Proyecto de Software 2023
Trabajo Integrador (TI)

CIDEPINT es el Centro de Investigación y Desarrollo en Tecnología de Pinturas de Argentina, nacido en los años 70 a partir de una colaboración entre varias instituciones. Su objetivo principal es promover la competitividad de los productos de pintura argentinos a nivel nacional e internacional mediante investigaciones y desarrollos en tecnología de recubrimientos. Además, se dedica a la formación de profesionales especializados y a la creación de normas en la industria. Con el tiempo, ha ampliado sus áreas de enfoque para incluir temas como el tratamiento de aceros, la protección contra la corrosión y soluciones ecológicas. Sus objetivos incluyen investigar, formar recursos humanos, difundir resultados, organizar cursos y colaborar con instituciones afines.
CIDEPINT plantea la necesidad de que exista una plataforma para mostrar y ofrecer los servicios que prestan las diferentes Instituciones.

La aplicación tendrá un aplicación interna de administración (para usuarios y administradores) en Python y Flask, y un portal web en Vue.js que será donde se podrán buscar los servicios ofrecidos por las instituciones registradas. Utilizaremos una base de datos PostgreSQL y se implementarán las API necesarias para las consultas.

Objetivo general

El objetivo de este trabajo integrador es desarrollar una aplicación web que permita registrar y gestionar los servicios ofrecidos por las instituciones o centros de Investigación y Desarrollo en Tecnología de Pinturas. Estos centros contarán con un conjunto de características como nombre, descripción, contacto, página web, redes sociales y ubicación geográfica.

Funcionalidad de la aplicación

La aplicación permitirá:

- **Implementar un buscador de servicios:** se debe proporcionar a los usuarios un buscador que les permita encontrar servicios ofrecidos por las distintas instituciones.
- **Permitir a entidades/personas puedan dar de alta su organización** y publicar los servicios que ofrecen.
- **Realizar un pedido de servicio:** los usuarios podrán solicitar un servicio, que puede incluir la reserva de un turno para utilizar algún equipamiento o simplemente dejar un mensaje para establecer el contacto.
- **Seguimiento del pedido.**
- **Obtener reportes estadísticos:** la aplicación debe generar reportes estadísticos que muestren los servicios más requeridos, las zonas más demandadas y los turnos no cumplidos.

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 acceder al mapa con las ubicaciones de los centros, utilizar el buscador de servicios y realizar pedidos de servicio.
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 de los centros y servicios.
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 todas las funcionalidades requeridas y su correcto despliegue en un entorno de prueba.

Etapa 1

Aplicación Privada

Esta aplicación será utilizada tanto por los administradores del sistema que tienen el acceso a la administración de los usuarios y también por usuarios asociados a cada una de las instituciones para que puedan administrar las mismas agregando los servicios que ofrecen.

1.1 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.

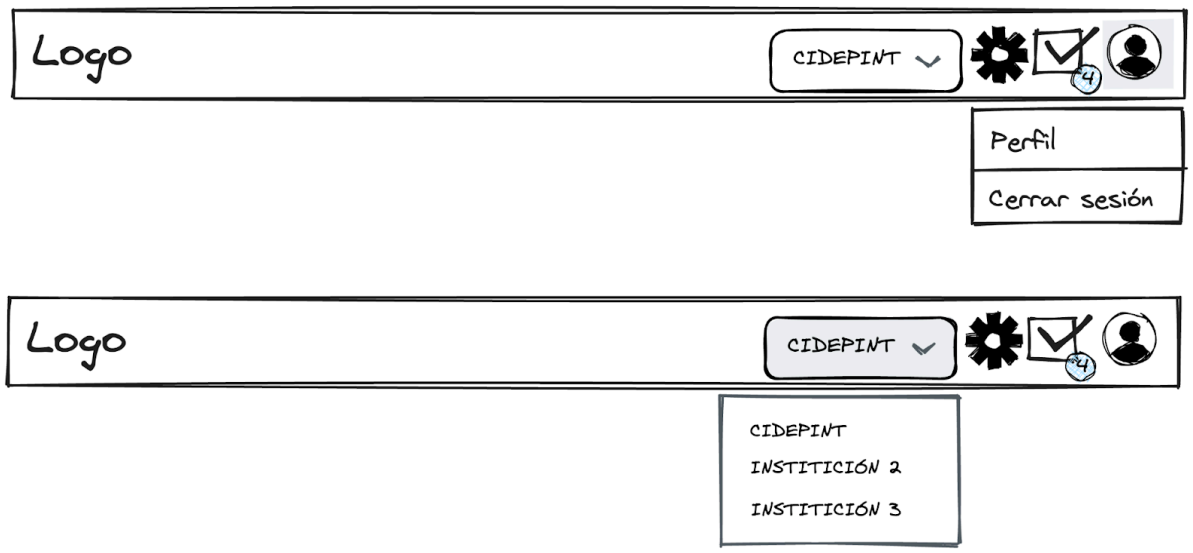

Figura 1. Posible visualización de top bar de aplicación privada.

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 los seguimientos**. En este ejemplo dejamos un indicador de los seguimientos que están en curso.

- Botón que nos permite **modificar la información de la institución** seleccionada actualmente.
- **Dropdown que nos permite cambiar de institución** (si es que el usuario pertenece a más de una)

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.
También pueden crear un logo para la aplicación y mostrar el mismo en forma coherente en todas las secciones de la aplicación.

1.2 Registro de usuarios

La aplicación de administración debe permitir el registro de 2 formas: un registro simple, que permite al usuario registrarse de la forma convencional y también se debe permitir registrarse en el sistema con google (se trabajará en la Etapa 2).

Registro simple

Implementar las páginas necesarias para realizar el flujo de registro de usuario al sistema. El registro de usuario deberá ser público, permitiendo que cualquier persona pueda crear una cuenta.
Es necesario que estén registrados en el sistema tanto los usuarios que van utilizar la parte administrativa del sistema como los que realicen las solicitudes de servicios que se ofrecen en el sistema.
El formulario de registro debe solicitar los siguientes datos:

- **Nombre**
- **Apellido**
- **Correo electrónico**

Una vez cargados los datos en el formulario de registro, se enviará un correo de confirmación que el usuario deberá aceptar y será redirigido a la aplicación obligando a que complete usuario y contraseña para terminar el proceso. El usuario no podrá realizar ninguna operación en la plataforma hasta tanto se le asigne el rol correspondiente que se lo permita.

1.3 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), deberá existir la opción de poder iniciar **sesión con Google** (se trabajará en la Etapa 2).
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.
Tener en cuenta que un usuario puede tener distintos roles en distintas instituciones.
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.

1.4 Módulo de usuarios

Desarrollar el **módulo de usuarios** que debe permitir al **Super Administrador** (no pertenece a ningúna institución) realizar distintas operaciones sobre los usuarios del sistema, ya sea personal administrativo o usuario común.
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:

- **Nombre**
- **Apellido**
- **Email**
- **Nombre de usuario**
- **Password**
- **Activo**: SI | NO
- **Roles por Institución**: Operador/a | Administrador/a

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

- **Email**
- **Activo/Bloqueado**

El resultado de la búsqueda debe estar paginado en base a la configuración del sistema (ver **módulo de configuración**). La paginación deberá realizarse del lado del servidor, es decir, la cantidad de registros retornados debe ser la indicada en el módulo de configuración, por ej. 25 registros por página.
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 **Super Administrador**.
Además se debe poder asignar o desasignar roles de un usuario para una institución. En principio se proponen los roles **Dueño/a, Administrador/a** y **Operador/a** pero pueden agregarse más si se lo cree conveniente.
En referencia a los roles anteriormente mencionados podemos decir que **Super Administrador** es el rol con menos restricciones del sistema y que no pertenece a ningúna

institución. En contrapunto los roles **Dueño/a, Administrador/a** y **Operador/a** son los roles que se pueden asignar a los usuarios que trabajan para las distintas instituciones.
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 con rol de **Super Administrador** 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.

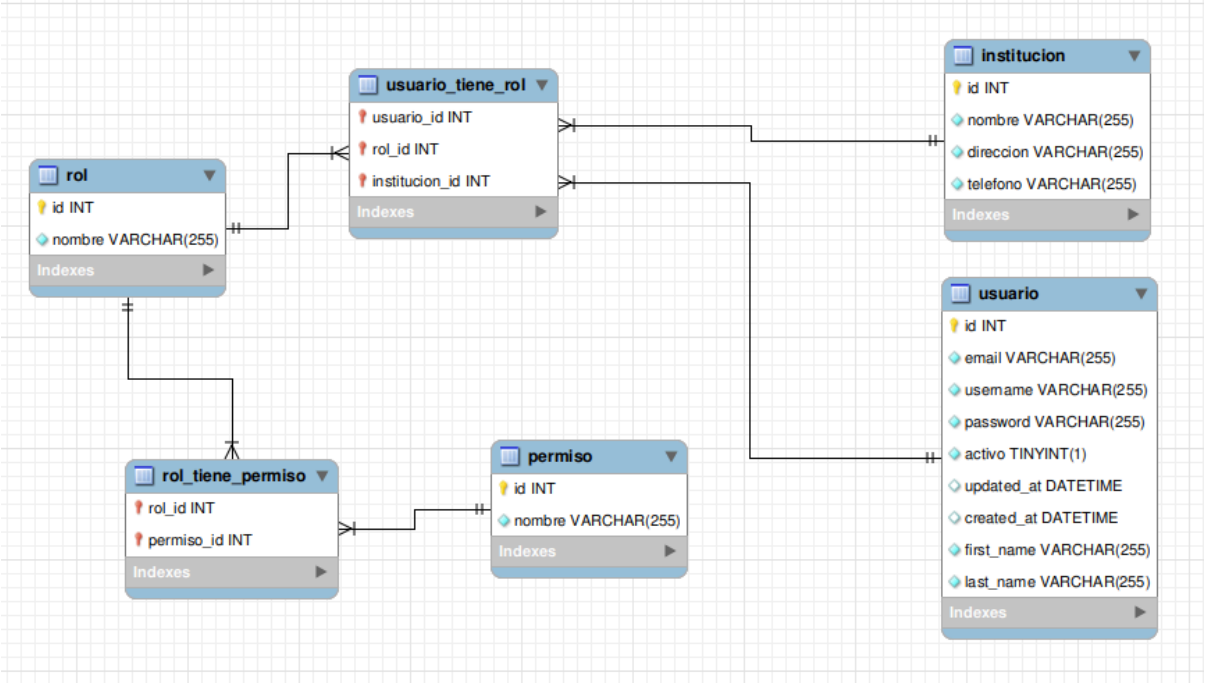

Figura 2. Posible esquema para el manejo de usuarios

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

Super Administrador/a: index, show, update, create, destroy.

1.5 Módulo de administración de Instituciones

Este módulo permite la gestión de las instituciones habilitadas para brindar servicios en el sistema. Incluye la siguiente funcionalidad.
El módulo debe permitir el **CRUD** de instituciones. Se deben poder listar las mismas paginadas según las páginas del **módulo de configuración**.
Los datos necesarios para cargar una nueva institución son los siguientes:

- **Nombre de la institución**
- **Información de la institución**
- **Dirección**
- **Localización**
- **Web**
- **Palabras claves de búsqueda**
- **Días y horarios de atención**
- **Información de contacto**
- **Habilitado**: SI | NO

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

- **Super Administrador/a:** index, show, update, create, destroy, activate, deactivate.

1.6 Módulo de administración de usuarios de la institución

Este módulo permite a los usuarios con rol **Dueño** de una institución agregar a otros usuarios registrados del sistema a participar en su institución.
La acción consiste básicamente en agregarle a ese usuario un rol dentro de la institución.
El mismo debe poder realizar la acción con el email del usuario al que quiere asignarle un rol dentro de la institución.
Todas las asignaciones de rol a los usuarios deben listarse paginados según se indique en el **módulo de configuración** y se debe permitir cambiar el rol para un usuario dentro de la misma institución.
A continuación se definen los roles y las acciones que pueden realizar sobre este módulo:

- **Dueño/a:** index, create, destroy, update.

1.7 Módulo de administración de servicios

Este módulo permite la gestión de los servicios brindados por la institución. Incluye la siguiente funcionalidad.
Se deben poder realizar todas las operaciones **CRUD** de cada servicio. Los servicios deberán verse paginados según se indique en el **módulo de configuración**.

Los datos necesarios para cargar un servicio son los siguientes:

- **Nombre**: nombre del servicio. Ejemplo: “Ensayo de corrosión”.
- **Descripción del servicio**: breve descripción del servicio prestado por la institución.
- **Palabras claves de búsqueda.**
- **Tipo de servicio (Análisis, Consultoría, Desarrollo).** Puede implementarse con un Enum en Python o podría simplemente utilizar la tabla precargada. No es necesario implementar el CRUD.
- **Habilitado**: SI | NO

A continuación se definen los roles necesarios para las acciones:

- **Dueño/a:** index, show, update, create, destroy.
- **Administrador/a:** index, show, update, create, destroy.
- **Operador/a:** index, show, update, create.

1.8 Módulo de gestión de solicitud de servicio

Este módulo permite realizar la gestión de las solicitudes realizadas por los clientes a la institución.
Deberán listarse las solicitudes de servicios recibidas desde la aplicación pública (API), que fueron realizadas por cada cliente. Pudiendo filtrar por tipo de servicio, rango de fechas, estado de solicitud, usuario que realizó la solicitud.
Los resultados también deben estar paginados tomando los parámetros de cantidad de páginas del **módulo de configuración**.
Al ingresar al detalle del pedido, podrá visualizarse información del cliente (que debe ser un usuario registrado), detalle del pedido de servicio, archivos adjuntos recibidos.
La solicitud de servicio podrá marcarse con alguno de los siguientes estados: **aceptada**, **rechazada**, **en proceso**, **finalizada**, **cancelada**. En todos los casos se deberá indicar la fecha en la que se realizó el cambio de estado y un campo para las observaciones.
Dado un pedido determinado, se tendrá la posibilidad de ingresar un comentario o nota sobre el mismo, el cual podrá ser leído por el cliente. Podemos decir que este módulo permite una interacción directa entre la persona que solicita el servicio y quien lo ofrece.
A continuación se definen los roles necesarios para las acciones:

Dueño/a: index, show, update (cambiar estado, comentar, etc.), destroy.
Administrador/a: index, show, update (cambiar estado, comentar, etc.), destroy.
Operador/a: index, show, update (cambiar estado, comentar, etc.).

1.9 Módulo de configuración

Este módulo permitirá administrar la configuración del sistema, como mínimo deberá contemplar la siguiente configuración:

- Cantidad de elementos por página en los listados del sistema (todos los listados deberán respetar este valor para el paginado).
- Información de contacto se mostrará en la app pública.
- Posibilidad de deshabilitar el sitio para mantenimiento
- Mensaje de mantenimiento de sitio

La configuración del sistema sólo podrá modificarla un usuario con el rol de **Super Administrador**. Pueden agregar, en caso de que lo consideren necesario y con acuerdo previo con el ayudante asignado, alguna configuración más para ser administrada por este módulo.
A continuación se definen los roles y las acciones que pueden realizar sobre este módulo:

- **Super Administrador/a:** show, update.

1.10 API

Se deberán implementar en la aplicación privada una serie de endpoints que permitan realizar distintas operaciones u obtener contenido desde la aplicación pública.
Pueden acceder a la especificación de la API publicada en nuestra web en el siguiente enlace. En la especificación publicada se determinan las restricciones de acceso a cada uno de los servicios (qué servicios requieren login y cuáles no).
Nota: La especificación es una guía para el/la estudiante que podrá modificar en caso que necesite agregar nuevos endpoints.
Para la Etapa 1 el método de autenticación sólo deberá responder un {“result”: “success”} o {“result”: “fail”}. En la Etapa 2 del trabajo se responderá un JWT que permitirá realizar todas las operaciones.
Todo endpoint que requiera autenticación para esta entrega no realizará el chequeo dado que aún no se dió el tema, pero deben implementarlos y en la segunda etapa se agregará el chequeo del JWT. En lugar de JWT podrían enviar el ID del usuario para poder implementar todos los endpoints que requieran ese dato.

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.
- 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.
- No pueden utilizar un framework/generador de código para el desarrollo de cada una de las API requeridas en el enunciado.
  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 **Bootstrap** u otro framework similar.
  Todas las vistas deben ser web responsive y visualizarse de forma correcta en distintos dispositivos. Al menos deben contemplar 3 resoluciones distintas:
  - res < 360
  - 360 < res < 768
  - res > 768

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:

Servidor de Base de Datos: Postgres 15
Intérprete Python: Python 3.8.10
Servidor web: Nginx 1.18.0-0ubuntu1.3