Proyecto de Software 2024

Trabajo Integrador (TI) - Etapa 1

Contexto

El Centro de Equitación para Personas con Discapacidad y Carenciadas, CEDICA, es una Asociación Civil sin Fines de Lucro, fundada en 1994 en la ciudad de La Plata, Provincia de Buenos Aires, Argentina. CEDICA trabaja con la finalidad de IGUALAR OPORTUNIDADES en procura de la reinserción familiar, el logro de la escolaridad, la integración laboral y la mejora integral de la calidad de vida de los J&A ("Jinetes" y "Amazonas") que forman parte de la institución.

Las actividades que se realizan se denominan Terapias y Actividades asistidas con Caballos –TACAs-, aunque históricamente también fueron conocidas con otros términos como Rehabilitación Ecuestre o Equinoterapia. Se trata de una terapia integral que apunta a la recuperación de la persona en sus dimensiones biológica, psíquica y social. El principal recurso de trabajo es el Caballo, como mediador y facilitador de las intervenciones terapéuticas que cada profesional de la salud o la educación planifica para las diferentes Personas con Discapacidad que asisten a la institución.

CEDICA cuenta con un Equipo de trabajo integrado por personal ecuestre, técnicos y profesionales: Terapistas Ocupacionales (T.O.), Psicólogos, Psicopedagogos, Psicomotricistas, Profesores de Educación Física y Equitación; Instructores de Volteo; Médico Veterinario, capacitados en TACAs. A ello se suman anualmente una gran cantidad de voluntarios, en su mayoría estudiantes universitarios de la UNLP y la UNQUI; y el plantel se completa con los caballos en actividad y potrillos en etapa de entrenamiento.

Este Equipo de trabajo se divide en 5 grandes Áreas, a saber: Área Técnica, Área Ecuestre, Área de Voluntariado, Área de Gestión y Administración; y Área de Capacitación. Cada una de ellas, cuenta con sus propios Procesos de Trabajo para cumplir con la tarea y los objetivos institucionales.

Objetivo general

El objetivo de este trabajo integrador es desarrollar una Aplicación Web que permita contar con mucha de la información en formato digital que contienen los diferentes procesos de trabajo de una institución de TACAs. Los usuarios directos serían los integrantes del equipo de CEDICA.

La solución tendrá un aplicación interna de administración (para usuarios y administradores) desarrollada en Python y Flask, y un portal web desarrollado en Vue.js donde se podrán visualizar los servicios ofrecidos por la institución. Utilizaremos una base de datos PostgreSQL y se implementará una API con los endpoints necesarios para la integración de estas dos aplicaciones.

Funcionalidad de la aplicación

La aplicación permitirá:

- **Mantener un registro histórico de los legajos de los J&A (Jinetes y Amazonas), incluyendo anexos de documentación necesaria.**
- **Mantener un registro de los Legajos de los profesionales del equipo.**
- **Registrar la información de los caballos.**
- **Obtener reportes estadísticos:** la aplicación debe producir reportes que presenten estadísticas relevantes sobre los datos manejados.

Componentes de la aplicación

Aplicación de administración en Python y Flask: desarrollar una aplicación que permita la administración de altas, bajas y modificaciones de los recursos necesarios para el sistema. También se deberá implementar un registro de usuarios y un sistema de autenticación para operar con la aplicación.
Portal en Vue.js: crear una interfaz de usuario interactiva que permita visualizar los diferentes servicios que brinda la institución (segunda etapa).
Base de datos PostgreSQL: diseñar la estructura de la base de datos para almacenar la información de las instituciones, servicios y usuarios de la aplicación.
API para consultas: implementar API que permitan realizar las consultas necesarias para obtener la información que se quiera visualizar en el portal.
Es importante que el trabajo sea desarrollado en equipo, asignando roles y tareas a cada miembro para garantizar una implementación eficiente y exitosa de la aplicación web. Durante el cuatrimestre, se realizarán reuniones periódicas para monitorear el progreso y resolver dudas.
Al finalizar el cuatrimestre, cada equipo deberá presentar las aplicaciones web completas y funcionales, demostrando conocimiento sobre todas las funcionalidades requeridas y su correcto despliegue en un entorno de prueba.

Aplicación Privada

Esta aplicación será utilizada tanto por los administradores del sistema que tienen el acceso a la administración de los usuarios, como también los distintos usuarios con roles delimitados según el área correspondiente.

Esta aplicación será desarrollada utilizando el framework Flask con el lenguaje python y la base de datos Postgres.

1 Layout

Se deberá implementar el layout de la aplicación que es la base para todas las vistas de la aplicación privada. El resto de las vistas estarán contenidas en este layout sobreescribiendo el contenido central.

La aplicación deberá contar con un menú de navegación que tenga los enlaces a todos los módulos del sistema y esté visible en aquellas vistas del sistema que se consideren necesarias.

Se debe incluir una barra superior en la aplicación que nos permita acceder a las operaciones principales de nuestra aplicación, Ejemplo en Figura 1.

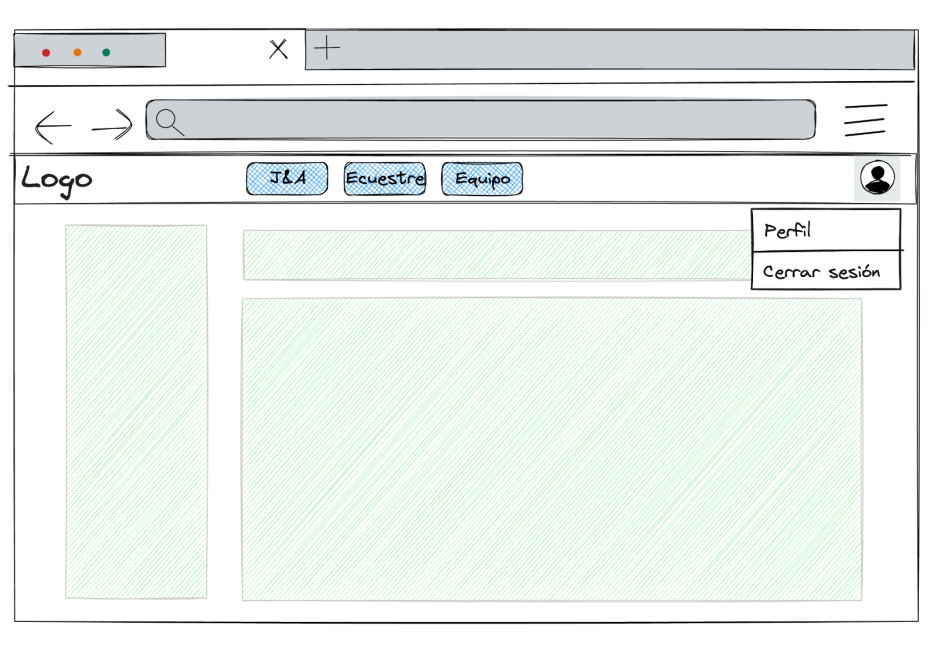


Figura 1. Posible visualización de top bar de aplicación privada.

*Logo de referencia***:**

En la Figura 1 se pueden ver las siguiente opciones de derecha a izquierda:

- **Dropdown de usuario/cuenta**. Permite realizar operaciones sobre la cuenta de usuarios y la sesión.
- **Botón de acceso a cada una de los módulos.** Permite acceder a la funcionalidad que gestiona los usuarios, jinetes y amazonas, caballos y portal público. Dependiendo del rol de usuario tendrá accesos de escritura o solo lectura.

Aclaración: No es necesario seguir este boceto a la perfección. El objetivo del mismo es sólo mostrarles la funcionalidad que debería estar fácilmente disponible para el usuario en la barra superior, menú o submenú.

Se debe usar el logo suministrado para la aplicación y mostrar el mismo en forma coherente en todas las secciones de la aplicación.

2 Módulo de usuarios

Desarrollar el **módulo de usuarios** que debe permitir al **System Admin** realizar distintas operaciones sobre los usuarios del sistema.

El módulo debe permitir el **CRUD** de usuarios. Deben validar que no existan dos usuarios con el mismo nombre de usuario (mismo email).

Se considerarán al menos los siguientes datos para cada usuario:

- **Email**
- **Alias**
- **Password**
- **Activo**: SI | NO
- **Roles**: Técnica | Ecuestre | Voluntariado | Administración

Se deben poder realizar búsquedas sobre los usuarios, **al menos** por los siguientes campos:

- **Email**
- **Activo:** SI | NO
- **Rol:** Técnica | Ecuestre | Voluntariado | Administración

Los resultados se deben poder ordenar por: **email**, **fecha de creación** del usuario. Se debe implementar el orden de manera **ascendente y descendente**.

Atención: El resultado de la búsqueda debe estar paginado. La paginación deberá realizarse del lado del servidor, es decir, la consulta a la base de datos debe pedir un número máximo de registros en la respuesta, por ej. 25 registros por página. Tomaremos como un error si se consulta a la base de datos sin definir un límite y se página posteriormente.

Se debe desarrollar la funcionalidad para activar o bloquear un usuario: un usuario bloqueado no podrá acceder al sistema. Se deberá validar que los únicos usuarios que no puedan ser bloqueados, sean aquellos con el rol **System Admin**.

Además se debe poder asignar o desasignar el rol correspondiente al usuario.

En referencia a los roles anteriormente mencionados podemos decir que un **System Admin** es un usuario que no tiene restricción de acciones (es decir, **puede hacer todo**). En contrapunto los usuarios con los roles **Técnica, Ecuestre, Voluntariado y Administración** son los roles que se pueden asignar a los usuarios que trabajan para las distintas áreas de la institución.

Para el desarrollo del TI *no será obligatorio desarrollar el CRUD de los roles y permisos, podrán administrarse* desde la base de datos. Los usuarios, roles y permisos sólo podrán ser administrados por un usuario **System Admin** para esta sección.

Los permisos necesarios asociados a cada rol deberán deducirse del enunciado, ante la duda pueden **consultar a su ayudante**.

El nombre de los permisos deberá respetar el patrón **modulo_accion**, por ejemplo, el módulo de gestión de usuarios deberá contemplar los siguientes permisos:

- **user_index**: permite acceder al index (listado) del módulo.
- **user_new**: permite crear un usuario.
- **user_destroy**: permite eliminar un usuario.
- **user_update**: permite actualizar los datos de un usuario.
- **user_show**: permite visualizar la información de un usuario.

Nota: Es importante entender el porqué del uso de esta solución para implementar la autorización y seguir el esquema de forma correcta. Este tema será explicado oportunamente en los horarios de práctica.

La Figura 2 muestra un posible modelo de usuarios, roles y permisos, que pueden utilizar en el trabajo.

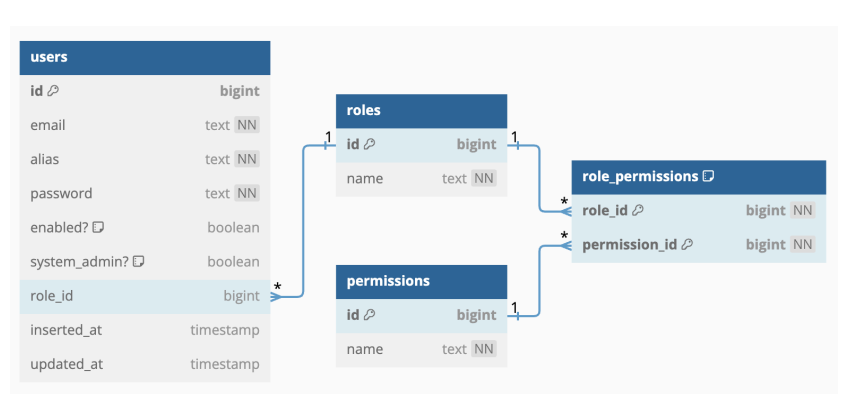


Figura 2. Posible esquema para el manejo de usuarios.

A continuación se definen los roles y las acciones que pueden realizar sobre este módulo:

- **System Admin:** puede hacer todo.

3 Login y manejo de sesiones

El sistema deberá contar con un **formulario de login** que permita a los usuarios iniciar sesión en el sistema. Además del inicio de sesión convencional (con correo electrónico y clave), **en la etapa 2** deberá existir la opción de poder iniciar **sesión con Google.**

Deberá implementarse un manejo de sesiones adecuado, verificando la sesión y permisos cuando corresponda. Para cada módulo se indicará si requiere autenticación o no y los permisos necesarios.

Atención: No está permitido el uso de ninguna librería externa que simplifique el proceso de login (ejemplo: flask-login). Deberán realizar la implementación completa de esta funcionalidad sin el uso de librerías que podrían ocultar comportamiento importante que queremos que aprendan y fijen en esta materia.

4 Módulo de Equipo

En este módulo se realiza la administración de todos los empleados de CEDICA. Usuarios con rol de **Administración** podrán dar de alta, modificar o eliminar los distintos empleados de CEDICA en este módulo.

Se debe tener en cuenta que un miembro del equipo además **puede estar asociado a un usuario del sistema o no.**

La información mínima que debemos almacenar de cada miembro del equipo es:

- **Nombre**
- **Apellido**
- **DNI**
- **Domicilio**
- **Email**
- **Localidad**
- **Teléfono**
- **Profesión** (Psicólogo/a, Psicomotricista, Médico/a, Kinesiólogo/a, Terapista Ocupacional, Psicopedagogo/a, Docente, Profesor, Fonoaudiólogo/a, Veterinario/a, Otro.)
- **Puesto laboral en la Institución** (Administrativo/a, Terapeuta, Conductor, Auxiliar de pista, Herrero, Veterinario, Entrenador de Caballos, Domador, Profesor de Equitación, Docente de Capacitación, Auxiliar de mantenimiento, Otro.)
- **Fecha de inicio**
- **Fecha de Cese**
- **Contacto de Emergencia** (Indicar Nombre y teléfono)
- **Obra Social** (Texto libre. Es opcional si quieren administrar las obras sociales en una tabla)
- **N° de Afiliado**
- **Condición** (Voluntario, Personal Rentado)
- **Activo** (si o no)

A cada miembro del equipo se le debe poder subir archivos con documentación complementaria. Ejemplo:

- **Título**
- **Copia DNI**
- **CV Actualizado**

Nota: Habrá una explicación práctica que explique cómo realizar el upload de distintos documentos. Deberán utilizar la herramienta Minio instalada por la cátedra como object storage.

Se deben poder realizar búsquedas sobre los miembros del equipo, **al menos** por los siguientes campos:

- **Nombre**
- **Apellido**
- **DNI**
- **Email**
- **Puesto Laboral:** Administrativo/a | Terapeuta | Conductor | Auxiliar de pista | Herrero | Veterinario | Entrenador de Caballos | Domador | Profesor de Equitación | Docente de Capacitación | Auxiliar de mantenimiento | Otra.

Los resultados se deben poder ordenar por: **nombre**, **apellido, fecha de creación**. Se debe implementar el orden de manera **ascendente y descendente**.

A continuación se definen los roles y las acciones que pueden realizar sobre este módulo:

- **Administración:** index, show, update, create, destroy.

5 Registro de pagos

Se debe permitir a los usuarios con rol gestión administrativa gestionar el registro del pago de honorarios a los empleados de CEDICA o cualquier otro gasto que desee registrarse de la institución . Este será un módulo muy simple que sólo permite llevar el registro de esos pagos.

Por cada registro se debe solicitar:

- **Beneficiario** (empleado dado de alta en el sistema si es pago de honorarios, si aplica)
- **Monto**
- **Fecha de pago**
- **Tipo de pago** (Honorarios, proveedor, gastos varios)
- **Descripción**

Se deben poder realizar búsquedas, **al menos** por los siguientes campos:

- **Rango de fechas de pago.**
- **Tipo de pago:** Honorarios | proveedor | gastos varios

Los resultados se deben poder ordenar por: **fecha de pago**. Se debe implementar el orden de manera **ascendente y descendente**.

A continuación se definen los roles y las acciones que pueden realizar sobre este módulo:

- **Administración:** index, show, update, create, destroy.

6 Módulo J&A (Jinetes y Amazonas)

Esta sección es donde se realiza la administración de todo lo relacionado con Jinetes y Amazonas. Muestra un listado de los mismos permitiendo entrar a cada registro para ver la información o para realizar operaciones sobre cada uno de manera individual, según el rol que se tenga asignado.

Este módulo permite a un usuario de **Técnica** la carga de nuevos legajos de J&A y otra documentación complementaria.

El módulo debe permitir el **CRUD** de J&A. Se deben poder listar las mismas paginadas y se debe incluir filtro que busque al menos en los atributos nombre y apellido.

Los datos necesarios para cargar una nuevo legajo incluye entre otros los siguientes (ver link a la Ficha Completa para los demás campos necesarios) :

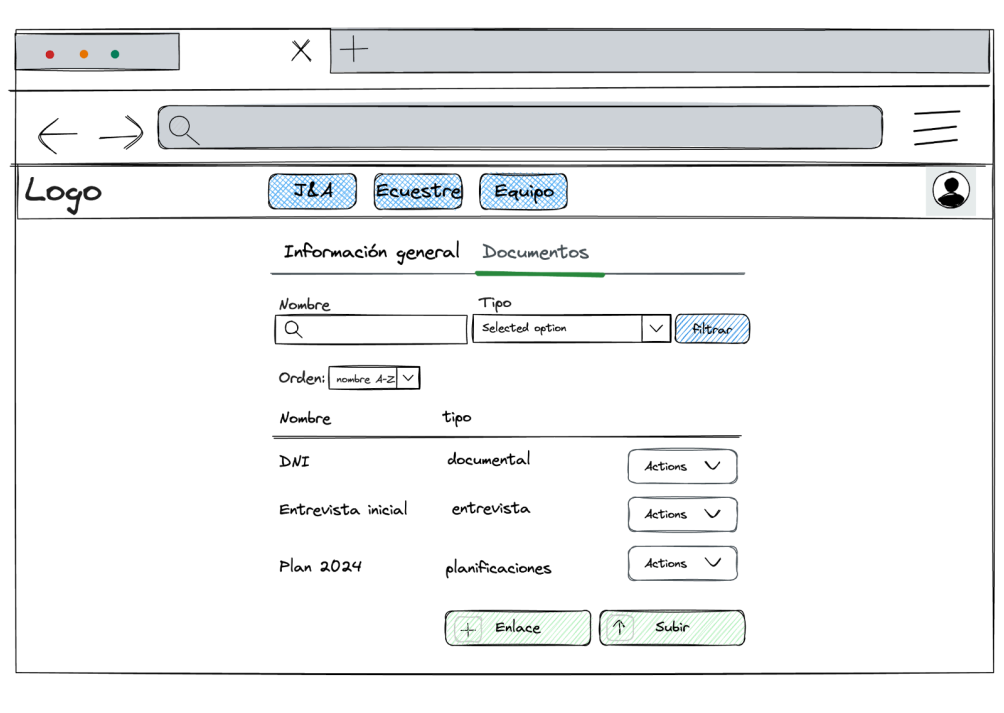
- **Nombre**
- **Apellido**
- **DNI**
- **Edad**
- **Fecha de nacimiento**
- **Lugar de nacimiento** (localidad y provincia)
- **Domicilio actual** (calle, número, departamento, localidad, provincia)
- **Teléfono actual**
- **Contacto de emergencia**
- **Tel**
- **Becado** (si/no)
- **Porcentaje de beca**
- **Profesionales que lo atienden** (campo libre)

Se deben poder realizar búsquedas, **al menos** por los siguientes campos:

- **Nombre**
- **Apellido**
- **DNI**
- **Profesionales que lo atienden**

Los resultados se deben poder ordenar por: **nombre** y **apellido.** Se debe implementar el orden de manera **ascendente y descendente**.

Adicionalmente a cada legajo del jinete o amazonas se le deberá poder cargar **múltiples** archivos (formatos permitidos PDF, DOC, XLS, JPEG) o enlaces a archivos externos (por ej, a un Drive, Dropbox, etc). Los mismos aportan información complementaria. Junto con la subida del archivo o link deberá asociarse un tipo de documentación (*entrevista, evaluación, planificaciones, evolución, crónicas, documental*) como una forma de clasificarlos. El tipo de documentación nos va a permitir organizar los archivos y encontrarlos fácilmente.



De esta manera cuando entremos al legajo de la persona, se deberá mostrar la información de su legajo y se listarán todos los documentos asociados. Este listado deberá incluir el título, fecha de subida, tipo y las operaciones permitidas para el documento: editar, eliminar, descargar.

En el caso de la documentación asociada con links externos, se deberá incluir el título, fecha en la que se agregó el enlace, tipo y las operaciones permitidas para el enlace: editar, eliminar, ir al documento.

Se deben poder realizar búsquedas sobre los documentos, **al menos** por los siguientes campos:

- **Título**
- **Tipo:** entrevista | evaluación | planificaciones | evolución | crónicas | documental

Los resultados se deben poder ordenar por: **título** y **fecha de subida/enlace.** Se debe implementar el orden de manera **ascendente y descendente**.

A continuación se definen los roles y las acciones que pueden realizar sobre este módulo:

- **Técnica:** index, show, update, create, destroy.
- **Administración:** index, show, update, create, destroy.
- **Ecuestre:** index, show.

7 Registro de cobros

El sistema debe permitir registrar los pagos de los J&A, los cuales deberán mostrarse listados por fecha. Simplemente se permitirá llevar el registro de los mismos. No hace falta incluir una fórmula de cálculo. Los usuarios con el rol de **Administración** serán los encargados de dar de alta los registros y marcar de alguna manera si algún jinete o amazona está en deuda.

El mismo debe incluir los siguientes campos:

- **J&A** (referencia al J&A cargado en el sistema.)
- **Fecha del Pago**
- **Medio de Pago** (Efectivo, Tarjeta de Crédito, Tarjeta de Débito, etc)
- **Monto**
- **Quien recibe dinero** (miembro del equipo dado de alta en el sistema)
- **Observaciones**

Adicionalmente el usuario debe poder registrar que la persona se encuentra al día con los pagos o tiene deudas.

Se deben poder realizar búsquedas, **al menos** por los siguientes campos:

- **Rango de fechas de pago**
- **Medio de pago** (Efectivo, Tarjeta de Crédito, Tarjeta de Débito, etc)
- **Nombre de quien recibe el dinero**
- **Apellido de quien recibe el dinero**

Los resultados se deben poder ordenar por: **fecha de pago**. Se debe implementar el orden de manera **ascendente y descendente**.

A continuación se definen los roles y las acciones que pueden realizar sobre este módulo:

- **Administración:** index, show, update, create, destroy.
- **Técnica:** index, show.

8 Módulo Ecuestre

Este módulo permite la gestión de la información relativa a los caballos utilizados en las terapias.

Se deben poder realizar todas las operaciones **CRUD** de cada registro del animal. Los registros deberán verse paginados.

Al igual que la sección anterior muestra inicialmente un listado de caballos con filtros y al entrar a un caballo particular se pueden realizar las distintas operaciones.

Los datos básicos necesarios para cargar un registro son los siguientes:

- **Nombre**
- **Fecha Nacimiento**
- **Sexo**
- **Raza**
- **Pelaje**
- **Compra/Donación**
- **Fecha de Ingreso**
- **Sede Asignada**
- **Asociar Entrenadores y Conductores** (son miembros del Equipo dados de alta en el sistema)
- **Tipo de J&A Asignados** (Hipoterapia, Monta Terapéutica, Deporte Ecuestre Adaptado, Actividades Recreativas, Equitación)

Se deben poder realizar búsquedas, **al menos** por los siguientes campos:

- **Nombre**
- **Tipo de J&A Asignados:** Hipoterapia | Monta Terapéutica | Deporte Ecuestre Adaptado | Actividades Recreativas | Equitación.

Los resultados se deben poder ordenar por: **nombre, fecha de nacimiento, fecha de ingreso**. Se debe implementar el orden de manera **ascendente y descendente**.

De la misma forma que en el módulo de J&A, se deberá permitir adjuntar documentación complementaria y/o enlaces a archivos externos, lo cual deberá ser listado al momento de ingresar a la ficha del animal. El tipo de documentación a anexar incluye:

- **Ficha general del caballo**
- **Planificación de Entrenamiento**
- **Informe de Evolución**
- **Carga de Imágenes**
- **Registro veterinario**

Aplicar las mismas posibilidades de filtros y ordenamiento de la sección de documentos para los J&A pero adaptada a estos tipos de documentos.

A continuación se definen los roles necesarios para las acciones:

- **Ecuestre:** index, show, update, create, destroy.
- **Administración:** index, show.
- **Técnica:** index, show.

Importante

Consideraciones técnicas

- El prototipo debe ser desarrollado utilizando Python, JavaScript, HTML5, CSS3 y PostgreSQL, y **respetando el modelo en capas MVC**.
- El código deberá escribirse siguiendo las guías de estilo de Python.
- El código Python deberá ser documentado utilizando docstrings.
- **El uso de jinja como motor de plantillas es obligatorio para la aplicación privada**.
- Se debe utilizar Flask como framework de desarrollo web para la aplicación privada.
- Se deberán realizar **validaciones de los datos de entrada** tanto del lado del cliente como del lado del servidor. Para *las validaciones del lado del servidor se deben realizar en un módulo aparte* que reciba los datos de entrada y devuelva el resultado de las validaciones. En caso de fallar el controlador debe retornar la respuesta indicando el error de validación. Para los campos de texto se deben hacer las validaciones básicas como por ejemplo validar sólo números es un DNI, sólo caracteres válidos para un nombre, etc. Para los campos de varias opciones se debe chequear que la opción ingresada sea una de las opciones válidas.
- Podrán utilizar librerías que facilitan algunas de las tareas que deben realizar en el trabajo como pueden ser: conexión a servicios externos, librerías de parseo, librerías con patrones para buenas prácticas, validaciones de datos, etc. **Pero todos los miembros del equipo deben demostrar en la defensa pleno conocimiento del funcionamiento de estas librerías y una idea de cómo solucionan el problema**.
- Para la implementación del Login **no se podrá utilizar librerías como Flask-Login** dado que consideramos que ocultan implementación que queremos asegurarnos que entiendan en el transcurso de esta materia.
- Para la interacción con la base de datos se deberá utilizar el ORM SQLAlchemy que nos permita además tener una capa de abstracción con la BD.
- Debe tener en cuenta los conceptos de Semántica Web proporcionada por HTML5 siempre y cuando sea posible con una correcta utilización de las etiquetas del lenguaje.
- Cada entrega debe ser versionada por medio de git utilizando el sistema de Versionado Semántico para nombrar las distintas etapas de las entregas. Ejemplo: para la etapa 1 utilizar la versión v1.x.x.
- Puede utilizar frameworks de CSS como Bootstrap, Foundation, Bulma o Tailwind CSS, entre otros.
- Todas las vistas deben ser web responsive y visualizarse de forma correcta en distintos dispositivos. Al menos deben contemplar 3 resoluciones distintas:
  - res < 767px (móviles)
  - 768px < res < 1024px (tablets)
  - 1025px < res (portátiles y escritorio)

Consideraciones generales

- La entrega es obligatoria. Todos y todas los/as integrantes deben presentarse a la defensa.
- El/la ayudante a cargo **evaluará el progreso y la participación** de cada integrante mediante las consultas online y el seguimiento mediante GitLab.
- El proyecto podrá ser realizado en grupos de tres o cuatro integrantes (será responsabilidad de los y las estudiantes la conformación de los equipos de trabajo). Todos y todas los/as estudiantes cumplirán con la totalidad de la consigna, sin excepciones.
- Deberán visualizarse los aportes de cada uno/a de los/as integrantes del grupo de trabajo tanto en Git como en la participación de la defensa.
- La defensa será de forma virtual a convenir con el ayudante asignado. En caso de no tener los medios necesarios para realizar la defensa de forma correcta (micrófono y cámara), se deberá realizar la defensa de manera presencial.
- El trabajo será evaluado desde el servidor de la cátedra que cada grupo deberá gestionar mediante Git. **NO se aceptarán entregas que no estén realizadas en tiempo y forma en el servidor provisto por la cátedra**.
- Toda funcionalidad que no se haya terminado en la etapa 1 puede completarse en la siguiente.

Información del servidor

Tener en cuenta que el servidor de la cátedra utiliza los siguientes servicios y versiones:

- **Lenguaje**: Python 3.12.3.
- **Servidor Web**: nginx/1.24.0 (Ubuntu).
- **Dependencias Python**: Poetry (1.8.3).
- **Servidor de Base de Datos**: PostgreSQL 16.
- **Node**: v22.7.0 (npm 10.8.2).