Para resolver la problematica planteada nos enfocamos en seguir un orden y en hacer que el sistema desarrollado sea flexible, sostenible y escalable a lo largo del tiempo, por lo cual decidimos hacer uso de una Clean Architecture lo cual nos permitio separar en capas nuestro proyecto para tener una clara visión de como debian comunicarse entre si las partes del sistema, de esta manera logramos tener una clara estructura del mismo.

Ademas de la arquitectura planteada que es esencial, hay mas factores involucrados, como los principios SOLID, estos principios hacen posible que un sistema sea fácil de escalar y de mantener.

A lo largo de la construcción de la aplicacion hicimos mucho énfasis en el principio DRY por sus siglas en ingles Dont Repeat Yourself para la reutilización de funcionalidad, ya que esto además de hacer que seamos menos redundantes, agiliza en gran medida el desarrollo ahorrando tiempo de manera significativa.

Ampliando un poco mas acerca de la reutilización de funcionalidad, decidimos que el uso de patrones de diseño era también algo fundamental para el desarrollo de la aplicación, ya que esto nos permite no tener que volver a inventar la rueda cuando esta ya existe, con esto queremos decir que estos patrones son soluciones reutilizables a problemas comunes durante el desarrollo de sistemas los cuales nos guian en la creación de estructuras eficientes y escalables y pueden ser adaptados a distintos contextos dependiendo de las necesidades de un proyecto, un ejemplo que podríamos dar seria para el desarrollo de un sistema de notificaciones para el usuario, el enfoque que tomamos para resolver esta necesidad fue implementar un patron de diseño muy conocido llamado Observer.

Volviendo hacia lo que son los principios aplicados para las buenas practicas, nos concentramos en escribir funcionalidad que todo el equipo fuese capaz de leer ya que es mucho mejor que todos comprendan el código escrito a que solo una persona pueda hacerlo, lo cual impediria el trabajo en equipo. Este principio en el ambiente tech se conoce como KISS por sus siglas Keep It Simple Stupid.

Hablando acerca de seguridad no nos tomamos a la ligera el procedimiento en la reventa y compra de entradas del sistema y realizamos una serie de procedicimientos en cuanto a las verificaciones que debe hacer el usuario para poder revender y comprar una entrada, estas serian que debe verificar su numero de teléfono, verificar su email y agregar a su información su DNI. Tambien implementamos DTOs por sus siglas en ingles Data Transfer Object siguiendo las practicas actuales en el desarrollo ya que nos permite realizar ajustes en cuanto al ingreso y egreso de datos sensibles.

Mediante todo lo que se comento anteriormente aplicamos exitosamente la practica conocida como encapsulamiento de responsabilidades, respetando el principio de responsabilidad unica logrando que cada método se ocupe de realizar solo una parte de la funcionalidad proporcionada por el software, esto lleva a tener una mayor claridad en el diseño volviendo a la clean architecture ya que nos permite que todo se encuentre según estándares de buenas practicas de programación y tambien nos brinda mayor facilidad en su mantenimiento.

Con este claro enfoque logramos hacer que la aplicación sea super flexible, escalable, testeable y mantenible a lo largo del tiempo.

V2:

Buenas tardes a todos, mi nombre es Ezequiel Feola y ocupo el rol de líder del equipo de backend, del cual forman parte Daniel Nina, Gonzalo Rosofsky y Jorge Gomez.  
  
Para abordar la problemática planteada en GoPass, nos enfocamos en desarrollar un sistema flexible, sostenible y escalable a largo plazo siguiendo el estándar REST para la API. Para esto implementamos una **Clean Architecture**, lo que nos permitió separar el proyecto en capas claramente definidas, facilitando la comunicación entre las diferentes partes del sistema y garantizando una estructura sólida y comprensible.

Además de la arquitectura, consideramos seguir los principios **SOLID** y la filosofía Clean Code, siendo ambos esenciales en el desarrollo hoy en dia. Esto se ve reflejado en la implementación del patron repositories que centraliza todo el acceso a datos y en los servicios que se ocupan de contener toda la lógica de negocio dándole a los controladores un código mucho mas limpio.

También nos basamos en el principio **DRY** por sus siglas en ingles (Don't Repeat Yourself), que nos ayudaron a evitar ser redundantes reutilizando funcionalidad en lugar de repetirla constantemente, esto no solo agilizó el desarrollo sino que también permitió que el código fuera más fácil de mantener y extender.

Para ampliar esta reutilización, recurrimos a **patrones de diseño**, ya que estos proporcionan soluciones probadas a problemas comunes en el desarrollo, como solemos decir, "no hace falta reinventar la rueda cuando esta ya existe". Un claro ejemplo de esto fue la implementación del patrón **Observer** además del de repositories mencionado anteriormente, con el observer nos ocupamos de hacer un sistema de notificaciones, lo que nos sirvió como guía y permitió crear una solución eficiente y adaptable sin tener que partir de cero.

En cuanto a la simplicidad y legibilidad, aplicamos el principio **KISS** por sus siglas (Keep It Simple Stupid), asegurándonos de que el código fuera entendible para todo el equipo, fomentando así una mejor colaboración.

En lo que concierne a la seguridad del sistema, implementamos una serie de rigurosas verificaciones para la compra y reventa de entradas, incluyendo la verificación de número de teléfono, email y la incorporación del DNI, sin pasar estas verificaciones por completo no se le permite a los usuarios comprar ni vender entradas, cabe destacar que se cifraron los datos sensibles como la contraseña del usuario, su numero de teléfono y DNI. También utilizamos **DTOs** por sus siglas (Data Transfer Objects) los cuales nos permiten manejar de forma segura la transferencia de datos disminuyendo la exposicion de datos sensibles a lo largo del sistema.

Finalmente, respetamos el principio de **responsabilidad única**, encapsulando responsabilidades en métodos y funciones que se centran en tareas específicas, lo que contribuyó a un diseño más claro y fácil de mantener. Esto, junto con la Clean Architecture, nos permitió crear una aplicación altamente flexible, escalable, testeable y mantenible.

Esto fue todo por parte del equipo de backend, sin mas que agregar le cedo la palabra al equipo de QA.