

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN



IIC2513 - Sección 1 — Tecnologías y Aplicaciones Web

Tarea 2

Entrega

- **Fecha y hora**: sábado 13 de septiembre del 2025, a las 22:00.
- **Lugar:** Repositorio individual en la organización del curso en GitHub, main branch.
- **Link de Invitación**: este corresponde a su repositorio individual. Pueden haber recibido un mail con una invitación.

Objetivos

- **Diseñar y desarrollar** una API que siga los principios RESTful.
- **Definir** los recursos de la API, tales como las entidades y las rutas necesarias para acceder a estas.
- **Producir** documentación efectiva y clara del proceso realizado.

Descripción

En la tarea anterior, tuvieron la oportunidad de diseñar, crear y probar una aplicación tipo marketplace de Pokémon, consumiendo una API. ¡Ahora ha llegado el momento de revertir los roles! Para esta tarea, les corresponderá crear la API que será consumida por un frontend preexistente de PokePlace, el cual les será entregado por el equipo docente.

El objetivo principal de esta tarea es que logren hacer funcionar la aplicación entregada construyendo una API RESTful desde cero con Express y Node.js.

Este proyecto les permitirá comprender cómo se construyen y se manejan las interfaces de programación de aplicaciones en el contexto del desarrollo web moderno, enfocándose en la integración entre el backend y el frontend, y en la importancia de la documentación adecuada.

Para esta tarea, se les proveerá un código base que incluirá algunas configuraciones básicas de Express y Node.js para su API. Dentro de este marco de trabajo, deberán desarrollar una API que gestione recursos para permitir que PokePlace pueda quedar operativa. Cada usuario podrá crear pokémons, publicarlos y venderlos a otros usuarios de la aplicación. Todo como si fuese un marketplace, pero de Pokémon. Y la versión que se les ha entregado de PokePlace tiene una feature nueva! Ahora puedes dejar reseñas a los vendedores de pokémons.

Para todas estas acciones, deberán crear y utilizar métodos HTTP como `GET` para recuperar información, `DELETE` para eliminar recursos preexistentes, `POST` para crear nuevos recursos y `PUT/PATCH` para actualizar recursos existentes, manejando toda la lógica que hay detrás.

Adicionalmente, dado que el frontend ya está desarrollado y utiliza `fetch` para las conexiones, deberán asegurarse de que las respuestas de su API se alineen con las expectativas del frontend, facilitando una integración fluida y funcional.

1. Especificaciones

1.1. Entidades y modelos

Se les entrega un diagrama de clases, el cual se encuentra en la carpeta `docs`. Este tiene como finalidad que puedan entender la relación entre las 3 entidades principales a desarrollar en la base de datos de la aplicación de PokePlace:

- **User**: Esta entidad representa al usuario de la aplicación. Cada usuario es identificado de manera única por su *id*, que actúa como la *primary key* de la entidad. Este atributo es necesario para la autenticación y para vincular los pokemons al usuario correcto. Además, los usuarios cuentan con los siguientes atributos:
  - `name`: un campo de tipo *string* que almacena el nombre del usuario
  - `image`: un campo de tipo *string* que almacena la url de donde se encuentra la imagen de usuario o foto de perfil.
  - `password`: un campo de tipo *string* que almacena la contraseña del usuario.
  - `balance`: un campo de tipo *float* que almacena el monto o saldo de pokecoins de un usuario.
- **Pokemon**: Esta entidad representa un registro de un Pokemon del usuario. Cada registro tiene un identificador único *id* de tipo *int*, que sirve como la *primary key*. La entidad Pokemon incluye los siguientes atributos:
  - `name`: un campo de tipo *string* que almacena el nombre del Pokemon.
  - `level`: campo de tipo *int* que almacena el nivel del Pokemon. Este va del 1 al 100.
  - `types`: es un campo que almacena un arreglo de *strings* (`array<str>`) indicando los tipos de un Pokemon.
  - `price`: campo de tipo *int* que almacena el precio de un Pokemon. Este va de 50 a 1000 pokecoins.
  - `image`: un campo de tipo *string* que almacena la url de donde se encuentra la imagen del Pokemon.
  - `onSale`: un campo de tipo *booleano* que permite identificar si el Pokemon se encuentra a la venta o no.
  - `userId`: un campo de tipo *int* que indica el id del usuario que es dueño del Pokemon.
- **Review**: esta entidad representa el registro de la reseña que deja un usuario comprador al usuario vendedor. Las reseñas poseen los siguientes atributos:
  - `authorId`: es un campo de tipo *int* que funciona como la *foreign key* del modelo, estableciendo la relación con la entidad User, indicando qué usuario dejó la reseña.
  - `reviewedId`: es un campo de tipo *int* que funciona como la *foreign key* del modelo, estableciendo la relación con la entidad User, indicando a qué usuario se le dejó la reseña.
  - `comment`: un campo de tipo *string* que es el texto de la reseña creada por el usuario.
  - `rating`: campo de tipo *int* que indica que tan conforme ha quedado el usuario con la compra-venta. Toma valores del 1 al 5.

Basándose en la descripción proporcionada, deberán implementar los 3 modelos en su base de datos. Es fundamental prestar atención a los tipos de datos específicos de cada atributo y asegurarse de que las relaciones entre los modelos reflejen fielmente la estructura y las conexiones indicadas en el diagrama relacional. Esto significa, por ejemplo, que deberán establecer una relación uno a muchos entre usuarios y pokémons, donde un único usuario pueda estar vinculado a múltiples pokémons, pero cada pokémon pertenezca exclusivamente a un solo usuario. Deberán hacer todas las relaciones, al igual que con las *reviews*.

1.2. Migraciones y *Seeds*

Después de crear correctamente las entidades, el siguiente paso es aplicar migraciones para estructurar la base de datos según estos modelos.

Además, es necesario generar *seeds* para poder probar la funcionalidad y la integridad de su aplicación. Para esto, deben crear *seeds* que incluyan, como mínimo, tres instancias de la entidad *User*, además de crear 9 instancias de pokémons (3 para cada usuario), con diferentes combinaciones de pokémons en estado *onSale*, y deberán generar 6 instancias de la entidad *review*, con al menos una reseña en cada usuario.

1.3. Configuración de Rutas en `routes.js`

En el archivo `routes.js`, su tarea es completar las rutas que aún no se han definido. Deberán importar los módulos de rutas necesarios desde el directorio `routes`, utilizando la función `require`, y luego registrarlos con el enrutador de su aplicación. Las rutas deben configurarse de la siguiente manera:

- Asigne la ruta `/users` para manejar las peticiones relacionadas con la entidad **User**.
- Asigne la ruta `/pokemons` para manejar las peticiones relacionadas con la entidad **Pokemon**.
- Asigne la ruta `/reviews` para manejar las peticiones relacionadas con la entidad **Review**.

Es importante que las rutas se definan exactamente como se especifica arriba. Cualquier diferencia de las rutas puede resultar en que el frontend, ya desarrollado, no sea capaz de comunicarse correctamente con los endpoints del backend.

1.4. Rutas

Para cada modelo, se deberá implementar las siguientes rutas.

- **Rutas para la entidad `User`**:
  - **Registro de Usuario**:
    - Método `POST` en la ruta `/users/register`. Este *endpoint* permite registrar un nuevo usuario en la aplicación, validando que el nombre no esté en uso ni con campos vacíos. Retorna el usuario creado sin incluir la contraseña.
  - **Login de Usuario**:
    - Método `POST` en la ruta `/login`. Este endpoint permite autenticar a un usuario registrado mediante su `name` y `password`.
    - En caso de un login correcto, se retorna un token y también al usuario logeado, sin incluir su contraseña.
    - En caso de credenciales inválidas, el endpoint responderá con código `401`.
    - La lógica del token ha sido implementada por el equipo docente y se detalla su uso en el punto 1.5. del enunciado.
  - **Obtención de Todos los Usuarios**:
    - Método `GET` en la ruta `/users`. Retorna un listado con todos los usuarios registrados, excluyendo sus contraseñas.
  - **Obtención de Usuario por ID**:

- Método `GET` en la ruta `/users/:id`. Retorna la información de un usuario específico, identificado por su `id`. Excluye la contraseña.

- **Actualización de Usuario:**
  - Método `PUT` en la ruta `/users/:id`. Este *endpoint* actualiza los datos de un usuario específico (nombre, contraseña, balance, imagen). Solo el propio usuario autenticado puede realizar esta acción. Debe utilizarse el middleware auth para proteger esta ruta. Es decir, esta acción solo se permite si existe un token válido. Para más información ver item 1.5. del enunciado.
- **Eliminación de Usuario:**
  - Método `DELETE` en la ruta `/users/:id`. Este *endpoint* elimina a un usuario específico de la aplicación. Solo el propio usuario autenticado puede realizar esta acción, por lo que debes utilizar el middleware `auth` especificado en el item 1.5. del enunciado.

- **Rutas para la entidad** `Pokemon`:
  - **Creación de Pokémon:**
    - Método `POST` en la ruta `/pokemons`. Permite crear un nuevo Pokémon asociado a un usuario. Valida que se proporcionen los campos requeridos (`name`, `types`, `level`, `image`, `userId`).
    - En caso de datos inválidos, retorna `400`.
  - **Obtención de Todos los Pokémons:**
    - Método `GET` en la ruta `/pokemons`. Retorna un listado de todos los Pokémons registrados en el sistema.
  - **Obtención de Pokémons de un Usuario:**
    - Método `GET` en la ruta `/pokemons/user/:id`. Retorna todos los Pokémons pertenecientes a un usuario específico.
  - **Obtención de Pokémons en Venta de un Usuario:**
    - Método `GET` en la ruta `/pokemons/user/:id/on-sale`. Retorna los Pokémons de un usuario que están actualmente a la venta. (*id* corresponde al id del usuario)
  - **Poner un Pokémon en Venta**:
    - Método `POST` en la ruta `/pokemons/:pokemonId/put-on-sale/:userId`. Permite que el propietario de un Pokémon lo ponga a la venta, cambiando *onSale* a *True*.
    - Si el Pokémon ya está en venta, retorna `400`.
    - Si el usuario no es el propietario, retorna `403`.
  - **Cancelar la Venta de un Pokémon:**
    - Método `POST` en la ruta `/pokemons/:pokemonId/cancel-sale/:userId`. Permite cancelar la venta de un Pokémon previamente puesto en el mercado. Solo el propietario puede realizar esta acción. Se debe cambiar el atributo *onSale* a *False*
    - Si el Pokémon no está en venta, retorna `400`.
    - Si el usuario no es el propietario, retorna `403`.
  - **Compra de un Pokémon:**
    - Método `POST` en la ruta `/pokemons/:pokemonId/buy/:userId`. Permite a un usuario autenticado de *userId* comprar un Pokémon de *pokemonId* que se encuentre a la venta. La transacción solo puede ocurrir si se tiene el saldo suficiente. Si tiene el saldo (*balance*) suficiente, se hace el descuento de los pokecoins de su cuenta junto con aumentar los pokecoins del vendedor.
    - Debes utilizar el middleware `auth` para proteger este endpoint y verificar que el usuario esté autenticado.

    - Si el balance del usuario es insuficiente, retorna 400.
- **Actualización de Pokémon:**
    - Método PUT en la ruta /pokemons/:pokemonId/:userId. Permite al usuario de userID, dueño de un pokemon de *pokemonId*, actualizar sus atributos (name, level, types, price, image).
    - Si no se recibe ningún campo válido, retorna 400.
    - Si alguno de los campos es inválido (por ejemplo, nivel fuera de rango), retorna 400.
    - Si el usuario no es el propietario, retorna 403.
- **Eliminación de Pokémon:**
    - Método DELETE en la ruta /pokemons/:pokemonId/:userId. Permite al propietario eliminar un Pokémon permanentemente.
    - Si el usuario no es el propietario, retorna 403.

- **Rutas para la entidad** Review:
    - **Obtención de Reseñas de un Usuario:**
        - Método GET en la ruta /reviews/user/:userId. Retorna todas las reseñas asociadas al usuario identificado por userId, es decir, retorna todas las reseñas que le han dejado a dicho usuario.
    - **Creación de Reseña**:
        - Método POST en la ruta /reviews. Permite crear una reseña indicando explícitamente authorId, reviewedId, rating y opcionalmente comment.
        - Si faltan campos requeridos, retorna 400.
        - Si el rating no es un número entero entre 1 y 5, retorna 400.
    - **Actualización de Reseña**:
        - Método PUT en la ruta /reviews/:id/:userId. Permite que el autor de la reseña (identificado por userId) modifique el comentario o la puntuación de la reseña identificada por id.
        - Si el usuario (userId) no corresponde al autor de la reseña, retorna 403.
        - Si el nuevo rating no es un entero válido entre 1 y 5, retorna 400.
    - **Eliminación de Reseña:**
        - Método DELETE en la ruta /reviews/:id/:userId. Permite eliminar la reseña identificada por id, siempre que el usuario autenticado sea su autor (parámetro userId).
        - Si el usuario no es el autor de la reseña, retorna 403.

Cada *endpoint* debe ser diseñado para reflejar correctamente la respuesta mediante códigos de estado HTTP consistentes y mensajes informativos. Para lograr esto, cada *endpoint* debe incluir una respuesta (res) incluso para los casos de error, siguiendo el siguiente formato estandarizado:

- En caso de que un recurso o entidad no se encuentre:

```
res.status(404).json({ error: '{Entidad} not found' }).
```

- En caso de un error durante la operación:

```
res.status(codigo).json({ error: error.message })
```

donde `codigo` debe corresponder al estado HTTP adecuado según la naturaleza del error (2XX, 4XX y 5XX), garantizando que el cliente reciba una señal clara del resultado de su solicitud.

Ahora bien, en caso de éxito, por ejemplo, al crear exitosamente un usuario, la respuesta se debe ver de la siguiente manera:

```
res.status(201).json({ user })
```

Es importante que sean consistentes con los códigos HTTP que se utilizan en el frontend para evitar problemas, es decir, no enviar una respuesta con status 203 cuando se espera un status 201 desde el frontend.

1.5. Token y Middleware

Se les ha entregado el código de un middleware (o función intermedia) que protege rutas verificando que el cliente envíe un token válido. Las rutas que deberán proteger son:

- POST `pokemons/:pokemonId/buy/:userId` para comprar un pokemon.
- PUT `users/:id` para actualizar un usuario
- DELETE `users/:id` para eliminar un usuario

Es decir, para realizar estas acciones, se necesita de un token válido enviado desde el frontend, garantizando que existen los permisos para realizar esta acción. Para esto, deberán utilizar el middleware `auth` que se les entregó listo en el archivo `middlewares/auth.js`.

Para el correcto funcionamiento del middleware, simplemente deberán seguir los siguientes pasos:

1. Generar un token, escribiendo el siguiente código en el endpoint para el `login`. Recordar que este token debe ser retornado por el endpoint, junto con el usuario.

```
    const token = jwt.sign(
        { id: user.id },
        'PANCONQUESO',
        { expiresIn: '1h' }
    );
```

   Ojo que 'PANCONQUESO' es un string y no debe tener ningún espacio en su interior. Por eso, se recomienda escribir el código a mano, no copiar y pegarlo directamente (tanto el del paso 1 como el del paso 2).

2. Para que lo anterior funcione, es necesario importar las siguiente dependencia en el archivo donde estén creando el endpoint del `/login`. Es decir, en el archivo donde crearon el token, deben importar jsonwebtoken:

```
    const jwt = require('jsonwebtoken');
```

En resumen, con esos simples dos pasos van a poder crear un token para los usuarios que inician sesion. Lo que deben hacer es proteger las 3 rutas mencionadas anteriormente, usando el middleware `auth` que se les entregó (y que no requiere de modificación).

1.6. Documentación

Este ítem tiene como propósito:

- Indicar apropiadamente cómo correr su programa.
- Completar todos los puntos especificados en el `README.md`

2. Consideraciones

- **Código Frontend**:

  En el siguiente link se les entrega el frontend ya implementado. Deben hacer un fork y seguir las instrucciones del Readme.md para correr la aplicación.

  Este sistema está configurado para interactuar con el backend de manera automática. No deben realizar modificaciones en el código del frontend, con la única excepción de actualizar el archivo `config.js` para definir la URL del servidor local (localhost) que usará la aplicación.

  En caso de que presenten alguna dificultad con algún endpoint, se recomienda fuertemente ver cómo se está utilizando el endpoint a nivel de frontend. Esto les puede dar indicios de cómo hacerlo correctamente en el backend.

- **Código Backend**: Para el código base proporcionado del backend no se permite la adición de nuevas librerías. Solo están permitidas aquellas ya incluidas en el archivo `package.json`, con la excepción de reemplazar Express por Koa si así lo desean. Este cambio no afectará negativamente su calificación. Sin embargo, cualquier otra modificación fuera de esta excepción resultará en una calificación automática de 1.0, sin posibilidad de re-corrección.

- **Video Explicación** En el siguiente link pueden encontrar un video en el cual se introduce la aplicación que deben hacer funcionar, junto con un par de tips para abordar la tarea. Recordar que el frontend está listo. Simplemente deben crear la API (backend) con todos los endpoints especificados en el enunciado.

3. Rúbrica

A continuación, se describe el criterio de cada uno de los ítems de la rúbrica con la que se evaluará esta entrega. La nota se calcula al 50 % considerando el puntaje descrito.

- **Entidades [19 puntos]**: Definir correctamente las entidades, junto con sus atributos y relaciones especificadas en el punto 1.1.
- **Migraciones y *seeds* [13 puntos]**: Migrar correctamente el modelo y que las seeds cumplan con el mínimo estipulado en el punto 1.2.
- **Routes.js [3 puntos]**: Importar e instanciar correctamente las rutas especificadas en el puntos 1.3.
- ***Endpoints* [55 puntos]**: Crear y exportar correctamente todos los metodos especificados en el punto 1.4. Además de incluir las respuestas de estado HTTP correspondientes a los endpoints.
- **Documentación [10 puntos]**: Deben indicar cómo correr tu programa y responder las preguntas indicadas en el *template* que te facilitamos en tu repositorio.

4. Bonus

Se entregará un bonus de 5 décimas a aquellos que entreguen una documentación en Swagger o en un archivo `.collections` de Postman. Esto debe realizarse para todos los endpoints de las entidades *User* y *Pokemon*. La documentación debe subirse en la raíz del repositorio o indicar en el README cómo acceder a ella.

Para cada *endpoint*, deberán entregar:

- Método del *endpoint*

- Url completa del *endpoint*
- Body con los parámetros necesarios para el *endpoint*
- Ejemplo de respuesta exitosa (200/201)
- Ejemplo de respuesta errónea (4XX)

El bonus se otorgará de la siguiente forma:

- 5 décimas: documentando la totalidad de endpoints de las entidades *User* y *Pokemon*, además de contar con todo lo solicitado para cada *endpoint*.
- 2.5 décimas: documentando más del 50 % de los endpoints de las entidades *User* y *Pokemon*, además de contar con todo lo solicitado para cada *endpoint*.
- 0 décimas: documentando menos del 50 % de las entidades *User* y *Pokemon*

Cabe destacar que, para recibir el bonus completo o parcial, deberán entregar toda la información solicitada de cada *endpoint*.

5. Integridad académica

Cualquier situación de **falta a la integridad académica** detectada en el contexto del curso (por ejemplo, durante alguna evaluación) tendrá como **sanción un 1**,**1 final en el curso**. Esto sin perjuicio de sanciones posteriores que estén de acuerdo a la Política de Integridad Académica de la Escuela de Ingeniería y de la Universidad, que sean aplicables al caso. Rige para este curso tanto la política de integridad académica del Departamento de Ciencia de la Computación (ver anexo) como el **Código de honor de la Escuela de Ingeniería**.

Dudas

Cualquier duda que se presente acerca del enunciado debes consultarla en las ***issues*** del repositorio del curso. Recuerden que como equipo docente estaremos atentos para poder ayudarlos :)