



TRABAJO CUDECÁ



ANEXOS

- **Actores y Requisitos**
- **Extracción de casos de uso**
- **Sesión de desarrollo de modelado de clases**
- **Sesión Entidad-Relación**
- **Sesión Diagrama de secuencias**
- **Sesión Penpot**
- **Fotos adjuntas**



ACTORES

- **CRM:** conexión con la base de datos para la web de Cudeca, la cual, almacena información de los usuarios.
- **Usuario:** persona que compra entradas para los eventos organizados por Cudeca.
- **Servidor de Cudeca:** Modela los servidores encargados del despliegue y de la gestión de la aplicación de compra de entradas que estamos desarrollando. Inicialmente se planteó dividir al actor en 2, Servidor Cudeca y Servidor email. Prácticamente todos los casos de usos relacionados con este actor representan una tarea conjunta de ambos actores, motivo por el que hemos decidido unificarlos para mayor claridad.
- **Pasarela de pago:** se encarga de traspasar los pagos de los usuarios.
- **Admin Cudeca:** persona que se encarga de organizar y administrar los eventos, encontrar una ubicación, asignar precios a las entradas, etc.



REQUISITOS FUNCIONALES Y NO FUNCIONALES

REQUISITOS FUNCIONALES

- RF1. El sistema debe ofrecer la venta de un número ilimitado de entradas para distintos tipos de eventos. [pg.3 §2-3; pg.14 §1]
- RF2. En el sistema se deben poder configurar distintos tipos de eventos (loterías, conciertos, cenas, etc). [pg.3 §2-3]
- RF3. El sistema debe poder recopilar los datos de los compradores de eventos de forma voluntaria, para poder generar un certificado de donaciones. [pg.3 §4]
- RF4. La aplicación debe poder utilizar los datos de usuarios registrados anteriormente para que estos no se tengan que volver a llenar en cada compra. []
- RF5. El sistema ha de informar a los usuarios, que hayan previamente aceptado el envío de correos de Cudeca, de eventos próximos y otras notificaciones. [pg.4 §1]
- RF6. La aplicación debe permitir el uso de múltiples métodos de pagos para la compra de entradas. [pg.2 §4]
- RF7. El sistema debe soportar la cancelación de eventos por parte de los organizadores del mismo y que gestione la devolución de los pagos de todas las entradas. [pg.7 §1]
- RF8. Cada evento puede incluir una meta de recaudación, actualizando el total recaudado tanto automáticamente por la venta de entradas como de forma manual por los organizadores del evento. [pg.9 §2]
- RF9. El sistema debe permitir la distinción de distintos tipos de entradas para un mismo evento (correspondientes a distintas localidades en un concierto por ejemplo), con distintos precios. [pg.15 §2]



- RF10. El usuario debe poder pagar un extra a la hora de comprar la entrada a modo de donación. [pg.9 §4]
- RF11. Las entradas deben poder identificarse con un código QR que pueda permitir la gestión de entrada al evento el día de su celebración. [pg.9 §7]
- RF12. El sistema debe enviar recordatorios a los compradores una vez se acerque la fecha de celebración del evento, preferiblemente por SMS. [pg.16 §5; pg.17 §1]
- RF13. La aplicación contará con una interfaz simplificada para facilitar las gestiones de compra de entradas para personas con poca experiencia digital. [pg.11 §4]
- RF14. La aplicación se incorporará en la ya existente página web de Cudeca. [pg.18 §6]
- RF15. El sistema debe conectarse con el CRM ya existente de Cudeca. [pg.11 §6]
- RF16. En la aplicación se deben poder consultar eventos pasados. [pg.14 §1]
- RF17. Los nombres y descripciones de los eventos deben traducirse al inglés automáticamente. [pg.13 §4]
- RF18. La aplicación debe permitir múltiples métodos de pago para la compra de entradas. [pg.2 §4]

REQUISITOS NO FUNCIONALES

- RNF1. El sistema debe cumplir la ley de protección de datos. [pg.3 §2]

EXTRACCIÓN CASOS DE USO



Posteriormente, mostraremos los casos de uso que hemos usado en nuestra aplicación:

- **CU01 (Usuario)←Alta usuario:** el usuario se da de alta en la web de Cudeca como socio. Sus datos se almacenan en su base de datos.
- **CU02 (Usuario)←Seleccionar evento:** los usuarios seleccionan el evento al que quieran asistir, para obtener información del evento y poder pasar a la fase de compra.
- **CU03 (Usuario)←Comprar entrada:** los usuarios pueden comprar todas las entradas que quieran para los eventos de la fundación.
- **CU04 (Usuario)←Elegir tipo de entrada:** cada evento podrá tener distintos tipos de entradas, con distintos precios y beneficios. El usuario podrá elegir el tipo que más conveniente le parezca.
- **CU05 (Usuario)←Rellenar datos:** el usuario debe llenar sus datos para poder comprar entradas.
- **CU06 (Usuario)←Rellenar datos adicionales:** el comprador podrá llenar algunos datos adicionales, con la posibilidad de registrarse en la página web de Cudeca.
- **CU07 (Usuario)←Donar extra:** además de comprar las entradas, los usuarios tienen la posibilidad de donar un dinero extra.
- **CU08 (Servidor Cudeca)←Enviar entrada:** el servidor de Cudeca se encarga de enviar la entrada al correo electrónico del usuario.
- **CU09 (Servidor Cudeca)←Recordatorio evento:** Cudeca enviará un recordatorio días antes de la celebración del evento a los asistentes.
- **CU10 (Servidor Cudeca)←Enviar Spam:** si el usuario está dado de alta, este podrá ser informado de próximos eventos similares.



- **CU11** (Administrador Cudeca) ← Crear evento: el administrador Cudeca configura los nuevos eventos que van celebrarse y prepara su entrada para que los usuarios puedan comprar entradas
- **CU12** (CRM) ← Almacenar datos entrada: los datos del usuario se almacenan en la base de datos de Cudeca.
- **CU13** (CRM) ← Almacenar información eventos: la base de datos de Cudeca almacena la información relativa a los eventos para futuras consultas.
- **CU14** (CRM) ← Generar certificados: la base de datos de Cudeca se encarga de generar y enviar los certificados de donación a los usuarios donantes.
- **CU15** (Pasarela de pago) ← Realizar pago: para comprar las entradas el usuario deberá realizar el pago mediante la pasarela de pago.

En esta sección explicaremos nuestras discusiones y decisiones sobre los aspectos de nuestro diagrama.

El proceso de comprar una entrada (CU04) conlleva diversas acciones. En primer lugar el usuario deberá haber seleccionado el evento del cual quiere comprar una entrada (CU02), justificando la relación de dependency. Una vez seleccionado el evento, se deberán llenar los datos del usuario (CU05) admitiendo la posibilidad de llenar datos adicionales (CU06). El usuario ha podido previamente darse de alta en nuestra página de venta de tickets (CU01), lo cual acelerará el proceso de llenar los datos. Estos datos se enviarán a la base de datos de Cudeca, su CRM, el cual almacenará los datos (CU12) y, si se disponen de los datos suficientes podrá generar un certificado de donaciones (CU14).

No podemos olvidar la realización del pago (CU15), con la que el usuario se relaciona con la pasarela de pago. Existe la posibilidad de pagar una cantidad superior al precio base de la entrada (CU07), lo cual se trata de una acción completamente opcional por lo que la representamos con una relación de extends del caso de uso base. Finalmente, al usuario se le deberá enviar la entrada que ha comprado (CU08), de lo cual se encarga el actor Servidor Cudeca. Como podemos apreciar, prácticamente todos los casos de uso descritos guardan una relación de include con el caso de uso de comprar, puesto



que consideramos que este es el uso principal de un usuario y hemos modelado el resto de situaciones como parte de ese proceso.

En lo relativo al Servidor Cudeca, en el diagrama mostramos una visión bastante simplificada de las relaciones, de hecho la mayoría de sus casos de uso están simplemente relacionados con el actor (CU08, CU09, CU10). Todos estos casos de usos consisten en enviar algo, ya sea un recordatorio o algún tipo de publicidad, a un usuario, por lo que podríamos conectar en el diagrama estos casos de uso con el actor usuario. Sin embargo, hemos decidido obviarlo pues en esta situaciones el usuario es un elemento pasivo y no hace nada más que recibir y además complicaría visualmente el diagrama.

Otra cuestión que ya ha sido brevemente mencionada es la posible relación entre este actor y un hipotético actor Servidor email. El Servidor Cudeca no se encargará directamente de comunicarse con los usuarios, sino que creará los mensajes los cuales serán enviados por correo. Es decir cada uno de los casos de uso del actual Servidor Cudeca podría pasar de ser un envío a una creación del mensaje, y estos casos de uso se relacionarían con un actor Servidor email quien ya realiza el envío y tendría los casos de uso pertinentes. Decidimos no seguir este diseño para este diagrama pues, una vez más, complica innecesariamente las relaciones y el diagrama en su totalidad. Por eso motivo, hemos considerado que Servidor Cudeca incluye a al Servidor de email o de envío (pues quizás se envían estos datos por más de una vía), aunque para diagramos futuros y para la implementación de la aplicación será necesario deshacer esta abstracción.

La última peculiaridad a señalar sobre los casos de uso de este actor hace referencia al envío del spam o publicidad de la empresa (CU10). Para que el Servidor Cudeca pueda llevar a cabo este caso de uso, es necesario que el usuario al que va dirigido se haya previamente dado de alta (CU01). En lugar de crear una relación de dependencia entre ambos casos de uso, volviendo a nuestra filosofía de simplicidad en los diagramas, hemos decidido no relacionarlos explícitamente y en su lugar dejar una nota señalando esta precondition.

Para terminar comentaremos brevemente al Administrador Cudeca y al CRM, así como sus casos de uso. El administrador Cudeca tiene un único caso de uso sin ninguna relación peculiar, crear evento (CU11). Lo interesante de este caso es que, de forma similar a comprar entrada, guarda una relación de include con el caso Almacenar información evento (CU13) del CRM. El CRM en nuestro



EXTRACCIÓN MODELADOS DE CLASES

Una parte sustancial de la arquitectura y planificación del proyecto es el modelado de las clases y paquetes del sistema, en lo que se conoce como diagrama de clases. En este Diagrama quedan recogidas todas las clases (dentro del paradigma de la programación orientada a objetos) así como sus atributos, su funcionalidad y las relaciones entre ellas.

De esta forma conseguimos una documentación viva y muy visual que permite una evolución progresiva del proyecto gracias a su escalabilidad así como una forma sencilla de adaptar a nuevos integrantes del proyecto en éste, pudiendo entender de una vista rápida la estructura y funcionamiento del diseño.

Paquetes

En primer lugar, para facilitar la comprensión y el trabajo, decidimos dividir el proyecto en una serie de paquetes de clases, lógicamente independientes aunque luego puedan estar relacionadas. Así facilitamos el trabajo grupal y la separación de tareas, además de añadir una capa más de significado al diagrama.

Los principales paquetes detectados son:

- Usuarios: en este paquete está recogida toda la información acerca de los usuarios y clientes que visiten la página, incluidos aquellos que estén registrados o sean socios.
- Eventos: Recoge todos los tipos de eventos así como las diferentes entradas que podemos encontrar para cada uno de ellos.
- Tickets: Una especie de paquete intermedio entre usuarios y eventos, recoge la información de los tickets de entradas que los usuarios han comprado.



Clases y relaciones

Una vez identificados los paquetes en los que dividiremos la aplicación, definimos su composición interna, es decir, las clases que los componen.

Paquete Usuario:

Dentro de este paquete definimos los tipos de usuario que existen así como los atributos y funciones que debe tener cada uno.

En primer lugar, siempre es buena práctica definir una interfaz y una clase abstracta que nos defina aquello que tienen en común todos los usuarios. Por ello creamos la interfaz *Usuario*, que usarán todas las clases que quieran implementar alguna relación con las clases que hereden de esta.

Para definir aquellos atributos que son comunes, creamos la clase *Usuario_Base*, que implementa la interfaz *Usuario* y contiene los atributos mínimos que debe tener un usuario.

Estos son *nombre*, de tipo string; *email*, de tipo string; *dni*, de tipo string y *spam*, de tipo bool. Dado que un usuario puede ser completamente definido por estos atributos (aquellos que no decidan llenar los datos adicionales [requisito funcional]), no se trata de una clase abstracta sino de una clase normal.

Por otra parte, necesitamos una forma de salvar la información de aquellos usuarios que sean socios o que estén registrados para poder recibir los certificados de donaciones ([requisito funcional]). Para ello creamos la clase *Usuario_Registrado*, que extiende a *Usuario_Base* y que además de los datos ya mencionados contendrá *direccion*, de tipo string; *telefono*, de tipo integer y *socio*, de tipo bool. Con estos atributos obtenemos los datos necesarios para generar los certificados de donaciones y podemos comprobar si un usuario es además socio, es decir que paga la cuota mensual de socio. Pese a existir la posibilidad de crear una clase adicional *Usuario_Socio* para expresar esta funcionalidad, como equipo hemos preferido mantener la simplicidad de una sola clase y definirlo a través del atributo *socio* dado que los usuarios socio no tienen atributos adicionales que los registrados; simplemente son socios. En el caso de que existiesen diferentes planes de socio se consideraría la opción de



crear dicha clase adicional puesto que habría que recoger dicho plan como un atributo extra de los socios.

Paquete Eventos:

Este es el paquete *core* de la aplicación, en el sentido de que es el más grande, técnico, importante y repleto de decisiones de desarrollo importantes. En éste se refleja la estructura de los eventos y tipos de eventos que existen, los tipos de entrada que están relacionadas con los eventos y en general cuestiones de manejo de eventos.

Empezamos definiendo primero la estructura de los eventos en sí mismos. Seguimos una estructura similar a la empleada con los usuarios, creando una interfaz *IEvento* en la que se recoge la funcionalidad más básica y la clase *Evento_abstracto* con los atributos genéricos, que son *nombre*, de tipo string; *recaudacion*, de tipo integer que representa la cantidad recaudada y *objetivo_recaudacion*, de tipo integer que representa el objetivo a conseguir. También incluirá una lista de patrocinadores ([requisito funcional]) y de Tags para categorizar los eventos. Estos serán usados de cara a enviar publicidad específica a usuarios que han asistido a ciertos eventos ([requisito funcional]). Estos tipos pueden ser tanto clases con atributos de tipo string o pueden ser enums para simplificar su uso. Cabe destacar que, a diferencia de en el caso de la clase *Usuario_Base*, hablamos de una clase por definición abstracta, puesto que no existe ningún evento que sea suficiente por sí mismo con estos atributos.

Tras haber definido la raíz, creamos 3 clases diferentes, una para cada tipo de evento:

- [Evento_Concierto](#): Clase creada para definir los eventos de tipo concierto, en estos existen los atributos específicos de *artista*, de tipo string; *ubicación*, de tipo string y *fecha*, de tipo string o Date.

- [Evento_Carrera](#): Con esta clase se recoge la información de aquellos eventos de tipo carrera, cuyos atributos son *ubicación*, de tipo string y *recorrido* de tipo integer.

- [Evento_Rifa](#): Los eventos rifa son eventos un tanto diferentes, puesto que a diferencia de los demás no son presenciales. Estos eventos son un sorteo por lo que sus atributos son *premios*, de tipo lista de premios que pueden ser económicos o de otro tipo (se puede definir con posterioridad puesto que no

tiene ninguna otra relación con el resto del sistema) y *fecha*, de tipo string o Date.



Esta estructura de clases, como se verá más adelante, se va a repetir en reiteradas ocasiones. Hemos tomado la decisión de seguir este tipo de jerarquía debido a varios motivos. En primer lugar nos permite gran escalabilidad y adaptabilidad de cara a añadir nuevos formatos de evento, creando simplemente una nueva clase *Evento_TipoDeEvento*. Gracias a la interfaz *IEvento* no es necesario cambiar el código del resto de secciones del programa.

Además de esta ventaja de compatibilidad, es una arquitectura muy simple e intuitiva, fácil de entender para nuevos desarrolladores del equipo y lo suficientemente compleja como para cumplir todos los requisitos del cliente e incluso posibles futuros requisitos (nuevos eventos de otro tipo, que requieran otros atributos diferentes o nuevas funcionalidades).

Una vez definidos los eventos vamos a definir los tipos de entrada. El concepto de entrada surge del debate alrededor de la necesidad del cliente de disponer de distintos precios para distintas zonas de un concierto ([requisito funcional]). Inicialmente parecía lógico indicar que el precio fuera un atributo de *Evento_Abstracto*, puesto que resulta natural hablar del precio de un evento. Sin embargo, el hecho de que existan diferentes precios para un mismo evento nos lleva a necesitar definir una arquitectura que no solo soporte la idea inicial sino que además pueda llevar a cabo esta nueva necesidad. Para ello creamos la clase entrada. Esencialmente cada *IEvento* va a tener una serie de tipos de entrada diferentes (por ejemplo entrada primera fila, segunda fila; en una rifa puede haber distintos boletos con distinto precio y posibilidades de ganar un premio; dentro de una cena puede haber diferentes entradas para diferentes menús). Estos tipos de entradas van a estar organizados en la jerarquía de clases de entradas que describimos a continuación.

La estructura es exactamente igual que la comentada para los eventos: una interfaz *IEntrada* con la funcionalidad más básica y una clase abstracta *Entrada_Abstracta*. Esta clase va a tener los atributos *precio*, de tipo integer; el atributo *sub_aforo*, de tipo integer, para poder definir cuántas entradas de ese tipo se pueden vender y *evento*, de tipo *IEvento*, que indica el evento al que da acceso la entrada. Es importante notar que en el desarrollo se deberá tener la precaución de que la suma de los subaforos de los tipos de entrada sea igual al aforo del evento al que hacen referencia. Otra posible solución sería eliminar el



atributo *aforo* de la clase *Evento_Abstracto*, sin embargo hemos tomado la decisión de mantenerlo y tener las precauciones pertinentes por simplicidad.

En el árbol jerárquico aparecen entonces las clases *Entrada_Concierto*, *Entrada_Carrera* y *Entrada_Rifa*. En cuanto a las entradas de concierto, el único atributo adicional es *fila*, de tipo integer que indica la fila del concierto correspondiente a ese precio para el evento correspondiente. Las otras dos clases definidas están vacías. Si bien antes hemos comentado el caso del usuario socio, cuya clase preferimos no implementar dado que no poseía atributos específicos; para esta ocasión con problemática similar hemos tomado la decisión contraria y hemos mantenido las clases. Esto es así más que nada por un valor semántico y arquitectónico de la aplicación, de cara a mantener la congruencia con la jerarquía de los eventos y de los tickets que veremos con posterioridad. Además de ello, consideramos que es mucho más probable que eventualmente se requieran atributos específicos para entradas de tipo rifa o carrera, como los comentados anteriormente para permitir boletos con más posibilidades de ganar o posiciones adelantadas en la carrera.

Paquete Tickets:

Finalmente encontramos el paquete referente a los tickets. Como hemos comentado con anterioridad, este es el nexo que une los paquetes de usuarios y eventos, define lo que comúnmente llamamos entrada a un evento (el nombre entrada ya ha sido usado para definir los tipos de entrada de un evento, de ahí la elección de este otro nombre). Estos tickets recogen la información que es enviada al usuario para confirmar la compra de la entrada y permitir el acceso al evento.

Siguiendo el estilo estructural previo, tenemos una interfaz *ITicket* y una clase genérica *Ticket* con los atributos comunes de todos los tipos de ticket, a entender *id*, tipo integer, es un identificador (potencialmente convertible a qr); *comprador*, de tipo *IUsuario*; *evento*, de tipo *IEvento*; *nombre_beneficiario*, de tipo string y *dni_beneficiario*, de tipo string. Estos dos últimos atributos fueron una fuente de argumentación dentro del equipo.

Originalmente, se planeó la clase *Ticket* sin contar con estos atributos, suponiendo que el atributo de tipo *IUsuario* contendría la información del asistente del evento. Sin embargo releyendo la transcripción, vimos que el cliente propone la idea de que una persona pueda comprar varias entradas y poner los datos de otras personas en ellas (ejemplo típico de cena de empresa o



carrera de empresa en la que el jefe compra las entradas pero pone los datos de los empleados que asisten ([requisito funcional]). Por ello decidimos usar un esquema en el que se diferencia comprador y beneficiario, en el que el comprador es quien recibe la información de la donación pero puede existir un beneficiario ajeno, identificado con nombre y dni, que es el asistente al evento.

Se llegó a plantear la idea de que el beneficiario fuese también de tipo *IUsuario*, puesto que muchas veces comprador y beneficiario son la misma persona. Sin embargo consideramos innecesario que el beneficiario tenga que entregar el dato de correo electrónico (que es necesario para formar un usuario), por ello y para evitar duplicidad de los datos en el caso de que comprador y beneficiario son la misma persona, se indicará dejando *nombre_beneficiario* y *dni_beneficiario* como valores nulos o vacíos, de forma que se supondrá que los datos del beneficiario son los del comprador y se obtendrán los datos del atributo *comprador*.

Completando la jerarquía genérica, tenemos las clases *Ticket_Concierto*, *Ticket_Carrera* y *Ticket_Rifa*. *Ticket_Concierto* tiene el atributo *asiento*, de tipo integer; que indicará, en caso de existir varios asientos para ese tipo de entrada, cuál de ellos le corresponde.

Ticket_Carrera contiene el atributo *dorsal*, de tipo integer; que indica el dorsal que corresponde para la carrera. Finalmente *Ticket_Rifa* que contiene el identificador numérico *n_boleto* ([requisito funcional]) para el sorteo. Pese a poder usar el atributo *id* de la clase padre, preferimos incluir este atributo en la clase *Ticket_Rifa* para poder repetir números. Los identificadores de la clase padre deberán ser únicos, para poder ser guardados correctamente en la base de datos. Con este atributo adicional podemos permitir que haya dos tickets de rifa con el mismo número de dos eventos de rifa distintos. Será necesario entonces desde el punto de vista del programador asegurar que no haya duplicidad de asientos o números de boleto dentro de un mismo evento.

Otras arquitecturas planteadas

Durante el desarrollo del modelo, se propuso otra posible arquitectura que creemos que merece la pena comentar. El planteamiento venía a sugerir que dentro de nuestra clase *Evento_Abstracto* hubiese una lista con todas las entradas del evento individuales, de forma que las clases de *IEntrada* no representarían tipos de entrada sino entradas en sí mismas. Además de esta



forma la clase Ticket no tendría por qué incluir la compleja jerarquía y se limitaría a indicar la entrada a la que hace referencia y el usuario que la compra.

Esta idea se descartó por el simple motivo de que existen eventos que pueden tener entradas “infinitas”, o por lo menos no determinadas. Ello implicaría una indeterminación en el tamaño de la lista de entradas de la clase representante del evento que se debería resolver con un tamaño arbitrario. Además sería necesario alojar espacio para todas y cada una de las entradas del evento, hayan sido o no compradas, cada vez que se quiera crear el objeto de tipo *Evento_Abstracto*. Por ello optamos por esta otra implementación que, aunque es más compleja y requiere de más cuidado en la implementación por parte del programador, es más óptima y legible.

SESIÓN ENTIDAD-RELACIÓN



Entidades y Atributos

Usuario:

Usuario es una de las entidades mas importantes de nuestro diagrama, este tiene la función de representar el usuario que utilizará la plataforma y es aquella entidad la cual guardara los datos mas importantes de los usuarios. Es primordial que esta exista, ya que cualquier transacción parte de un usuario y como ahora veremos sus atributos son necesarios para la correcta gestión y comunicación con estos.

Como clave primaria tiene el atributo DNI el cual es único para cada usuario y funcionara como el identificador. También contaremos con otros atributos como nombre y apellidos del usuario, los cuales son obligatorios pero no únicos, ya que puede existir diferentes personas con mismo nombre y/o apellidos. Por último, dispone de los atributos email y SPAM, el cual también sera obligatorio y único ya que cada correo es diferente para cada persona. Aclaramos que SPAM se trata de un flag booleano. Estos atributos son necesarios para poder mandarles correos, ya sean recordatorios o anuncios de eventos similares. Como ahora veremos, un usuario debe tener la capacidad de registrarse y de comprar un ticket, por tanto necesitaremos otras dos entidades las cuales estarán relacionadas con usuario.

Registrado:

Registrado es una subentidad de usuario. Esta entidad representa a los usuarios que se registran formalmente en la plataforma, la cual proporcionan datos adicionales. Esta entidad es necesaria ya que los usuarios tienen la capacidad de registrarse, función que no es obligatoria.

Al principio tuvimos ciertas dudas de como representar a los usuarios registrados y de si crear un usuario “empresa” para la compra de entradas de las empresas para sus empleados. Al final decidimos implementar los usuarios registrados como una subentidad de usuario. Esto se debe a que llegamos a la conclusión de que es simplemente un usuario, pero con datos adicionales y registrado. El tema de la compra de ticket como empresa se explicara más adelante, concretamente en la entidad “ticket”. Sus únicos atributos son



teléfono y dirección postal los cuales simplemente guardan información adicional del usuario registrado.

Evento:

Evento otra entidad indispensable en nuestro diagrama. La entidad tiene como función representar los eventos disponibles en la plataforma. La entidad tiene gran importancia ya que tanto las entradas como los patrocinios giran en torno a los eventos. Los atributos de esta entidad permiten detallar la información necesaria para su gestión y publicación.

Como clave primaria tenemos a ID. Esta ID será único para cada evento y estará vinculado al evento. No se repetirán IDs. Para esta entidad quizás podríamos haber utilizado una clave primaria compuesta formada por el nombre del evento y el lugar por ejemplo, pero por simplicidad preferimos crear un ID único. Esta decisión también la tomaremos en otras entidades. Los demás atributos de esta entidad simplemente añadirán información necesaria de dicho evento. Como atributos contamos con nombre, el cual será simplemente el nombre del evento. La fecha y el lugar, que son necesarios para saber cuando se realizará el evento. Otro atributo que hemos añadido se llama tipo el cual es un identificador numérico que sirve para poder identificar los diferentes tipos de eventos como carreras, conciertos, rifas etc. Esta decisión se ha tomado por encima de crear subentidades debido a que no creemos que hayan suficiente información que almacenar para los diferentes tipos de eventos. No obstante, a nivel lógico de clases esto sí que se tendrá en cuenta. Y por último tenemos la descripción el cual tendrá una breve información adicional relevante sobre el evento.

Un evento tiene unas etiquetas, ofrecen unas entradas y puede tener o no, unos patrocinadores. Por tanto necesitaremos una entidad para cada uno de dichos aspectos

.

Patrocinador:

Como hemos mencionado, necesitaremos una entidad para poder representar a los patrocinadores. Los patrocinadores serán aquellos que patrocinan un evento. Es necesario ya que necesitamos establecer una relación entre los eventos y los patrocinadores.



Como atributos tenemos la clave primaria ID que, igual que los eventos, cada patrocinador tendrá una y sera única, y como el otro atributo tendremos el nombre de dicho patrocinador.

Etiquetas:

Esta sera una entidad la cual solo estará relacionada con evento. Cada evento esta asociada con una etiqueta. Esta etiqueta sera necesaria para dar información adicional del evento (saber que tipo de evento estamos acudiendo). También es útil ya que necesitaremos saber que tipo de eventos acude cada usuario para poder mostrarle a estos, evento similares. Sin la etiquetas, no podríamos aplicar esta funcionalidad.

La clave primaria sera ID que sera un único para cada etiqueta. Y como otro atributo, el nombre de

cada etiqueta.

Entrada:

Esta entidad es una entidad débil de evento, ya que no puede existir sin estar asociada a un evento. Representa los diferentes tipos de entradas a eventos. Es necesaria ya que no todos los eventos tiene el mismo tipo de entrada y por lo tanto es necesaria para poder tener controlado y asociado la entrada con el evento, y como veremos, con el ticket que dispondrá el usuario.

Cada evento singular puede tener distintos tipos de entradas. Por ejemplo un concierto puede tener entradas a distinto precio en función de la localidad. Distinguir estos tipos es el objetivo fundamental de esta entidad.

Cada diferente tipo de evento tendrá su tipo de entrada, la cual estará identificada por su clave primaria, ID, la cual asignara a cada tipo de entrada un identificador único. Dispondrá también de la clave foránea ID de evento. Esto se debe a que entrada es entidad débil de evento y necesitaremos saber a que evento pertenece la entrada. El atributo información sirve para concretar la información sobre el tipo de entrada. Esta información variara según el tipo de evento al que se acuda, siendo este diferente para un concierto, rifa, carrera etc. Por simplicidad hemos decidido no crear subentidades para los diferentes tipos de entrada, ya que nos parece mas simple y cómodo. Esto si que se tendrá en cuenta en el modelado lógico. Y como atributo final tenemos el precio, el cual guardara la información del precio de la entrada.



Cada tipo de entrada estará asociada a un ticket que es aquel que comprara y dispondrá el usuario. Por lo tanto, entrada tendrá que estar relacionada con ticket.

Ticket:

Ticket es otra entidad indispensable en nuestro diagrama. Esta entidad tiene la función de llevar el control de las compras realizadas por los usuarios y sus atributos son necesarios para controlar y personalizar correctamente cada ticket.

Ticket es una entidad débil de usuario y de entrada ya que este no puede existir sin un usuario y sin el tipo de entrada. Realmente se trata de una relación muchos a muchos con atributos.

Esta entidad nace de la necesidad de necesitar algo que relacione los tipos de entrada con el usuario. A su vez esta entidad nos generaba un debate. ¿Las entradas son intransferibles?. Si la respuesta es sí, entonces, ¿qué pasa con las empresas que desean comprar muchas entradas para sus trabajadores? La solución a esto la resolvimos implementando un atributo DNI_beneficiario el cual, es opcional y se explicara en el párrafo de los atributos.

Su clave primaria será id, la cual será única para cada ticket. Esta entidad contará con dos claves foráneas, tanto la clave principal de usuario que servirá para identificar el comprador (el cual será el que recibiría el certificado de donación en caso de proporcionar sus datos), que será el usuario y tanto la clave foránea de entrada, la cual identificara el tipo de ticket que estamos comprando. Como atributo opcional contamos con DNI_beneficiario. Este atributo esta puesto como opcional ya que contamos con que hayan empresas que compren entradas en masa ya sean para sortearlas, o regalárselas a sus empleados, y por lo tanto, serán transferibles y no nominales. En cambio habrán usuario las cuales solo compren entradas para ellos mismos u otras personas identificadas, las cuales sí serán nominales y estarán registradas con este atributo. Cabe la opción de que el atributo de la clave foránea del usuario como DNI_beneficiario sea el mismo ya que aquel usuario que combre una entrada y se ponga como beneficiario a si mismo, tendrá el mismo DNI.

También hemos añadido un atributo pago extra para reflejar la cantidad que un usuario puede llegar a pagar adicionalmente al comprar una entrada, la cual se reflejaría después en el certificado de dicho usuario como una donación. Por ultimo también hemos añadido un atributo información análogo al atributo



información de la entidad entrada, la cual especificara información adicional en base al tipo de entrada. La justificación de este atributo es prácticamente la misma que en Entrada Con esto finalizamos la explicación de cada entidad y sus atributos.

Relaciones

Ahora pasaremos a explicar las relaciones entre las diferentes entidades.

Usuario 1 – N Ticket: Un usuario esta relacionado con ticket ya que el usuario es aquel que compra un ticket y el ticket es de un usuario. Un usuario no esta obligado a comprar un ticket, por tanto, es opcional pero tiene la capacidad de comprar mas de un ticket. En cambio, si existe un ticket, este debe haber sido comprado por un usuario, por tanto la relación es obligatorio y un ticket solo puede ser comprado por un usuario.

Ticket N -1 Entrada: Un ticket solo puede ser un tipo de entrada, y es obligatorio por lógica, de que debe existir un tipo de entrada para tu ticket. Por otro lado, pueden existir muchos tickets que sean de un mismo tipo de entrada pero no es obligatorio que exista, ya que puede ocurrir de que nadie compre un tipo específico de entradas.

Evento 1 – N Entrada: Un evento siempre dispondrá de al menos un tipo de entrada lo cual es obligatorio. Una entrada solo estará relacionada con un evento, y esta relación tiene que ser obligatoria.

Evento N – N Patrocinador: Un evento puede o no tener uno o varios patrocinadores. Un patrocinador puede o no estar patrocinando uno o varios eventos. Es una relación de muchos a muchos opcionales

Evento N – N Etiquetas: Un evento puede tener una o más etiquetas, las cuales no son obligatorias. Una etiqueta puede estar relacionada con uno o mas eventos. A su vez, si existe una etiqueta esta tiene que tener un evento asociado por lo cual seria opcional la relación.



SESIÓN DIAGRAMAS DE SECUENCIAS

Los diagramas de secuencia resultan auto explicativos, ya que no son más que una especificación y desarrollo adicional de los casos de uso ya expuestos en su correspondiente diagrama. En este documento nos centraremos en tratar las cuestiones que puedan generar dudas.

Cabe destacar que los nombres de las funciones utilizados así como sus parámetros no son más que indicativos, para que el lector pueda entender qué información se pasa, aproximadamente, con cada mensaje. No hemos entendido los diagramas de secuencia como documentación de la implementación, sino como documentación de un nivel más abstracto que nos permita entender de un vistazo rápido las comunicaciones entre los distintos actores.

Hemos realizado un total de 6 diagramas con los cuales cubrimos la totalidad de los casos de usos. Para cada uno de ellos, detallaremos los actores involucrados, los casos de uso que cubren y los mensajes que se usan.

COMPRAR ENTRADA:

De todos los diagramas de secuencia, este es en el que se envían más mensajes y se involucran más actores, debido a que, tal y como explicamos en su respectivo documento, hemos modelado la mayoría de acciones como partes del caso de uso de comprar entrada (CU03). En este diagrama intervienen los actores Usuario, Servidor Cudeca, CRM y Pasarela de Pago. Además, existen un par de actores adicionales que no están presentes en el diagrama de casos de uso

Por un lado, tenemos al actor InterfazWeb, el cual los usamos como representación de la página web, el frontend, de nuestra aplicación. Sigue siendo un objeto algo abstracto pues los detalles de la implementación aún no se han decidido. Por otro lado tenemos el actor Servidor email, el cual también planteamos cuando discutimos qué actores incluir en el diagrama de casos de usos. En este primer diagrama unificamos a Servidor Cudeca y Servido Email,



pero ahora que estamos estudiando cada mensaje es necesario realizar esta distinción.

En primer lugar el usuario selecciona el evento al que quiere asistir (CU02). Esta acción podría haberse separado como un diagrama independiente pero vimos más conveniente su inclusión en este caso. Posteriormente se elige el tipo de entrada y se rellenan los datos del usuario. De esta manera se cubren los casos de uso CU04, CU05 Y CU06, así como el caso de uso CU07, donar extra, el cual no aparece mencionado explícitamente en este diagrama pero que agrupamos junto con llenar los datos pues asumimos que la cantidad adicional a donar se trata de un dato más a llenar. Todos estos mensajes siguen la misma estructura de envío de mensaje Usuario->InterfazWeb->Servidor Cudeca->CRM. En última instancia siempre consultamos al CRM pues es quien almacena los datos de los eventos.

A continuación se introducen los datos de pagos, los cuales llegan y son procesados por la Pasarela de Pago (CU15), y el Servidor Cudeca se encarga de almacenar esta nueva entrada en el CRM (CU12). Finalmente, a través del Servicio Email se envía la entrada al usuario (CU08).

En este diagrama hemos asumido que en efecto el usuario ha podido comprar la entrada que ha seleccionado pues no ha habido problemas de disponibilidad, y hemos devuelto true. Sin embargo, realmente ese mensaje daría una respuesta booleana que en caso de ser false devolvería al usuario al paso de seleccionar entrada para que elija otro tipo. A la hora de implementar el sistema, la interfaz en principio no permitiría al usuario elegir un tipo de entrada no disponible, pero aún así debemos considerar esta posibilidad. Aún considerándola, hemos decidido no plasmarla en el diagrama pues para ello aumentaríamos severamente la complejidad visual de un diagrama que ya es bastante completo de por sí.

CREAR EVENTO:

En este diagrama modelamos el proceso para que el Administrador Cudeca pueda crear un evento (CU11). Este se trata de un proceso sencillo en el que el administrador se comunica con la interfaz web, elige el tipo de evento que quiere crear y en función del tipo introduce los datos correspondientes, los cuales podrían ser distintos para cada tipo de evento por lo que hemos decidido utilizar un bloque alt para representar estas posibilidades.



Una vez recogidos los datos, InterfazWeb se comunica con Servidor Cudeca quien a su vez almacenará estos datos en el CRM (CU13).

Una peculiaridad del diagrama en su estado actual es que al terminar la creación del evento, no se manda ningún mensaje de vuelta al Administrador.

Inicialmente se planteó así debido a algunas limitaciones de representación del diagrama. Podríamos haber hecho que introducir_datos_[tipo_evento] fuesen mensajes síncronos pero habría una superposición en los bloques de activación. La otra opción hubiese sido plasmarlo como un mensaje asíncrono, lo cual tampoco sería especialmente fiel a la realidad. Finalmente decidimos dejarlo tal y como está y en lugar de mandar explícitamente un mensaje al administrador, simplemente se modificaría la interfaz web (que sí recibe un mensaje al finalizar el proceso) para reflejar la exitosa creación del evento.

DAR ALTA:

En este diagrama modelamos el proceso mediante el cual el usuario puede proporcionar sus credenciales para “darse de alta” o “iniciar sesión” (CU01). Este rellena sus datos a través de la interfaz web, la cual le cede los datos a Servidor Cudeca. Este actor consulta al CRM para saber si este usuario ya estaba presente en la base de datos o no. En caso de que fuese un usuario existente, se le devuelve sus datos a la interfaz, en caso contrario, se almacena al nuevo usuario en el CRM antes de que la interfaz pueda continuar.

Un pequeño detalle a aclarar es que, aunque en el diagrama cuando la respuesta del CRM a la existencia del usuario es positiva, el mensaje que se envía está usando una flecha de mensaje asíncrono. Realmente es la respuesta del mensaje síncrono, cuyo bloque de activación hemos tenido que extender a lo largo del bloque alt.

Este “inicio de sesión” está entrecerrillado pues realmente nuestra página de venta de tickets no tendría un sistema de usuarios como tal, sino que simplemente almacenaría los datos de los compradores (uno de los requisitos funcionales) y permitiría la importación rápida de los mismos para poder ofrecer mayor facilidad a la hora de llenar los datos a aquellos usuarios que ya han comprado entradas con anterioridad.

ENVIAR CERTIFICADOS:

Diagrama sencillo en el que el CRM genera los certificados de donación, proporciona la lista de certificados y los usuarios a los que les corresponden al



servidor Email, que se encargará de enviar dichos certificados a los respectivos usuarios (CU14).

Ambos mensajes son asíncronos pues no esperan respuesta y la temporalización entre ellos no ha de ser continua. En este diagrama asumimos que el CRM es quien genera los certificados aunque normalmente lo tratamos como una simple base de datos y que al servidor de email le podemos encolar el envío de múltiples mensajes.

ENVIAR SPAM:

Otro diagrama sencillo en el que Servidor Cudeca solicita al CRM la información acerca de todos aquellos usuarios que aceptaron recibir notificaciones sobre futuros eventos de la fundación Cudeca, para poder enviarles dicha información a través de Servidor Email (CU10). Una vez más, usamos mensajes asíncronos por los mismos motivos que en el diagrama anterior.

RECORDATORIO EVENTO:

Último diagrama, con el que desarrollamos cómo Servidor Cudeca envía recordatorios de eventos próximos (CU09) a aquellos usuarios que habían comprado entradas para dichos eventos. Esto se consigue mediante un par de consultas al CRM que proporcionan a Servidor Cudeca la información necesaria para delegar a Servidor Email el envío de recordatorios a los usuario pertinentes.



SESIÓN PENPOT

Para la realización del diseño se han seguido **los principios de Nielsen y Norman**, que se relatan en los siguientes puntos:

- Se cumple la visibilidad del sistema, mostrando en una página que cada vez que se pulsa un botón, se pasa al estado “cargando...”, visualizando el estado del sistema en el momento.

- La previsión de errores se consigue mediante distintas técnicas. Para empezar en cada lugar en el que el usuario debe introducir un “input” se expresa claramente lo que se debe introducir (Ej: “Introduzca su correo electrónico”) y además hay botones (tales como los de + y -) que expresan qué ocurrirá si se pulsa, evitando errores del usuario. Además en algunas casillas para llenar datos se escribe un ejemplo para demostrar lo que se tiene que introducir, por ejemplo si se tiene que introducir un nombre, se escribe “Abel Fernández Palomo”.

- Para cada página del diseño, se implementa una con mayor accesibilidad, que consiste en un tiempo de espera para comprobar que la persona que está llenando los datos no es capaz de hacerlo, tras lo que aparece un mensaje en la pantalla que te dice que pulses el botón de abajo a la izquierda si necesitas ayuda. Esta implementación se decidió así ya que el mensaje puede resultar molesto si sale cada vez que entras en la web, y de esta forma si una persona tiene problemas se puede solucionar sin molestar al resto de usuarios. Una vez se pulsa el botón de mayor accesibilidad los cambios principales son letras, casillas y botones más grandes, y además cuando se tienen que llenar datos se hace una interfaz mucho más sencilla en la que se muestra únicamente la casilla a



rellenar y lo que se debe introducir, para darle una mayor facilidad a personas que tienen problemas para llenar los datos.

En general, se trata de una interfaz sencilla y minimalista, con las opciones justas y además con un diseño agradable a la vista y muy fácil de entender.

Venta Entradas

Para comprar una entrada decidimos establecer una estructura con páginas desplegables, para que todo estuviese reunido en una misma zona de forma organizada y así se pueda acceder a cada paso de manera progresiva.

En primer lugar va ***el tipo de entrada*** donde se ven para cada evento sus respectivos tipos (menús si se trata de una cena, o una entrada VIP si fuese un concierto), junto a su precio individual y en caso de haber demasiadas entradas para ajustarse al tamaño de la ventana, se implementa una barra de desplazamiento. Justo a la derecha se puede seleccionar la cantidad de entradas que se quieren comprar de cada uno, poniendo un número o simplemente pulsando el símbolo “+” hasta llegar al número acordado. Justo debajo se habilita la opción de las donaciones que se pedía que se habilitase, hay un campo vacío en el que se introduce voluntariamente una cantidad y luego se debe pulsar el botón “REALIZAR DONACIÓN”. También se implementa más abajo un apartado en el que se puede observar las entradas seleccionadas con su importe total, para poder ver de forma detallada de dónde sale la suma total a pagar.

La siguiente pestaña desplegable trata de ***los datos del comprador***. Lo primero que aparece es una ventana en la que te pregunta si quieres llenar los datos manualmente o introducir tu dni para buscarlos ya que, en caso de haber comprado entradas anteriormente, tus datos se guardan y no tienes que volver a meterlos en la página web. En ambos casos se lleva a la página de llenar datos, ya sea para introducirlos o para confirmar que están bien. Aquí también se implementa una barra de desplazamiento. También se implementan dos casillas de verificación para consentir el tratamiento de datos y además por si el usuario quiere hacerse socio de Cudeca.



La siguiente pestaña es la de **los datos del asistente**. Lo primero que se implementa aquí es un botón para llenar con los datos del comprador, en caso de que sean los mismos. Si este no es el caso, se habilitan varias casillas para llenar los datos de cada entrada de forma individual, introduciendo así la información del asistente.

Por último tenemos **la pasarela de pago**. Lo primero que hay que destacar es que una vez en comprar entrada se le da a pagar, se lleva a una página distinta, para llenar los datos que son más personales. Aquí se puede elegir el método de pago y a continuación se llenan los datos que sean necesarios (por ejemplo teléfono móvil para Bizum o email y contraseña para Paypal).

Eventos

A esta sección se accede desde el apartado “Eventos” existente en la página principal de Cudeca. Esta página está dividida en dos: **eventos pasados y eventos próximos**. Al entrar en cualquiera de ellas, se muestran los eventos de dos en dos ordenados verticalmente.

Para cada evento, mostramos una imagen representativa del evento, su nombre y, a su derecha, información básica sobre el evento. Primero, se muestra la ubicación donde ocurre el evento. Abajo, informamos sobre la fecha en la que empieza. Finalmente, mostramos dos campos llamados “*Comprar Ticket*” y “*Leer más*” (el inicial no se mostrará en eventos pasados). El primero llevará a la página anteriormente explicada en “*Venta entradas*”. Por otro lado, al pulsar el segundo aparecerá un pop-up donde, aparte de la información detallada anteriormente, se explica en más detalle el evento. En la parte baja del pop-up se ve, finalmente, una barra que indica la meta de donación del evento y el porcentaje donado.

Crear eventos

Para acceder a esta ventana, el administrador tiene una visión de la página única, donde en el apartado eventos se encuentra un desplegable en el que puede seleccionar si quiere ver los eventos o crear uno. Este diseño solo puede ser visualizado por gente que administre la página de *cudeca*.



Como en el apartado de “Ventas de entrada”, hemos aplicado una estructura de páginas desplegables para guiar al usuario paso a paso.

El primer apartado, “Seleccionar tipo de evento”, muestra los principales tipos de evento(carrera, cena, concierto) más el tipo “otro” para cualquier otro evento. Tras seleccionar el evento deseado, se desplegará el apartado “Datos Evento”. En este apartado se introducirán los datos: nombre del evento, localización, objetivo de recaudación y fecha del evento. Aparte, para cada tipo de evento se añadirá la información específica de cada evento(por ejemplo, en concierto se pedirá el nombre de los artistas que participan). Se podrá añadir como datos adicionales, descripción del evento y una imagen del mismo y se podrán especificar distintos tipos de entradas con distintos: precio, nombre y cantidad de tickets. Después, antes de introducir el evento a la base de datos, se desplegará el apartado “Verificar Evento” donde se verá un resumen con la información principal del evento, para poder asegurarse de que los datos sean correctos. Finalmente, al asegurarse de que todo está bien, el botón confirmar finalizará la operación.

FOTOS ADJUNTAS



