2024-2 / IIC2173 - E1 | PPE CoolGoat Lists Async

aka. Procesamiento de Pagos como Eventos

Fecha de entrega: *01/10/2024 - 3 Semanas*

Después de un alto éxito en la obtención y recopilación de partidos, su consultora favorita Legit Business los contrata full-time (con beneficio de taca-taca, pizza y mesa de ping-pong en vez de seguro de salud :)), y les pide que pasen a una versión mas poderosa de su plataforma. Dado que los partidos, las probabilidades y sus filtros por si solos dan valor pero incompleto para sus Stakeholders, se les ha pedido que mejoren su sistema para que permita a usuarios y clientes poder apoyar a sus equipos de forma monetaria a traves de bonos, tal que si gana el equipo recibiran un beneficio los clientes, usando sus servicios de pagos de forma que se puedan coordinar entre todos los grupos fácilmente.

Objetivo

Asumiendo una cantidad finita de bonos para cada partido, deben extender su API web para que sea capaz de conectarse a un canal del broker para publicar sus solicitudes de compra, y luego validarlos con mensajes que les enviaremos desde otro canal para que entre todos sean capaces de mantener consistencia y no comprar bonos fuera de plazo.

Utilizando la conexión que ya poseen con el broker de eventos, su app debe seguir recolectando los datos según el formato de la E0:

```
{
    "fixtures":[
        "fixtures":
        {
        "id":<int>,
        "referee": <string|null>,
        "timezone": <string>,
        "date": <date>,
        "timestamp": <int>,
        "status": {
            "long":<string>,
            "short": <string>,
            "elapsed": <int|null>,
        },
    },
    "league": {
        "id": <int>,
        "name": <string>,
        "country": <string>,
        "logo": <string>,
        "flag": <string>,
        "season":<int>,
        "round": <string>,
    },
```

```
    "teams": {
        "home": {
            "id":<int>,
            "name":<string>,
            "logo": <string>,
            "winner": <bool|null>,
        },
        "away": {
            "id": <int>,
            "name": <string>,
            "logo": <string>,
            "winner": <bool|null>
        },
    },
    "goals": {
        "home":<int|null>,
        "away":<int|null>,
    },
    "odds":[ {
        "id":<int>,
        "name":<string>,
        "values": any[],
    },...]
]
}
```

Usando esta data, le presentará a los usuarios una lista de partidos. Los usuarios podrán buscar y escoger un partido y poder ver su detalle, junto con solicitar la compra de un bono del resultado (quién fue el ganador o si hubo empate), donde **en caso de ser correcta la prediccion de su bono, este recibira el valor del bono por la probablidad del resultado en su cuenta.** Por ultimo, se habilitara un nuevo canal en el broker donde se entregara la informacion de los resultados de los partidos de semanas anteriores.

Los bonos tendran un valor de **1000**. Ademas, en caso de ser incorrecta la prediccion, el usuario no gana nada.

Los usuarios deben tener un wallet donde tener su dinero para las compras. En esta entrega, la recarga deben manejarla de una manera simple, donde ustedes controlan el proceso de agregar fondos. Así, provisionalmente un usuario puede agregar cuantos fondos desee.

Será importante manejar una correcta gestión de usuarios, y para evitar problemas de colapso en su aplicación deberán usar un servicio de terceros y conectarlo con su aplicación mediante una API Gateway.

Adicionalmente, para empezar a lograr un proceso de implementación más expedita de su proyecto, se les pedirá que creen su solución en frontend y backend separados, además de implementar un proceso de *Continuous Integration*.

Solicitudes de compra de bono

El flujo para la compra de bonos consistirá en poder manejar una **integración masiva entre diversas apps que estarán queriendo comprar bonos por un equipo en cada partido**, teniendo un limite de bonos por equipo. Para esto se les pedirá que se conecten a un nuevo canal de broker con un nuevo script para que

puedan publicar eventos de solicitud de compra de bono, y que luego reciban por otro canal las validaciones de las solicitudes de compra de bono correspondientes.

Para explicar el flujo la idea es que un usuario ya ingresado en su aplicación y con dinero en su waller, busque y vea un partido que le interese y aprete en su detalle. Acá le saldrá la opción de compra de bono con valor 1000 , junto a las probabilidades de los resultados, donde podrá colocar a qué resultado quiere apoyar. Luego de indicar cuántos bonos está dispuesto a pagar, ustedes en su aplicación gestionarán: **cuánto ganara en cada caso**, **si les paga correctamente** (para esta entrega siempre lo hará así) y la **asignación del premio a ese usuario**. Ahora viene lo entretenido, validarlo entre todos.

Una vez que su usuario les indique que quiere apostar, **deberán publicar en el broker en el canal** `fixtures/requests` que su aplicación quiere apostar cierta cantidad de plata con el siguiente formato:

```
{
    "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,
    "seller": 0
}
```

En el body que mandará, `request_id` será un uuid de su solicitud que manejarán ustedes que para cada una deberá ser distinta (en la sección de enlaces útiles hay links que les pueden servir), `fixture_id` que sera el id del fixture al cual se quiere comprar un bono, y `group_id` es su número de grupo. El valor de `league_name` seria el nombre de la liga, `round` la ronda, `result` seria el nombre del equipo ganador y en caso de empate se debera dejar como `---` , y para terminar de hacer el match `date` indica la fecha del partido. También, el atributo `seller` siempre deberá ser 0 para esta entrega. Finalmente, el atributo `datetime` deberá ser la fecha y hora exacta en la que se realizó la solicitud de compra.

Como todos los grupos mandarán mensajes por este canal, deberán estar escuchándolo para recibir las solicitudes de los otros grupos.

Para poder validar las compras, tendrán que escuchar el canal `fixtures/validation` **.** Acá se les enviarán respuestas en el formato:

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

Si reciben una respuesta de que esa transacción es válida, entonces deben marcarla como completada. En otro caso (solicitud inválida), deberán marcarla como que no se logró. Para esta entrega una vez que se reciba esta validación se terminará el flujo.

Por ultimo, habilitamos un nuevo canal donde pueden recibir los resultados de los partidos de las semanas anteriores, para que manejen la entrega de la plata. El canal es `fixtures/history`. El cual entregara un json con el siguiente formato.

```
{
    "fixtures":[
        "fixture":
        {
        "id":<int>,
        "referee": <string|null>,
        "timezone": <string>,
        "date": <date>,
        "timestamp": <int>,
        "status": {
            "long":<string>,
            "short": <string>,
            "elapsed": <int|null>,
        },
    },
    "goals": {
        "home":<int|null>,
        "away":<int|null>,
    }
]
}
```

*Nota: En esta entrega el canal de validaciones **responderá `true` si es que la requests que publican en el canal respectivo se envió con el formato correcto**, en otro caso responderá con `false`.*

Ejemplo de flujo a nivel de usuario

1. Un usuario ingresa en la plataforma con credenciales creadas anteriormente a su aplicación.
2. El Usuario rellena una wallet en el sistema, con el dinero para participar.
3. Luego va a una lista de partidos filtrado con bonos disponibles y los revisa.
4. Para comenzar, escogera una y revisara el detalle de las probabilidades y la cantidad de bonos en este partido para ver si le interesa. Tendrá la opción de poder comprar bonos a este partido si quedan cupos:
5. (Solicitud) Se descontará la cantidad de bonos totales del partido para reservarlos mientras se valida. Se verá si puede pagarlo, Si el usuario no tiene fondo para esto aparece la opción de recargar wallet. En el caso contrario, se le creará una solicitud de compra avisando que su compra está en proceso pasando al paso 7.
6. El usuario debera poder ver una interfaz para recargar dinero, para asi hacerlo.

7. (Validación) Luego de un rato el usuario revisará su listado de compras y su compra se marcará de *en proceso* a *completada correctamente* o no de acuerdo a lo entregado en el canal de validaciones. Si resulta inválido se vuelven a agregar esas al partido.
8. Una vez decidido el ganador, en caso de que fuera validada la compra este recibira la cantidad de **bonos *1000 * odds** y se le mostrara un mensaje en la plataforma.

En caso de que ustedes reciban una solicitud de otro grupo por el broker, deben descontar esos bonos de igual manera ya que están reservados por ese grupo mientras se valida el pago. Una vez recibida la validación si es correcta lo mantienen igual y si no, devuelven los bonos del partido.

Requisito de Integración Continua

Para el requisito de continuous integration, les recomendamos usar los siguientes proveedores junto a su repo de GitHub (esto para evitar cobros):

- CircleCI
- Github Actions

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*. NO se revisaran entregas que no estén en la nube bajo ningún concepto.

Deben tener todos los requisitos no variables de la E0 para esta entrega o NO se les corregirá.

Para esta entrega y el resto del proyecto les pediremos que creen un repositorio personal y que luego integren a su ayudante respectivo con accesos de administrador.

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, y pregunten a su ayudante antes de usarlo.

Requisitos funcionales (15 ptos)

- **RF01 *(2 ptos) (Esencial)*:** Sus usuarios deben poder registrarse en la plataforma con datos de contacto y un correo electrónico.
- **RF02 *(2 ptos) (Esencial)*:** Los usuarios deben poder ver una lista de los partidos disponibles dado un filtro de búsqueda sobre los destinos y la fecha. Esta lista debe estar paginada.

- **RF03** ***(2 ptos)***: Debe poder verse el detalle de cada partido, incluyendo la cantidad de bonos disponibles (por default son 40) y dar la opción de compra.
- **RF04** ***(2 ptos)***: Deben obtener la ubicación desde donde el usuario hizo la compra desde su dirección IP. Para esto pueden usar ChatGPT para apoyarse y transformar la ip en una dirección.
- **RF05** ***(2 ptos)***: Deben poder mostrarle a su usuario los bonos comprados y si su solitud se completó correctamente.
- **RF06** ***(2 ptos) (Esencial)***: Al comprar un bono se deberá enviar la solicitud por el canal `fixtures/requests` y esperar la respuesta de si es válida por el canal `fixtures/validation`.
- **RF07** ***(2 ptos) (Esencial)***: Deberán estar escuchando los canales de `fixtures/requests`, `fixtures/validation` y `fixtures/history`.
- **RF08** ***(1 pto) (Esencial)***: Los partidos por defecto siempre tendrán incialmente 40 bonos disponibles. Deben gestionar correctamente la cantidad de bonos tal y como se les describe en el ejemplo más arriba.
- **RF09** ***(2 ptos) (Esencial)***: Cada usuario debe tener la capacidad de agregar dinero a una "billetera" dentro de su aplicación.
- **RF10** ***(2 ptos) (Esencial)***: Cuando un usuario compre un bono dentro de su aplicación, se debe validar que tenga el dinero suficiente en su billetera, y si es así, descontarle el dinero internamente para enviarlo a la API central.
- **RF11** ***(2 ptos) (Esencial)***: Cuando el usuario sea certero con la prediccion de un bono que compro. Debe recibir de premio 1000*odd siendo odd la probabilidad que entregaba ese resultado.

Requisitos no funcionales (39 ptos)

- **RNF01** ***(6 ptos) (Esencial)***: Deben usar un formato de Backend-Frontend separado: una API con respuestas JSON y un frontend. Esto es muy importante puesto que es crítico para las siguientes entregas. El Frontend debe ser ojalá una SPA **con un Framework que permita exportar el build de su frontend**, como React o Vue.
    - *Les recomendamos usar dos repositorios distintos para manejar su backend y su frontend.*

- **RNF02** ***(2 ptos) (Esencial)***: Sus aplicaciones en backend deben estar cada una en un contenedor Docker distinto. Debe coordinarse el levantamiento mediante docker compose.

- **RNF03** ***(2 ptos) (Esencial)***: Deben tener configuradas *Budget alerts* en la cuenta que ocupen como grupo, para no alejarse del Free tier de AWS y **que se den cuenta si les cobran para que cierren esos servicios**.

- **RNF04** (***6 ptos***) ***(Esencial)*** : Su API debe estar detrás de una AWS API gateway tipo REST o HTTP, con los endpoints declarados en esta. Debe asociarse un subdominio a esta (e.g. api.miapp.com) y debe tener CORS correctamente configurado.

- **RNF05** (***3 ptos***) ***(Esencial)*** : Su app debe ofrecer su backend y frontend utilizando HTTPS.

- **RNF06** (***6 ptos***) ***(Esencial)***: Deben implementar un **servicio de autenticacion/autorización** (auth). Este servicio puede ser en base a un servicio de terceros como Auth0 (recomendado), cognito o pueden hacerlo ustedes. Este RNF requiere que ustedes extraigan toda la lógica de los usuarios de la app principal y la trasladen a el servicio hecho por ustedes o el externo. Recomendamos fuertemente usar el modelo Oauth o como mínimo intercambiar tokens JWT con la audiencia e issuer correctos.
    - Si hacen un servicio ustedes desde 0, tienen un **bonus de 5 ptos**, esto implica gestión de tokens JWT con OAuth2, gestión de la información de usuarios, una gestión segura de los tokens a nivel de frontend, y **debe correr como un contenedor/servicio aparte**.

- **RNF07** (***3 ptos***): Su frontend debe estar desplegado en S3 con una distribución Cloudfront.

- **RNF08** (***3 ptos***): Su API Gateway debe poder usar al servicio del RNF06 para autenticar a los usuarios directamente, es decir que antes de enviar la request a su API, API Gateway verifica que el token sea correcto. Dentro de API Gateway deben crearle un Custom Authorizer si usan tipo REST para poder autenticar sus requests previos a mandarlos a su API.

- **RNF09** ***(8 ptos)*****:** Deben implementar un pipeline de CI. Como proveedores aceptados están CircleCI, Github Actions y AWS codebuild. Recomendamos los dos primeros porque los ayudantes tienen experiencia en estos dos. Esta implementación debe correr un linter que revise su código.
    - Implementar un build simple que resuelva un test trivial que pueda fallar solo para el backend (tipo `assert false` o similar) tiene un **bonus de 3 ptos**
    - Implementar un pipeline CI para su frontend que revise con un linter su aplicación y haga uso de revisiones de performance de lighthouse tiene **bonus de 3 ptos**.

Documentación (6 ptos)

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

- **RDOC01** ***(3 ptos)*****:** Deben crear un diagrama UML de componentes de la entrega, con **explicaciones y detalle** sobre el sistema. Esto deben tenerlo para la fecha final de entrega.
- **RDOC02** ***(2 ptos)*****:** Deben documentar los pasos necesarios para replicar el pipe CI que usaron en su aplicación (Qué pasos sigue si CI).
- **RDOC03** ***(1 ptos)*****:** Deben dejar una documentación de alguna forma de correr su aplicación en un ambiente local para propósitos de testeo (que instalar, que poner en el .env, como correr la app, etc).

Descuentos

Está totalmente prohibido subir su archivo .env o .pem 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. Se les dio plazo extra para que se adecuen a sus equipos de trabajo.
- 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 (Opcional)**: 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. Si tienen problemas con costos que no saben como detener hablenlos con su ayudante designado.
- **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.
- Pueden usar librerías para generar el gráfico de precios semanales.

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
- 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 y ocuparán typescript)
- *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	6
18	24:00 - 41:59	4.5
30	42:00 - 71:59	4

...	72:00 en adelante	1

Grupal

La nota se calcula como:

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

Siendo $E_{1\ RF}$ el puntaje ponderado de los requisitos funcionales, y $E_{1\ RNF}$ el correspondiente a los requisitos no funcionales y $E_{1\ RDOC}$ de la documentación.

Individual

Segun el programa del curso[5] , esto se evalua como:

$$E_1 = 1 + ((E_{1\ grupal} - 1) * F_g)$$

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

- API Gateway - Rest API
- Auth0 - Get Started
- Documentación de Auth0 con API Gateway
- Circle CI Blogs - CI para Django
- Librerías UUID
  - Python
  - NodeJS
  - Ruby
- Linters
  - Flake8 - Python
  - EsLint - JavaScript
  - RuboCop - Ruby

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, es válido negociar siempre y cuando no sea en base a requisitos esenciales.

Pueden usar el Slack del curso para dudas más rápidas. Usen el canal #e1 para sus dudas.

Las ayudantías programadas relevantes para esto por ahora son:

- Continuous Integration - 30/08
- S3 Upload [Cápsula] - 11/09
- API Gateway + CORS - 13/09