

Evidencia N°2

Plan de seguridad del Proyecto Integrador

1. Antes de comenzar:

- Descargar este documento de la Evidencia N°2, como Documento Word, así completan las consignas directamente y no tienen que empezar un documento desde cero.
- Un solo compañero debe subir el mismo documento descargado y subirlo a un Drive e invitar a los compañeros del mismo equipo.
- Leer y estudiar el material correspondiente de la Unidad II, que se encuentra en la sección de Material Obligatorio, en el Aula Virtual.

2. Requisitos de entrega:

- Nombrar el archivo *Documentación_Ciberseguridad_App[NombreDelProyecto]*, reemplazando *[NombreDelProyecto]* por el nombre real de su proyecto integrador (por ejemplo, *Documentación_Ciberseguridad_SangreYa*).
- Todos los integrantes deben participar y editar dicho documento.
- Una vez terminado y revisado por todo el equipo, descargarlo en formato PDF y agregar el enlace en la Wiki del repositorio del Proyecto Integrador
- Un solo compañero del equipo deberá compartir el enlace del documento colaborativo en la entrega en el Aula Virtual (configurar el Acceso general como Comentarista para la posterior revisión de la Docente).

3. Datos de los integrantes y del proyecto

- Integrantes:
 - o BALBASTRO Carlos Eduardo
 - o BENTIVOGLIO Ignacio Martín
 - o BOSQUE Ruben Dario
 - o FERNANDEZ Kiara

- o HEREDIA Eric Victor Hugo
- o LORENZATI María Florencia
- Enlace al repositorio de Programación Web (primer cuatrimestre): [1er cuatrimestre](#)
- Enlace al repositorio de Proyecto Integrador:
<https://github.com/1Adevelopers/FlorAppMovil>
- Descripción y objetivos del proyecto:

Descripción del proyecto: FlorApp es una solución tecnológica integral orientada a la concientización sobre la flora autóctona de Córdoba. El proyecto se compone de una Plataforma Web (desarrollada con Angular y Django REST Framework) pensada para la gestión administrativa central y el ABM completo de especies, la cual se extiende en una Aplicación Móvil nativa para Android para que los usuarios puedan consultar toda la información desde sus teléfonos de forma práctica.

Objetivos del proyecto

- **Enfoque Web:** Proveer una plataforma de gestión centralizada orientada a los administradores y docentes para dar de alta, modificar y administrar las fichas botánicas y los usuarios del sistema. Además se puede consultar la galería que contiene las fichas de plantas tanto en modo invitado como con acceso autenticado.
- **Enfoque Móvil:** Extender la experiencia a dispositivos Android mediante una aplicación orientada a estudiantes, niños, docentes y público en general, facilitando la consulta ágil de galerías y fichas de plantas tanto en modo invitado como con acceso autenticado.

4. Datos y principios CIA

Los datos de los usuarios que maneja nuestro proyecto **FlorAppMovil** son:

1. **Nombre y apellido:** se utiliza para identificar al usuario dentro de la aplicación.
2. **Correo electrónico:** se utiliza principalmente para el registro e inicio de sesión.
3. **Contraseña:** permite al usuario acceder a su cuenta.
4. **Rol del usuario:** permite identificar si se trata de un docente o un administrador y así determinar qué funciones puede utilizar.

De estos datos, consideramos que la **contraseña es el dato que necesita mayor protección**, ya que permite acceder a la cuenta del usuario. El nombre, apellido y correo electrónico también son datos personales, por lo que deben tratarse de forma segura y no exponerse innecesariamente.

Para protegerlos, se deberían utilizar mecanismos de autenticación seguros, no guardar las contraseñas directamente en texto plano y controlar el acceso según el rol de cada usuario.

También aplicamos los tres principios de seguridad **CIA**:

- **Confidencialidad:** los datos personales de los usuarios deben ser accesibles solamente para las personas autorizadas.
- **Integridad:** la información de los usuarios y los contenidos de la aplicación no deberían poder ser modificados por personas que no tengan los permisos necesarios.
- **Disponibilidad:** la aplicación y sus datos deberían estar disponibles cuando los usuarios autorizados necesiten utilizarlos.

Estos principios son importantes para que **FlorAppMovil** pueda proteger la información de sus usuarios y funcionar de una manera segura.

5. Vulnerabilidades del Backend

Reúnanse como equipo y revisen juntos, con el backend a mano, cada ítem de la siguiente lista.

Para cada punto, además de describir lo que encontraron, indiquen a cuál o cuáles categorías de OWASP Top 10 corresponde (puede ser más de una, según lo que vieron en el material): Pérdida de Control de Acceso, Configuración de Seguridad Incorrecta, Fallas en la Cadena de Suministro de Software, Fallas Criptográficas, Inyección, Diseño Inseguro, Fallas de Autenticación, Fallas en la Integridad del Software o de los Datos, Fallas en el Registro y Monitoreo, o Manejo Inadecuado de Condiciones Excepcionales.

a. Secretos escritos directamente en el código

Revisen si hay contraseñas, claves de acceso o datos de conexión a la base de datos escritos directamente en algún archivo de código, en vez de en un archivo de variables de entorno que no se suba al repositorio. Por ejemplo: `'PASSWORD': '1234'` escrito directo en el archivo de configuración, en vez de `'PASSWORD': os.getenv('DB_PASSWORD')`. Para revisarlo, busquen en los archivos de configuración de su proyecto cualquier contraseña o clave escrita literalmente entre comillas.

- **Resultado de lo encontrado:**

Se revisó el archivo de configuración central del backend (`backend/floraCbapp/settings.py`) y se constató que el proyecto **no tiene secretos escritos directamente en el código**, ya que, tanto la `SECRET_KEY` de Django como las credenciales de conexión a la base de datos MySQL (`DB_NAME`, `DB_USER`, `DB_PASSWORD`, `DB_HOST`, `DB_PORT`) se leen mediante `os.getenv()`, con el cual se cargan los valores reales desde un archivo `.env` externo.

Se verificó que el archivo `.env` se encuentra correctamente excluido del control de versiones, figurando en el `.gitignore` de la raíz del proyecto como en el propio de la carpeta `backend/`.

Se realizó una revisión por archivos Python del backend (modelos, vistas y serializers) y tampoco se encontraron contraseñas, claves de API ni otros datos secretos.

- **Categoría de OWASP Top 10 correspondiente:**

No aplica, ya que no se detectó ningún secreto hardcodeado en el código fuente de la app.

b. Modo de desarrollo activo

Revisen si el proyecto tiene activado, en el ambiente donde está corriendo ahora, algún modo de desarrollo o depuración que muestre información técnica detallada ante un error. Por ejemplo: `DEBUG = True` escrito fijo en el archivo de configuración, en vez de depender de una variable que se pueda cambiar a `False` al desplegar. Para revisarlo, provoquen un error a propósito (por ejemplo, pidiendo algo que no existe con la siguiente URL: `http://127.0.0.1:8000/noExiste`) y vean si la respuesta muestra código interno, rutas del servidor o datos de configuración, en vez de un mensaje simple.

- **Resultado de lo encontrado:**

Al revisar el archivo de configuración central (`floraCbapp/settings.py`), se constató que el modo de depuración está configurado como **DEBUG = os.getenv('DEBUG') == 'True'**. Si bien se utiliza una variable de entorno para evitar dejar el valor `True` fijo en el código fuente, en el entorno de desarrollo local la variable se encuentra activa (**DEBUG=True**). Al realizar la prueba de provocar un error intencional solicitando una ruta inexistente (por ejemplo, **http://127.0.0.1:8000/noExiste**), el servidor responde con la pantalla de error detallada de Django en lugar de un error 404 genérico. Esta vista expone información técnica crítica: el listado completo de URLs registradas en el backend (**api/usuarios/**,

api/flora/, api/interacciones/), rutas absolutas del servidor en disco, la versión exacta de Django y detalles del entorno de ejecución, lo cual facilita la recolección de información a posibles atacantes.

- **Categoría de OWASP Top 10 correspondiente:**

A05:2021 – Configuración de Seguridad Incorrecta (Security Misconfiguration).

c. Orígenes permitidos

- **Resultado de lo encontrado:**

Luego de la revisión del backend, notamos, que la configuración de CORS (CORS_ALLOWED_ORIGINS) dentro de la aplicación principal de django, en "settings.py" actualmente permite peticiones desde <http://localhost:4200> y desde <http://127.0.0.1:4200>, que corresponden al entorno virtual de desarrollo de FlorApp en el cuatrimestre pasado (peticiones de Angular, permitiendo que se comuniquen el Frontend con el Backend y que no se repelan entre sí). Sin embargo, aún no está configurado para funcionar en la nueva extensión móvil de FlorApp en Android nativo (FlorAppMovil).

- **Categoría de OWASP Top 10 correspondiente:** A02 - Configuración de seguridad incorrecta.

d. Endpoints sin autenticación

Revisen si alguno de los endpoints que devuelven datos de usuarios, pedidos o carritos responde igual sin haber iniciado sesión. Por ejemplo: pedir la lista completa de usuarios o pedidos desde el navegador o Postman o Swagger, sin haber iniciado sesión, y recibirla igual. Para revisarlo, prueben acceder a esos endpoints sin loguearse y vean si el sistema les niega el acceso o les devuelve los datos igual.

- **Resultado de lo encontrado:**

Al evaluar el comportamiento de la API REST en Django, se constató que la ruta destinada a la gestión y consulta de usuarios (/api/usuarios/) no exige un token de autenticación JWT para peticiones de lectura (GET). Al realizar una solicitud HTTP directa desde herramientas como Postman o el navegador sin enviar el encabezado Authorization: Bearer <token>, el backend responde con un código 200 OK y entrega la lista completa con los datos de las cuentas registradas en el sistema. Esto expone públicamente información sensible como nombres de usuario, correos electrónicos y roles de los docentes, facilitando la recolección de datos por parte de terceros no autorizados.

- **Categoría de OWASP Top 10 correspondiente:**

A01:2021 – Control de Acceso Roto (*Broken Access Control*) / A07:2021 – Fallos de Identificación y Autenticación (*Identification and Authentication Failures*).

e. **Control de acceso por ID**

Revisen si el sistema permite acceder, editar o eliminar registros pertenecientes a otros usuarios cambiando únicamente el identificador numérico (ID) en la URL sin validar quién es el propietario o si tiene permisos suficientes

- **Resultado de lo encontrado:**

Durante la prueba de control de acceso, se constató que al iniciar sesión con una cuenta de docente (ej *lucho@hotmail.com*) y modificar de forma manual el parámetro numérico en la barra de direcciones de la interfaz de edición (cambiando la URL de */docentes/editar-planta/14* a */docentes/editar-planta/13*), la aplicación no valida los permisos de autoría. Como resultado, el sistema expone y permite visualizar la ficha de especie "Margarita punzo" que pertenece a otro usuario. Al alterar algún campo (ej categoría de "arbusto" a "hierba") y presionar guardar, el registro se modifica y la propiedad de la planta se reasigna automáticamente al usuario que realizó la edición. Posteriormente, al

ingresar con otra cuenta de docente distinta (ej *ana@example.com*), es posible acceder a esa misma ficha, modificarla nuevamente y eliminarla por completo del sistema sin ningún tipo de restricción de seguridad o validación de propietario.

- **Categoría de OWASP Top 10 correspondiente:** A01:2021 - Broken Access Control (Control de acceso roto - Exposición y modificación no autorizada de recursos por ID secuencial)

f. Dependencias reconocidas

Revisen si reconocen para qué sirve cada librería o paquete externo que usa su proyecto. Por ejemplo: encontrar en el archivo de dependencias dos o tres librerías que nadie del equipo recuerda haber agregado ni para qué sirven. Para revisarlo, repasen el archivo donde su proyecto lista sus dependencias (requirements.txt, package.json, pom.xml, u otro según su lenguaje), y marquen las que no sepan identificar o no estén usando realmente.

- **Resultado de lo encontrado:**

Se revisaron las dependencias declaradas en los archivos requirements.txt del backend y package.json del frontend. Las dependencias encontradas son reconocidas por el equipo y corresponden a herramientas utilizadas para el desarrollo y funcionamiento del proyecto tanto en frontend como en backend.

En el backend se utilizan Django, Django REST Framework, MySQL Client, python-dotenv, entre otras dependencias necesarias para el funcionamiento de la aplicación. En el frontend se utilizan las principales dependencias de Angular, Bootstrap, RxJS y TypeScript, además de herramientas de desarrollo y testing.

Durante el proceso de la revisión no se han encontrado dependencias desconocidas para el equipo y paquetes sin utilizar

- **Categoría de OWASP Top 10 correspondiente:**

A06:2021 – Componentes Vulnerables y Desactualizados (*Vulnerable and Outdated Components*).

Aunque en este chequeo en particular el resultado fue positivo (no se detectaron paquetes desconocidos o en desuso), el análisis de la lista de dependencias (requirements.txt en Django y package.json en Angular) se realiza bajo el marco del riesgo **A06:2021**, cuyo objetivo es prevenir la inclusión de bibliotecas obsoletas, no identificadas o con vulnerabilidades conocidas en la cadena de suministro de software (*Software Supply Chain*).

A06:2021 – Componentes vulnerables y desactualizados.

Esta categoría se relaciona con el control de las dependencias utilizadas por el proyecto, ya que es importante mantenerlas identificadas y actualizadas para reducir posibles vulnerabilidades. Durante la revisión realizada no se detectó una dependencia desconocida o claramente innecesaria.

6. Plan de Seguridad

A partir de lo que encontraron en el punto anterior, armen un plan concreto para cada ítem donde detectaron un problema. Este plan será la base de lo que van a implementar en el código durante la Evidencia N°3, así que piensen en soluciones que realmente puedan llevar a cabo con lo que van aprendiendo en la materia. Para cada ítem de la lista anterior donde hayan encontrado un problema, respondan:

a. Secretos escritos directamente en el código

- ¿Qué van a hacer para solucionarlo?

Como en la revisión del backend no se encontraron secretos expuestos en el código, no se requiere una corrección puntual. No obstante sería oportuno optar como medida preventiva, para mantener buenas prácticas, se establece:

- 1- Revisar en cada Pull Request que no se agreguen variables sensibles directamente en el código.
- 2- Mantener actualizado el archivo `.env_modelo` cada vez que se sume una nueva variable de entorno, documentando su nombre sin exponer su valor verdadero.
- 3- Verificar antes de cada despliegue que el archivo `.env` de producción no quede expuesto públicamente ni se suba por error al repositorio.

b. Modo de desarrollo activo

- ¿Qué van a hacer para solucionarlo?
- **Desactivación estricta en producción:** Asegurar que en el archivo `.env` del entorno de producción o despliegue la variable `DEBUG` esté explícitamente configurada en `False` (`DEBUG=False`), garantizando que por defecto nunca se active el modo depuración si la variable no está definida.
- **Ajuste de hosts permitidos (`ALLOWED_HOSTS`):** Definir y restringir mediante variables de entorno los dominios e IPs autorizados que podrán consultar el backend cuando `DEBUG=False`, evitando solicitudes con encabezados `Host` manipulados.
- **Páginas y respuestas de error personalizadas:** Implementar vistas de error estándar (handlers para HTTP 404, 403 y 500) en el backend y en Django REST Framework que respondan únicamente con mensajes simples en formato JSON (por ejemplo, `{"detail": "Recurso no encontrado"}`), neutralizando cualquier fuga de rutas internas, configuraciones o trazas de código (*tracebacks*) ante fallos imprevistos.

c. Orígenes permitidos

- ¿Qué van a hacer para solucionarlo?
 - **Actualización de orígenes autorizados en el Backend:** Configurar e integrar dentro de la aplicación principal de django "[settings.py](#)", la dirección IP del entorno de desarrollo móvil (dominio del servidor), dentro de la lista de CORS (`CORS_ALLOWED_ORIGINS`), permitiendo

que la app nativa Android (FlorAppMovil) pueda consumir la API sin ser bloqueada por el servidor.

d. Endpoints sin autenticación

- ¿Qué van a hacer para solucionarlo?

Se implementará autenticación mediante **JWT** en los endpoints que manejan información de usuarios y otros recursos que no deban ser públicos. Se configurarán permisos en Django REST Framework para que los endpoints que contienen información privada requieran un usuario autenticado y un token válido. De esta manera, una petición realizada sin autenticación deberá ser rechazada por el servidor. También se revisarán los permisos de cada endpoint para determinar cuáles pueden ser consultados públicamente y cuáles deben estar protegidos según el rol del usuario, evitando que cualquier persona pueda obtener información de otros usuarios.

e. Control de acceso por ID

- ¿Qué van a hacer para solucionarlo?
 - **Validación de autoría en el Backend:** Implementar una verificación en los endpoints del servidor (controladores de plantas) para que, antes de permitir actualizar o eliminar un recurso, se compruebe que el ID del usuario autenticado (extraído de la sesión o token JWT) coincida exactamente con el docente propietario de esa planta. Si no coincide, el servidor deberá rechazar la petición retornando un error de acceso denegado (403 Forbidden).
 - **Blindaje de rutas:** Asegurar que las consultas a la base de datos incluyan siempre el filtro por el ID del usuario logueado, evitando que se puedan buscar o modificar registros ajenos simplemente alterando los parámetros numéricos de la URL.

f. Dependencias reconocidas

- ¿Qué van a hacer para solucionarlo?

Durante la revisión no se encontraron dependencias desconocidas o claramente innecesarias, por lo que no se requiere eliminar ninguna dependencia en este momento.

Como medida preventiva, el equipo realizará revisiones periódicas de los archivos `requirements.txt` y `package.json` para mantener identificadas las dependencias utilizadas por el proyecto.

También se revisarán las actualizaciones disponibles antes de incorporarlas, verificando su compatibilidad con la aplicación y evitando mantener paquetes que dejen de ser necesarios. De esta manera se busca reducir posibles riesgos relacionados con componentes vulnerables o desactualizados.

Nota: si en algún ítem del punto anterior no encontraron ningún problema, pueden mencionar que no han encontrado ningún problema.

7. Autenticación

Este punto depende de en qué etapa esté su proyecto, respondan la opción que corresponda a su equipo:

- **Si ya implementaron JWT u otro método de autenticación en su backend:** describan cómo lo armaron. ¿Qué información guarda el token? ¿Por cuánto tiempo es válido antes de vencer? ¿Qué reglas de complejidad exige el sistema para las contraseñas al registrarse (cantidad mínima de caracteres, combinación de letras, números y símbolos)?
- **Si todavía no lo implementaron:** este es el momento de empezar a diseñarlo, aunque sea a nivel de idea general. Un token es una credencial que el servidor le entrega al usuario después de un login exitoso, y que la aplicación vuelve a presentar en cada pedido posterior para demostrar que ya inició sesión, sin tener que enviar la contraseña de nuevo cada vez. ¿Qué método de autenticación piensan usar (JWT es el más común, pero no el único)? ¿Por qué les parece la opción adecuada para su proyecto? ¿Qué reglas de

complejidad piensan exigir para las contraseñas (cantidad mínima de caracteres, combinación de letras, números y símbolos)?

Estado Actual

El backend todavía no implementa ningún mecanismo de autenticación basada en tokens. El endpoint de login (*POST/api/usuarios/login/*) compara el email y la contraseña ingresados directamente contra valores guardados en la base de datos.

Esto significa que las contraseñas se guardan y se comparan en texto plano, sin ningún tipo de hash. Esto significa e implica una falla de autenticación concreta, ya que si alguien accediera a la base de datos, tendría las contraseñas de todos los usuarios.

Por lo tanto se propone el siguiente diseño:

- 1- Hashear las contraseñas usando las utilidades de Django para dejar de guardarlas y compararlas con texto plano. Teniendo en cuenta que hay que resolverlo antes de la migración a JWT para que la contraseña viaje y se almacene cifrada.
- 2- Utilizar JWT con *django-rest-framework-simplejwt* ya que se integra directamente con Django REST Framework y se puede consumir desde Android con alguna librería como Retrofit adjuntando el token en el header *Authorization*.
- 3- El token debe guardar: ID de usuario, email y rol, para identificar permisos sin consultar la base de datos en cada petición.
- 4- Implementar una duración *access token* de 15-30 minutos y un refresh token de 7 días, para renovar sin volver a loguearse.
- 5- Establecer políticas de contraseñas que determinen cantidad mínima de 8 caracteres, combinando mayúsculas, minúsculas y números.