# Implementación y Arquitectura (y para que):

## Primer entregable

En nuestro primer entregable hemos utilizado avanzado de una manera más directa y sencilla que el resto de apartados. Debido a la simpleza del planteamiento original (transformas preguntas y respuestas) Se ha llevado a cabo el problema mediante un **Big Ball of Mud**.

Comenzando a programar sin una arquitectura realmente definida y simplemente buscando la funcionalidad del problema. En entregas posteriores, ya tendríamos que decidir antes la arquitectura a utilizar.

De todas formas no hubo problemas de flexibilidad de cara a avanzar con la aplicación, en parte debido a que se trataba de una edad muy temprana de la aplicación.

Esta parte se separo en:

* Mostrar y ejecutar la pequeña lógica.
* (ETL) Extraer la información de los ficheros del formato que más adelante comentamos, transformandolos, y guardandolos para su posterior utilización.

Las preguntas se recogen en tiempo de ejecución y guardan en estructuras simples. En posteriores entregas integramos una base de datos para su almacenamiento.

## Segundo entregable

Al ser la aplicación muy básica y sencilla en su versión anterior, tendríamos que replantearnos la manera para abordar su consecuente aplicación de escritorio, así como utilizar nuevas tecnologías.

En el segundo entregable hemos optado por afrontar el problema desde otra perspectiva. Mediante elmodelo **Vista-Controlador**. Con el cual podremos separar la vista del usuario de nuestro modelo.

Gracias a ellas podríamos:

* Por un lado separar las responsabilidades y funcionalidad de la aplicación del modelo. Pudiendo desarrollar independientemente la interfaz gráfica del resto de aplicación en un principio. Por ejemplo, para la integración de la base de datos, mientras un grupo del equipo se podía centrar en la base de datos el otro se encargaba de diseñar una de las vistas.
* Por otro lado, tenemos la posibilidad de la creación de diferentes vistas para el usuario. Gracias a que será independiente del resto de aplicación. Por ejemplo, mediante los *look and feel*  de la tecnología usada, podremos presentar nuestra aplicación de forma diferente sin necesidad de cambiar nada.

Aunque aún así tenemos lógica con un grado de acoplamiento no excesivamente grande, no toda la lógica ha sido separada de la parte gráfica, con lo que perdimos cierto grado en la distribución por capas que pretendíamos abarcar, estando algo acoplada la capa lógica con la presentación. Esto no pasa con la capa de datos.

Para integrar la base de datos hemos utilizado un patrón ***DAO***, para las posibles diferentes implementaciones que pudiéramos querer utilizar. De esta forma, suministraremos una punto de conexión entre la interfaz de la aplicación las bases de datos que tengamos, en este caso una, consiguiendo así separar las responsabilidades.

Para las partes más complejas de nuestra aplicación, véase por ejemplo el cálculo de tiradas con un dado. Hemos optado por realizar algoritmos eficientes, que no hagan aumentar demasiado nuestros tiempos de ejecución (cumplir atributo de calidad). En este ejemplo se utilizado un algoritmo *Backtracking*.

## Tercer entregable

En esta última parte seguiremos con un modelo **Vista-Controlador**

Reciclando la mayoría de las cosas de nuestra segunda aplicación, ahora solo tendríamos que utilizar las librerías y frameworks adecuados para poder transformar nuestra aplicación al tipo web, siguiendo la misma arquitectura.

Cabe destacar, que podríamos tener funcionando a la vez nuestras tres aplicaciones anteriores por separado. Utilizarían una **base de datos compartida**. Aunque no se ha llegado a trabajar en ese apartado, y por lo tanto nuestras aplicaciones no estarán sincronizadas adecuadamente, lo que podría producir problemas de integridad en la base de datos.

# Herramientas y tecnologías utilizadas:

## Primer entregable

Para la recuperar la información de las preguntas y respuestas de la aplicación se utilizará en las tres partes preguntas en formato **.GIFT** y en el primer entregable también en formato .JSON. Haremos la conversión de estos formatos para poder utilizarlos posteriormente en nuestra aplicación.

Para la implementación de la aplicación se escribirá en lenguaje **Java** y se utilizará la herramienta **eclipse** para la programación.

Utilizaremos proyectos **Maven** en eclipse, sincronizándolos mediante la plataforma **GitHub**, con la cual tenemos la posibilidad de hacer un desarrollo sincronizado entre el equipo.

Estas características se mantendrán a lo largo de las tres aplicaciones.

## Segundo entregable

Además de lo anterior hemos tenido que utilizar las siguientes herramientas para posibilitar el desarrollo de la aplicación de escritorio.

Ahora no se mostrará todas las opciones en una simple consola de comandos, si no que tendremos que darle una representación visual. Para ello hemos utilizado la él Framework para Eclipse llamado ***Java WindowBuilder***. Con el cual podremos editar y mostrar de manera visual por fin nuestro trivial.

Con esta herramienta además hemos podido seleccionar un *look and feel* para nuestra aplicación, que además si queremos cambiar a lo largo de la vida de la aplicación, se podrá cambiar sin ningún problema.

Por otro lado, para llevar a cabo un cierto control de los usuarios registrados del sistema y las partidas jugadas, utilizaremos una base de datos ***HSQL*** en local. Con ella podremos, registrar usuarios, identificarlos y por cada partida que juegue guardarla. Gracias a guardar las partidas de cada usuario podemos mostrar en una sencilla pantalla de estadísticas estos datos, para información del administrador.

La implementación en código para las consultas con la base de datos se hace mediante las librerías *java.sql*. Usadas para programar las diferentes implementaciones ***DAO*** que comentamos con anterioridad.

Además se han incorporado las historias de usuario en el proyecto a través de la herramienta de ***Cucumber***. Gracias a esta podremos localizar fácilmente en nuestro código donde se encuentran las funcionalidades concretas de dicha historia.

## Tercer entregable

Para la última parte, al tratarse de un proyecto web, no nos bastará con lo que tendíamos con anterioridad. Tendremos que hacer uso de algún Framework que nos permite su representación en web.

Para ello utilizaremos el Framework **Play** que nos permitirá representar cada una de nuestras pestañas de aplicación en web. Mediante **HTMLs**.

Además se ha introducido el uso de **Javascripts** para por ejemplo mostrar el chat incorporado a nuestra aplicación.

En cuanto al aspecto de la aplicación, hemos optado por utilizar el framework de **boobstrap**, para la generación de .**CSS** que nos ayuden a cambiar el estilo de nuestra aplicación de manera sencilla.

# Atributo de calidad y cuántos de ellos se han llevado a cabo

* Disponibilidad 24x7, siempre y cuando la base de datos este arrancada y no tenga ningún problema con la conexión este atributo de calidad no se verá perjudicado.
* Elección de formatos para las preguntas. Damos la posibilidad de elegir .GIFT y .JSON

Pero principalmente .GIFT siempre se podrá, por lo que se satisface completamente este atributo.

* Versatilidad en cuanto a más juegos se refiere. Se podrían reaprovechar ciertos puntos de la aplicación para poder hacer otros juegos del estilo sin problema.
* Formatos de texto para preguntas. Se tendría que reescribir las preguntas en función del formato.
* La extracción de las preguntas debe de ser fluida y no demorarse al menos no más de 15 segundos. Mediante el testeo (current.time.millis) podemos comprobar que en principio, los tiempos son muy buenos.
* Acceso a base de datos solo autorizado. Desde la aplicación nadie que no deba podrá ver datos que no debiera. Por ejemplo, mediante el login y contraseña del Administrador dejaremos únicamente a él, ver datos estadísticos de las partidas.

Al tratarse de una aplicación en local, no está controlada la intrusión directa en la propia base de datos.

* Facilidad testeo de salvado y recuperación de preguntas. Mediante unas clases test podemos comprobar que todo se introduce correctamente. Depende del formato algunas clases test.
* Localización de fallos mediante arquitectura modular, no se llevo a cabo finalmente este punto.
* Sostenibilidad del sistema. Depende en que área nuestra aplicación es más o menos sostenible. En el primer entregable lo era completamente, en el segundo habría dependencias entre algunas partes de la interfaz y la lógica. Y por último en el tercero hay poco acoplamiento y es sencillo de realizar cambios.
* Cumplir intervalo de fallos, sin sobrepasar una tasa. Mediante las anteriores pruebas y la implementación de Excepciones para controlar los fallos, podemos prever algunos problemas.