# ACTIVIDADES UD8 - SERVIDOR

Mediante las tareas que se detallan en este documento vamos a conseguir la parte de los resultados de aprendizaje cubiertos por la UD8, atendiendo a sus correspondientes criterios de evaluación:

| RESULTADOS DE APRENDIZAJE | CRITERIOS DE EVALUACIÓN |
|---|---|
| RA4. Desarrolla aplicaciones Web embebidas en lenguajes de marcas analizando e incorporando funcionalidades según especificaciones. | d) Se han identificado y caracterizado los mecanismos disponibles para la autentificación de usuarios.<br>e) Se han escrito aplicaciones que integren mecanismos de autentificación de usuarios. |
| RA9. Desarrolla aplicaciones Web híbridas seleccionando y utilizando librerías de código y repositorios heterogéneos de información. | e) Se han utilizado librerías de código para incorporar funcionalidades específicas a una aplicación web. |

Consideraciones adicionales:

- Estas actividades se presentan en forma de **reto**, con unas pautas iniciales y el objetivo que se pretende conseguir, por tanto son actividades menos guiadas que las que hemos realizado hasta ahora.
- Se ha de continuar comentando el código mediante el código de tarea.
- Cualquier copia (ya sea de fuentes externas, literal de los apuntes...) que no sea una respuesta original será calificada con 0. El código no estructurado, que presente dificultades para se leído no podrá ser evaluado.

| **Actividad 1: Modificación del modelo de usuario** | | | | |
|---|---|---|---|---|
| **Código**: UD8.1 | **CE**: RA4.e (**20%**) | IE: I3, I6 | **Puntos**: 10 | **Estimación**: 1h |

**DESCRIPCIÓN**

Hasta el momento solo nos hemos autenticado mediante el administrador de Django en nuestra aplicación avaluapp. Para ello, hemos utilizado el nombre de usuario y la contraseña, pero queremos cambiar este comportamiento para que sea el e-mail el campo principal que identifica al usuario, no su nombre de usuario. En la actividad 5 trabajaremos más con el e-mail del usuario.

a) Primero crea una nueva app llamada "usuarios" y añádela a settings.py. (**1 punto**)

b) A continuación, en models.py de la nueva app, extiende la clase AbstractBaseUser para crear un nuevo modelo de usuario, llamado MyUser, cuyo campo principal sea el e-mail. Los campos del modelo serán: (**7 puntos**)

- username: puede ser nulo (**1 punto**)
- email: ha de ser único (**1 punto**)
- activo: verdadero o falso (**1 punto**)
- create_date: DateTimeField, con auto_now_add a True. (**1 punto**)
- update_date: DateTimeField, con auto_now a True. (**1 punto**)
- is_staff, is_active: dos campos booleanos, por defecto valor falso (**1 punto**)
- Además, has de definir un atributo USERNAME_FIELD, con el valor 'email'. Tienes un enlace más abajo donde explica el propósito de este atributo. (**1 punto**)

**NOTA**: Crea un Manager para este nuevo modelo, según el Anexo I, de forma que el comando de creación de superusuarios reconozca el e-mail como el campo identificador de usuario. Define este manager antes que el modelo MyUser en models.py. Tendrás que referenciar este manager en el modelo MyUser mediante el atributo objects (consulta el tercer enlace).

c) Aplica las migraciones. (**1 punto**)

d) Configura el administrador para poder gestionar el nuevo modelo. (**1 punto**)

**ENLACES**

- USERNAME_FIELD
- How to use email as username for Django registration
- Use the email field as username in Django

| **Actividad 2: Login y logout** | | | | |
|---|---|---|---|---|
| **Código**: UD8.2 | **CE**: RA4.d, RA4.e (**60%**) | IE: I3, I6 | **Puntos**: 10 por cada CE | **Estimación**: 3 h |

**DESCRIPCIÓN**

a) (RA4.d) Visualiza el siguiente vídeo, e implementa las funcionalidades de los siguientes

apartados (**10 puntos**)

b) (RA4.e) Pantalla de Login (**7 puntos**)

c) (RA4.e) Botón de Logout (**3 puntos**)

Tienes cierta libertad para implementar estas funcionalidades, y puedes utilizar más fuentes de información.

| **Actividad 3: Protección de vistas mediante decoradores** | | | | |
|---|---|---|---|---|
| **Código**: UD7.3 | **CE**: RA4.e (**20%**) | IE: I3, I6 | **Puntos**: 10 | **Estimación**: 30 min |

**DESCRIPCIÓN**

Visualiza primero el siguiente vídeo introductorio a los decoradores de Python. A continuación, visualiza este otro vídeo donde se explica cómo utilizar los decoradores para proteger determinadas vistas si el usuario no está autenticado en el sistema.

Con todo esto, esta actividad consiste en proteger todas las vistas de la parte privada de la aplicación, mediante el decorador login_required.

Si intentas acceder a una URL protegida sin estar autentificado en el sistema, la aplicación debería redireccionarte a la pantalla de login.

| **Actividad 4: Django Allauth** | | | | |
|---|---|---|---|---|
| **Código**: UD7.4 | **CE**: RA9.e (**80%**) | IE: I3, I6 | **Puntos**: 10 | **Estimación**: 4 h |

**DESCRIPCIÓN**

En la actividad 2 hemos implementado el login/logout de nuestra aplicación, pero los usuarios no pueden resetear su contraseña en caso de que la olviden, entre otras acciones.

En esta actividad vamos a explorar el paquete Django Allauth, que nos aporta multitud de formas de realizar la autenticación en nuestro sistema, resetear contraseñas, etc. En este enlace encontrarás todo lo referente a este paquete.

Con Django Allauth se pueden realizar multitud de tipos de autenticación (local, a través de plataformas de terceros como Facebook). Este paquete también viene con sus propias plantillas (pantalla de login, etc), aunque necesitan ser personalizadas para tener un aspecto más atractivo visualmente.

Esta actividad va a consistir en seguir los pasos de este enlace, instalar la última versión de Django Allauth y utilizar la plantilla de login que viene incorporada en el paquete. Esta plantilla nos aporta un enlace para resetear la contraseña. Nos hemos de poder autenticar en el sistema de igual modo que lo hacíamos en la actividad 2, pero en este caso a traveś de Django Allauth. Realiza los comentarios pertinentes para que las dos versiones de autenticación (según la actividad 2 y la 4) queden reflejadas en tu código.

Aquí tienes otro enlace que explica también este proceso, y que utilizaremos en la siguiente actividad.

| **Actividad 5: Envío de e-mails** | | | | |
|---|---|---|---|---|
| **Código**: UD7.5 | **CE**: RA9.e (**20%**) | IE: I3, I6 | **Puntos**: 10 | **Estimación**: 3h |

**DESCRIPCIÓN**

En la actividad 4 hemos implementado el login mediante Django Allauth, y en la actividad 1 hemos modificado el modelo de usuario para tomar el e-mail como el campo principal del modelo de usuario, y no el nombre de usuario.

En esta actividad vamos a configurar el envío de e-mails desde Django, para poder cerrar el ciclo de alta de usuario/login/logout/reseteo de contraseña.

Básate en el segundo enlace aportado en la actividad anterior para configurar el envío de e-mails desde Django, y así poder resetear la contraseña de nuestros usuarios.

Puedes utilizar tanto tu cuenta de Microsoft365 del centro, o cualquier otra cuenta personal (gmail por ejemplo) de la que dispongas

NOTA: Te puedes basar en cualquier otro enlace que encuentres de tu interés, pero por favor refléjalo como comentario en tu código.

## ANEXO I: MyUserManager

```
class MyUserManager(BaseUserManager):
    def create_user(
        self, email, first_name=None, last_name=None, password=None, type=None
    ):
        """
        Creates and saves a User with the given email and password.
        """
        if not email:
            raise ValueError("Ha de proporcionar un e-mail válido")

        user = self.model(
            email=self.normalize_email(email),
            first_name=first_name,
            last_name=last_name,
        )

        user.is_active = False
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, password):
        """
        Creates and saves a User with the given email and password.
        """
        if not email:
            raise ValueError("Ha de proporcionar un e-mail válido")

        user = self.model(email=self.normalize_email(email))

        user.set_password(password)
        user.is_staff = True
        user.is_active = True
        user.is_superuser = True
        user.save(using=self._db)
        return user
```

Acuérdate de insertar la siguiente línea en el modelo MyUser:

```
    objects = MyUserManager()
```