

# Proyecto de Software 2022

## Trabajo Integrador (TI)

Todos los años se trata de coordinar el trabajo integrador (TI) de la materia con necesidades y/o demandas de la comunidad. Para la cursada 2022, se trabajará en una aplicación que permita gestionar un club barrial. Este trabajo surge ante el pedido del Club Deportivo Villa Elisa, quien en este año cumple 100 años desde su fundación.

El trabajo se llevará a cabo en dos (2) etapas, en las cuales se deberá entregar un prototipo en funcionamiento en el servidor provisto por la cátedra.

Se implementarán dos aplicaciones que permitirán gestionar y brindar distinta información a las personas asociadas del club.

Por un lado, se desarrollará una *aplicación privada* que potencialmente proveerá la siguiente funcionalidad:

- Gestión de las personas asociadas del club (con alguna funcionalidad para la importación y exportación de datos).
- Gestión de las disciplinas y/o actividades.
- Gestión de pagos.
- Mantener opciones de configuración del sistema.
- Administración de roles para aplicación privada.

También se desarrollará una *aplicación pública*, que permitirá visualizar la información de interés para la comunidad y personas asociadas al club. A través de esta aplicación se podrá:

- Obtener información para establecer contacto con el club.
- Conocer los precios de las actividades del club y sus disciplinas.
- Tramitar la credencial digital.
- Visualizar estado de cuenta/socio.
- Registrar la realización de un pago de cuota.

# Etapa 1

Fecha de entrega: 21/10 a las 23:59

## Aplicación Privada

### 1.1 Manejo de sesiones

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.

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.

### 1.2 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 un espacio donde se pueda visualizar la cuenta del usuario con acceso al perfil y link para cerrar la sesión, similar a la Figura 1.

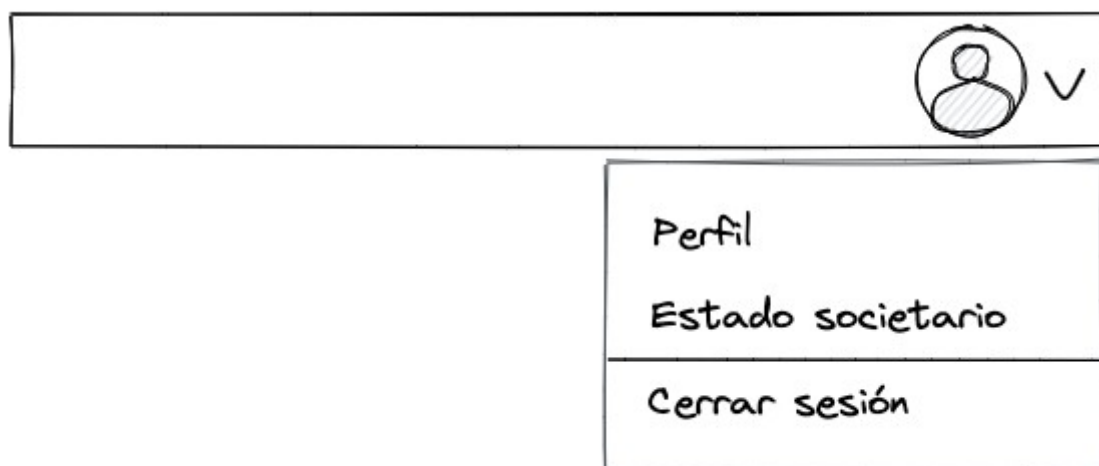


Figura 1. Posible visualización de la cuenta del usuario en el sistema.

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

## 1.3 Módulo de usuarios

Desarrollar el **módulo de usuarios** que deberá contemplar **al menos** la siguiente funcionalidad:

- CRUD de usuarios: en esta sección deben validar que no existan dos usuarios con el mismo nombre de usuario, o mismo email, es decir, tanto el nombre de usuario como el email son únicos.
- Se considerarán al menos los siguientes datos para cada usuario: nombre, apellido, email, nombre de usuario, password, activo (sí o no) y roles (Socio/a, Operador/a o 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 retornada debe ser la indicada en el módulo de configuración, por ej. 25 registros por página.

- Activar/Bloquear 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 **Administrador**.
- Asignar o desasignar roles de un usuario, pueden ser varios. En principio se proponen los siguientes roles: **administrador/a, operador/a y socio/a**.

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 **Administrador**.

Considerar que un usuario podrá tener más de un rol, y para cada rol se pueden configurar varios permisos. 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 personas asociadas (ver **módulo de gestión de personas asociadas**) deberá contemplar los siguientes permisos:

- member\_index: permite acceder al index (listado) del módulo.
- member\_new: permite crear una persona asociada al club.
- member\_destroy: permite eliminar una persona asociada al club.
- member\_update: permite actualizar una persona asociada al club.
- member\_show: permite visualizar una persona asociada al club.

**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.

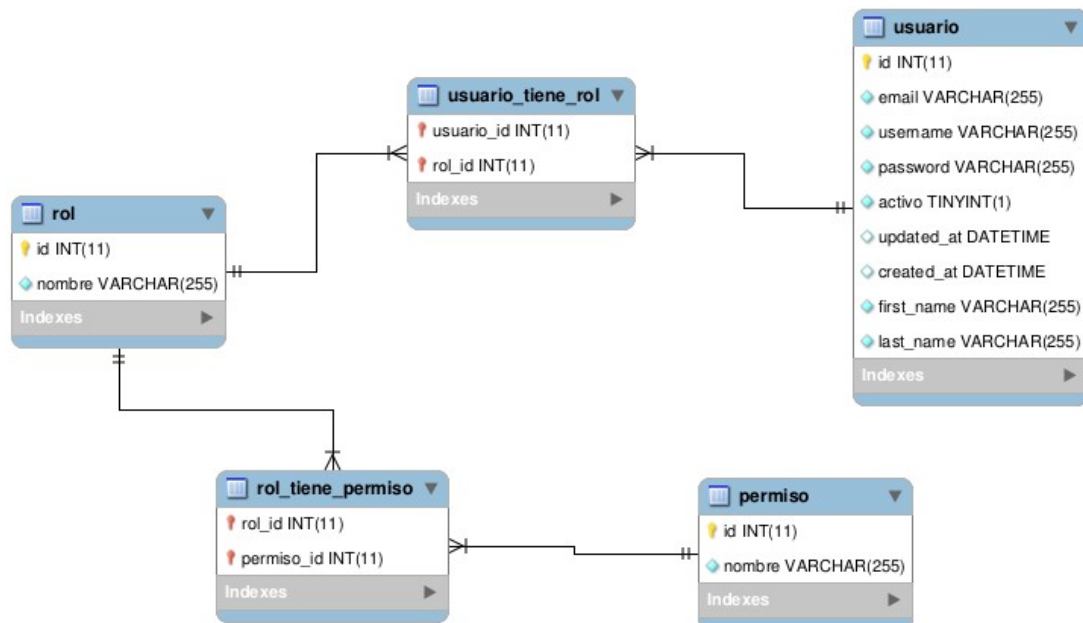


Figura 2. Posible esquema para el manejo de usuarios

## 1.4 Módulo de gestión de personas asociadas

Este módulo permite a un usuario con el rol de **Operador/a** o **Administrador/a**, gestionar las altas/bajas/modificaciones de las personas asociadas al club.

### 1.4.1 CRUD de Personas asociadas

El módulo debe brindar **al menos** la siguiente funcionalidad:

- CRUD de personas asociadas.
- Se deben listar los campos que permitan ver la información relevante (consultar con el ayudante en caso de alguna duda).
- El listado de personas asociadas deberá permitir exportar a CSV y PDF, respetando el filtro aplicado.
- Se deben poder realizar búsquedas, **al menos** por los siguientes campos:
  - Apellido de la persona asociada (texto).
  - Estado: activo o no-activo (select).

Los resultados también deben estar paginados tomando los parámetros de cantidad de páginas del **módulo de configuración**.

Los datos necesarios para cargar una persona asociada son los siguientes:

- Apellido: apellido de la persona asociada (text).
- Nombre: nombre de la persona asociada (text).
- Tipo de documento: listado de tipo de documentos, por ej. DNI, LE, LC, etc (select)
- Número de documento: número (text)
- Género: M|F|Otro
- Número de socio/a: único, generado por sistema.
- Dirección: domicilio de la persona asociada (text).
- Estado: activo o no-activo.
- Teléfono (opcional): número de teléfono (text).
- Email (opcional): dirección de correo electrónico (text).
- Fecha de alta: fecha generada por sistema (datetime)

A continuación se definen los roles necesarios para el resto de las acciones:

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

## 1.5 Módulo de gestión de disciplinas

Este módulo permite la gestión de las **disciplinas** del club. Incluye la siguiente funcionalidad.

### 1.5.1 CRUD de disciplinas

- Operaciones CRUD de cada disciplina
- Se debe poder listar las disciplinas, paginadas según las páginas del **módulo de configuración**.

Los datos necesarios para cargar una disciplina son los siguientes:

- Nombre: nombre de la disciplina (text)
- Categoría: nombre de la categoría (text)
  - por ejemplo: Basket Pre-mini (5 a 8 años), Basket mini (8 a 11), Fútbol Categoría 2010 etc.
- Nombre y apellido de instructores: nombre y apellido del/los instructor/es (text)
- Días y horarios: día y horario en el que se realiza la disciplina (text)
- Costo mensual: costo mensual de la disciplina (text). El formato de la moneda deberá respetar el formato configurado en el módulo de configuración.
- Habilitada: SI | NO

A continuación se definen los roles necesarios para el resto de las acciones:

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

### 1.5.2 API Disciplinas

Se deberá desarrollar **en la aplicación privada** un servicio público que permita obtener el listado de las disciplinas desde la aplicación pública.

- [Documentación del servicio GET /api/club/disciplines](#)

## 1.6 Módulo de Inscripción a disciplinas

Este módulo permite a un usuario con el rol de **Operador/a** o **Administrador/a**, realizar la carga inscripción de una persona asociada a una disciplina particular.

### 1.6.1 Inscripción

Desarrollar la funcionalidad que permita asociar a un usuario a una o más disciplinas dentro del club.

Existen muchas variantes posibles para realizar esta funcionalidad:

- Desde el listado de personas asociadas puede existir un botón que redirija a una vista donde para ese socio seleccionado se le asigne la nueva disciplina.
- Otra alternativa puede ser que desde el listado de disciplinas se permita agregar a una nueva persona asociada a la misma.
- Una tercera alternativa es entrar a una vista totalmente independiente donde se tenga que realizar el filtro de la persona y la disciplina para realizar la asociación.

Dejamos a criterio del grupo en debate con el ayudante para decidir cuál solución más conveniente para esta funcionalidad.

Algunas restricciones que se deben tener en cuenta:

- Verificar que el estado del socio sea el correcto para realizar la asociación. Por ejemplo, no se debe poder registrar a socios morosos. Considere este estado y algún otro que si es que como grupo vieron conveniente agregarlo y lo consideran como un estado no válido.
- Verificar que la disciplina está habilitada a la hora de realizar la asociación.

### 1.6.2 API de Disciplinas de socio

Se deberá desarrollar un servicio privado (requiere autenticación) que permita obtener todas las disciplinas a las que una persona asociada está inscrita.

- [Documentación del servicio GET /api/me/disciplines](#)

## 1.7 Módulo de gestión de pagos

Este módulo permite a un usuario con el rol de **Operador/a** o **Administrador/a**, realizar la carga de los pagos de las cuotas.

La cuota del club está definida de la siguiente manera: cuota base (monto definido en el módulo de configuración) + adicionales según monto de cada disciplina que practique el/la asociado/a.

En caso de modificarse el valor de una cuota se aplica para los meses siguientes.

Las cuotas vencen del día 1 al 10. En caso de producirse mora en el pago se aplica el porcentaje de recargo configurado en el **Módulo de configuración**.

### 1.7.1 Registro de pago de cuotas

Desarrollar la funcionalidad que permita registrar el pago de las cuotas de sus asociados/as y generar un recibo de pago.

El módulo brinda la siguiente funcionalidad:

- Se deben listar los campos que permitan ver la información relevante (consultar con el ayudante en caso de alguna duda).
- Se debe poder realizar búsquedas, **al menos** por los siguientes campos:
  - apellido del/a asociado/a (texto).
  - número de socio/a (texto)

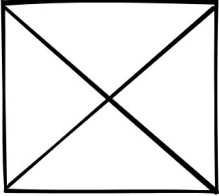
Los resultados también deben estar paginados tomando los parámetros de cantidad de páginas del **módulo de configuración**.

Para registrar los pagos se proponen dos estrategias que pueden (o no) tomar cada grupo:

- 1) Registrar todas las cuotas como IMPAGAS con sus datos necesarios (número de cuota, asociado/as que realiza el pago, fecha, monto, disciplina), y al momento de realizar el pago se cambia el estado de la cuota a PAGADO.
- 2) El registro del pago de cada cuota se realiza en el momento en que el/la asociado/a cancela la cuota de una disciplina en particular.

En cualquier caso, sean estas dos propuestas o una propuesta diferente del grupo de trabajo, el módulo deberá permitir descargar el recibo de pago, con la información correspondiente (fecha, número de cuota, monto, disciplina, datos del/la asociado/a).

Mínimamente, cada recibo de pago deberá contener la siguiente información:

	Recibo # 000000	Fecha 22/08/2022
	Recibimos de Cosme Fulanito el importe en pesos (2000) dos mil Por el concepto de cuota societaria mes Agosto 2022	

A continuación se definen los roles necesarios para el resto de las acciones:

- index, show, import: **Operador/a, Administrador/a**
- destroy: **Administrador/a**

### 1.7.2 API Pagos

Se deberán desarrollar dos servicios públicos que permitirán obtener los pagos de las cuotas realizados por un asociado/a desde la aplicación pública y generar un pago:

- [Documentación del servicio GET /api/me/payments](#)
- [Documentación del servicio POST /api/me/payments](#)

## 1.8 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).
- Habilitar/Deshabilitar tabla de pagos de la app pública.
- Información de contacto se mostrará en la app pública.
- Texto para mostrar en el encabezado del Recibo de pago.
- Valor de la cuota mensual base
- % de recargo para cuotas adeudadas

La configuración del sistema sólo podrá modificarla un usuario con el rol de **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.



## Consideraciones generales:

- 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 un ORM que nos permita además tener una capa de abstracción con la BD.
- Para la aplicación pública se debe utilizar el Framework web [VueJS](#) versión 3. Se deberá utilizar la librería [vue-router](#) para implementar el ruteo de la aplicación.
- 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.
- 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.**
- 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.
- 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.
- El/la ayudante a cargo **evaluará el progreso y la participación** de cada integrante mediante las consultas online y el seguimiento mediante GitLab.
- Seleccione una vista (**HTML5** y **CSS3**) para realizar validaciones de las especificaciones de la W3C (<http://validator.w3.org/> y <https://jigsaw.w3.org/css-validator/> respectivamente). En esta oportunidad puede utilizar **Bootstrap** u otro framework similar. En caso de que alguna de las vistas no valide, deberá realizar un breve informe indicando cuales son los errores encontrados (este informe deberá realizarse para la etapa 1).

- Todas las vistas deben cumplir ser web responsive y visualizarse de forma correcta en distintos dispositivos. Al menos deben contemplar 3 resoluciones distintas:
  - $res < 360$
  - $360 < res < 768$
  - $res > 768$
- La entrega es obligatoria. Todos y todas los/as integrantes deben presentarse a la defensa.
- El sistema no debe ser susceptible a SQL Injection, XSS ni CSRF.
- **Importante:** 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.

## Información del servidor

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

- Servidor de Base de Datos: Postgres 13
- Intérprete Python: Python 3.10.6
- Node JS: v14.20.0 (npm 6.14.17)
- Servidor web: Nginx 1.18.0-Ubuntu1.3