2024-2 / IIC2173 - E2 | CPE CoolGoat List Async

aka. Coordinación de Pagos en base a Eventos

Fecha de entrega: *28/08/2024 - 3.5 Semanas*

Dado que se lograron acoplar correctamente al servicio asíncrono, su empresa favorita LegitBusiness les da el visto bueno para formalizar estos pagos que antes se hacían a mano. De esta forma se espera que puedan integrarse con WebPay para luego indicarle al resto de los grupos cuando la transacción se haya completado con exito.

Objetivo

La entrega está intencionada para que creen un diseño sólido de una aplicación utilizando diversas herramientas clave a la hora de ofrecer una aplicación cualquiera. Usando esto la finalidad funcional de esta entrega es que puedan gestionar el sistema de pagos asíncrono con una integración con WebPay síncrono. Además, se introducirá el uso de Serverless como tecnología altamente útil y escalable.

Compra de bonos

Su flujo de compra desde la entrega 1 consiste en enviar mensajes a través de un broker a todos los grupos esperando la validación con los siguientes estándares:

Canal **`fixtures/requests`**

```
{
    "request_id": uuid,
    "group_id": string (número de grupo),
    "fixture_id": <int>
    "league_name": string,
    "round": string,
    "date": date,
    "result":string,
    "deposit_token": "",
    "datetime": string (YYYY-MM-ddThh:mm:ss UTC),
    "quantity": number,
    "wallet": bool,
    "seller": 0
}
```

Canal **`fixtures/validation`**

```
{
{
    "request_id": string,
    "group_id": string,
    "seller": 0,
    "valid": bool (indica si fue válida la solicitud)
}
}
```

Como se puede ver se agrego un nuevo dato al request que se debe mandar a `fixtures/requests` que es si es wallet, esto es un boleano para que quede mas claro si tienen que esperar de otros grupos la validation o de nosotros.

Ahora tendrán la posibilidad de realizar un flujo entre medio para poder "pagar" formalmente mediante ambientes de pruebas de WebPay.

Para esto, cuando quieran realizar una solicitud de compra con webpay deberán:

1. Mandar la solicitud via WebPay para obtener el **token de la solicitud** de compra.

2. Enviar el mensaje por el canal de `fixtures/requests` indicando la solicitud, donde el **token de la solicitud** se enviará en el campo `deposit_token` .
3. Iniciar un proceso de pago a través de WebPay.
4. Usuario paga a través de WebPay.
5. **Enviar la respuesta** por el canal `fixtures/validation` indicando si se realizó con exito o no.
6. Si es certero el resultado, se recibe la ganancia en su wallet.

En resumen, la idea es que ahora envíen la solicitud de compra al broker en `fixtures/requests` con el `deposit_token` específico y luego inicien el flujo de pago para un usuario con WebPay desde su frontend. Finalmente deberán responder que se hizo la solicitud correctamente a su usuario y enviar el resultado por el canal `fixtures/validation` para que el resto de grupos lo vea.

Integración con WebPay

Pueden ver el detalle de la integración en el repositorio de la ayudantía de la API de Transbank (https://github.com/iic2173/ayudantia-webpay). Este paso se debe hacer para realizar el pago con los usuarios.

Flujo de recomendación de partido

Para esta entrega además se les pedirá poder crear recomendaciones de resultado para un usuario y poder revisarlas a futuro para proponerle futuras compras facilmente.

El flujo principal de esto es que un usuario, luego de realizar una compra, se correrá un script que tomará todos sus partidos revisados, comprados, y su eleccion para obtener al menos 3 recomendaciones.

Una vez iniciada la recomendación, deberá haber una vista de recomendaciones generadas para el usuario donde podrá ver todas las recomendaciones de partidos generadas en orden de la más reciente a la más antigua. Si un partido se repite en las recomendaciones solo será adelantado en la lista.

Como este proceso puede tomar una cantidad de tiempo significativa se les pedirá que lo implementen usando el concepto de **workers asíncronos** que se explica más adelante para poder mantener una buena performance.

Para el cálculo de las recomendaciones deberán:

1. Obtener el resultado de todas sus compras.
2. Obtener los proximos partidos de los equipos que estaban involucrados en todas sus compras, importante que aparezcan por liga.
3. Se calcula cuantas veces a acertado de manera historica(en todas sus apuestas) a los resultados, cuando uno de los equipo de los proximos partidos a estado presente.
   pd: esto toma en cuenta cuantas veces aposto en el mismo partido
4. Se calculan los beneficios de los partidos para cada liga tomando en cuenta la cantidad de aciertos antes calculada, la cantidad de partidos de los equipos con los que acerto(en que round van) y la probabilidad, y se ordenan en base a la siguiente fórmula:

$$pond = \frac{Aciertos * round\_de\_liga}{\sum odd\_de\_equipo\_involucrado}$$

5. Se obtienen los 3 partidos con los mejores ponderadores y se los muestran al usuario

Implementacion de los workers

Un worker es una instancia de un código generada específicamente para cumplir una tarea asignada, luego de lo cual podría tomar otra, esperar o desaparecer, en base a la alimentación que reciba de una cola de órdenes.

Estos workers se coordinan con un broker y una instancia maestra de coordinación.

Afortunadamente, existen paquetes de software que simplifican el trabajo de la implementación de workers.

Recomendamos las siguientes implementaciones para los siguientes lenguajes

- FastAPI, Django
  - Celery
    - Pueden investigar sobre la herramienta *Flower* para la visualización de estos
- Node
  - Bull
- Ruby on rails
  - Sideqik

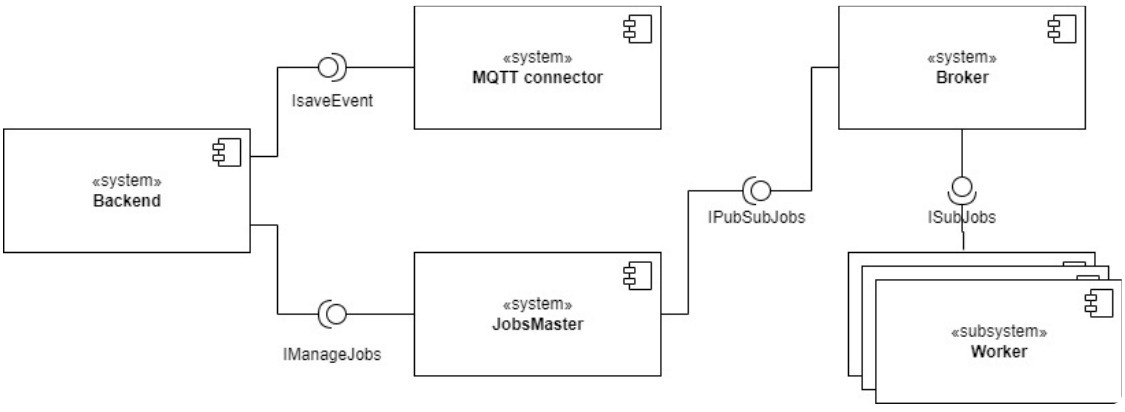
Estos workers deben estar coordinados por un maestro independiente de su aplicación. Este maestro es completamente separado de su aplicación y ofrece una API HTTP para recibir las órdenes y ver los resultados. Ofrecerá tres endpoints

- **GET** /job/(:id)
  - :id representa el id de un job creado
- **POST** /job
  - Recibe los datos necesarios para el pago y entrega un id del job creado
- **GET** /heartbeat
  - Indica si el servicio está operativo (devuelve `true`)

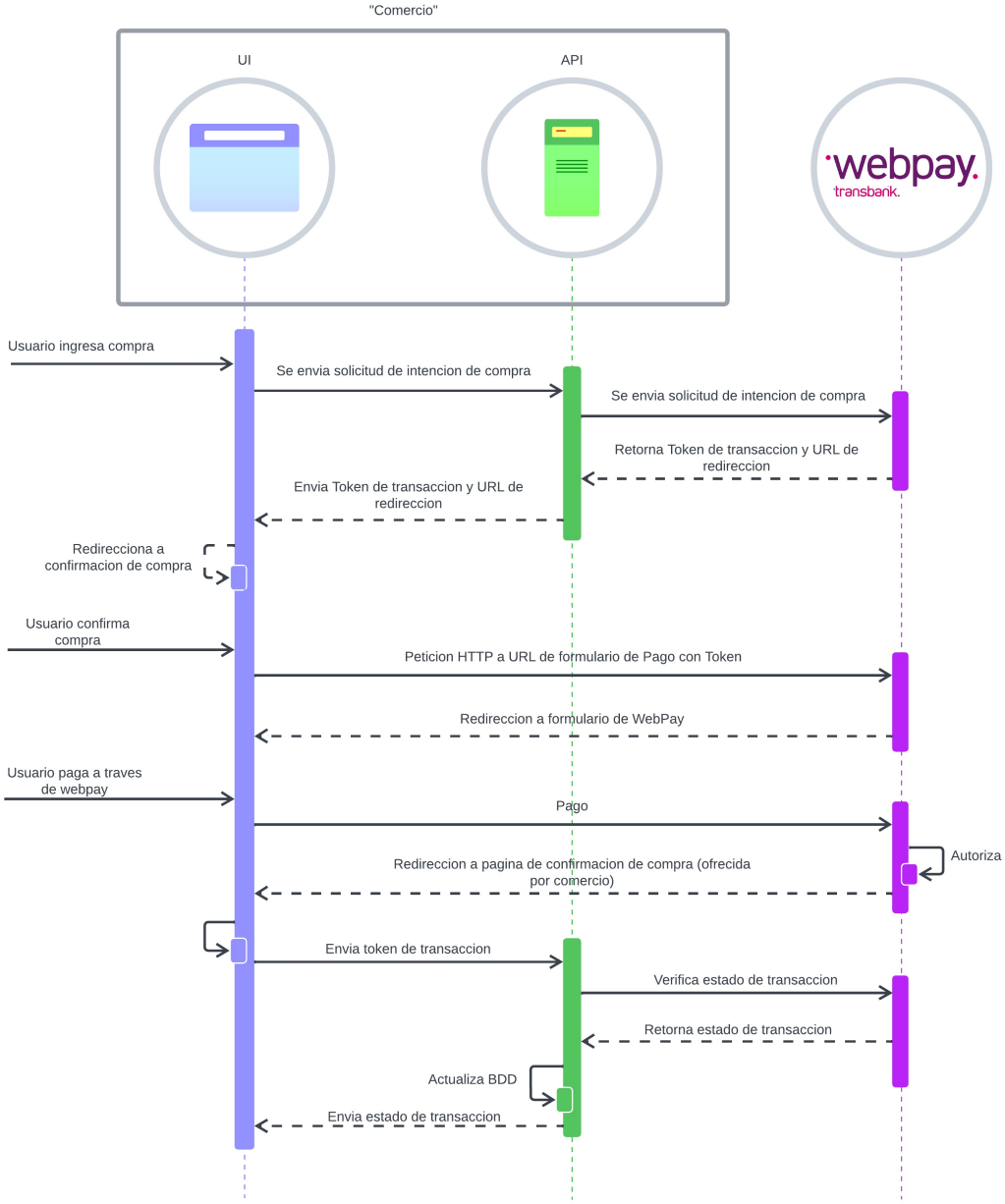
Pueden agregar los datos que deseen (u otros endpoints) mientras se calcule el indice que se solicite. Es clave recalcar que se adhieran al *single responsability principle*, y hagan este servicio lo más pequeño posible.

Debe estar disponible en otro container y debe llevar el tracking de los trabajos, cosa que se los puede proporcionar el framework que usen (usualmente es así por defecto).

En un diagrama simplificado, se vería así



Ejemplo de flujo a nivel de usuario de implementacion de WebPay



Este flujo muestra como se vería para darle una respuesta al usuario. La idea es que tomen esta señal para empezar el flujo por WebPay, esperen la respuesta final, y luego retornen el resultado.

Especificaciones

Si un requisito está marcado como *Esencial*, el no cumplirlo en un grado mínimo (al menos un punto) reducirá la nota máxima a *4.0*. Lo mismo ocurrirá si su entrega no cumple con los requisitos esenciales de las entregas pasadas. NO se revisaran entregas que no estén en la nube.

Deben tener todos los requisitos esenciales de la E1 para esta entrega o NO se les corregirá.

Por otro lado, debido a que esta entrega presenta una buena cantidad de *bonus*, la nota no puede sumar más de 8.
Decidan con sabiduría que bonus les gustaría aprovechar o usar para compensar.

Al final de la entrega, la idea es que se pongan de acuerdo con su ayudante para agendar una hora y hacer una demo en vivo para su corrección. Se revisarán algunas features esenciales de código (conexión al broker).

Pueden utilizar Github Copilot (recomendado) para multiplicar su productividad y ChatGPT (poco recomendado) para dudas de implementacion, pero la información más completa la encontrarán en las documentaciones oficiales y preguntando a sus ayudantes / profesores. Si utilizan código

copiado de una plataforma deben citarlo en el README.md (http://README.md), y pregunten a su ayudante antes de usarlo.

Requisitos funcionales (13 ptos)

- **RF01** ***(2 ptos)*****:** Cuando un usuario compre un partido se debe realizar la generación de las recomendaciones mediante workers.
- **RF02** ***(3 ptos)*****:** Cada usuario debe tener una vista para poder ver las recomendaciones que se le han generado. Debe mostrar la fecha de la ultima actualización de las recomendaciones y el link al partido para poder comprarlo.
- **RF03** ***(7 ptos) (Esencial)*****:** Cuando un usuario compre un bono de partido dentro de su aplicación y seleccione la opcion webpay(hacer una vista para esto), se debe validar su compra a través de para luego enviar la respuesta mediante el canal `fixtures/validation` .
- **RF04** ***(1 ptos)*****:** Debe haber un indicador visible en su front que muestre si el ***servicio*** maestro de workers está disponible (para eso es el endpoint `/heartbeat`).
- **RF05** ***(2 ptos)*****:** Los usuarios deben poder descargar una boleta de su compra si esta se validó correctamente desde su vista de compras realizadas.

Requisitos no funcionales (38 ptos)

- **RNF01** ***(12 ptos) (Esencial)*****:** Deben crear el servicio que hace el cálculo de recomendaciones de partidos indicada en el enunciado, el cual asigna tareas a *workers*, lleva el registro de trabajos y los resultados. Este servicio existe en un **container independiente**, se conecta via HTTP ofreciendo una API REST y posee workers conectados mediante un broker con capacidad de encolado/pubsub (*Redis*/*rabbitMQ*), así como conexión a la base de datos del backend principal.
    - Separar los workers en contenedores propios tiene un bonus de **5 ptos**
- **RNF02** ***(5 ptos) (Esencial)*****:** La aplicacion debe ser capaz de manejar los tres casos posibles al momento de realizar un pago con Webpay (pago aprobado, pago rechazado y compra anulada por el usuario). De forma que se le informe al usuario de manera clara el estado final de la transaccion.
- **RNF03** ***(3 ptos)*****:** Una vez que se reciba una validación de un pago hecho en su aplicación, deberán enviar una notificación vía correo a los usuarios que lo solicitaron.
- **RNF04** ***(5 ptos)*****:** La aplicación tiene que ofrecer un servicio de generacion de boletas PDF desde AWS Lambda. Esta boleta debe tener el nombre de su grupo, los datos del usuario y la informacion de los equipos del partido que compró. Además, debe almacenarse en S3 y se le debe entregar al usuario un enlace público para descargarlo desde S3. Deben utilizar Serverless.js o AWS SAM para manejar y desplegar esta función.
    - Crear un pipe CI/CD para este servicio tiene un bonus de **4 ptos**
- **RNF05** (***8 ptos***): Deben implementar CD en su pipeline CI/CD para **backend**, específicamente usando servicios de registro de contenedores para aprovechar las ventajas de Docker. Como proveedores aceptados de CI están Github Actions, Codebuild y CircleCI. Para deployment deben usar AWS codedeploy y AWS ECR en un registro público para evitar cobros.
- **RNF06** (***5 ptos***): Deben implementar CD en su pipeline CI/CD para **frontend**. Como proveedores aceptados de CI están Github Actions, CodeBuild y CircleCI. Para deployment deben subir su frontend a AWS S3 e invalidar la caché de Cloudfront que sirve su frontend. Les recomendamos aprovechar los pasos que hacen con el cli de AWS para que replique el deploy manual.

Documentación (9 ptos)

Todos estos documentos los deben dejar dentro de su repositorio de Github en una carpeta `/docs` para la fecha de entrega.

- **RDOC1 *(2 ptos)*:** Deben actualizar su diagrama UML de componentes con lo realizado en esta entrega, con explicaciones y detalle sobre el sistema.
- **RDOC2 *(2 ptos)*:** Deben crear un archivo de documentación indicando los pasos que siguieron para implementar la integración con WebPay.
- **RDOC3 *(2 ptos)*:** Deben incluir una documentación de cómo subir su aplicación en Serverless/SAM, paso a paso
- **RDOC4 *(3 ptos)*:** Deben documentar todas las posibles llamadas a sus APIs expuestas a sus clientes con algún estandar (Postman, Swagger u otra).

Descuentos

Está totalmente prohibido subir su archivo .env, .pem u alguna credencial al repositorio, en caso de subir su archivo .env se les descontará 5 decimas, y en el caso del .pem no se corregirá su entrega.

Recomendaciones

- Comiencen la entrega lo antes posible, puesto que es mas sencillo ir trabajando de a partes y seguro tendrán dudas.
- Planifiquen con antelación: pregunten dudas o ambigüedades a sus ayudantes.
- Ojo con los deploys a última hora, la maldición de la demo es muy real.
- Ocupen el Free Tier de AWS, que tiene capacidad para todos estos requerimientos. Deberían usar los siguientes servicios:
  - **EC2**: AWS les proporciona una instancia t2.micro gratuita al mes.
  - **S3**: Tienen 5 GB de almacenamiento y 20000 solicitudes GET.
  - **RDS** (Opcional, Recomendado): Tienen 20GB y una instancia básica al mes.
  - **API Gateway**: 1 MM de llamadas al mes
  - **Lambda**: Tienen 400K GB/s y 1 MM de solicitudes.
  - **EBS**: 30 GB al mes para almacenamiento de discos de sistema.
  - **Cloudfront**: 50 GB al mes de transferencia.
  - **Amazon SES**: 62000 mensajes salientes / mes.
- **NO** está planificado hacer devolución por uso de dolares en AWS. Para la entrega el free tier de AWS es suficiente para conseguir todos los puntos. En caso de utilizar dólares en su solución, el curso no puede hacerles devolución de estos bajo ninguna causa.
- Usen una cuenta nueva o de alguien que no tenga otras cargas en AWS, para evitar cargos por ahora, además de usar una tarjeta prepago y los budget alerts de AWS para evitar costos oculto.

Consideraciones

No se considerarán entregas:

- Con componentes que corran en sus computadores o servidores que no sean los básicos de AWS. Algunos ejemplos, los servicios de AWS serían:
  - EC2
  - VPC
  - IAM
  - S3
  - AWS API Gateway
  - CloudFront
  - Lambda
  - SES
- Montadas en Heroku/Firebase/Elastic beanstalk/Lightsail/Netlify o similares.

- Que no estén documentadas.

Pueden utilizar cualquiera de los siguientes lenguajes. Los frameworks son los recomendados. Sugerimos que utilizen en cada grupo el lenguaje que todos sepan usar más o les parezca más sencillo de aprender. Los lenguajes en cursiva poseen ayudantes capaces de ayudarlos en su implementación de código.

- *Python*
  - FastAPI
  - Django
- *Javascript (js)*
  - Koa
  - Express
  - NestJS (solo si tienen muuuucha experiencia, aprenderan mucho)
- *Ruby*
  - Rails
- C#
- Go
- Rust

Puntaje

Atraso

Para esta entrega se les descontará 0.5 puntos en la nota máxima por horas Fibonacci con F1 = 6 y F2 = 6.

Se considerará como atraso cualquier modificación en features o implementación que tenga que ver solo con lo que se pide en esta entrega.

Fibonacci	Hora	Nota maxima
6	0:01 - 5:59	6.5
6	6:00 - 11:59	6
12	12:00 - 23:59	5
18	24:00 - 41:59	4.5
30	42:00 - 71:59	4
…	72:00 en adelante	1

Grupal

La nota se calcula como:

$$\mathbf{E_{2\ grupal} = 1 + E_{2\ RF} + E_{2\ RNF} + E_{2\ RDOC}}$$

Siendo $\mathbf{E_{2\ RF}}$ el puntaje ponderado de los requisitos funcionales, y $\mathbf{E_{2\ RNF}}$ el correspondiente a los requisitos no funcionales y $\mathbf{E_{2\ RDOC}}$ de la documentación.

Individual

Segun el programa del curso , esto se evalua como:

$$\mathbf{E_2 = 1 + ((E_{2\ grupal} - 1) * F_g)}$$

Donde F_g es un factor de coevaluación asignado por el grupo que va de 0 a 1.2. Para esto se enviará un form de coevaluación donde cada integrante deberá evaluar a sus compañeros de grupo con una puntuación entre 1 y 5.

No podrán asignarle 5 puntos a más de un compañero, y sí lo hacen, se considerará que se entregó un máximo de 4 puntos a cada compañero.

De no realizar la coevaluación, asumiremos que se le entregó el mismo puntaje de coevaluación a cada integrante, es decir 4 puntos.

Links útiles

- Documentación de Workers - Celery (https://docs.celeryq.dev)
- Documentación de Workers - Bull (https://optimalbits.github.io/bull/)
- API de IP a coordenadas:
  - ip-api.com (https://ip-api.com/docs/api:json)
- API de transformación de dirección a coordenadas:
  - geocode.maps.co (https://geocode.maps.co)
  - Google Geocoding API (https://developers.google.com/maps/documentation/geocoding/overview?hl=es-419)
- Guia iniciacion productos Transbank (https://www.transbankdevelopers.cl/documentacion/como_empezar#como-empezar)
- Documentacion Webpay (https://www.transbankdevelopers.cl/documentacion/webpay-plus)
- Tutorial Serverless Framework (https://www.serverless.com/framework/docs/tutorial)

Apoyo

Cada grupo tendrá un ayudante asignado el cuál podrán elegir mediante un link que se les mandará oportunamente. Este ayudante está encargado de hacerles seguimiento y orientar sus dudas para responderlas ellos mismos y el equipo de ayudantes. Les recomendamos **fuertemente** que pregunten sus dudas a su ayudante de seguimiento puesto que conocen del proyecto o pueden dirigir sus dudas a otros ayudantes. Puede ser de enunciado, código o algún tópico que tenga que ver con el proyecto

Dado que cada ayudante puede tener pequeñas diferencias en sus correcciones, queda a criterio de este hacer relajos o hacer mas estrictas ciertas particularidades. Intenten tener un flujo de comunicación directo con sus ayudantes para aclarar posibles diferencias o decisiones de diseño.

Pueden usar el Slack del curso para dudas más rápidas. Usen el canal #e2 (https://arqui-software.slack.com/archives/C039314QMNU) para sus dudas.

Las ayudantías programadas relevantes para esto por ahora son:

- Conexión a API Transbank - 11/08
- Continouos Deployment [Cápsula] - 15/08
- Workers - 18/08
- Serverless + SAM - 25/08

¡Éxito en la entrega!

Column 1	Column 2	Column 3
Text	Text	Text