FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico

Técnicas de Diseño

Cuatrimestre y año: 1C 2023

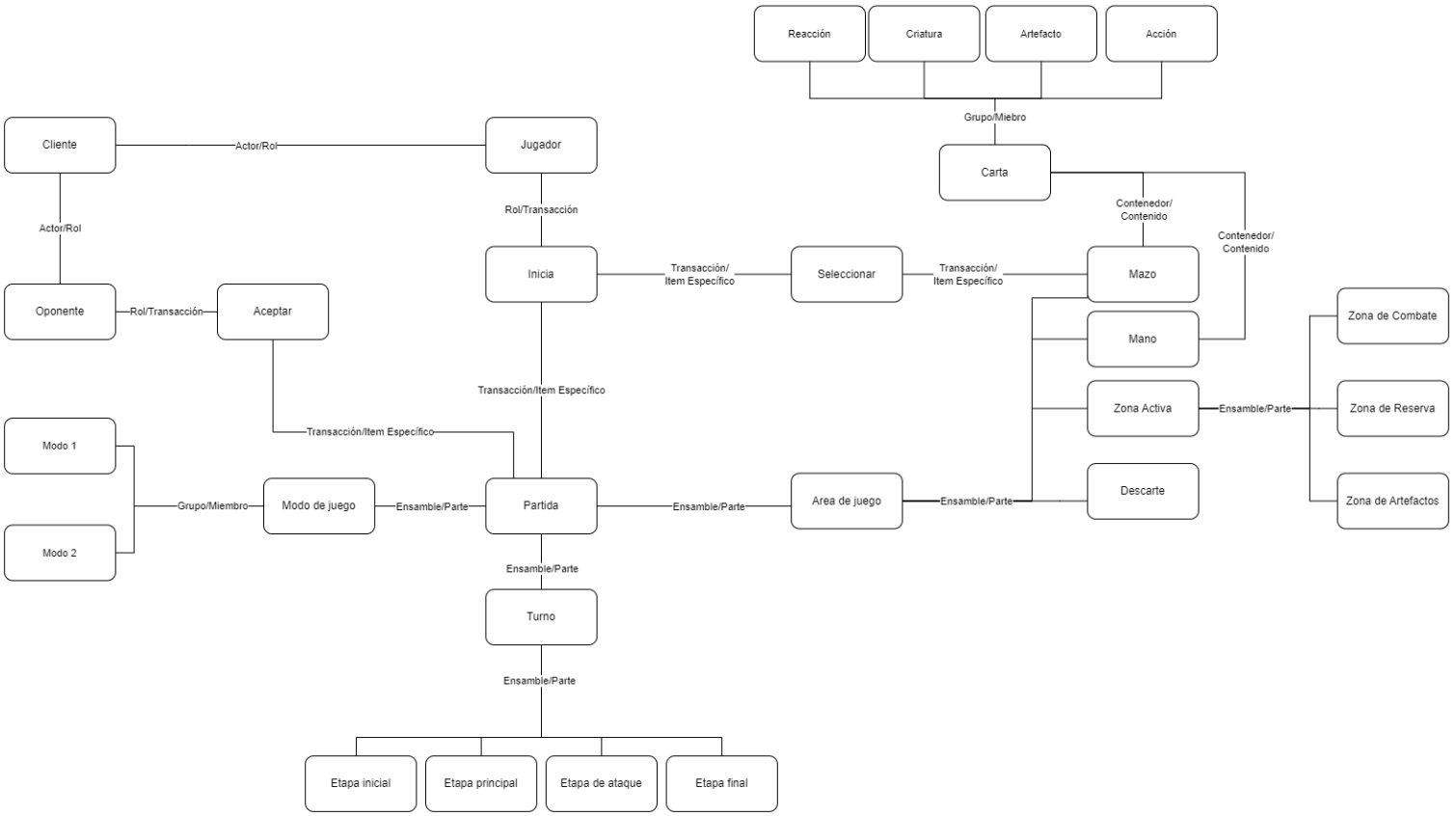
Fecha de entrega: 22 de junio de 2023

Alumnos:

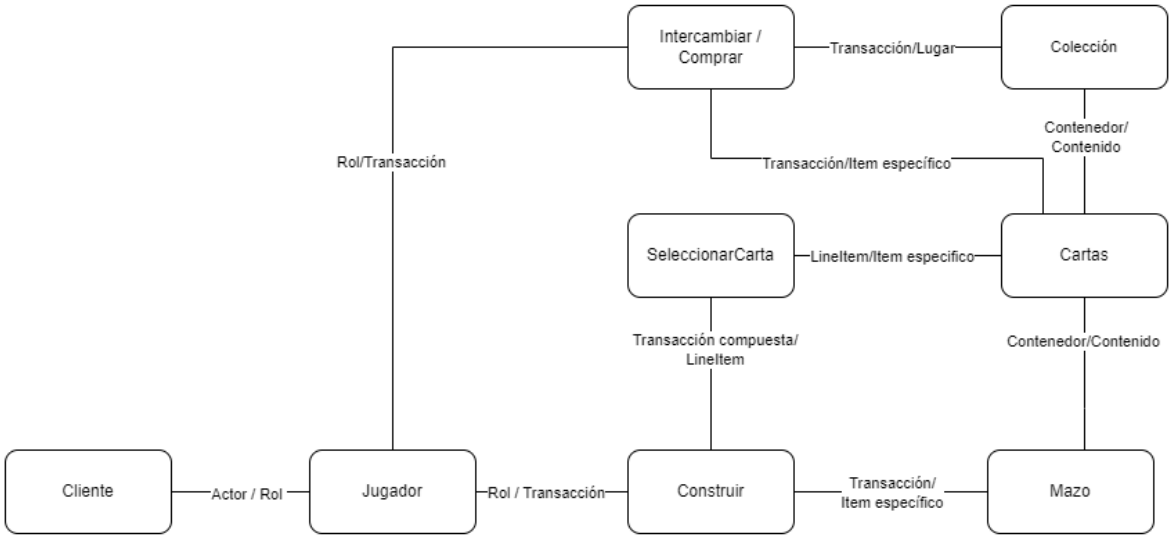
Nombre	**Padrón**
Lucas Montenegro	102412
Julieta Ponti	104694
Matías Rotondo	103852
Pablo Martín	105587
Ignacio Brusati	105586

Modelo de Dominio

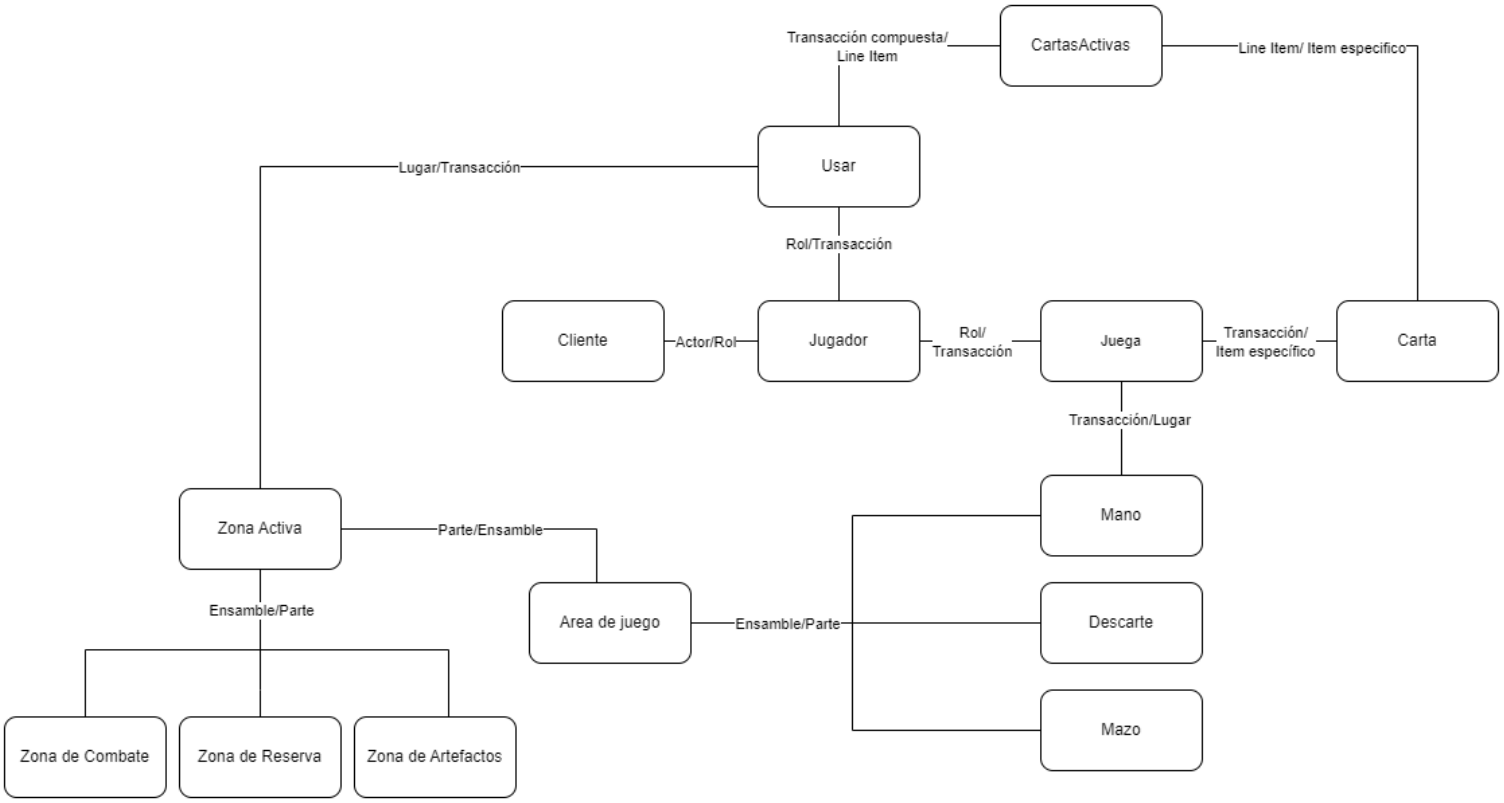
Partida



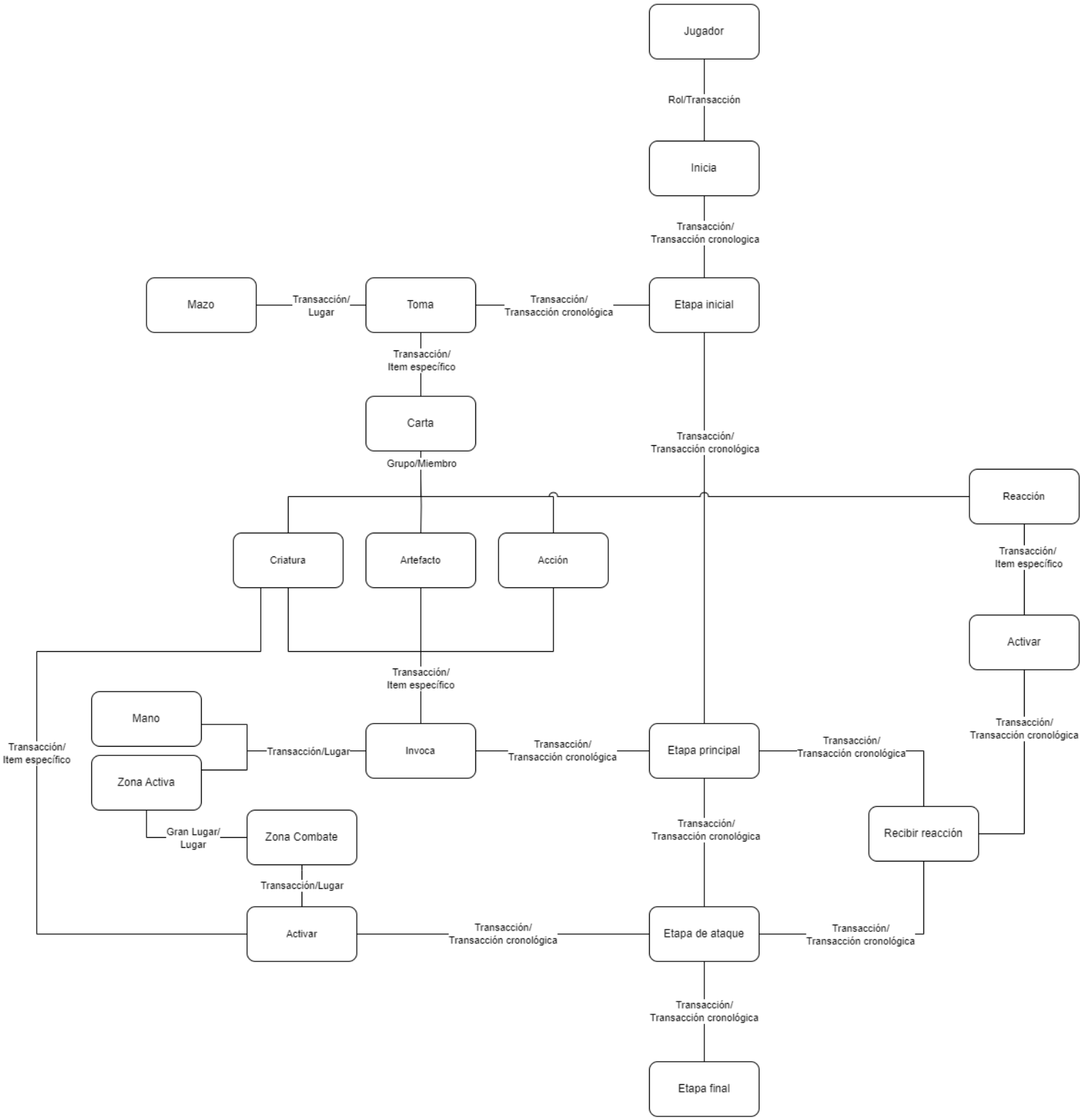
Mazos



Área de juego

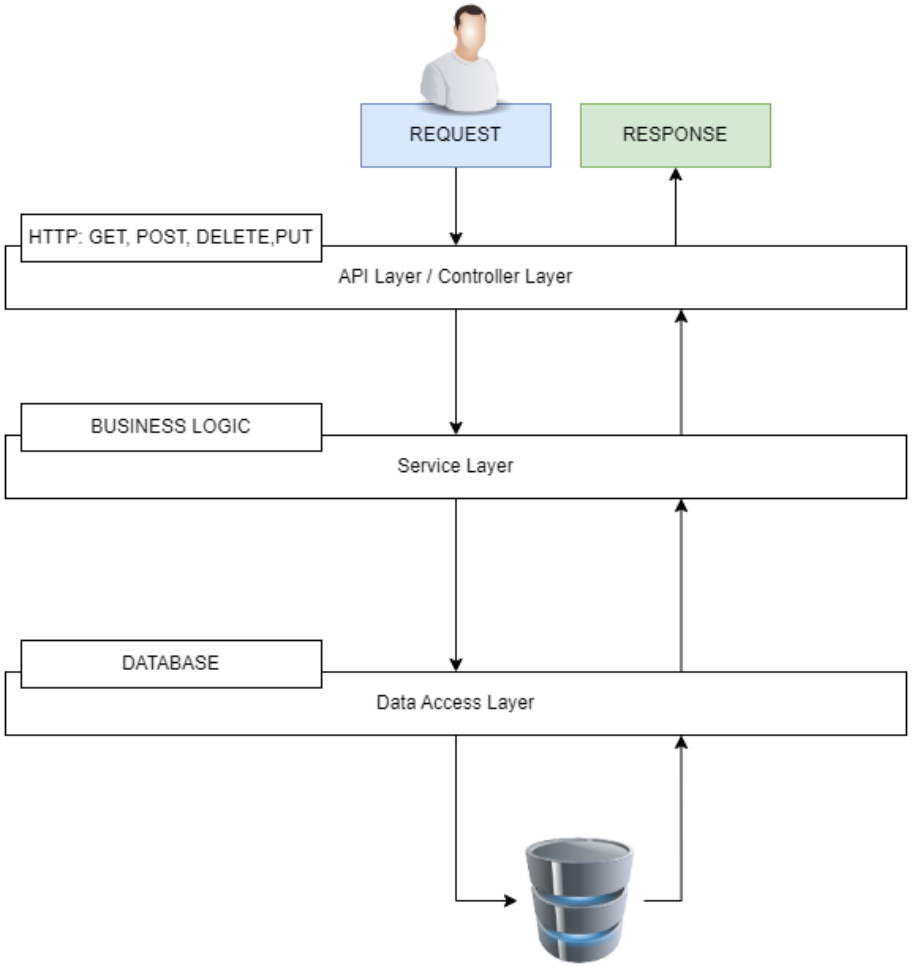


Turno


Jugador
Rol/Transacción
Inicia
Transacción/
Transacción cronologica
Mazo
Transacción/
Lugar
Toma
Transacción/
Transacción cronológica
Etapa inicial
Transacción/
Item específico
Carta
Grupo/Miembro
Transacción/
Transacción cronológica
Reacción
Criatura
Artefacto
Acción
Transacción/
Item específico
Activar
Transacción/
Item específico
Transacción/
Transacción cronológica
Mano
Zona Activa
Transacción/Lugar
Invoca
Transacción/
Transacción cronológica
Etapa principal
Transacción/
Transacción cronológica
Transacción/
Item específico
Gran Lugar/
Lugar
Zona Combate
Transacción/
Transacción cronológica
Recibir reacción
Transacción/Lugar
Activar
Transacción/
Transacción cronológica
Etapa de ataque
Transacción/
Transacción cronológica
Transacción/
Transacción cronológica
Etapa final

Arquitectura

Se propone una arquitectura en capas, la cual está compuesta por 3 capas. La primera capa es la capa de control, en la cual se va a encontrar la API, la segunda capa es el modelo de negocio, y como tercer capa está la capa de acceso a datos.
La API va a ser el medio por el cual cada cliente se comunique con el modelo, es decir que el cliente va a realizar las solicitudes a la API la cual se va a comunicar con el modelo para que este resuelva la solicitud y pueda ser retornada al cliente. Para resolver cada solicitud, se van a utilizar los datos obtenidos a partir de la capa de acceso a datos. Para el alcance de este trabajo no se utilizará una base de datos, pero se implementó la capa de acceso a datos como una abstracción para una posible implementación de base de datos. Es decir que esta capa solo utiliza datos que están en memoria.

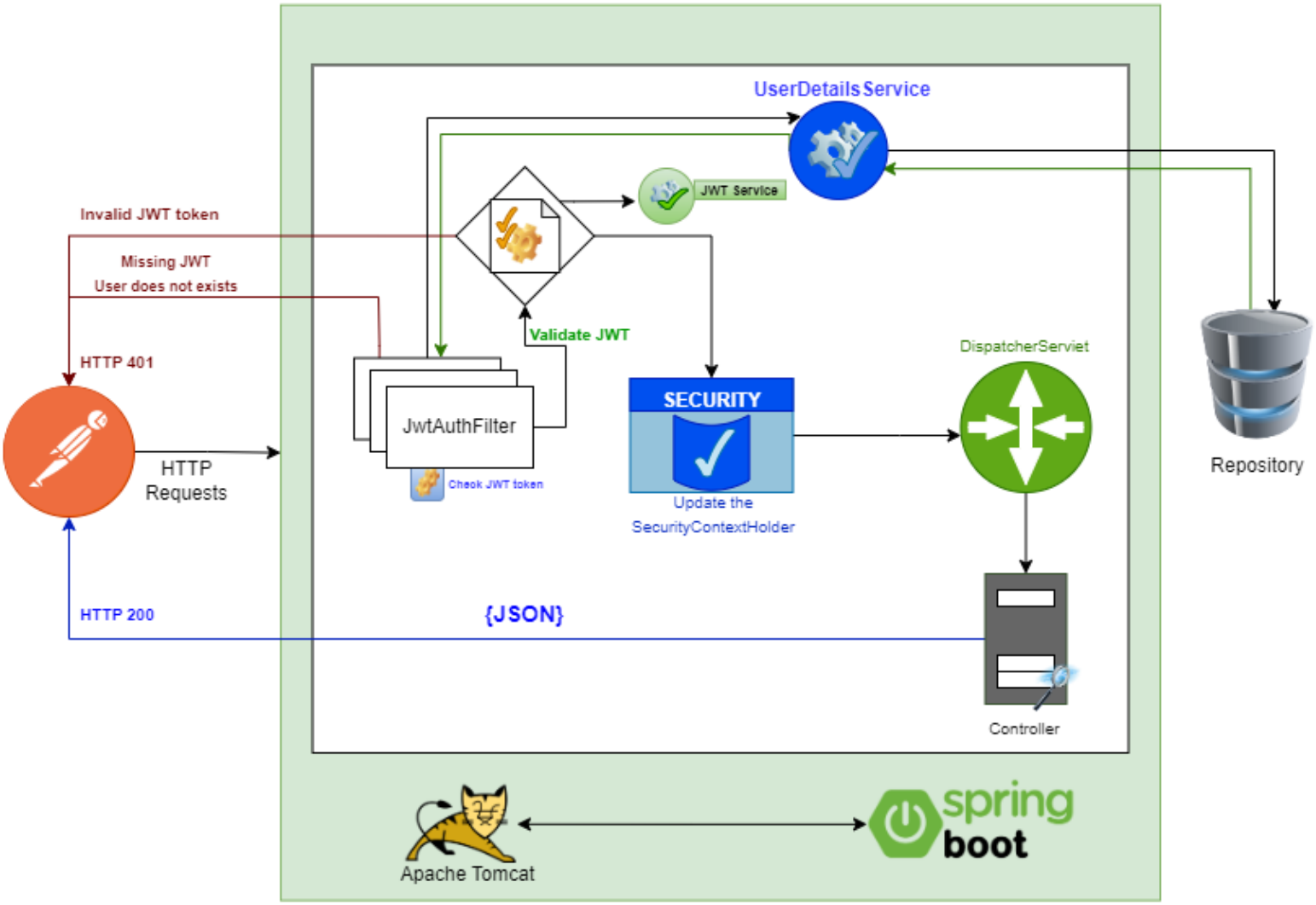


La aplicación está dockerizada y se puede correr de forma local utilizando el comando docker-compose up -d –build, el cual levanta el contenedor que ejecuta la aplicación y lo deja corriendo en segundo plano. Además se puede modificar el archivo *.env* de ser necesario, en el cual se encuentra la variable de entorno del puerto local que se va a utilizar, el cual mapea al puerto 8080 del contenedor. De esta forma podemos usar distintos puertos en distintos

entornos. Además en el trabajo se incluye una colección de POSTMAN para poder probarla de manera local una vez levantado el contenedor.

Seguridad de la API

En cuanto a seguridad de la API se utilizó el protocolo JWT para la autenticación y autorización de los clientes. En el siguiente diagrama se representa el funcionamiento del protocolo implementado.



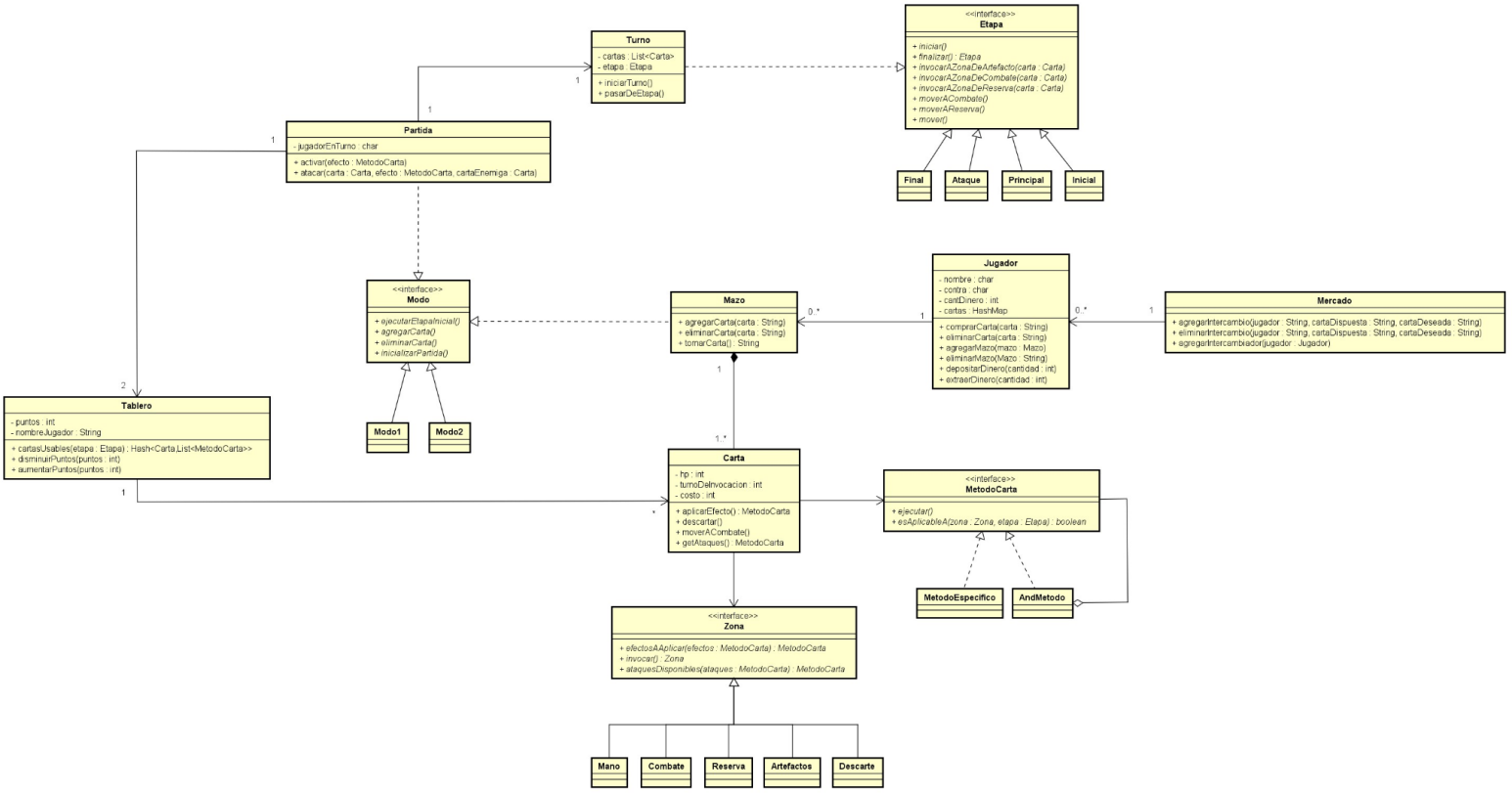
En este caso se decidió que todos los endpoints necesitan autenticación excepto los de registro y login. Es decir que solo pueden utilizar la API clientes los cuales se hayan registrado previamente y se autentiquen con un token válido. Los token generados expiran cada 15 minutos y tienen un único rol posible, que es el de jugador.

El flujo indicado por el diagrama es el siguiente. Cada solicitud recibida pasa por el filtro JwtAuthFilter, el cual va a filtrar todas las solicitudes que no tengan jwt y no sean solicitud de registro ni de login. En caso de recibir un jwt, este se va a encargar de obtener los datos del usuario a partir de UserDetailsServic, el cual en este caso solo va a tener usuarios cargados en memoria ya que como se mencionó anteriormente, no hay base de datos. Una vez hecho esto, a partir de los datos obtenidos, y el nombre de usuario que se extrae del jwt, se pasa a la etapa de validación, en la cual se verifica que el token sea válido, por ejemplo que no haya expirado, y que corresponda al usuario que está realizando la solicitud. Una vez hecho esto, se autoriza al token a realizar la consulta, el controler (en este caso el modelo) resuelve la consulta y se retorna al cliente.

Modelo de vistas 4 +1

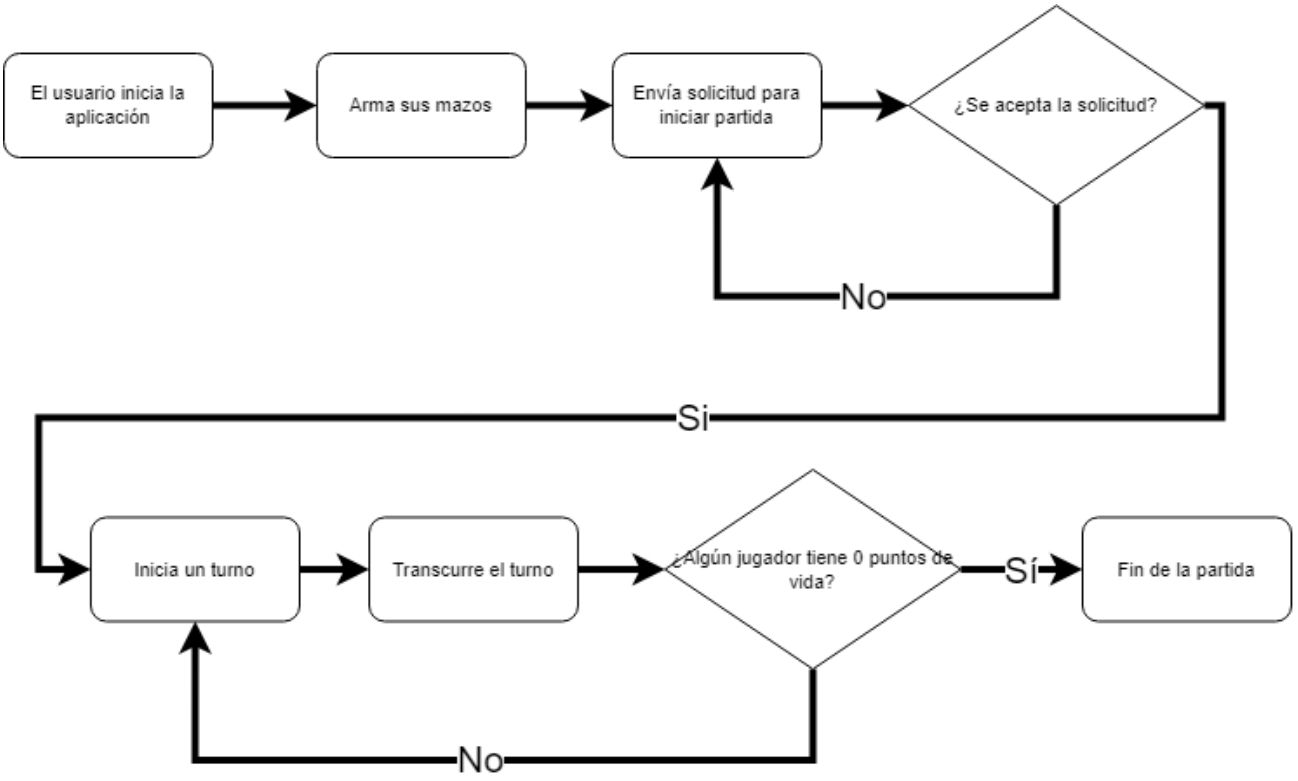
Vista lógica

A partir del dominio del problema planteado en el modelo de dominio se decidió realizar las abstracciones presentadas en el diagrama de clases. Cada cliente que se registre mediante la API, va a ser una instancia de la abstracción jugador. A su vez las interacciones entre clientes estará representada por medio de la abstracción partida, de la cual habrá una instancia por cada par de instancias jugadores que jueguen una partida. A continuación se muestra un diagrama de clases para entender la lógica del modelo.



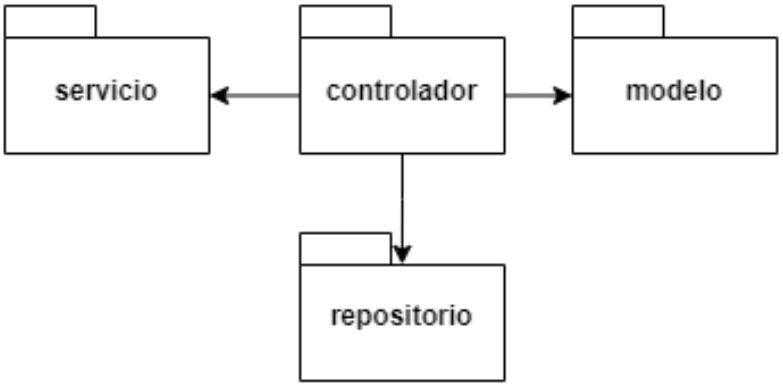
Vista de procesos

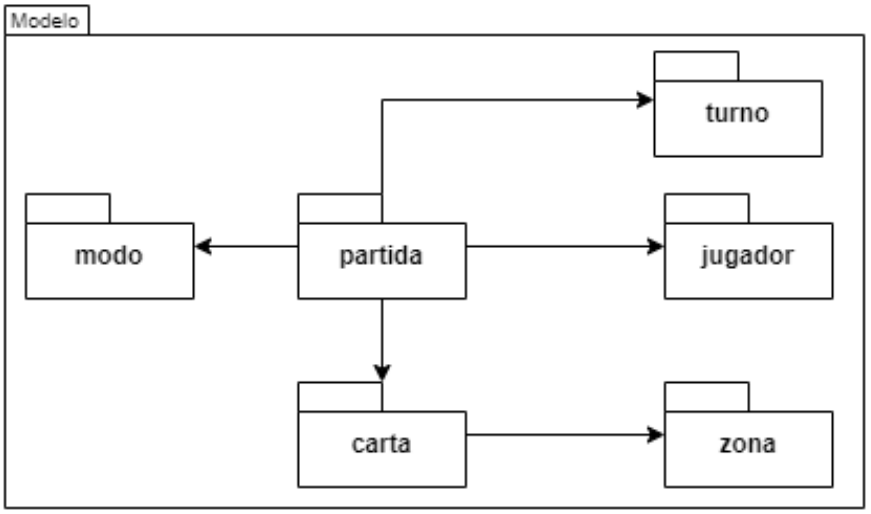
La aplicación va a tener principalmente dos procesos, donde cada proceso es una agrupación de tareas que forman una unidad ejecutable. El primer tipo de proceso es el proceso de un cliente activo, y el segundo proceso será la aplicación corriendo.
El proceso cliente se va a iniciar cuando un usuario levante un cliente, en este caso un cliente va a ser representado por POSTMAN, el cual va a realizar peticiones al otro proceso que está corriendo. Mientras que el otro proceso será el encargado de recibir consultas de procesos clientes, este proceso estará corriendo de manera local en un contenedor de docker.
A continuación se muestra un diagrama de flujos.



Vista de desarrollo

A continuación realizamos un diagrama de paquetes que muestra una idea general de cómo van a estar organizados los módulos de software en el ambiente de desarrollo. El software fue empaquetado en partes pequeñas con el fin de que puedan ser desarrollados en forma independiente y de manera sencilla.





Vista física

Se usará una computadora para poder levantar el backend dockerizado, de manera que se pueda puedan atender solicitudes de los usuarios y el desarrollo de las partidas de manera local. Además el cliente estará corriendo en los servidores de postman. Se necesitará tener conexión a internet para que se puedan comunicar.

Vista de escenarios

- Un jugador puede comprar una carta de otro jugador
- Un jugador puede realizar un intercambio de carta con otro jugador
- Un jugador puede iniciar una partida
- Un jugador puede tener varios mazos
- Un jugador no puede terminar una etapa cuando no es su turno

Preguntas entrega final

Agregar al informe final el análisis hecho durante la segunda entrega de qué cambios serían necesarios para implementar los siguientes pedidos, detallando de manera concreta los cambios a realizar

1. Agregar cartas nuevas
   - Pez: [Criatura] 4HP, 1 Agua. Ataque: Quita 2HP
   - Cese al fuego: [Reacción] Ante un ataque, ignorar todo daño y efecto de dicho ataque
   - Vampiro: [Criatura] 5HP, 1 Planta. Ataque: Quita 3HP, cura a sí mismo por 1HP con un límite máximo de sus puntos de vida iniciales
2. Agregar una nueva regla: Si un jugador tiene que tomar cartas de su mazo y no hay cartas suficientes, pierde inmediatamente
3. Modo combinado que trackea puntos de victoria y vida (Mismas condiciones para generar cada uno que sus modos originales)
4. Energía nueva: Metal
5. Para simplificar la regla de no activar artefactos en el mismo turno que se invocan, queremos separar la etapa principal en dos:
   a. Etapa de activación de artefactos (Solo activación de artefactos)
   b. Etapa de invocación (Toda la etapa principal excepto activar artefactos)
6. Al final de la partida, se le regala alguna cantidad de dinero a ambos jugadores

Respuestas:

1.

Carta Pez

Deberíamos agregar a nuestro enum CartasDisponibles una nueva carta, lo único que necesitamos en este caso sería pasar, a través del constructor, su nombre, "Pez", su vida, en este caso 4, su costoDeEnergia(fuego:0, planta:0, agua:1), su tipo (En este caso Tipo.Criatura) y por último el MetodoCarta Atacar con hp igual a 2.

Carta "Cese al fuego"

Nuevamente deberíamos agregar una nueva carta a nuestro enum CartasDisponibles, esta vez con el nombre "Cese al fuego", su tipo (Tipo.Reaccion) y su método. Al ser muy similar a impedir podría usarse su mismo método, pero refactorizando de manera que acepte ataques en vez de reacciones.

Carta "Vampiro"

En este caso, podríamos reutilizar el MetodoCarta curar, que ya se utilizaba para la carta "Hospital", pero con un ligero refactor donde se pase el limite de vida a curar, para que no sea mayor a la cantidad de puntos de vida inicial.

De esa manera utilizamos nuestro MetodoCartaCompuesto con atacar de hp 3 y curar con hp 1 y límite 5.
A su vez, deberíamos, como en los casos anteriores, agregar la carta a nuestro enum con los siguientes parámetros, Nombre "Vampiro", Tipo.Criatura, Hp : 5, MetodoCartaCompuesto de ataque y curar con sus correspondientes parámetros (para atacar 3 de hp y para curar 1 hp con límite 5)

2.

En este caso, vale aclarar que cada modo (Modo1, Modo2) implementa una interfaz Modo y cada uno tiene su forma de calcular el jugador ganador a través del método CalcularGanador, cada vez entonces que se ejecute la etapa inicial, deberíamos chequear en calcularGanador, a partir de esta nueva condición si es que hubo un ganador, luego, Partida chequea si es que hubo un ganador con su método VerificarGanador.

3.

Dada nuestra implementación, en la que cada modo implementa la interfaz modo, deberíamos realizar algo similar que lo que hicimos con los modos ya existentes. En este caso deberíamos crear un Modo3 que cumpla ambas condiciones simultáneamente, esto se puede realizar componiendo los modos 1 y 2 en el nuevo Modo3 creado, por lo que no habría que realizar cambios al código más que agregar la nueva funcionalidad.

4. Para agregar el nuevo tipo de energía, “Metal”, debemos realizar algunas modificaciones. En primer lugar, debemos modificar el enum Energía agregando el nuevo tipo. Por otro lado, como consecuencia debemos modificar el enum CartasDisponibles, modificando los costos de uso e invocación de cada carta. Por último, se realizarán las modificaciones necesarias en los métodos existentes que requieran el uso de esta nueva energía.

   En cuanto al jugador, es necesario realizar cambios en el Tablero para que acepte esta nueva energía en su hashmap.

5.

En nuestra implementación, todas las etapas implementan una interfaz Etapa.
Para separar la “etapa principal” en “Etapa de activación de artefactos” y “Etapa de invocación”, es necesario reorganizar la lógica correspondiente.
La lógica de invocación deberá ser trasladada a la “etapa de invocación”, mientras que el resto de la lógica será ubicada a la “etapa de activación de artefactos”.
Además, será necesario agregar las firmas de los métodos restantes de la interfaz en cada etapa y manejar adecuadamente las excepciones en los casos en que una etapa no tenga que implementar ciertos métodos.

6.

En el caso de nuestro modelo, el jugador de la partida (se podría decir qué Tablero) se encuentra desconectado del Jugador (que es quien tiene el dinero, realiza la compra y los

intercambios de cartas). Una posible solución, es que los tableros tengan la instancia de Jugador correspondiente. De esta manera al chequear la condición de victoria se podría llamar al jugador del tablero y utilizar el método depositarDinero para sumar la cantidad de dinero correspondiente.

Documentación endpoints

En el repositorio de GitLab se encuentra tanto el archivo .json de la colección de POSTMAN, como el archivo generado por el swagger.

Registro de jugador

```
path: /api/auth/registrarse:
  post:
    tags:
      - Auth
    summary: Registrar usuario
    requestBody:
      content:
        application/json:
          schema:
            type: object
            example:
              username: pedro
              password: '1234'
    security:
      - noauthAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Logueo de jugador

```
path:  /api/auth/loguearse:
  post:
    tags:
      - Auth
    summary: Loguear usuario
    requestBody:
      content:
        application/json:
          schema:
            type: object
            example:
              username: juan
```

```
            password: '1234'
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Cartas existentes

```
path:  /api/cartas:
  get:
    tags:
      - Cartas
    summary: Lista de cartas y precios
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Consultar el precio de una carta

```
path:  /api/cartas/{nombreCarta}:
  get:
    tags:
      - Cartas
    summary: Precio de una carta
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Cartas de un jugador

```
path:  /api/jugadores/cartas:
  get:
    tags:
      - Cartas
    summary: Cartas del jugador
    security:
      - bearerAuth: []
```

```
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Comprar una cantidad de cartas

```
path:  /api/cartas/comprar/{nombreCarta}/{cantidad}:
  post:
    tags:
      - Cartas
    summary: Comprar carta
    requestBody:
      content: {}
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Consultar jugadores logueados

```
path:  /api/jugadores:
  get:
    tags:
      - Jugador
    summary: Jugadores enfrentables
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Consultar el dinero de un jugador

```
path:  /api/jugadores/dinero:
  get:
    tags:
      - Jugador
    summary: Dinero disponible
    security:
```

```
    - bearerAuth: []
  responses:
    '200':
      description: Successful response
      content:
        application/json: {}
```

Depositar dinero

```
path:  /api/jugadores/depositar/2000:
  post:
    tags:
      - Jugador
    summary: Depositar dinero
    requestBody:
      content: {}
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener mazos existentes

```
path:  /api/mazos:
  get:
    tags:
      - Mazo
    summary: Mazos
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Agregar un mazo

```
path: /api/mazos/agregar/{nombreMazo}:
  post:
    tags:
      - Mazo
    summary: Crear mazo
```

```
    requestBody:
      content:
        application/json:
          schema:
            type: object
            example:
              AGUA: 40
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Eliminar un mazo

```
  path: /api/mazos/eliminar/{nombreMazo}:
    post:
      tags:
        - Mazo
      summary: Eliminar mazo
      requestBody:
        content: {}
      security:
        - bearerAuth: []
      responses:
        '200':
          description: Successful response
          content:
            application/json: {}
```

Agregar cartas a un mazo

```
 path:  /api/mazos/agregarCartas/{nombreMazo}:
    post:
      tags:
        - Mazo
      summary: Agregar cartas a un mazo
      requestBody:
        content:
          application/json:
            schema:
              type: object
              example:
                FUEGO: 1
      security:
        - bearerAuth: []
```

```
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Eliminar cartas de un mazo

```
path:  /api/mazos/eliminarCartas/{nombreMazo}:
  post:
    tags:
      - Mazo
    summary: Eliminar cartas de mazo
    requestBody:
      content:
        application/json:
          schema:
            type: object
            example:
              ESPADAMAGICA: 3
              GOBLIN: 2
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Ver mercado

```
path:  /api/mercado:
  post:
    tags:
      - Intercambio de cartas
    summary: Entrar al mercado
    requestBody:
      content: {}
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Ver los intercambios disponibles en el mercado

```
path: /api/mercado/intercambiosMercado:
  get:
    tags:
      - Intercambio de cartas
    summary: Intercambios disponibles
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Ver mis intercambios en el mercado

```
path: /api/mercado/intercambios:
  get:
    tags:
      - Intercambio de cartas
    summary: Mis intercambios
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Realizar un intercambio

```
path:  /api/mercado/intercambiar:
  post:
    tags:
      - Intercambio de cartas
    summary: Intercambiar carta
    requestBody:
      content:
        application/json:
          schema:
            type: object
            example:
              cartaDispuesta: FUEGO
              cartaDeseada: GOBLIN
    security:
      - bearerAuth: []
    responses:
```

```
      '200':
        description: Successful response
        content:
          application/json: {}
```

Eliminar un intercambio

```
path: /api/mercado/eliminarIntercambio:
  post:
    tags:
      - Intercambio de cartas
    summary: Eliminar intercambio
    requestBody:
      content:
        application/json:
          schema:
            type: object
            example:
              cartaDispuesta: AGUA
              cartaDeseada: GOBLIN
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener mi tablero en la partida

```
path: /api/partidas/tablero:
  get:
    tags:
      - Partida > Tablero
    summary: Tablero
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener el tablero de mi jugador contrincante en la partida

```
path: /api/partidas/tableroEnemigo:
```

```
  get:
    tags:
      - Partida > Tablero
    summary: Tablero enemígo
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener las cartas usables

```
path:  /api/partidas/cartasUsables:
  get:
    tags:
      - Partida > Tablero
    summary: Cartas usables
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener las cartas atacables

```
path:  /api/partidas/cartasAtacables:
  get:
    tags:
      - Partida > Tablero
    summary: Cartas atacables
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener las solicitudes de partidas

```
path: /api/partidas/solicitudes:
  get:
```

```
    tags:
      - Partida
    summary: Solicitudes de partida
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Solicitar una partida

```
path:  /api/partidas/solicitar/1:
  post:
    tags:
      - Partida
    summary: Solicitar partida
    requestBody:
      content:
        application/json:
          schema:
            type: object
            example:
              contrincante: pedro
              mazo: mazoOfensivo
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Aceptar una partida

```
path: /api/partidas/aceptarPartida:
  post:
    tags:
      - Partida
    summary: Aceptar solicitud de partida
    requestBody:
      content:
        application/json:
          schema:
            type: object
            example:
              contrincante: juan
```

```
            mazo: mazoOfensivo
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener los puntos en una partida

```
path:  /api/partidas/puntos:
  get:
    tags:
      - Partida
    summary: Obtener puntos
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener el ganador de una partida

```
path: /api/partidas/ganador:
  get:
    tags:
      - Partida
    summary: Ganador
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener información de quién es el turno

```
path:  /api/partidas/turno:
  get:
    tags:
      - Partida
    summary: Jugador en turno
    security:
```

```
    - bearerAuth: []
  responses:
    '200':
      description: Successful response
      content:
        application/json: {}
```

Obtener información acerca de qué etapa se encuentra el turno

```
path:  /api/partidas/etapa:
  get:
    tags:
      - Partida
    summary: Etapa
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener informacion sobre la energia de un jugador

```
path:  /api/partidas/energía:
  get:
    tags:
      - Partida
    summary: Energía
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Obtener información sobre la energía del jugador contrincante

```
path:  /api/partidas/energiaEnemigo:
  get:
    tags:
      - Partida
    summary: Energía enemiga
    security:
```

```
    - bearerAuth: []
  responses:
    '200':
      description: Successful response
      content:
        application/json: {}
```

Solicitar la invocación de una carta

```
path: /api/partidas/invocarCarta:
  post:
    tags:
      - Partida
    summary: Invocar carta
    requestBody:
      content:
        application/json:
          schema:
            type: object
            example:
              carta: FUEGO
              zona: ZonaArtefacto
    security:
      - bearerAuth: []
    responses:
      '200':
        description: Successful response
        content:
          application/json: {}
```

Solicitar la finalización de la etapa actual

```
 path:  /api/partidas/terminarEtapa:
   post:
     tags:
       - Partida
     summary: Pasar de etapa
     requestBody:
       content: {}
     security:
       - bearerAuth: []
     responses:
       '200':
         description: Successful response
         content:
           application/json: {}
```

Solicitar activar una carta

```
path: /api/partidas/activarCarta:
    post:
      tags:
        - Partida
      summary: Activar carta
      requestBody:
        content:
          application/json:
            schema:
              type: object
              example:
                carta: 0
                indiceMetodo: 0
                jugadorObjetivo: juan
                cartasObjetivos: []
                energia: ''
      security:
        - bearerAuth: []
      responses:
        '200':
          description: Successful response
          content:
            application/json: {}
```