Tarea 3
Bases de Datos
"Cryptomonedas: P-Coin III Final"

Diego Altamirano
diego.altamiranot@sansano.usm.cl

Javier Mendoza
javier.mendoza@sansano.usm.cl

Felipe Quintanilla
felipe.quintanilla@sansano.usm.cl

Kevin Reyes
kevin.reyesv@sansano.usm.cl

Rodrigo Rojas
rodrigo.rojasmu@sansano.usm.cl

Cecilia Reyes
reyes@inf.utfsm.cl

15 de Junio, 2021

1. Introducción

La implementación del sistema fue un éxito, la administración de la Universidad considera su plataforma un excelente avance dentro de la infraestructura de la universidad; por otro lado, nuestro Departamento de Informática se mantiene orgulloso de su producto y de los estudiantes que lo consiguieron desarrollar.

Con el prototipo listo y, como en todo desarrollo de sistemas, continúa una parte de revisión, mantenimiento y perpetuo Kaizen. Dentro de este último, el área de tecnologías del departamento ha propuesto una mejora logística al sitio, que permita simular y/o cotizar distintos tipos de inversiones dentro de la plataforma, donde el usuario pueda entender cómo funcionan las transacciones, fluctuaciones del mercado y la importancia de la toma de decisiones con su dinero.

Sin embargo, antes de su implementación definitiva, se le encargará a los alumnos de Bases de Datos que utilicen el prototipo actual y simulen una sección que consiga todas las funcionalidades solicitadas en la nueva iteración, donde en específico, se buscará probar el funcionamiento de API's para separar el front-end del back-end en la aplicación.

Figura 1: Sansan@ glorioso alcanzando el fin de semestre

2. Instrucciones generales

Ustedes como excelentes desarrolladores, deberán **añadir a su sitio de la Tarea 2** una sección "Simulacro"la cual debe cumplir con los nuevos requisitos solicitados en el apartado de **Instrucciones Técnicas**.

La nueva sección de prueba **Simulacro** tendrá la gran novedad de que, en vez de trabajar la comunicación con la base de datos a través de consultas directas en PHP, buscará la implementación de una API en Flask, que consiga manejar toda la modificación, creación, eliminación y transacción de datos pertinentes.

Sin embargo, para no irrumpir en la implementación ya realizada (y posiblemente crear un sistema no funcional), se solicita que el uso de API's sea exclusivamente de esta nueva sección, es decir, todas las implementaciones realizadas en la Tarea 2 deben continuar trabajando con PHP y PostgreSQL como lo han hecho desde el comienzo, y sólo esta nueva implementación requerirá el uso de API's.

Desde este contexto, se trabajará con dos bases de datos en una misma plataforma, las cuales son:

- **Back-end de PHP:** Representa lo que ustedes hayan implementado en su plataforma para la entrega anterior (Tarea 2).
    - El modelo de esta BD (junto con sus interfaces asociadas) es el mismo que ustedes diseñaron para la segunda entrega, el cual, de ser correcto, no necesita modificaciones.
- **Back-end en Flask:** Representa a los nuevos requisitos mencionados para la sección **Simulacro**.
    - Este modelo estará detallado en el apartado **Modelo de Datos**, el cuál **es el mismo de la primera entrega**.

Además, como se especificará en las Instrucciones Técnicas, deberá existir una mínima mecánica para la nueva sección, donde será necesario que **uno haya iniciado sesión** (independiente del nivel de acceso) para acceder a esta.

3. Modelo de Datos

3.1. Back-end de PHP

Utilizará el modelo que cada grupo haya diseñado para su segunda entrega.

3.2. Back-end de Flask

Utiliza el modelo de la primera entrega, el cual se representa con el siguiente diagrama:

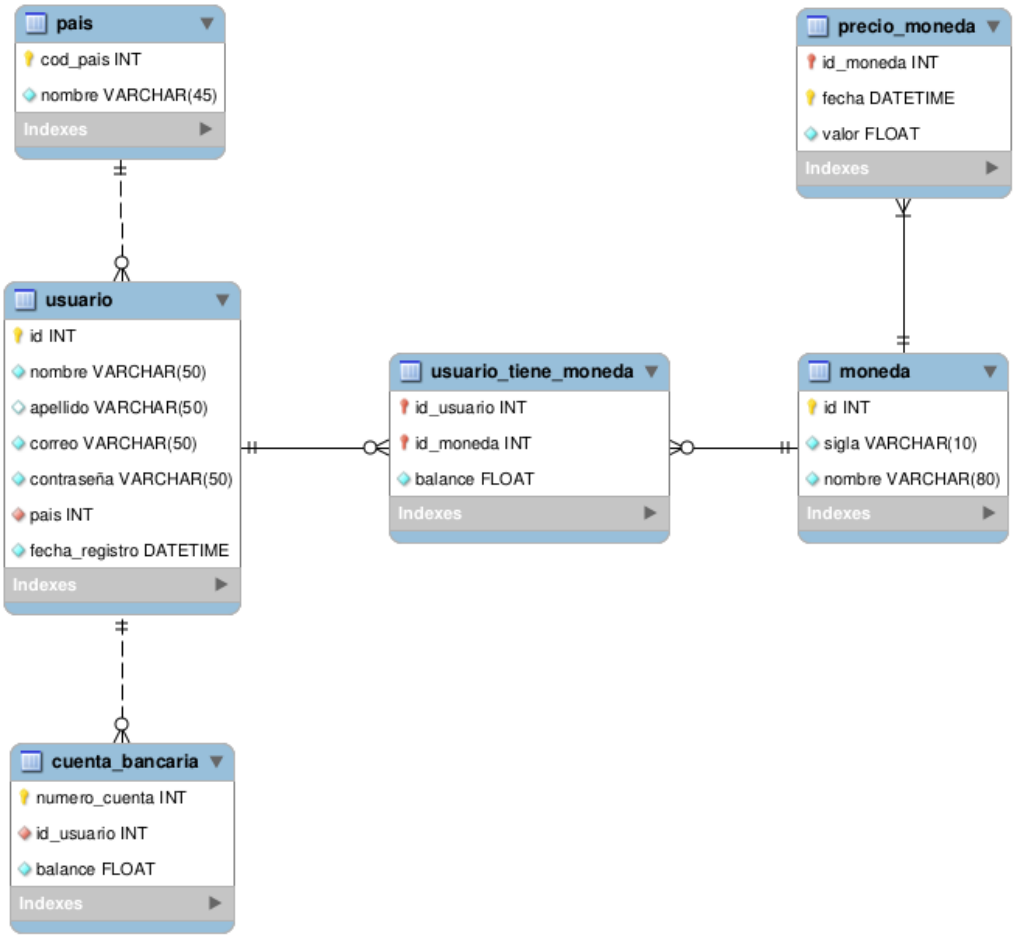


Figura 2: Modelo de Datos (para API)

4. Instrucciones Técnicas

La nueva pestaña de **Simulación** buscará conectarse con una base para la lectura y cotización de distintas monedas para un usuario, donde se puedan manejar los datos de manera separada del front-end, y sea mediante un lenguaje más familiar como lo es Python. A grandes rasgos, se buscará que el sistema soporte una funcionalidad como el siguiente diagrama:

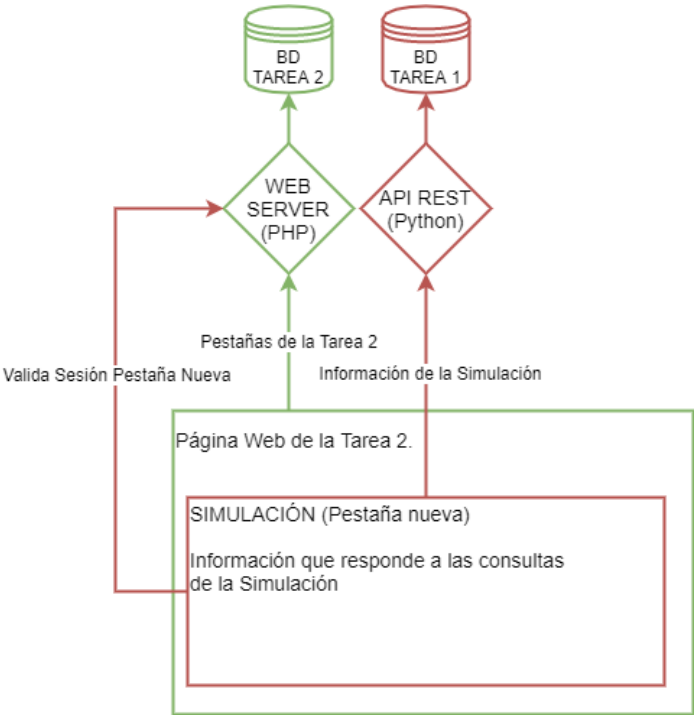


Figura 3: Diagrama de funcionalidad general para la Tarea 3

En el cual la nueva pestaña de Simulación interactúa con una API montada en Python, diseñada específicamente para el modelo de la primera entrega.

4.1. Interfaz Web

Respecto al sitio web, en la misma pestaña de Simulación deben existir dos subsecciones: **CRUD** y **Consultas**:

4.1.1. Interfaz CRUD

Desde las implementaciones de CRUD en la API, es necesario que el usuario tenga acceso a todas las interfaces creadas en esta. Concretamente, se necesita:

- Debe existir **un acceso al CRUD de las distintas tablas del modelo**.
  - Se recomienda que, para lograr esto, la interfaz CRUD tenga un link a seis sub-pestañas, donde cada una sea el CRUD de una tabla particular.

- Cada CRUD individual debe ser capaz de listar los atributos de la tabla, editarlos, eliminarlos o crear nuevos registros.
    - Para esto, es recomendable la típica tabla de datos, donde a cada dato se le asocian sus tres operaciones (Leer, Modificar y Eliminar).

El diseño específico sigue siendo a libertad de ustedes. Mientras cumpla con lo requerido, son libres de decorar la página a su gusto.

4.1.2. Interfaz Consultas

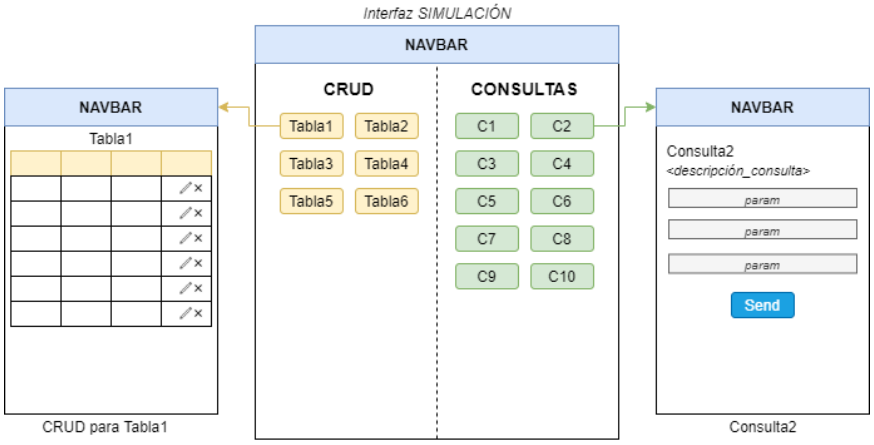
Esta interfaz consume la misma API que la anterior, sin embargo, se enfoca en utilizar las rutas diseñadas para las 8 consultas especificadas en el apartado **API en Flask**. Los detalles son:

- Debe existir un formulario o botón para cada una de las 8 consultas solicitadas, donde se le debe permitir al usuario la entrega de atributos (Input) y la visualización del resultado.
- Está permitido que, para aquellas consultas que no requieran input, se ejecuten automáticamente al cargar su vista respectiva.
- También, está permitido el juntar las consultas en una vista o separar la interfaz en distintas páginas para simplificar la lectura de resultados.
- Debe ser comprensible a cúal consulta corresponden los formularios/botones visibles para el usuario. Una solución a esto es etiquetarlos con un texto encima de los mismos.
- Además, para las consultas que requieran input, se debe indicar qué datos se deben ingresar, y en qué formato (una buena técnica es usar *placeholders* en cada espacio).

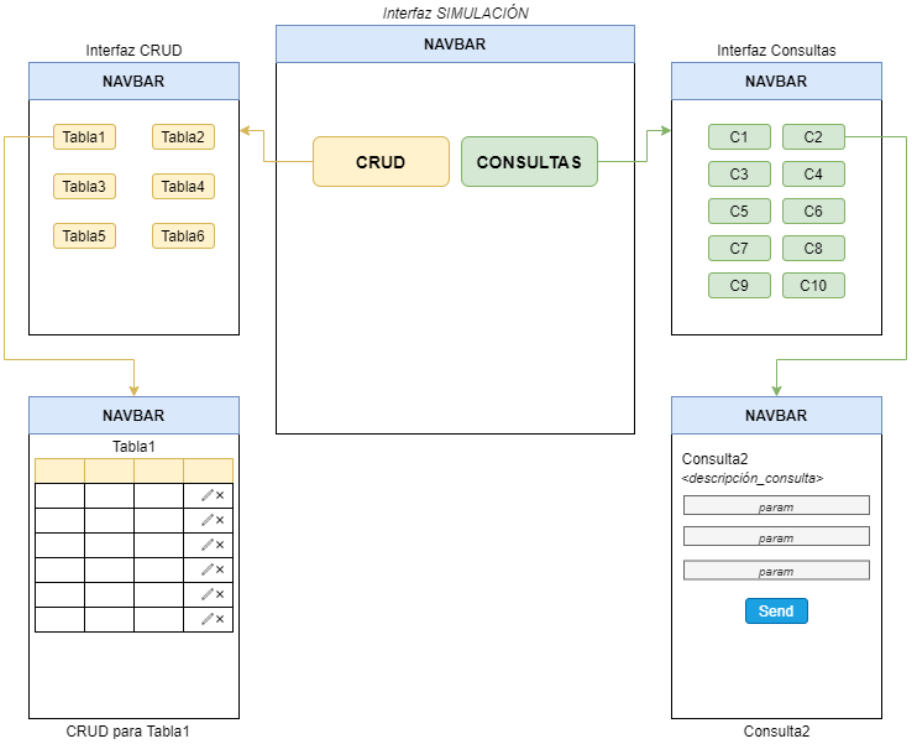
4.1.3. Ejemplos Gráficos

Para que puedan entender y/o guiarse en su desarrollo, aquí van dos ejemplos de cómo se **podría** distribuir su nueva interfaz:

- **Ejemplo concentrado**



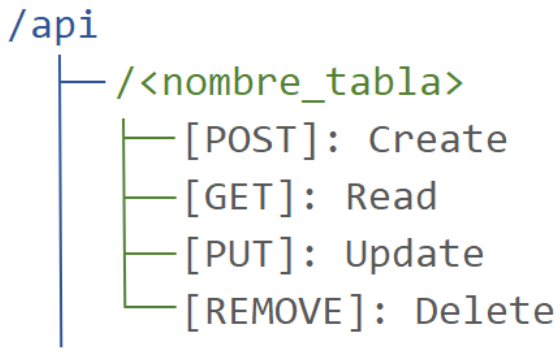
- **Ejemplo distribuido**



4.2. API en Flask

4.2.1. CRUD

Se solicita que se implemente un CRUD para **todas las entidades del modelo (T1)**, donde a través de distintas rutas, se pueda acceder a los métodos implementados en Flask. Se solicita que dichas rutas cumplan el siguiente **formato** base:



La ruta física de los métodos es **/api/nombre_tabla** y el CRUD se diferencia según el tipo de envío de formulario con el que se conecta. Por ejemplo, para añadir un **país**, debemos usar la dirección **/api/pais** y enviar la solicitud por método **POST**.

4.2.2. Soporte de Consultas

Se requiere además que su API posea rutas y métodos que soporten **las consultas solicitadas en la primera entrega** de este curso. Es decir, deben existir interfaces que permitan ejecutar las siguientes solicitudes:

- **1. Obtener todos los usuarios registrados durante el año X**
- **2. Obtener todas las cuentas bancarias con un balance superior a X**
- **3. Obtener todos los usuarios que pertenecen al país X**
- **4. Obtener el máximo valor histórico de la moneda X**
- **5. Obtener la cantidad de moneda X en circulación (Es decir, la suma de todas las cantidades de la moneda X que poseen todos los usuarios)**
- **6. Obtener el TOP 3 de monedas más populares, es decir, las que son poseídas por la mayor cantidad de usuarios diferentes**
- **7. Obtener la moneda que más cambió su valor durante el mes X**
- **8. Obtener la criptomoneda más abundante del usuario X**

Para cada una de estas consultas, debe existir una ruta que las pueda ejecutar. El formato de estas rutas debe ser, como base:

- **/api/consultas/<num_consulta>**

Donde **<num_consulta>** corresponde al número de la consulta.

5. Condiciones de Entrega

5.1. Repositorio

Para esta útlima entrega, será **obligatorio** el utilizar un **repositorio Git alojado en Gitlab** (si no poseen Gitlab, pueden usar **Github**), esto es, para que puedan mejorar su trabajo colaborativo y podamos realizar las revisiones intermedias sin que tengan que estar constantemente subiendo cosas al aula.

Dicho lo anterior, es necesario que en el primer avance, entreguen al Aula:

- Un **archivo .txt** detallando la **dirección de su repositorio**.

5.2. Revisiones iterativas

Su último proyecto es extenso y tiene numerosos requisitos a implementar. Es por esto que para apoyarlos, se irán solicitando avances iterativos de su desarrollo. Para que podamos hacer revisión, se le solicitará que nos den acceso a nosotros ayudantes a su Repositorio donde trabajarán. Los usuarios se detallan en la tabla 1.

En concreto existirán tres iteraciones, las cuales requerirán:

1. **Implementación del CRUD** (Fecha: 6 de Julio a las 23:59 hrs.)
   - Debe estar implementado el CRUD en la API
   - Los métodos deben tener listas las rutas
2. **Conexiones Iniciales y Primeras 5 consultas** (Fecha: 22 de Julio a las 23:59 hrs.)
   - Debe existir la interfaz para el CRUD en el sitio, con los formularios conectados a la API
   - Deben estar implementadas las cinco primeras consultas en código (Python), con la ruta asignada
3. **Final, 8 consultas** (Fecha: 30 de Julio a las 23:59 hrs.)
   - Deben estar las 8 consultas implementadas y con ruta asignada
   - Debe existir la interfaz de consultas para cada una de ellas
   - La interfaz debe consumir la API implementada

Nombre	Gitlab	Github
Diego	dialtami	dialtami
Felipe	fquintan	Feldoy
Kevin	kreyes	kevinReyesv7
Javier	jamendoz	jamendozc
Rodrigo	rgomunita	rgomunita

Cuadro 1: Usuarios de Gitlab y Github