|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | Data |
| Superclase: | | - |
| Nombre | **Descripción atributo** | **Colaboraciones** |
| dataType | Tipo de dato que representa | DataType |
| name | Nombre del campo del dato |  |
| rawData | Bytes de los datos |  |
| Nombre | **Descripción método** |  |
| getType | Devuelve el tipo de datos | DataType |
| getName | Devuelve el nombre del campo |  |
| getRawBytes | Devuelve los datos |  |
| getAsInteger | Devuelve una representación de entero |  |
| getAsString | Devuelve una representación como texto |  |
| getAsFloat | Devuelve una representación como float |  |
| getAsDouble | Devuelve una representación como double |  |
| getAsBoolean | Devuelve una representación como boolean |  |

## 3. Requerimientos funcionales:

*El proyecto se dividirá en cuatro subsistemas. La organización está basada en un sistema orientada a objetos, dónde se complementan las clases y los casos de uso. La base de datos, la seguridad y eficiencia juegan un rol muy importante.*

## Clases:

## El proyecto estará dividido en clases, cada una de la cual dispone de un prototipo de las funciones y atributos necesarios para que la aplicación funcione tal y como se planea, así mismo de un esquema general y tarjetas CRC en el anexo mostrando la interacción entre las clases y los tipos de dichas funciones y atributos.

## 1.1 Subsistema de gestión de usuarios:

***DATA***Representa un dato abstracto de una fila de resultados de una consulta a base de datos.

|  |  |  |
| --- | --- | --- |
| Nombre del enumerado: | | DataType |
| Superclase: | | - |
| Nombre | **Descripción literal** | **Colaboraciones** |
| Integer | Entero |  |
| Float | Número de coma flotante |  |
| Double | Número de coma flotante de doble precisión |  |
| Boolean | Valor de sí o no |  |
| String | Cadena de texto |  |
| File | Archivo |  |

***DATA TYPE***Tipo enumerado que representa un tipo de datos.

|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | DataRow |
| Superclase: | | - |
| Nombre | **Descripción atributo** | **Colaboraciones** |
| dataList | Lista de datos de la fila | Data |
| Nombre | **Descripción método** |  |
| getColumnCount | Devuelve el número de columnas |  |
| getColumnDataAtIndex | Devuelve el dato en el índice dado | Data |

***DATA ROW***Es una fila de datos devuelta por una consulta a una base de datos.

***DATABASE SERVICE***Clase abstracta que representa una conexión a una base de datos.

|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | DatabaseService |
| Superclase: | | - |
| Nombre | **Descripción método** |  |
| query | Devuelve un array de filas de la base de datos | DataRow |
| close | Cierra la conexión |  |

Al crear nuestras abstracciones para las bases de datos, podemos cambiar de servicio de base de datos sin tener que modificar el código, fomentando la **reusabilidad**, por ejemplo, teniendo una conexión a una base de datos MySQL y otra a SQLite. Se puede ver mejor la relación entre estas clases en el siguiente esquema:

***USER MANAGER***Clase con métodos estáticos que representa una lista de usuarios. Está conectada a una base de datos mediante un **DatabaseService.**

|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | UserManager |
| Superclase: | | - |
| Nombre | **Descripción atributo estático** | **Colaboraciones** |
| dataType | Tipo de dato que representa | DatabaseService |
| Nombre | **Descripción método estático** |  |
| getUser | Devuelve un usuario buscado | Data |
| deleteUser | Borra un usuario dado | User |
| modifyUser | Modifica un usuario dado actualizándolo | Data, User |
| insertUser | Inserta un nuevo usuario | User |

***USER***Usuario que utiliza la aplicación.

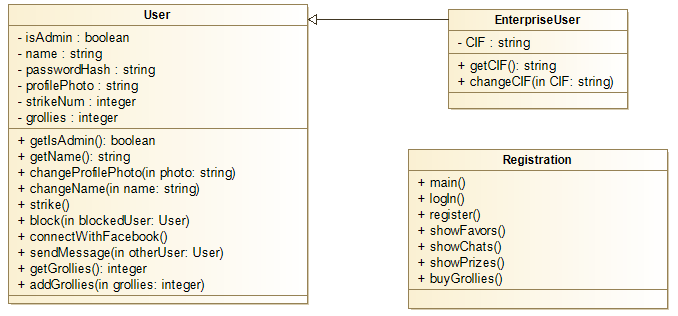
|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | User |
| Superclase: | | - |
| Nombre | **Descripción atributo** | **Colaboraciones** |
| isAdmin | ¿El usuario es administrador? |  |
| Name | Nombre del usuario |  |
| passwordHash | Hash seguro almacenado de la contraseña |  |
| profilePhoto | Ruta a la foto de perfil del usuario |  |
| strikeNum | Número de strikes que ha recibido |  |
| hrollies | Número de grollies que posee el usuario |  |
| Nombre | **Descripción método** |  |
| getIsAdmin | Devuelve si el usuario es administrador |  |
| getName | Devuelve el nombre del usuario |  |
| changeProfilePhoto | Cambia la foto de perfil del usuario |  |
| strike | Añade un strike al usuario |  |
| block | El usuario bloquea a otro usuario |  |
| connectWithFacebook | Conecta la cuenta con Facebook |  |
| sendMessage | Envía un mensaje a otro usuario |  |
| addGrollies | Añade (o resta) grollies |  |

***ENTERPRISE USER***Usuario especial que puede pagar por anunciar sus favores.

|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | EnterpriseUser |
| Superclase: | | User |
| Nombre | **Descripción atributo** | **Colaboraciones** |
| CIF | CIF de la empresa |  |
| Nombre | **Descripción método** |  |
| getCIF | Devuelve el CIF de la empresa |  |
| changeCIF | Cambia el CIF de la empresa |  |

***REGISTRATION***Clase principal que da acceso a todas las funcionalidades.

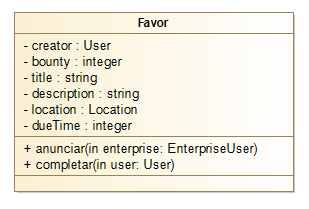
|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | Registration |
| Superclase: | | - |
| Nombre | **Descripción método** |  |
| main | Método principal de la aplicación |  |
| logIn | Permite al usuario iniciar sesión |  |
| register | Permite al usuario registrarse |  |
| showFavors | Muestra los favores cercanos | FavorManager |
| showChats | Muestra los chats del usuario | UserManager |
| showPrizes | Muestra los premios canjeables | PrizeManager |
| buyGrollies | Muestra la opción de comprar grollies | Transaction |

  
Esquema de las relaciones entre clases:

## 1.2 Subsistema de favores

***FAVOR***Representa un favor solicitado por un usuario.

|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | Favor |
| Superclase: | | - |
| Nombre | **Descripción atributo** | **Colaboraciones** |
| creator | Usuario que ha creado el favor | User |
| bounty | Recompensa recibida por completarlo |  |
| title | Título del favor |  |
| description | Descripción del favor |  |
| location | Posición del favor | Location |
| dueTime | Tiempo máximo para completar el favor |  |
| Nombre | **Descripción método** |  |
| anunciar | El favor es anunciado por una empresa | EnterpriseUser |
| completar | El favor es completado por un usuario | User |

Diagrama de la clase en UML que muestra los tipos de cada método y atributo:

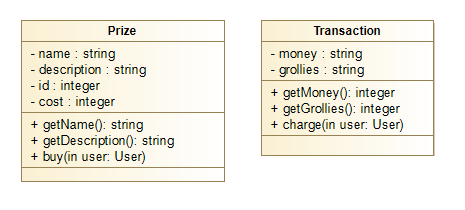
## 1.3 Subsistema de premios y compras

***PRIZE***Premio que el usuario puede intercambiar si tiene suficientes grollies.

|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | Prize |
| Superclase: | | - |
| Nombre | **Descripción atributo** | **Colaboraciones** |
| name | Nombre del premio |  |
| description | Descripción del premio |  |
| id | Identificación del premio (única) |  |
| cost | Coste en grollies del premio |  |
| Nombre | **Descripción método** |  |
| getName | Devuelve el nombre |  |
| getDescription | Devuelve la descripción |  |
| buy | El premio es comprado por un usuario | User |

***TRANSACTION***Transacción que representa un intercambio de dinero a cambio de grollies.

|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | Transaction |
| Superclase: | | - |
| Nombre | **Descripción atributo** | **Colaboraciones** |
| money | Dinero a cobrar |  |
| grollies | Cantidad de grollies a ingresar |  |
| Nombre | **Descripción método** |  |
| getMoney | Devuelve el dinero real a cobrar |  |
| getGrollies | Devuelve los grollies a ingresar |  |
| charge | Realiza el cargo a un usuario | User |

Esquema UML de las clases:

## 1.4 Subsistema de búsqueda y geolocalización

***FILTER***Filtro usado para restringir la búsqueda de favores.

|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | Filter |
| Superclase: | | - |
| Nombre | **Descripción atributo** | **Colaboraciones** |
| keywords | Palabras clave |  |
| position | Posición del favor | Location |
| minGrollies | Mínima cantidad de grollies |  |
| Nombre | **Descripción método** |  |
| getKeywords | Devuelve las palabras clave |  |
| getPosition | Devuelve la posición del favor | Location |
| getMinGrollies | Devuelve la cantidad mínima de grollies |  |
| changeKeywords | Modifica las palabras clave |  |
| changePosition | Modifica la posición del favor | Location |
| changeMinGrollies | Modifica la cantidad mínima de grollies |  |

***LOCATION***Posición en el globo terráqueo.

|  |  |  |
| --- | --- | --- |
| Nombre de la clase: | | Location |
| Superclase: | | - |
| Nombre | **Descripción atributo** | **Colaboraciones** |
| latitude | Latitud |  |
| longitude | Longitud |  |
| Nombre | **Descripción método** |  |
| getLatitude | Devuelve la latitud |  |
| setLatitude | Cambia la latitud |  |
| getLongitude | Devuelve la longitud |  |
| setLongitude | Cambia la longitud |  |

Esquema UML con las relaciones entre las clases:



## Casos de uso:

INSERTAR AQUÍ CASOS DE USO

## Requisitos de rendimiento:

La aplicación está orientada **a un público amplio**, y hemos planeado un **gran número de usuarios**, por lo tanto, estimamos que el número de terminales conectados al servidor puede ser bastante alto.

La actividad de los usuarios estará por lo general **repartida entre los días de la semana**, pero sufrirá picos de carga en **períodos de exámenes**, al estar el público objetivo más ocupado y por tanto con mayor necesidad de pedir favores. Es necesario adaptar los servidores para soportar dicha carga.

## Requisitos lógicos de la base de datos:

Todos los datos estarán almacenados en una **base de datos relacional**, y es importante mantener su **consistencia** y coherencia en todo momento, sobre todo con las transacciones monetarias. Deberá permitir **acceso concurrente** a un gran número de personas (ver requisitos de rendimiento). Se mantendrá una **copia de seguridad** que se renovará periódicamente.

Las principales clases que se relacionan con la base de datos son:

* **UserManager:** Gestiona los usuarios, administradores y empresas.
* **FavorManager:** Gestiona los favores solicitados por usuarios.
* **PrizeManager:** Gestiona los premios disponibles a cambio de grollies.

## Restricciones de diseño

La aplicación será implementada en un lenguaje **orientado a objetos**, con soporte para **módulos** y **paquetes**.

El almacenamiento se realizará en una **base de datos relacional**. Debido al diseño modular, pueden usarse distintos sistemas, aunque principalmente se usará MySQL.

## Atributos del sistema software

**DISPONIBILIDAD**La aplicación debe estar **disponible en todo momento** para el acceso por parte de los usuarios. El mantenimiento debe realizarse de manera que no interfiera con el uso normal de la aplicación.   
  
Deberán realizarse **copias de seguridad frecuentes y redundantes**, para poder reestablecer los datos en otros servidores en caso de daño. Dicha responsabilidad recae sobre los administradores.

**SEGURIDAD**Los datos del usuario seguirán la normativa recogida por la **Agencia Española de Protección de Datos** y se usará en todo momento **conexión cifrada** entre la aplicación y el servidor, para asegurarse que los datos de los usuarios no se vean comprometidos.

Se seguirá el protocolo **PCI** (Payment Card Industry) no almacenando ningún dato bancario de los usuarios, gestionándolo todo a través de una API.  
  
Las **contraseñas** de los usuarios serán hasheadas adecuadamente y no se almacenará la contraseña en texto plano o cifrada.

**MANTENIBILIDAD**Los administradores se encargan de que el **servidor no se sature**, eliminando los **datos no necesarios** y copias de seguridad obsoletas. Los administradores también son los encargados de reiniciar el sistema en caso de error.

**ACCESIBILIDAD**La aplicación debe permitir que múltiples usuarios se conecten **concurrentemente** al servidor y a la base de datos. Cualquier usuario con la **aplicación descargada e identificado** con conexión a Internet debería ser capaz de acceder a la misma.