

Índice

[**Introducción 1**](#_rutzxycyvsjc)

[**Planificación 2**](#_ca5hnh4quelg)

[**UML 6**](#_unr2do1o2pys)

[Definición de Clases, Atributos y Métodos 6](#_2nmqrgcxlmx8)

[Relaciones y Multiplicidad 7](#_6in9hqxktr0h)

[**Páginas web 9**](#_pkdr8085b74d)

[**Conversión a .php 13**](#_78y1yc29cjp8)

[**JSON/JSON Schema 13**](#_3e1bzih4a5oo)

[**Diagrama entidad/relación 18**](#_4tqw7skknpau)

[**Nuestra Base de Datos 20**](#_2fum5vhx84dj)

[Archivos .php de registro e inserción en profundidad 22](#_ohkqm8z0ynqv)

[Archivo de registro 23](#_6qaw2vdj3dfa)

[Archivo de inicio de sesión 23](#_a8y9rliqeckm)

[**Servidor EC2 24**](#_oexkq5ct3k0b)

[**Servidor RDS 25**](#_olfeeyj20jhb)

[**Programas en Java 26**](#_k20qg0mw323g)

[Elementos generales 26](#_pdp12ama4dmj)

[Nuevos comandos 26](#_tw5eoe5o4zqi)

[Programa de registro e inicio de sesión 27](#_plscoyhpclpg)

[Programa de menú de inicio 30](#_nrd3bukul65s)

[Programas de máquinas tragaperras 31](#_gr70mjcp0vp1)

[Jar 32](#_gk4p2hjitbyo)

[**Contratación de los empleados y sus nóminas 33**](#_2apvrpu43uy6)

[**Presentación de la página en Inglés 33**](#_y044vqbr9ye0)

[**Desviaciones 34**](#_43kao5l6heec)

[**Planes a futuro 34**](#_f5trv8irimkl)

[**Diagrama de Gantt definitivo 36**](#_m66f5c4wfnnc)

# Introducción

Esta es la documentación oficial de *Oliva Bet.* En este documento explicaremos paso a paso todo el proceso de nuestro proyecto. *Oliva Bet* es un casino en línea diseñado para mero entretenimiento, hemos diseñado y programado 5 máquinas tragaperras que están disponibles para el uso y disfrute de los usuarios. Tenemos claro que este tipo de juegos puede crear cierta adicción y esto es un gran problema hoy en día para algunas personas. Nuestro objetivo es diseñar este tipo de juegos y disfrutar de los mismos de manera gratuita, con saldo ficticio.

De igual manera, nuestras máquinas pueden ser de gran utilidad para todos aquellos que nunca han tenido la oportunidad de probar este tipo de juegos, al igual que puede aportar un grán servicio a modo de prueba para cualquiera que quiera apostar su dinero en una máquina similar de otro casino.

Una vez explicada la idea principal y general del proyecto, empezaremos con la base de la planificación.

# Planificación

En los primeros días de proyecto, estuvimos trabajando en la planificación del mismo. En esta planificación, decidimos y concretamos cuáles eran nuestros objetivos, el porqué de realizar el mismo, como idealizar e imaginar tanto la página web con sus servicios como los juegos en sí. Nos pusimos de acuerdo para decidir los recursos que íbamos a necesitar para llevar el proyecto a cabo, tanto el frontend, backend y la base de datos, los costes eran también muy importantes para el tema de planificar los gastos que tendríamos no solo con los ordenadores sino también con el dominio y la contratación de los empleados. Para ver los costes de una forma más gráfica, haz [clic aquí](https://www.canva.com/design/DAGDaC-UzKA/lizM466t292d5pF43XXBaA/edit)

A su vez, uno de los apartados más importantes en el tema de la planificación es el tema de los plazos y duración del proyecto ya que contábamos con tres semanas para ello. Para ello, hemos identificado las tareas que cada uno de nosotros tenía que realizar y las hemos plasmado en un diagrama de Gantt, visible en la **Imagen 1**.



**Imagen 1:** En esta imágen se muestran las distintas tareas que cada miembro del grupo tenía que realizar o llevar las riendas de las mismas

Las distintas tareas son las siguientes:

* Planificación: realizar el documento de la planificación general de proyecto.
* Servidor EC2 : instalar un servidor *EC2* en Amazon con servicio de página web.
* Base de datos: crear una base de datos mediante la interfaz de phpMyAdmin y mediante el lenguaje MySQL
* HTML y CSS: mediante una plantilla de [esta página web](https://www.free-css.com/free-css-template) editar dicho código para modificarlo en base a nuestros intereses.
* JSON Schema: diseñar un *JSON* Schema que cumpla con los requisitos de la base de datos.
* JSON: diseñar un *JSON* que cumpla con datos insertados en la tabla y obviamente que respete las normas del *JSON Schema*.
* Mover la base de datos al servidor de Nazaret: emplear las credenciales que el profesorado nos deje para poder migrar nuestra base de datos creada localmente a dicho servidor.
* Documentación: realizar este mismo documento con la mayor claridad posible explicando todo lo realizado en cada apartado.
* Vídeo de Inglés: hacer un vídeo en Inglés mostrando las funcionalidades principales de la página en un formato de anuncio, en él todos los miembros del grupo deberán participar.
* Presentación: diseñar la presentación final del proyecto, la cual mostraremos y explicaremos delante de toda la clase.
* FOL Contratación: realizar cinco contratos de cada uno de nosotros, con distintos tipos de contratos, exactamente tres tipos que explicaremos en dicho apartado.
* Nóminas: realizar una serie de nóminas relacionadas con los contratos antes mencionados.

Hay que recalcar y es evidente que, explicaremos todas las tareas en profundidad en sus respectivos apartados.

Por último, la búsqueda o la identificación de los riesgos ha sido de vital importancia para poder así crear un plan de respuesta correcto e ideal para las posibles consecuencias que tendrían esos riesgos tal y como se puede ver en la Tabla 1.

| Riesgo | Repercusión | Solución |
| --- | --- | --- |
| No asistir a clase | Retraso en las tareas y no ayudar al grupo | Si no hay justificación, echarle del grupo |
| Fallos en el servidor | El progreso se frenará durante un tiempo indefinido. | Hacer copias locales de cada avance para poder seguir avanzando. |
| Usar IA  (Inteligencia Artificial) | Suspender el reto | Prohibido usarla a no ser que algún profesor lo vea necesario |
| Festivos | No se trabajará durante los días festivos. | Hacer la planificación desde el primer dia que nos juntamos para poder organizarnos mejor |
| No trabajar en el reto | Perjudicar al grupo y retrasar los trabajos | Darle un aviso y en caso de que llegue a los 3 avisos, expulsarle del grupo |
| Usar imágenes con copyright | Que se deshabiliten las imágenes | Usar paginas especificas para obtener imágenes sin copyright |
| No cumplir con los plazos asignados para cada tarea | Pérdida de tiempo y descuadrar la planificación | Modificar los plazos marcados para cada tarea |
| No llevarse bien entre los compañeros | Mala comunicación en el grupo y poca coordinación en los trabajos | Aunque haya roces, intentar llevarse bien o respetar al compañero por el bien del grupo |
| Mal funcionamiento de algún programa | Retraso a la hora de poder avanzar con las siguientes tareas | Tener más alternativas de programas parecidos o buscar alguna otra herramienta |

**Tabla 1:** Esta tabla muestra tanto los riesgos, sus repercusiones y un plan de respuesta para los mismos

Para poder visualizar nuestra planificación, [haz clic aquí](https://docs.google.com/document/u/0/d/1PQ9NpRAavSOINUf0pFOhnL0lU24qwXvBtWvqUmS483o/edit).

Asignaturas

Entornos de desarrollo

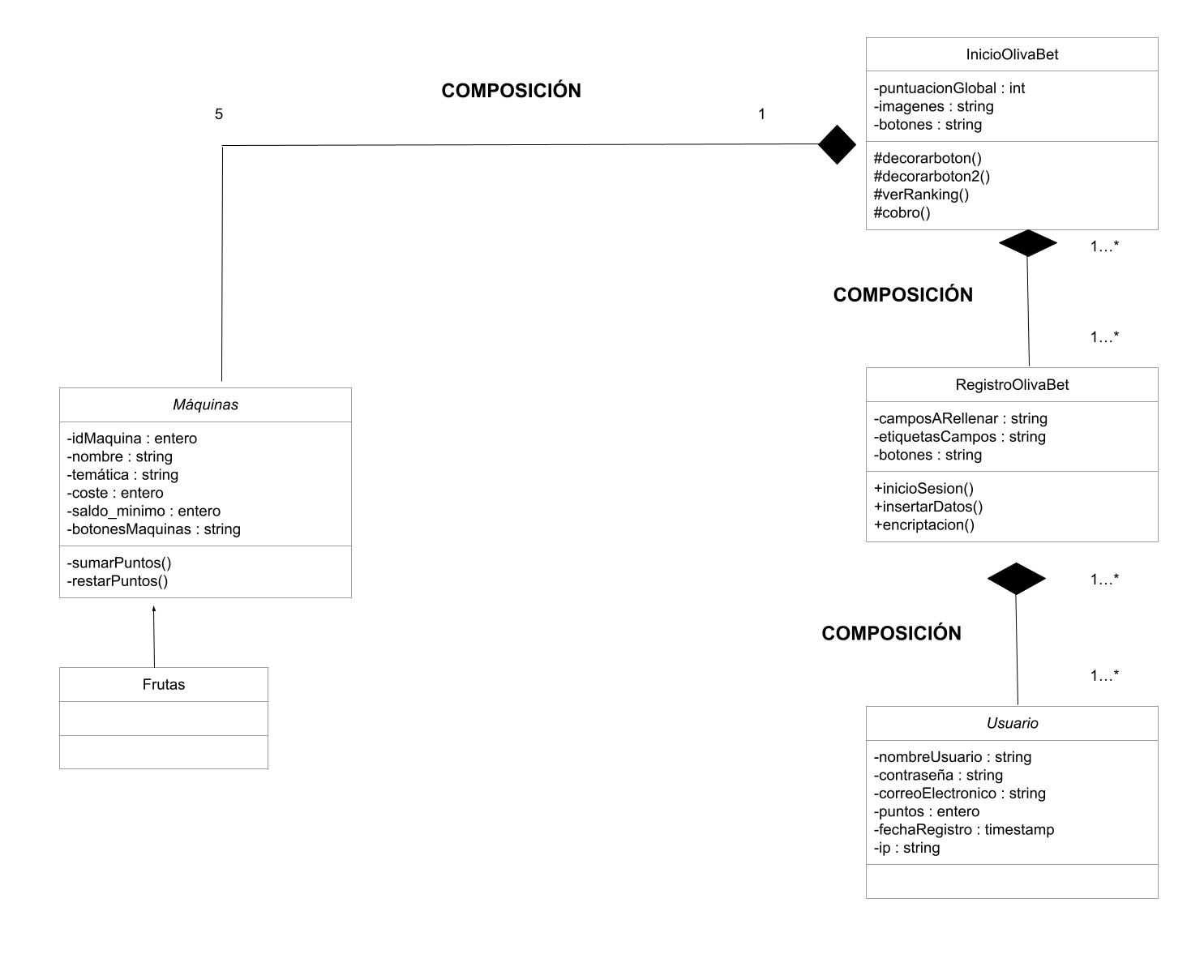
# UML

## Definición de Clases, Atributos y Métodos

En el diagrama visible en la **Imagen 3**, podemos observar 3 clases, 1 superclase y 5 subclases. Una de las primeras tres llamadas *Usuario* es abstracta, ya que en nuestro programa no está ni creada, definida y tampoco vamos a crear instancias de esa clase pero vemos realmente necesaria su explicación y representación para entender mejor el tema de registros e inicios de sesión. Además, otra de estas primeras tres clases es *RegistroOlivaBet* que representa el programa de registro e inicio de sesión que los usuarios tienen que realizar para poder jugar a nuestro juego. Por último, tenemos la clase *InicioOlivaBet*, un programa que representa el menú de inicio de nuestro juego, es aquí donde se podrá elegir cada máquina.

Por otro lado, tenemos la superclase abstracta llamada *Maquinas* querepresenta todas las *slots* que tenemos diseñadas. Es por esto que las subclases son los programas individuales de cada máquina tragaperras.

Cabe recalcar que hemos intentado seguir nuestro tipo de visibilidad empleada en el programa en cuanto a métodos y atributos. Todos los métodos y atributos de los mismos se pueden ver en la **Imagen 2**.



**Imagen 2**: Breve ejemplo de la representación de los atributos y métodos de nuestras clases

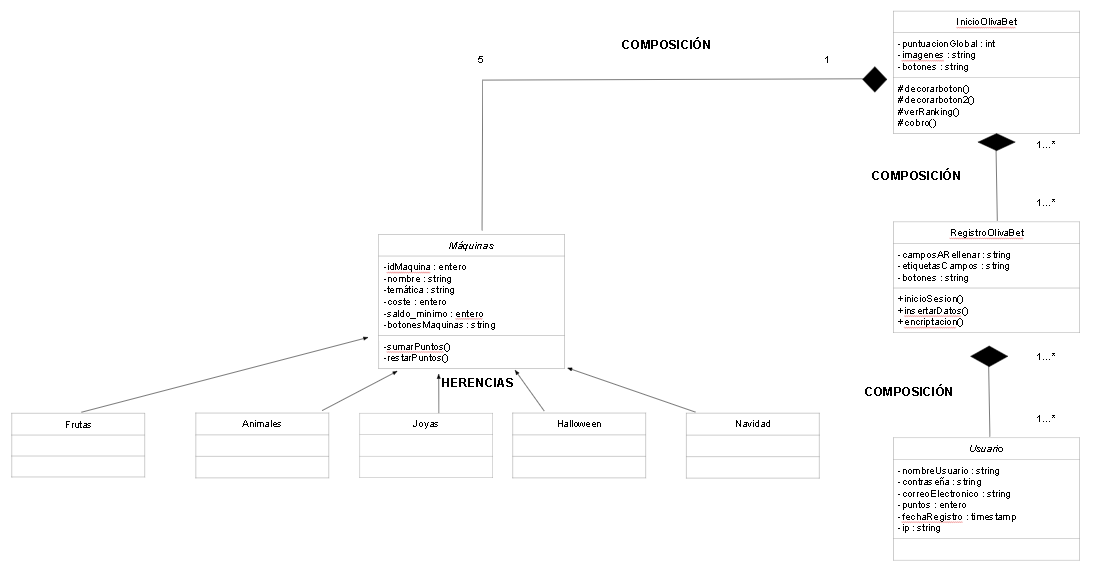
## 

## Relaciones y Multiplicidad

En cuanto a la **Imagen 2**, hemos usado una subclase de ejemplo, pero si seguimos el UML se pueden observar 4 más. Existe una relación de tipo composición entre la clase abstracta llamada *Usuario* y la clase *RegistroOlivaBet*, en este caso y basándonos en la multiplicidad, un usuario podrá registrarse o iniciar sesión (ya que ambos están en la misma clase) 1 vez o más veces. Además, este registro lo podrán hacer 1 o más personas a la vez.

Por otro lado, podemos observar una relación de tipo composición entre *RegistroOlivaBet*  e *InicioOlivaBet*. En cuanto a la multiplicidad respecta, hemos pensado quedebería existir una relación 1…\* entre ambas porque un usuario puede entrar varias veces al menú de inicio una mediante registro o mediante inicio de sesión, al menú de inicio a su vez pueden entrar 1 o más de 1 usuarios registrados o con la sesión iniciada.

Otra relación evidente en el diagrama, siendo también de tipo composición, se da entre la clase abstracta llamada *Maquinas* y la clase *InicioOlivaBet*. En cuanto a la multiplicidad, las máquinas estarán en 1 único menú de inicio pero en dicho menú, 5 máquinas estarán disponibles según sus puntos requeridos.



**Imagen 3:** Esta imagen muestra cómo está compuesto el UML de Oliva Bet [Clic aquí](https://docs.google.com/drawings/d/12ukLXAOACdrb3CbD9duzhsgxBoNVlkX-lEOfjGZoGys/edit) para visualizar el diagrama UML

Lenguajes de marcas

# Páginas web

En cuanto a la página web, hemos decidido obtener una plantilla en esta página, ya que veíamos más viable hacerlo de esta forma que hacerlo en wordpress y desde 0. Es notorio que nuestra página tiene cierta complejidad en el tema del diseño de la misma, entonces con ayuda de todos nosotros y encargados de partes diferentes para terminarla a tiempo ha sido posible finalizar la misma. Una de las partes importantes ha sido escoger imágenes sin copyright, ya que así nos ahorramos posibles líos a futuro con los dueños de estas[1].

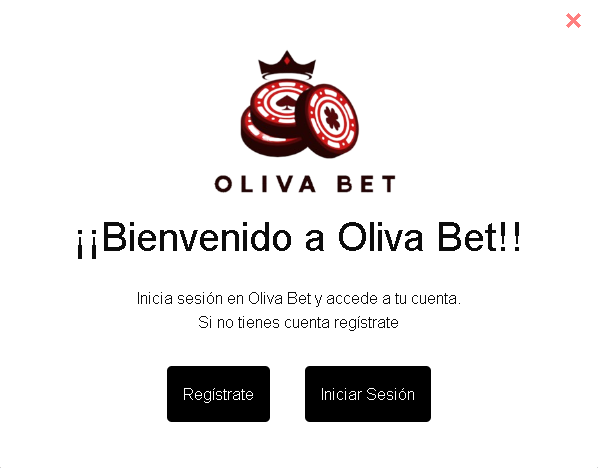
Tenemos una página de inicio con la información más importante sobre el juego. Además, la página dispone de una breve explicación sobre nuestras intenciones como se ve en la **Imagen 4** y lo que nos ha motivado a crear el juego. Sumando a eso, tenemos un apartado para descargar nuestro juego con una explicación sobre las recompensas de cada modalidad de juego, en la **Imagen 5**.

**Imagen 4:** Aquí se ve la página de inicio. Es lo primero que se ve al entrar en la WEB.

**Imagen 5:** Explicación del coste de cada tirada y los premios en cada juego.

**Imagen 6:** Ranking de los 10 mejores usuarios.

También hemos creado unos formularioscomo se puede ver en las **imágenes 7 y 8** para crear una cuenta e iniciar sesión, que permite a los usuarios guardar su puntuación para poder seguir desde donde dejaron el juego, recogiendo los datos desde nuestra base de datos. Esto también nos ha permitido crear un ranking de los jugadores con las puntuaciones más altas. Para poder crear y visualizar nuestro ranking en la página web que se puede ver en la **Imagen 6**, seguimos [este vídeo](https://www.youtube.com/watch?v=bHFoobciCTM) que combinaba php y html para poder visualizar una tabla de la base de datos[2].



**Imagen 7:** Ventana que sale al hacer clic en el botón “Cuenta”, para registrarse o iniciar sesión.

|  |  |
| --- | --- |

**Imagen 8:** Los formularios para crear una cuenta e iniciar sesión, para guardar la puntuación del usuario y mostrarlo en el ranking (en caso de alcanzar suficientes puntos).

Por otro lado, queremos también decir que nuestras reglas de las máquinas estarán disponibles para descargarlas en formato *pdf* en otro apartado de la web. Hay que tener en cuenta además, que hay ciertas paginas a las que no se puede acceder sin tener la sesión iniciada, por ejemplo, en el apartado de cuenta, el cúal muestra la información del usuario.

# Conversión a .php

Hemos tenido que cambiar todas las extensiones de .*html* a *.php* en las páginas para poder emplear sesiones en cada una de estas, gracias a las mismas, el usuario podrá ver su nombre introducido en el formulario una vez inicie sesión en *Oliva Bet*. La modificación más notoria pese a que la base de *html* siga siendo la misma, en ciertas partes del código, hacemos la llamada al lenguaje de programación *php.*

Tenemos dos archivos de validación *.php* que insertan y obtienen datos de la base de datos de *Oliva Bet*, estos mismos archivos los explicaremos más adelante en este documento. El resto de archivos con la misma extensión, son las antiguas páginas *.html.*

# JSON/JSON Schema

En cuanto al *Json Schema* hemos tenido en cuenta la tabla llamada *usuarios* de nuestra base de datos. Esta tabla la hemos representado con un *array* llamado *Casino* que a su vez se compone de los siguientes subelementos como *idUsuario, nombreUsuario, contrasena, correoelectronico, fecha\_Alta,* *IP\_Regsitro* y *puntos,* en los cuales hemos introducido un *pattern* en alguno de estos , es decir, que le hemos puesto una condición que en caso de no cumplirla se marcará un error en el *Json*, pese a que los datos que hemos utilizado para representar el mismo cumplan con esa condición.

Con esto queremos decir que un usuario puede registrarse sin cumplir estas condiciones, pero nosotros hemos empleado las mismas para visualizar este apartado mediante los propios *Json Schema* y *Json* e incluso los registros insertados en la base de datos.

En el elemento *idUsuario* hemos puesto que sea de tipo *string,* ya que los *patterns* se usan más que nada para ponerle una condición a dicho tipo de dato. En este caso el *pattern* añadido es que sea mínimo un número o más.

| "idUsuario": {  "type": "string",  "pattern": "^[0-9]+$"  }, |
| --- |

El siguiente elemento es *nombreUsuario* de tipo *string* que la condición que le hemos puesto es que empiece por una mayúscula y que seguido pueda ir cualquier carácter alfanumérico seguido de otro tipo de carácteres excepto el salto de línea.

Queremos especificar que pese a tener en la base de datos este campo como tipo *integer*, hemos decido definirlo como string para emplear su *pattern*.

| "nombreUsuario": {  "type": "string",  "pattern": "^[A-Z]{1}[a-zA-Z0-9].\*$"  }, |
| --- |

El elemento *contrasena* también es de tipo *string* con la condición de que sean letras mayúsculas, minúsculas o números, este campo contendrá 10 carácteres alfanuméricos o más.

| "contrasena": {  "type": "string",  "pattern": "^[a-zA-Z0-9]{10,}$"  }, |
| --- |

El elemento *correoelectronico* es el que más nos ha costado, ya que su *pattern* ha sido el más complicado a pesar de ser de tipo *string* como alguno de los anteriores*.* La condición que hemos asignado es que haya letras minúsculas, números, puntos, barra baja, porcentaje, plus o minus seguido de un arroba y de nuevo estos caracteres 1 o más veces. Después obligatoriamente deberá de ir un punto y dos o tres letras minúsculas que representan la extensión del dominio en cuanto al correo (*.com*, .*cus*, .*es*…) .

| "correoelectronico": {  "type": "string",  "pattern": "^[a-z0-9.\_%+-]+@[a-z0-9.\_%+-]{1,}[.][a-z]{2,3}$"  }, |
| --- |

El siguiente elemento que tenemos asignado es *fecha\_Alta* que es de tipo *string* y formato *date-time* que solo se usa para fechas y al contrario de las cuatro anteriores, esta se introduce de forma automática en cuanto al registro en el juego o en la web. En este caso no le hemos puesto ningún *pattern[3]*.

| "fecha\_Alta":{  "type":"string",  "format":"date-time"  }, |
| --- |

El penúltimo elemento es *IP\_Registro* de tipo *string* y en este caso le hemos puesto la condición de que haya 1, 2 o 3 números seguidos de un punto y así sucesivamente tres veces. Después de finalizar esa cadena de caracteres tendrá que ir de nuevo 1, 2 o 3 números. También se introduce automáticamente este atributo al hacer un registro.

| "IP\_Registro":{  "type":"string",  "pattern":"^([0-9]{1,3}[.]){3}[0-9]{1,3}$"  }, |
| --- |

Por último tenemos el elemento *puntos* de tipo *Integer* que no le hemos puesto ningún pattern y se introduce de forma automática una vez te registras, la cantidad de puntos iniciales concretamente es de 5000.

| "puntos":{  "type":"integer"  } |
| --- |

Ahora pondremos 5 ejemplos de cómo daría error algunos datos en caso de no cumplir los *patterns*:

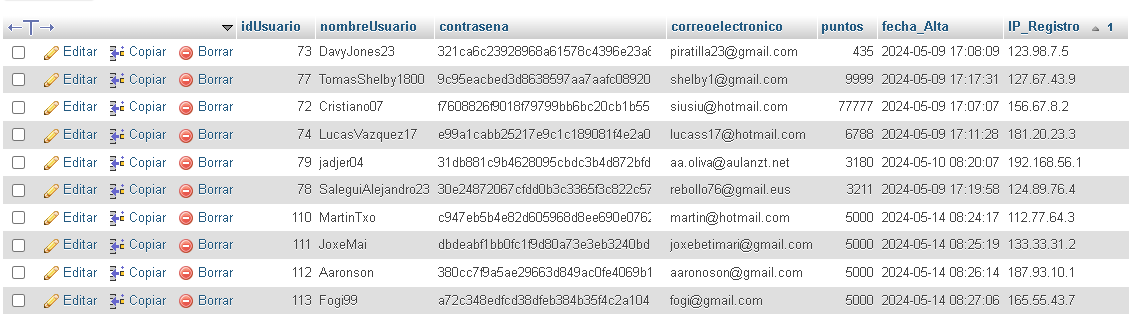
* Ejemplo 1:***idUsuario* = A1** Daría error ya que solo admite números y en este caso tiene una letra
* Ejemplo 2:  **nombreUsuario = sergio45** El error está ya que está obligado a empezar con una mayúscula y empieza por minúscula.
* Ejemplo 3: **contrasena = Iñigo27y** El error sería que mínimo debe de tener 10 caracteres y solamente tiene 8.
* Ejemplo 4 : **correoelectronico = Antonio(4)@gmail.com** El error está en la mayuscula ya que no se puede poner mayúsculas en el correo electrónico y el otro error serían los paréntesis que tampoco se podrían poner según nuestro *pattern*
* Ejemplo 5: **IP\_Registro = 192.168.4.67.** El error se encuentra en el punto final ya que la condición que hemos añadido es que haya 1, 2 o 3 números seguidos de un punto así sucesivamente 3 veces y después de cumplirse eso deben de ir 1, 2 o 3 números sin punto y en este caso finaliza con punto.

A continuación se mostraran 5 ejemplos de cómo estarían de forma correcta los datos con los *patterns*:

* Ejemplo 1:***idUsuario* = 34** Estaría bien ya que solo aparecen números en el id
* Ejemplo 2:  **nombreUsuario = RoDri56** Está bien escrito porque empieza por mayúscula y va seguido de letras, números o cualquier carácter.
* Ejemplo 3: **contrasena = Fernando3456** Está escrita de forma correcta porque solo hay números o letras, 10 veces o más
* Ejemplo 4 : **correoelectronico = pet4e\_r.@gmail.com** El correo esta bien escrito ya que solo hay minúsculas, números y barra baja, después un arroba de forma obligatoria después solo hay letras minúsculas y para acabar un punto y de 2 a 3 letras minúsculas
* Ejemplo 5: **IP\_Registro = 192.168.10.3** Es correcto porque hay una secuencia de 1, 2 o 3 números seguido de punto 3 veces y acaba con una secuencia de 1, 2 o 3 números sin punto



**Imagen 9:** Esta imagen se muestra el JSON SCHEMA



**Imagen 10**: Aquí se puede observar los datos del JSON schema

agregados en la base de datos



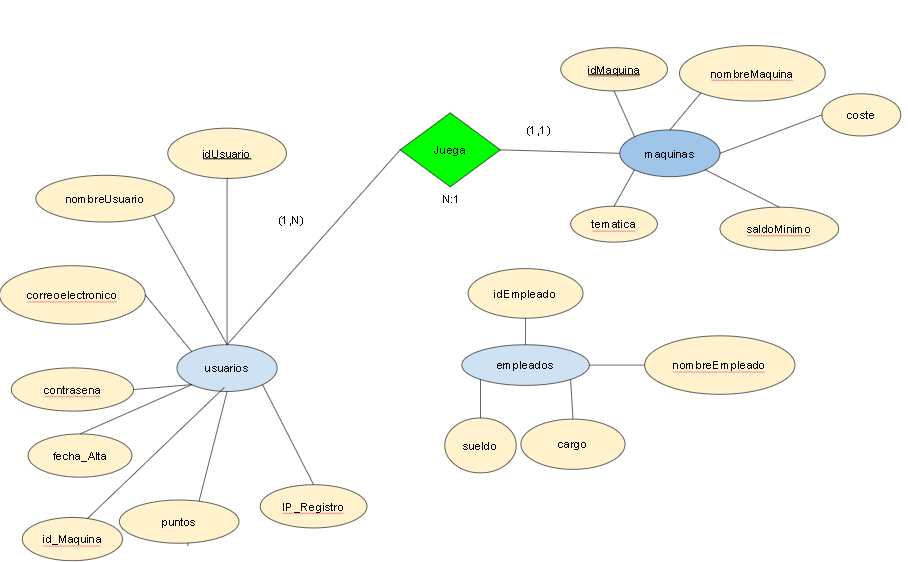
**Imagen 11:** En esta imagen se logra apreciar el JSON con los datos añadidos de 10 usuarios dentro del *array* general *Casino*.

Bases de datos

# Diagrama entidad/relación

Para nuestro diagrama de entidad/relación sobre nuestra base de datos, hemos decidido que contenga 3 entidades que son *usuarios***,** *máquinas*y*empleados,*tal y como se puede ver en la **Imagen 12.** Hemos decidido esto ya que nuestro servicio trata sobre *slots* online, y es necesario que todos los usuarios que quieran jugar a nuestro juego o registrarse en nuestra web deban cumplir con ciertos requisitos, los atributos encargados de esto son: *idusuario,**nombredeUsuario***,** *correoelectronico***,** *fecha-Alta, IP\_Registro, puntos*y*contrasena***.** Estas estarán encargadas en la entidad *usuarios* para que las personas puedan registrarse y jugar en las máquinas.

Ya que el juego es virtual cada máquina debe tener sus respectivos atributos que serán: *idMaquina, nombreMaquina, coste, tematica* y *creditosMinimos***.** Hemos puesto estos campos ya que cada máquina es única, y tendrán que tener información de cada uno de estos para que puedan ser usadas y diferenciarlas. Por último hemos hecho la entidad de *empleados*, que es una entidad creada específicamente para los programadores encargados del sitio web. Para tener los datos sobre ellos, su sueldo asignado y su respectivo cargo, se han hecho los atributos:  *idEmpleado, nombreEmpleado, cargo* y *sueldo.*



**Imagen 12:** Podemos observar en esta imagen nuestro diagrama entidad/relación completo

El temas de la relación entre la entidad *usuarios* con *maquinas* se ha asignado de la siguiente manera:

* Un usuario solo puede jugar en una máquina a la vez, pero una máquina puede ser jugada por 1 o por N usuarios al mismo tiempo. Es por esto, que nos encontramos con una relación de tipo N:1

Queremos también destacar que teníamos ideada otra relación de tipo N:1 entre *usuarios* y otra entidad llamada *usuariosRanking* que ya no es existente, es por esto, que esta idea la explicaremos en el apartado de desviaciones.

# Nuestra Base de Datos

En cuanto a las tablas creadas, sus atributos y las relaciones en nuestra base de datos, hemos seguido el mismo modelo diseñado en el diagrama entidad/relación en cuanto a entidades y atributos. Por otra parte, la relación existente en el diagrama la hemos omitido aquí, ya que hemos visto que carecía de sentido aplicar una clave foránea en la tabla de inserción de datos de usuario (*usuarios*), que en este caso ese campo sería una clave foránea de la tabla *maquinas* que representaría su mismo identificador, por ejemplo *id\_Maquina* al igual que en el diagrama; es por esto que *usuarios*está relacionada con la tabl*a maquinas* en cuanto al diagrama pero no en cuanto a la base de datos. El diseño general de la base de datos se puede observar en la **Imagen 13**.

Además, tenemos campos de todo tipo, enteros, de tipo *text, varchar()* y *timestamp*. Es evidente que hoy en día a todas las compañías con una base de datos se les requiera guardar la IP de los usuarios por si hay algún tipo de problema, para ello, hemos definido dicho campo como *varchar()* y en el archivo .php de validación de formulario, lo hemos definido de la siguiente manera:

| $ip = $\_SERVER['REMOTE\_ADDR']; |
| --- |

Hemos aplicado la funcionalidad en nuestro archivo de validación .php para poder encriptar las contraseñas de los usuarios y proteger su seguridad en caso de que alguien nos usurpe la base de datos. Hemos realizado este acto mediante *MD5* (Message Digest Version 5) en el archivo .php:

| $contrasena = $\_GET["contrasena"];  $str = $contrasena;  $encript = md5($str); |
| --- |

Por otro lado, hemos empleado otro archivo .php para el inicio de sesión, el cual explicaremos en profundidad en dicho apartado.



**Imagen 13:** Esta imagen muestra el diseño de la base de datos de *Oliva Bet*

Además, otro tema relacionado con la base de datos es el servidor RDS que hemos montado. Hemos decidido no poner ese apartado en la asignatura de Bases de Datos y hemos preferido asignar su propia sección en la asignatura de Sistemas Informáticos de Red.

## 

## 

## Archivos .php de registro e inserción en profundidad

Es evidente que estos archivos son necesarios para realizar una conexión a la base de datos y hemos hecho uso de los mismo, a continuación vamos a hacer la mención a los dos archivos que se conectan a la base de datos:

### Archivo de registro

En cuanto al archivo *.php* de registro, nos hemos basado en el modelo que un integrante tenía del anterior reto, modificamos ciertos aspectos como el nombre de la base de datos, la tabla, los campos… Como anteriormente hemos mencionado, hemos añadido ciertos aspectos como el tema de encriptar la contraseña del usuario mediante *MD5*, obtener la IP desde donde se hace el registro[4],[5]…

Por otro lado, en el apartado de desviaciones podemos observar que cambios hemos realizado en la base de datos en sí.

### Archivo de inicio de sesión

En cuanto a la validación de inicio de sesión del formulario, hemos seguido la misma estructura que en el archivo de inserción. Para poder iniciar sesión teníamos que comparar los datos que el usuario introducía con los datos de nuestra base de datos, pensábamos al principio que era una buena idea desencriptar la contraseña de la base de datos para compararla con la del usuario al iniciar sesión, pero decidimos que era una mejor idea encriptar la del usuario y compararla con la contraseña encriptada de la base de datos.

Para poder comparar todos los registros hemos realizado esta consulta:

| $nombreusuario = $\_POST["nombreUsuario"];  $contrasena = md5($\_POST["contrasena"]);  $correoelectronico = $\_POST["correoelectronico"]; |
| --- |

| $sql = "SELECT nombreUsuario, contrasena, correoelectronico FROM usuarios WHERE nombreUsuario = '$nombreusuario' AND contrasena = '$contrasena' AND correoelectronico = '$correoelectronico'"; |
| --- |

Por otro lado, en este mismo archivo, hemos necesitado usar sesiones, para que la sesión del usuario se mantenga iniciada, en este apartado tanto nuestro compañero de clase Andoni cómo el profesorado nos han ayudado a poder diseñar el código, he aquí el ejemplo del mismo:

| $\_SESSION['usuario'] = $\_POST['nombreUsuario'];  $\_SESSION['correo'] = $\_POST['correoelectronico'];  $\_SESSION['puntosSesion'] = $\_POST['puntos'];    echo $\_SESSION['usuario'];  echo $\_SESSION['correo'];  echo $\_SESSION['puntosSesion']; |
| --- |

Sistemas Informáticos de Red

# Servidor EC2

Para poder instalar un servidor *EC2* en Amazon hemos abierto una instancia en Amazon. El Sistema Operativo que hemos instalado es ubuntu *24.04 LTS (HVM)* , el tipo de instancia es una *t2.micro* y tiene una arquitectura de 64 bits (x86). También hemos empleado un par de claves y hemos permitido el tráfico de *HTTPS* desde Internet y el primer paso que hemos realizado en el servidor en sí ha sido instalar el paquete *apache2* y el conector de *php*  para que el servidor tenga servicio web. Por otro lado, hemos decidido instalar en cuanto a almacenamiento de raíz de 8 GiB, más que suficiente para almacenar nuestra web.

Para poder añadir nuestras páginas web y nuestros archivos de validación *.php* al servidor *EC2*, hemos instalado una aplicación llamada *WinSCP*. Gracias a ella, subimos los archivos necesarios para visualizar nuestra web en la dirección */var/www/html* del servidor *EC2*.

Para mover dichos archivos hemos usado el siguiente comando : *mv \* /var/www/html*

Esa era nuestra idea básica pero en el apartado de desviaciones explicaremos más a fondo el porqué no hemos podido finalizar esta tarea.

# Servidor RDS

Como alternativa para migrar nuestra base de datos local a un servidor decidimos migrar la misma a un servidor RDS de Amazon. Es cierto que tenías pensado emplear el servidor de Nazaret como bien se puede observar en el apartado de la planificación pero debido a la falta de tiempo, decidimos montar un servidor RDS en Amazon. Para instalar el mismo no tuvimos ningún problema pero a la hora de realizar la conexión con el cliente *MySQL Workbench*. Para solucionar el mismo, vimos [este vídeo](https://www.youtube.com/watch?v=07mAdMTwRHs) que nos ayudó a descubrir cuál era el problema[6].

El error consistía en que no todos los equipos podían conectarse a dicha base de datos, es por esto que modificamos el grupo de seguridad por defecto de este servidor añadiendo dos entradas nuevas. Ambas permitían todo tipo de tráfico, una para cualquier IPv4 (Internet Protocol version 4) y otra para cualquier IPv6(Internet Protocol version 6). Una vez hecho esto, introducimos las credenciales establecidas y copiamos el código *.sql* que *phpMyAdmin* genera al exportarlo y pegamos el mismo en nuestro servidor.

Por último en cuanto a este apartado respecta, modificamos los códigos *.php* y *.java* con las credenciales de nuestro *RDS*.

# Programas en Java

## Elementos generales

* La puntuación : en cuanto a la puntuación, esta es notoria en todas las clases del programa ya que tiene que variar en todas por igual. Al puntuar en una máquina y cobrar dichos puntos, se tiene que actualizar dicho campo en la base de datos.
* Condiciones para jugar a las máquinas : cada una de estas tiene un coste y para entrar a algunas de estas también se requiere un saldo.
* El usuario y la conexión constante con *MySQL* y la base de datos : es evidente que las clases deben de ser conscientes de quién es el usuario que está en el juego, para eso el campo que el programa de registro obtiene es el del nombre de usuario para que así si cumple con la condición de entrar en el ranking, entre una vez termina de jugar.

### Nuevos comandos

*panelEmojis.removeAll()*;: esto es útil cuando necesitas limpiar el contenido existente del panel antes de agregar nuevos componentes o actualizar su contenido. En este caso, se utiliza antes de cargar nuevos emojis en la máquina[7]

*panelEmojis.revalidate();*: es útil después de agregar o eliminar componentes de un contenedor en tiempo de ejecución, ya que permite al *LayoutManager* ajustar el tamaño y la posición de los componentes correctamente.

*panelEmojis.repaint()*;: este método solicita al sistema operativo que repinte el panel y sus componentes. Se utiliza después de *revalidate()* para asegurarse de que los cambios realizados en el diseño se reflejen visualmente en la interfaz de usuario.[8]

## Programa de registro e inicio de sesión

Para el programa de registro en *Java* basado en la superclase *JFrame* y con *ActionListener* implementado, hemos seguido la estructura de nuestro formulario de la página web. Se pueden observar en la **Imagen 14** dichos campos a rellenar y en la parte inferior dos botones, uno para registrarse y otro para iniciar sesión. También es notorio un título en la parte superior del programa.

En cuanto a la conexión con la base de datos para poder registrar los valores, hemos utilizado [esta librería](https://dev.mysql.com/downloads/connector/j/) para poder establecer y cerrar una conexión a través de *MySQL*. Para la inserción en sí, nos hemos validado de la siguiente consulta:

| String sql = "INSERT INTO usuarios (nombreUsuario, contrasena, correoelectronico,fecha\_Alta,IP\_Registro) VALUES ('"+ nombreUsuario+"','"+contrasena+"','"+ correoElectronico+"','"+time+","+ipaddress+")";  PreparedStatement stmt = conn.prepareStatement(sql); |
| --- |

El tema de las comillas individuales es indispensable para que este lenguaje de programación acepte y diferencie variables y campos de texto, ya que al introducir comillas dobles dentro de otras comillas dobles se producirá un error.

Por otro lado, ya que los campos de *fecha\_Alta* e *IP\_Registro* de la tabla de *usuarios* se rellenan automáticamente gracias a la propia base de datos en el primer caso y gracias al archivo *.php* de validación en el segundo, hemos tenido que buscar otros recursos para estos en el programa. Además el tema de encriptar la contraseña mediante *MD5* también lo hemos buscado ya que esa función también la realizaba el archivo de validación.

Para encriptar el campo de la contraseña, un compañero de clase, exactamente Beñat Ezquerro nos ayudó con una función para encriptar un campo de tipo *String*.

Para el campo de *fecha\_Alta*ya que requiere un elemento de tipo *timestamp* hemos empleado el siguiente código[9]:

| long time = System.currentTimeMillis();  java.sql.Timestamp timestamp = new java.sql.Timestamp(time);  System.out.println("Time in milliseconds :" + timestamp); |
| --- |

Para el campo de*IP\_Registro* ya que obtiene la IP desde el dispositivo que se realiza el registro, hemos tenido que implementar lo siguiente[10]:

| InetAddress my\_localhost = InetAddress.getLocalHost();  System.out.println("The IP Address of client is : " + (my\_localhost.getHostAddress()).trim());  String my\_system\_address = ""; |
| --- |

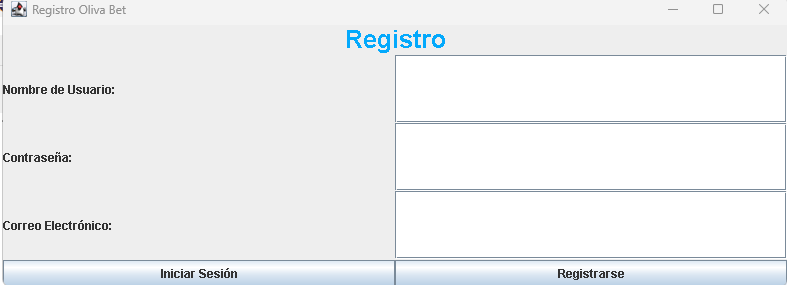
El método que hace posible la inserción de datos y que engloba los conceptos antes mencionados se llama *insertarDatos()*.

Por otro lado hemos aplicado la funcionalidad de inicio de sesión, al igual que hemos creado un método antes mencionado para el registro. En este caso hemos creado el método llamado *inicioSesion()*.

La estructura es muy similar respecto al registro, pero la consulta y otros apartados como un método que devuelve un *boolean* y la sintaxis cambian en algunos apartados. He aquí los cambios básicos respecto al registro para poder iniciar sesión:

| boolean funciona = false;  String iniciosesion = "SELECT nombreUsuario,contrasena,correoelectronico FROM usuarios WHERE nombreUsuario ='"+nombreUsuarioInicio+"'AND contrasena ='"+encriptadoInicio+"'AND correoelectronico ='"+correoElectronicoInicio+"'";  PreparedStatement stmt = conn.prepareStatement(iniciosesion);  System.out.println(iniciosesion);      ResultSet inicioSesion = stmt.executeQuery();  if(inicioSesion.next()) {  funciona = true;    JOptionPane.showMessageDialog(this, "Felicidades, has iniciado sesión","Inicio de sesión",JOptionPane.INFORMATION\_MESSAGE);  new InicioOlivaBet();  } else {  JOptionPane.showMessageDialog(this, "Tienes que introducir datos correspondientes","Error al inciar sesión",JOptionPane.ERROR\_MESSAGE);  } |
| --- |

En cuanto al *return* del *boolean*, este se puede ver en las líneas de código posteriores.



**Imagen 14:** En esta imagen se puede observar el programa de *RegistroOlivaBet*

## Programa de menú de inicio

Hemos ideado un programa en *Java* que nos muestra un menú de inicio para seleccionar la máquina que queremos, exactamente tenemos tres opciones: Frutas, Animales, Joyas, Halloween y Navidad. Este programa además implementa *ActionListener* y desciende de la superclase de *JFrame*.

Se puede observar en la **Imagen 15** que además de botones para entrar en dichas *slots* hay una imágen para cada una de ellas, para que se vea mejor que máquina es a la que vamos a jugar, por otro lado se puede observar un título en la parte superior de dicha ventana. Hay que mencionar también que esta ventana o este programa se visualiza una vez te hayas registrado o hayas iniciado sesión. Además, hay opciones de ver el ranking de los cinco mejores usuarios, de cobrar terminando así el juego, volver a jugar y el saldo de inicio se restablecerá en 5000 puntos.



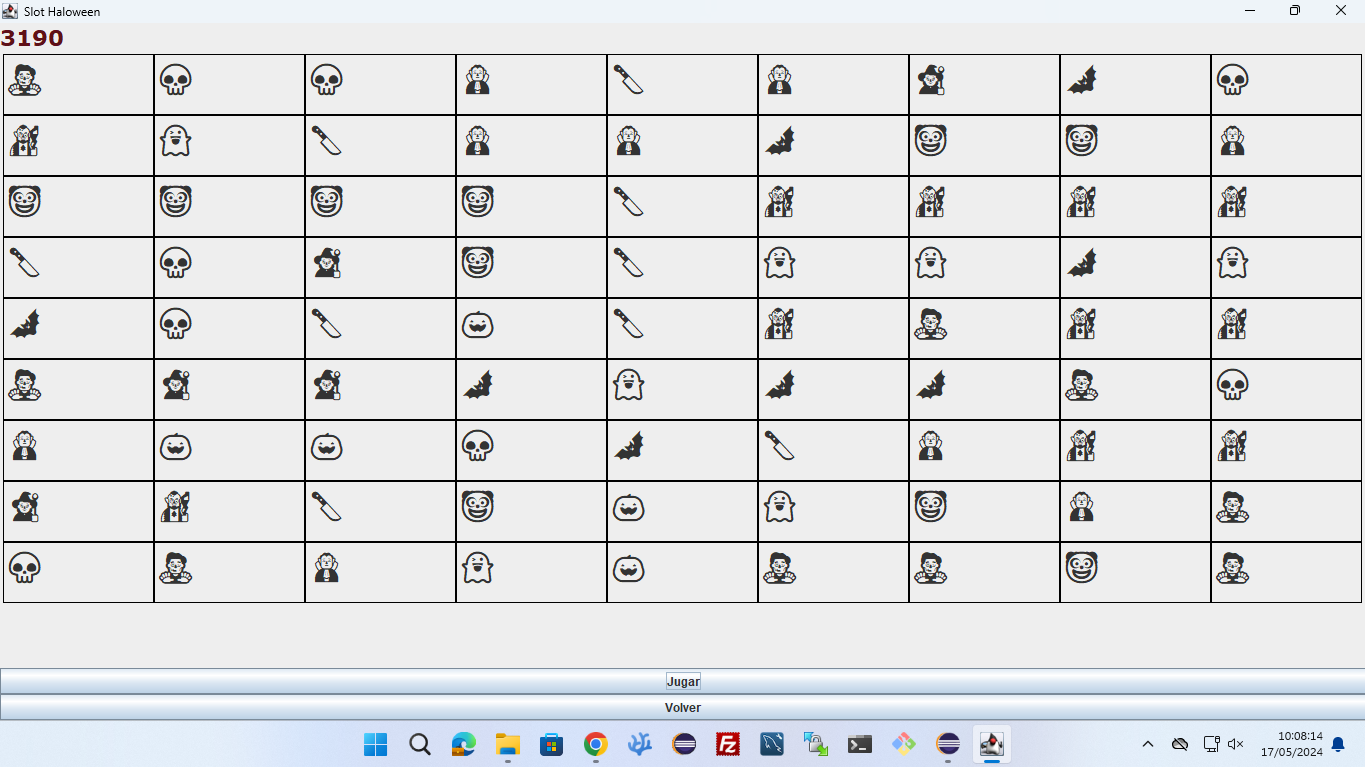
**Imagen 15:** Esta imagen muestra el menú de inicio de *Oliva Bet*

## Programas de máquinas tragaperras

En nuestras máquinas tragaperras, hemos implementado distintas puntuaciones según cuantas [frutas](https://www.canva.com/design/DAGEclg8Coo/SBw2yKRaIq-kucQyyXkAiQ/edit?utm_content=DAGEclg8Coo&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton), [animales](https://www.canva.com/design/DAGEcwlu_2s/HVQpCWh3iuNhZjgDCw8plA/edit) u otros símbolos iguales te salgan, se te asignará una puntuación diferente, por ejemplo en la tragaperras de frutas, con 5 frutas iguales haces un x0.5 del valor de las mismas, con 6 frutas iguales x1, con 7 x1.5 y con 8 o mas x2, esto también depende de las ganancias de cada fruta, en otras máquinas en cambio, hemos implementado el mismo sistema pero con distintas puntuaciones.

Estos programas los hemos hecho en *Java*, y para saber las diferentes puntuaciones hemos hecho un diseño en canva, pero el cúal estará en nuestra web para que los usuarios puedan saber cuánto pueden ganar. Por otro lado, en diferentes fechas como halloween o navidad, vamos a insertar nuevas máquinas.

En las tragaperras de [Halloween](https://www.canva.com/design/DAGEcWaJptI/770-6dtYprem9LLWDpgo0g/edit?utm_content=DAGEcWaJptI&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton) por ejemplo, con 8 símbolos iguales haces un x0.5 del valor de símbolo, con 9 símbolos iguales x1, con 10 x2, con 11 x4 y con 12 o más x7 esto también depende de las ganancias de cada símbolo. Ejemplo de la máquina en la imagen 16.

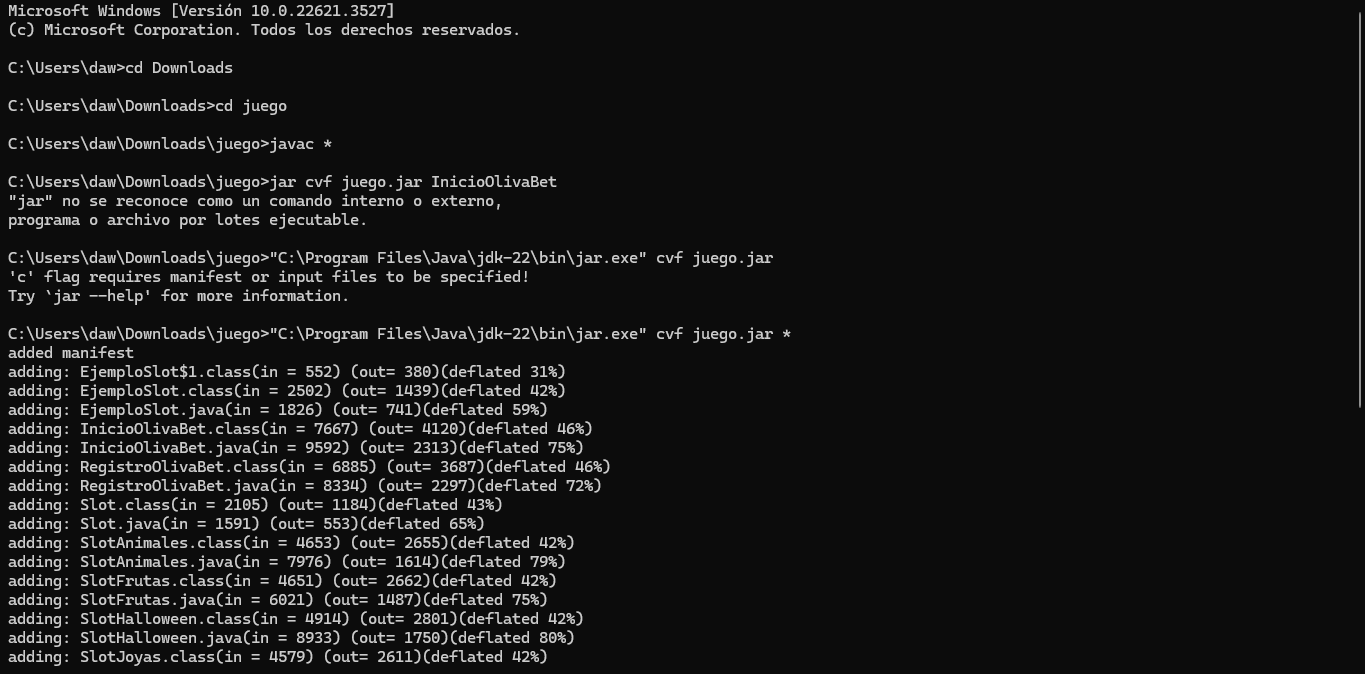


**Imagen 16:** Tragaperras de Halloween

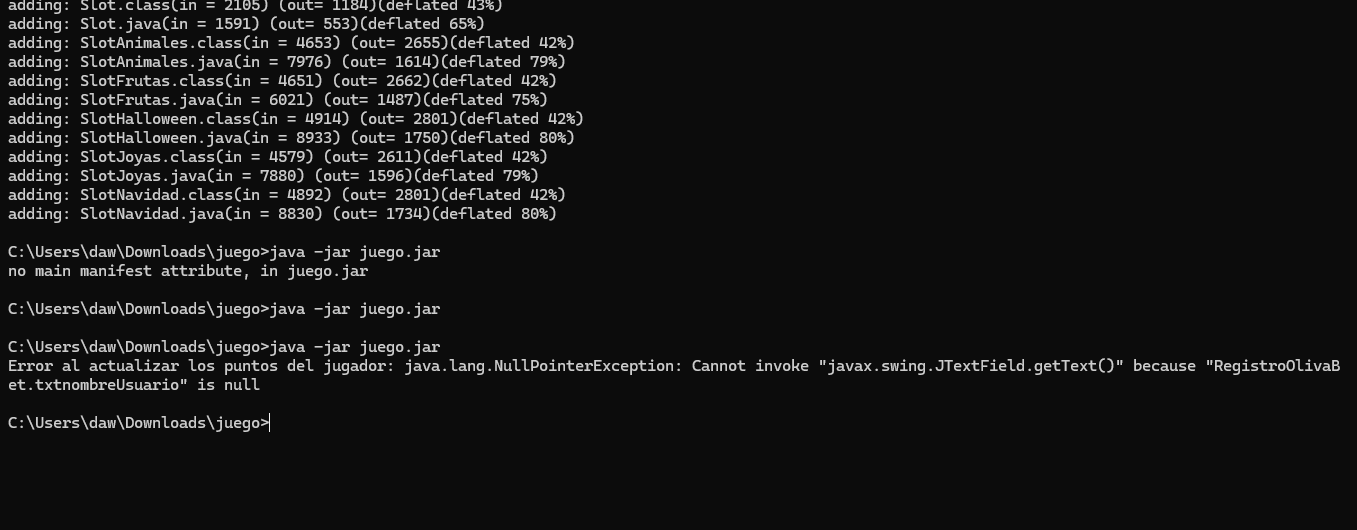
En las tragaperras de [Navidad](https://www.canva.com/design/DAGEhqupVvk/WaZmagL32AjnCreQa7B8ug/edit?utm_content=DAGEhqupVvk&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton) por ejemplo, seguimos el mismo esquema que en la de Halloween y la tragaperras de [Joyas](https://www.canva.com/design/DAGEb6PbjpI/KHJGloTnLhEgQ1cNVm7Mvg/edit) tiene otras normas diferentes que se pueden ver en dicho enlace.

## Jar

Hemos conseguido hacer el archivo *.jar*, pero hemos tenido unos errores para que se iniciase, ya que faltaba un conector y un archivo *Jni*, hemos decidio no hacerlo ya que no se abriría nunca



**Imagen 17:** Primera parte para crear el archivo *jar*



**Imagen 18:** Segunda parte para crear el archivo *jar*

Formación y Orientación Laboral

# Contratación de los empleados y sus nóminas

Para el tema de la contratación de los empleados con sus respectivas nóminas, hemos ideado cinco contratos, el tipo de estos cambia en algunos casos, es decir, para cinco empleados hemos usado concretamente tres tipos de contrato como son el contrato indefinido, el contrato de relevo y el contrato de prácticas. Nos pareció una buena idea usar nuestros nombres para referirnos a los empleados y cada uno de nosotros tiene como antes mencionado un contrato específico, no solo varía el tipo de contrato, sino que también cambian aspectos fundamentales como pueden ser el sueldo, la duración de dicho contrato, cláusulas y demás.

Todos los contratos siguen convenios colectivos, en nuestro caso, hemos necesitado dos:

* Para el empleado de atención al cliente: Convenio de Servicio al cliente: CCT 781/20
* Para los empleados con rol de programador Júnior y los empleados de mantenimiento web: Convenio de informáticos (programador y mantenimiento WEB): Convenio Colectivo de Trabajo 1696/23

Inglés

# Presentación de la página en Inglés

Para el apartado de esta asignatura, hemos pensado realizar un vídeo en dicho idioma a modo de anuncio para presentar y vender el proyecto. En este vídeo participaremos todos los integrantes del grupo, en cambio, para hacer más vistoso el vídeo antes mencionado, nos grabaremos dándole la entonación que un buen vídeo merece. Para ello, iremos a otra clase con un proyector para visualizar el contenido necesario y hemos diseñado un breve guión con lo que cada uno tiene que decir realmente en el vídeo.

Para visualizar el guión [clic aquí](https://www.canva.com/design/DAGEn7U2KXU/sPp53A6LpsFktg_oIcrl2w/edit)

Otros apartados

# Desviaciones

* Como primera desviación observable, teníamos diseñada tanto en el diagrama entidad/relación como en la base de datos otra tabla llamada *usuariosweb*. Nuestra idea era que los datos de los usuarios que se registraban tanto en el programa Java como en la página web se guardarán en tablas distintas de la base de datos. Al ver que era innecesario, decidimos borrar esta tabla mencionada y emplear solo la tabla *usuarios*.
* Como segunda desviación hemos eliminado la relación de la base de datos existente en el diagrama porque tenía sentido aplicarle una clave foránea a esa tabla usuarios. Es decir, no tenía sentido insertar o dejar vacío este campo, ya que esa tabla es empleada para el registro e inicio de sesión.
* Como tercera desviación hemos eliminado del diagrama y de la base de datos la tabla *usuariosRanking*, puesto que después de varias pruebas y comprobar que no había mucho sentido teniendo la tabla usuarios, la hemos eliminado ya que no era posible conectarla a esta, y la relación de puntos se la hemos asignado a usuarios.
* Otra de las desviaciones ha sido la funcionalidad del servidor *EC2*. Nuestra página se visualizaba en dicho servidor, pero el tema de inserción de datos en el formulario no funcionaba. En cuanto a esto, movimos nuestras páginas al servidor de Nazaret para intentar solucionar el problema pero no fué posible. En definitiva, las páginas web están alojadas en uno de los equipos del grupo, para poder entrar a la misma: <http://192.168.1.12/finter-html/index.php>

# Planes a futuro

1. Uno de los principales planes para un futuro es el tema de cambiar por completo la lógica de nuestros juegos. Como anteriormente hemos mencionado, según la cantidad de “emojis” que haya en el juego, ganas o pierdes, pero nuestra idea para un futuro es el poder ganar o perder con la condición de que haya dichos “emojis” seguidos, bien sea verticalmente u horizontalmente. Ya que dicha lógica es muy compleja para el poco tiempo que teníamos, optamos por una opción más sencilla. Una vez combinados, hacer que se note dicha combinación, cambiando el color del borde por ejemplo.
2. Diseñar un botón para salir de cada programa. Esta idea la pensamos a mitad de proyecto y como los juegos en Java están basados en un *setLayout(new BorderLayout())* tendríamos que haber modificado los cinco programas.
3. Otro de los planes para el futuro, es poder poner sonidos en cada uno de los programas de las máquinas, tanto si ganas como si pierdes. Además, sería un detalle apreciable notificar al usuario si ha ganado o ha perdido en dicha partida con la cantidad de puntos correspondientes.
4. Aplicar el juego descargable .jar para que cualquier usuario de cualquier Sistema Operativo pueda disfrutar del mismo

# Diagrama de Gantt definitivo



**Imagen 19**: Diagrama Gantt definitivo

En este diagrama Gantt se pueden observar los cambios y modificaciones que hemos realizado desde el día en que tuvimos que entregar la planificación. Han habido cambios tanto en la duración de las tareas como en el reparto de tareas y los objetivos del proyecto.

Los cambios más notables ha sido la sustitución de la base de datos al servidor de Nazaret por el servidor RDS, además de las complicaciones que tuvimos a la hora de hacer la base de datos que nos llevó más tiempo del esperado.

Bibliografía

[1]

<https://www.free-css.com/>

[2]

<https://www.youtube.com/watch?v=bHFoobciCTM>

[3]

<https://opis.io/json-schema/2.x/formats.html#:~:text=invalid-,date%2Dtime,%3A%3ADDThh%3Amm%3Ass>

[4]

<https://stackoverflow.com/questions/3003145/how-to-get-the-client-ip-address-in-php>

[5]

<https://www.w3schools.com/php/func_string_md5.asp>

[6]

<https://www.youtube.com/watch?v=07mAdMTwRHs>

[7]

[Java ArrayList removeAll() (programiz.com)](https://www.programiz.com/java-programming/library/arraylist/removeall)

[8]

[Java Swing revalidate() vs repaint() - Desbordamiento de pila (stackoverflow.com)](https://stackoverflow.com/questions/1097366/java-swing-revalidate-vs-repaint)

[9]

<https://stackoverflow.com/questions/15534775/how-to-insert-current-time-in-mysql-using-java>

[10]

<https://www.tutorialspoint.com/java-program-to-find-ip-address-of-the-client>