PROYECTO INFORMÁTICO

Tecnicatura Universitaria en Tecnologías de Programación

Facultad de la Micro, Pequeña y Mediana Empresa. UPSO

Docente: Carlos Berger

Año: 2023

Grupo 5:

- Aniñir Lionel
- Galvez Juan Manuel Ignacio
- Gómez Duilio Enrique (Project Manager)
- Leal Patricia Guillermina

Proceso de creación del proyecto FULL STACK API REST FULL BD

El Proyecto busca una solución FULL STACK, que incluye una Aplicación API REST FULL complementado con una Base de Datos SQL para la persistencia de los datos.

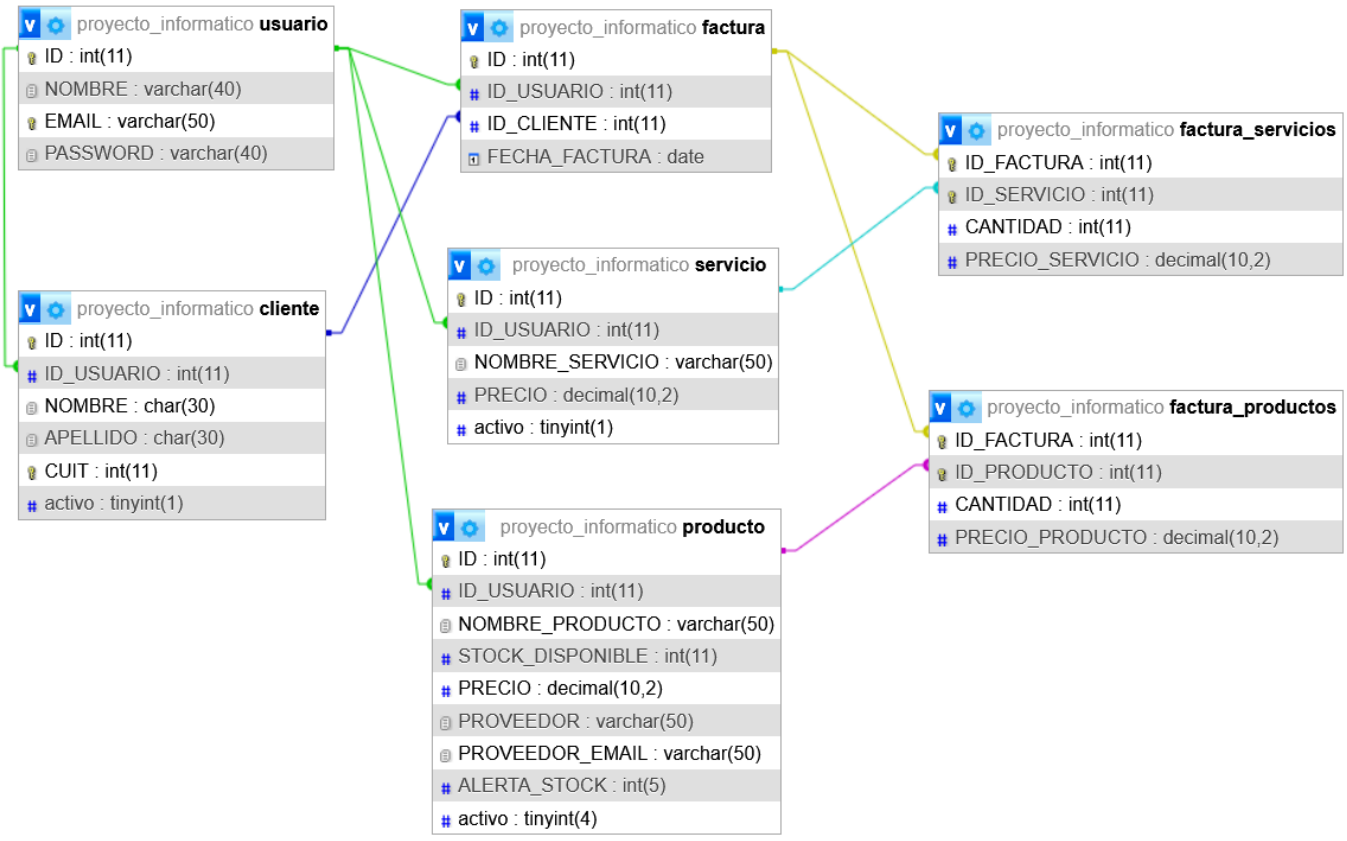
La idea principal de este modelo API REST FULL es que en el Frontend se hagan las peticiones entregando en el body de la consulta un archivo de tipo JSON, también se pueden hacer operaciones sobre la Base de Datos utilizando la URL preparada para recibir dicha petición, y sabiendo la ID de una factura o un cliente o producto se puede eliminar un ítem de la Base de Datos. Este borrado es LÓGICO.

Las facturas no se borran de la base de datos desde el Frontend (el usuario no puede borrarlas); para hacerlo se debe tener acceso al administrador de Base de Datos, por ejemplo PHPADMIN, pero según nuestro modelo de tablas interrelacionadas hay que borrar en un orden específico.

El proyecto se inició eligiendo cuál era la forma más óptima de crear las tablas y sus relaciones en la base de datos. Esto llevó varios intentos y pruebas, y se perdió mucho tiempo porque no estábamos muy prácticos en el tema de base de datos, no debido a

problemas de desconocimientos de comandos o el contexto de uso de una base de datos sino a obtener la funcionalidad acorde y óptima para poder vincular todo el Backend, Frontend y Base de datos.

Gráfica del modelo relacional que usamos en la base de datos:



Rutas del Servidor Backend API REST FULL (con sus métodos):

'/login' (POST, OPTIONS) -> login>,

En la ruta /login, luego de que el usuario carga sus datos, el backend valida los datos contra la base de datos y genera el token y USER ID únicos para cada usuario que le permitirán validar sus operaciones en la base de datos, el acceso a las rutas del backend y sus operaciones CRUD.

En todas las siguientes rutas se usa la id_user para direccionar el backend al usuario correspondiente y el acceso a sus recursos.

'/user/<id_user>/cliente' (HEAD, GET, OPTIONS) -> get_all_clients_by_user_id>,

'/user/<id_user>/historial' (HEAD, GET, OPTIONS) -> get_historial>,

'/user/<id_user>/ranking_clientes' (HEAD, GET, OPTIONS) -> get_ranking_clientes>,

'/user/<id_user>/cliente' (POST, OPTIONS) -> create_client>,

'/user/<id_user>/cliente/<id_client>' (PUT, OPTIONS) -> update_client>,

'/user/<id_user>/cliente/<id_client>' (DELETE, OPTIONS) -> delete_client>,

'/user/<id_user>/factura' (POST, OPTIONS) -> create_factura>,

'/user/<id_user>/factura/<id_factura>' (PUT, OPTIONS) -> update_factura>,

'/user/<id_user>/factura/<id_factura>' (HEAD, GET, OPTIONS) -> get_factura_by_user>,

'/user/<id_user>/facturas' (HEAD, GET, OPTIONS) -> get_facturas_by_user_id>,

'/user/<id_user>/factura/<id_factura>/factura_productos' (POST, OPTIONS) -> create_factura_productos>,

'/user/<id_user>/factura/<id_factura>/factura_productos' (PUT, OPTIONS) -> update_factura_productos>,

'/user/<id_user>/factura/<id_factura>/factura_productos/<id_producto>' (DELETE, OPTIONS) -> delete_factura_productos>,

'/user/<id_user>/factura_productos_total/<id_factura>' (HEAD, GET, OPTIONS) -> get_total_factura_producto>,

'/user/<id_user>/factura/<id_factura>/factura_servicios' (POST, OPTIONS) -> create_factura_servicios>,

'/user/<id_user>/factura/<id_factura>/factura_servicios' (PUT, OPTIONS) -> update_factura_servicios>,

'/user/<id_user>/factura/<id_factura>/factura_servicios/<id_servicio>' (DELETE, OPTIONS) -> delete_factura_servicios>,

'/user/<id_user>/factura_servicios_total/<id_factura>' (HEAD, GET, OPTIONS) -> get_total_factura_servicio>,

'/user/<id_user>/ranking_servicios' (HEAD, GET, OPTIONS) -> get_ranking_servicios>,

'/user/<id_user>/servicios' (HEAD, GET, OPTIONS) -> get_servicios_by_user>,

'/user/<id_user>/servicios' (POST, OPTIONS) -> create_servicio>,

'/user/<id_user>/servicios/<id_servicio>' (PUT, OPTIONS) -> update_servicio>,

'/user/<id_user>/servicios/<id_servicio>' (DELETE, OPTIONS) -> delete_servicio>,

'/user/<id_user>/ranking_productos' (HEAD, GET, OPTIONS) -> get_ranking_productos>,

'/user/<id_user>/stock' (HEAD, GET, OPTIONS) -> get_product_by_user_id>,

'/user/<id_user>/producto' (POST, OPTIONS) -> create_producto>,

'/user/<id_user>/producto/<id_producto>' (PUT, OPTIONS) -> update_producto>,

'/user/<id_user>/producto/<id_producto>' (DELETE, OPTIONS) -> delete_producto>.

En todas las operaciones el backend retorna un archivo JSON con datos de retorno de acuerdo a cada operación, así como errores de validación de *schema* (esquema) y/o tipos de datos u otros tipos de errores.

A continuación, mostramos algunos de los formatos de JSONs usados. Estos son simples pero suficientes para poder hacer el CRUD:

Alta de Factura:

ruta: /user/ID/factura
metodo POST

formato del JSON:

```
{
 "id_usuario" : 1,
 "id_cliente" : 9,
 "fecha_factura" : "2023-11-30"
}
```

retorno del backend un JSON:

```
{
  "fecha_factura": "2023-11-30",
  "id": 28,
  "id_cliente": 9,
  "id_usuario": 1
}
```

****

Alta de Producto a la Factura:

ruta:
/user/1/factura/28/factura_productos
metodo POST

```
{
  "id_producto" : 35,
  "cantidad" : 2,
  "precio_producto" : 150500.99
}
```

JSON retornado por el backend :

```
{
  "cantidad": 2,
  "id_factura": 28,
  "id_producto": 35,
  "precio_producto": 150500.99
}
```

****

Alta de Servicio a la factura:

ruta:
/user/1/factura/28/factura_servicios
metodo POST:
JSON enviado desde el Frontend:

```
{
  "id_servicio" : 8,
  "cantidad" : 1,
  "precio_servicio" : 15000.00
}
```

JSON retornado por el backend:

```
{
  "cantidad": 1,
  "id_factura": 28,
  "id_servicio": 8,
  "precio_servicio": 15000.0
}
```

Los siguientes archivos JSONs son más cargados de información, logrando enviar varios productos de una sola vez, que luego el backend procesará en tiempo y forma. Así se logra hacer la carga de varios productos en un solo envío de un archivo JSON.

/// En Alta Productos la idea es subir en un JSON todos los productos a dar de alta en la factura X

```
{ "alta productos" : [
        {
          "id_producto" : 26,
          "cantidad" : 4,
          "precio_producto" : 8799.99
        },
        {
          "id_producto" : 31,
          "cantidad" : 1,
          "precio_producto" : 225000.99
        }
     ]
}
```

La misma metodología, pero para el alta de servicios a la factura F en un solo envío JSON:

// Alta Servicios a la factura F del usuario X, formato JSON a enviar, ejemplo:

```
{ "alta servicios" : [
```

```
        {
          "id_servicio" : 11,
          "cantidad" : 2,
          "precio_servicio" : 3800.00
        },
        {
          "id_servicio" : 16,
          "cantidad" : 1,
          "precio_servicio" : 75989.00
         },
        {
          "id_servicio" : 5,
          "cantidad" : 3,
          "precio_servicio" : 7900.00
        }
    ]
}
```

Clases usadas y organización del directorio Backend:

Existen varias clases que solo fueron creadas para poder retornar un JSON organizado y prolijo cuando su clase es instanciada, por ejemplo, todas las clases cuyos nombres comienzan con “ranking_ventas_por_*.py
Esto es muy práctico, ya que permite poder editar o personalizar el JSON dependiendo de la respuesta de la base de datos a la solicitud de la misma.

Las demás clases en la carpeta MODELS son multifunción, porque poseen los métodos, como por ejemplo CHECK SCHEMA, que valida si el JSON proveniente del Frontend contiene forma y tipos de datos correctos.
Después están los del tipo CRUD: crear, verificar si existe el ítem, actualizar, consultar ID y borrar.

En la carpeta ROUTES se incluyen el manejo de rutas que en algunos casos hacen consultas a la Base de Datos, ya que se hicieron al principio del proyecto y no tuvimos tiempo de mover esas funciones a las clases correspondientes de la carpeta MODELS.

En el directorio Backend se encuentra main.py que es el archivo que inicia el servidor de Backend, a través del comando py main.py, habiendo previamente inicializado el entorno virtual.

En el Directorio API, directorio padre de MODELS Y ROUTES, se encuentra además el directorio DB donde se guarda la configuración de la Base de Datos que permite una forma correcta de conectar a la misma.

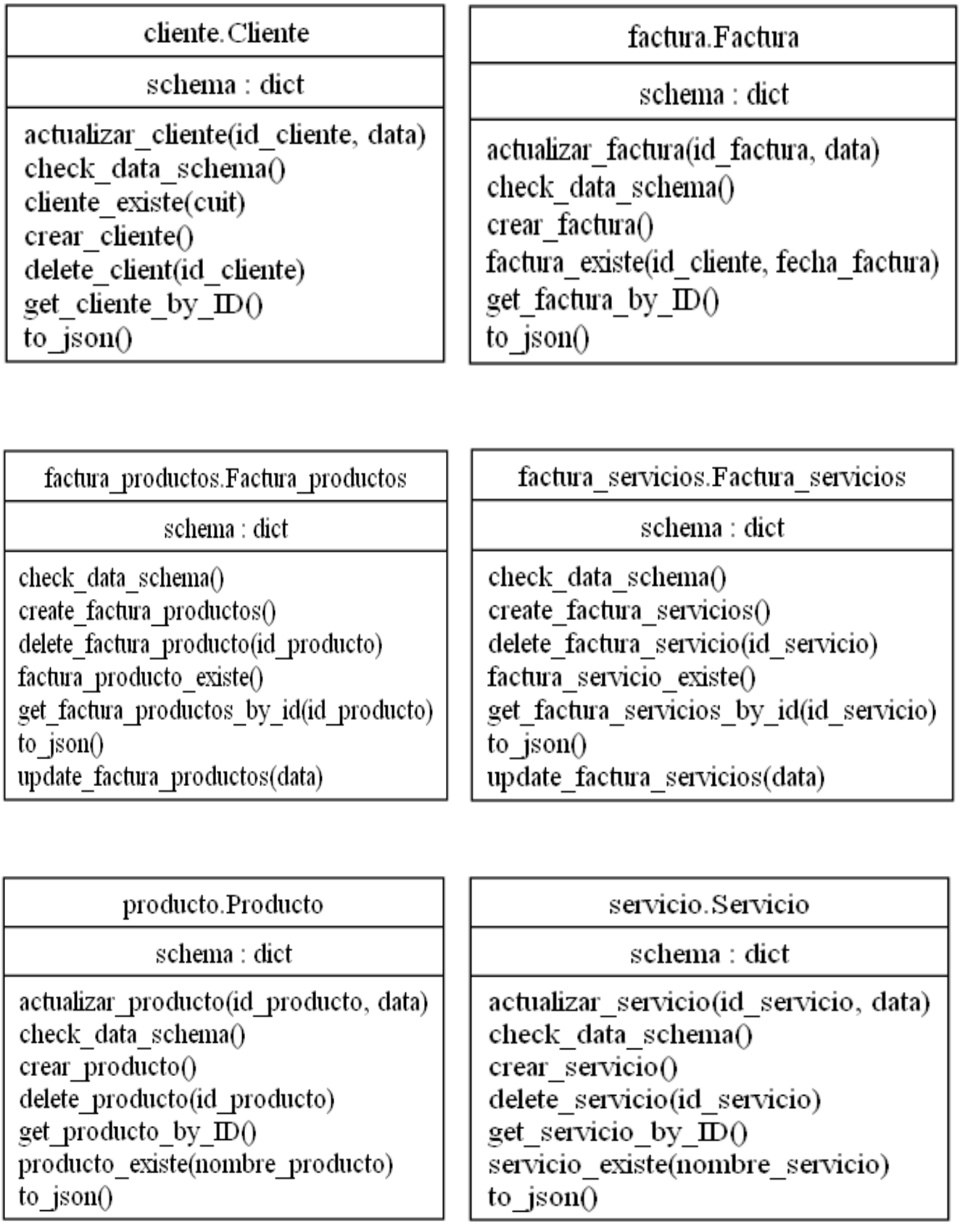
También en el directorio API se encuentran __init__.py y utils.py
En el primero se encuentran configuraciones de FLASK, CORS y la SECRET KEY que se usará como parte de la creación del TOKEN; esto se hace con varios datos: nombre usuario, clave y fecha actual más fecha de vencimiento del TOKEN.

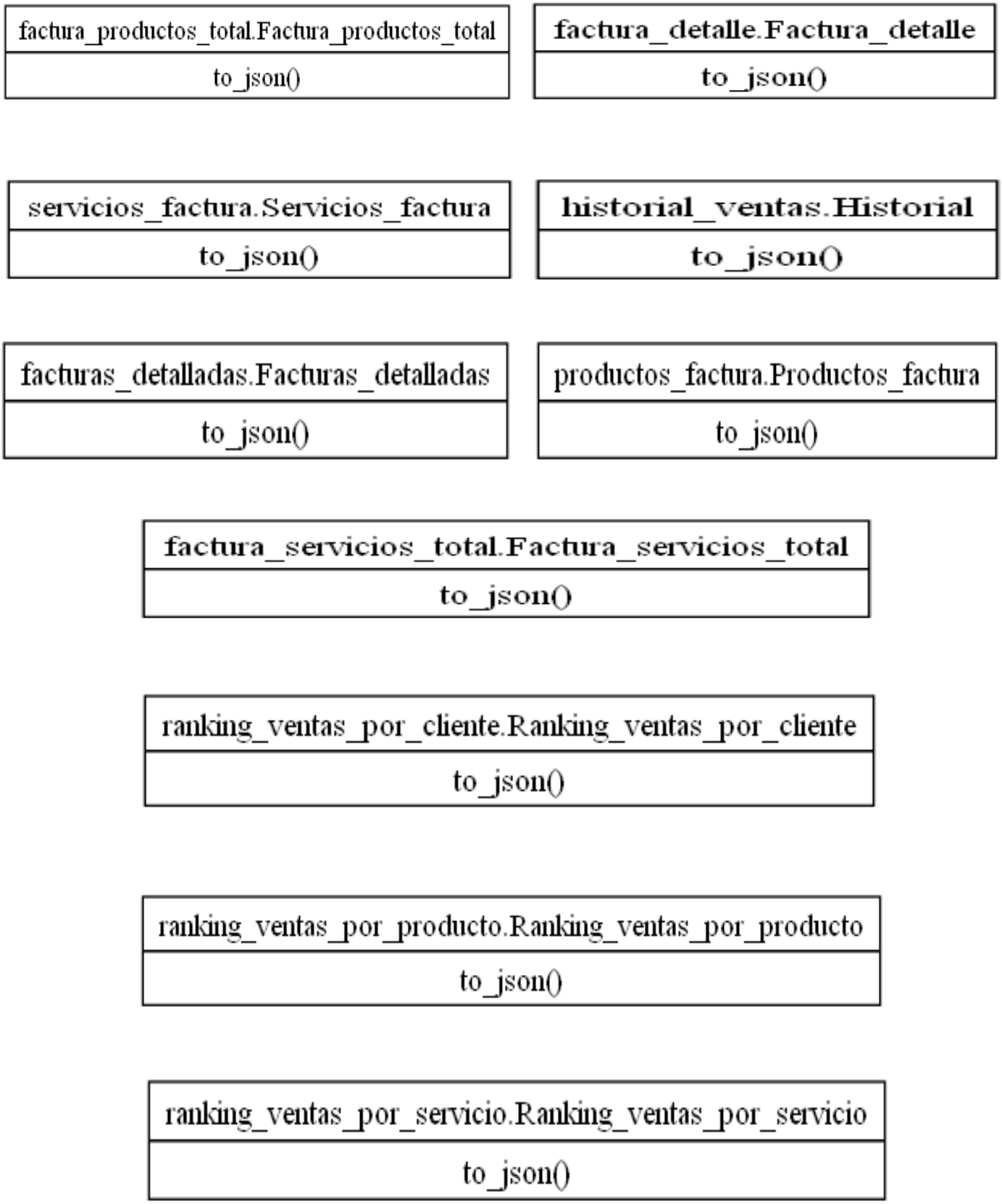
En utils.py encontramos los RESOURCE, los cuales servirán para validar que los usuarios solo puedan acceder a sus recursos y no de terceros, como por ejemplo client_resource, user_resource, token_requiered y factura_resource.

En utils.py se crean los DECORADORES que se usarán en las ROUTES.

También existe un directorio SQL_Scripts donde se guardan scripts de inicialización de la base de datos, para que en caso de que se produzca algún inconveniente se cree de nuevo desde el PHPADMIN importando el script SQL.

UMLs de las clases MODELS




factura_productos_total.Factura_productos_total
to_json()
factura_detalle.Factura_detalle
to_json()
servicios_factura.Servicios_factura
to_json()
historial_ventas.Historial
to_json()
facturas_detalladas.Facturas_detalladas
to_json()
productos_factura.Productos_factura
to_json()
factura_servicios_total.Factura_servicios_total
to_json()
ranking_ventas_por_cliente.Ranking_ventas_por_cliente
to_json()
ranking_ventas_por_producto.Ranking_ventas_por_producto
to_json()
ranking_ventas_por_servicio.Ranking_ventas_por_servicio
to_json()

GitHub

Tuvimos dificultades con GitHub, viéndonos en la necesidad de hacer varios repositorios para poder trabajar sin que se rompa. No perdimos código pero sí se perdieron muchos registros realizados con el comando *commit*. Debido a que al ir desarrollando código del backend muchos de los progresos, cambios, mejoras y correcciones se perdieron después de luchar con las ramas, decidimos trabajar en la rama *main* y subir con mucho cuidado las actualizaciones. Así, luego de varias semanas, logramos tener el repositorio operativo y sin problemas gracias a las recomendaciones que nos fueron brindadas en clase. Por eso usamos el repositorio creado a partir de entonces trabajando en una sola rama (*main*), porque resultó la forma más rápida y sencilla.

Creación de Rutas API REST FULL:

Al principio era fácil crear las rutas y los JSON de envío y de retorno, pero con el tiempo y después de consultas y sugerencias del profesor, hicimos algunos cambios significativos como el entregar un JSON con mucha información para que, al recibirlo, el backend pudiera cargar en secuencia varios productos o servicios a la vez. Esto fue un desafío interesante de aprendizaje y práctica.

También usamos ese tipo de JSON en algunas de las rutas para poder entregar más información y más completa, para que el Frontend pueda mostrar mejor la información que retorna el Backend.

Quisimos hacer una ruta para poder calcular los subtotales de productos y los de servicios desde una URL API REST FULL con solo la ID USER y la ID FACTURA. Perdimos mucho tiempo colocando el código en una función o dentro de otra función, y no funcionaba así que decidimos hacer los cálculos en el Frontend, y son mostrados en el detalle de la factura.

Cómo crear una Factura:

Para crear una factura hicimos que el usuario elija entre sus clientes disponibles y luego elija los productos que se cargarán en la factura, y seguidamente los servicios, y después aparece el botón de crear la factura que invoca varias rutas API REST FULL:

- 1ero se crea la factura
- 2do se cargan los productos seleccionados por el usuario a la factura recién creada
- 3ro se cargan los servicios seleccionados por el usuario a la factura recién creada

Y entonces, si no hubo problemas, ya estaría cargado en la base de datos en las respectivas tablas.

El frontend tiene dos formas de listar y presentar las tablas dinámicas.
Un método es específico para cada tipo de solicitud, por ejemplo: cliente, producto y servicio.
En cambio: stock, historial y los rankings usan una tabla dinámica que según el JSON que recibe se adapta y cambia su cantidad de columnas de manera específica.

Cuestiones que no pudieron desarrollarse o concretarse por falta de tiempo:

No se pudo hacer que se muestre y/o dispare una alarma o cartel cuando el stock es igual o menor a la columna (ALERTA STOCK) que posee la tabla producto.

No tuvimos oportunidad de hacer la página responsive y sólo es recomendable acceder a la página del sistema desde una PC, tablet o celular en modo apaisado.

No alcanzamos a hacer un diseño mucho más estético y estilizado.

Cuando se da de alta un producto o servicio y el precio termina en .00 el Javascript lo convierte a entero y el Backend espera un dato tipo float y no pasa la validación del SCHEMA que posee el backend. SCHEMA valida los datos según su tipo, un tipo compatible con el INSERT que se va a efectuar en la Base de Datos.

Otra cosa que no llegamos a realizar es la validación de datos en el frontend.

Trabajo en equipo:

El encargado de dirigir y guiar el proyecto fue Duilio, a quien todos consideramos la persona más adecuada para ello. El trabajo de equipo resultó muy gratificante, con una buena coordinación, asignación de roles y división de tareas a medida que se desarrollaba el proyecto, pero con una participación activa de todos los miembros en la toma de decisiones y en cada paso del proceso. Todos se involucraron de una u otra forma con cada aspecto del trabajo coordinando acciones y prestándonos diferentes tipos de colaboración mutua. Hicimos muchas reuniones por Zoom desde el inicio hasta el último momento, y entre muchas otras cosas también nos compartimos información, nos hicimos consultas y nos solicitamos opiniones y propuestas a través de un grupo de Whatsapp que creamos especialmente y exclusivamente para este proyecto. El grupo de Whatsapp fue la herramienta más cómoda y sencilla que utilizamos, y estuvo sumamente activo durante todo el proyecto.

El equipo fue muy positivo. Considerando que nos corrieron los tiempos, que se sumaron complicaciones personales de cada uno, además del cansancio y las exigencias del último cuatrimestre con la mayor cantidad de cursados de toda la carrera, haber logrado llegar a esta instancia en este proyecto que involucra tantos aspectos y requerimientos no es otra cosa que el resultado de un buen trabajo de equipo, explotando adecuadamente las habilidades y facilidades de cada miembro del grupo en la selección de las diferentes tareas; y algo que fue muy destacable y fundamental fue el apoyo moral mutuo y el compañerismo. Si de algo estamos seguros es de que no hubiésemos logrado llegar a este punto trabajando solos, así que destacamos la importancia y el valor del trabajo en grupo, el cual además es muy propio de nuestra profesión. En este tipo de experiencia pudimos vivenciar esto más que en ninguna otra anterior dentro de la carrera.

Conclusión sobre el trabajo:

Fue un aprendizaje sumamente interesante y enriquecedor, lleno de desafíos. Aún nos falta aprender cosas y practicar mucho, lo que llevará tiempo y se irá produciendo con la experiencia de trabajo cotidiana y la capacitación constante y gradual. Nos hubiese gustado tener la oportunidad de contar con un cuatrimestre entero exclusivamente para este proyecto y posterior a la finalización de cursados de todas las materias de la carrera, lo cual nos hubiese permitido tener más tiempo y más conocimiento previo para hacer un trabajo

mejor, más complejo y sofisticado, más completo y detallado. De cualquier forma, estamos satisfechos con lo logrado, pero sobre todo con la experiencia vivida.
