DP1 2020-2021

Documento de Diseño del Sistema

SPRINGFEST

Proyecto SpringFest

https://github.com/gii-is-DP1/dp1-2020-g3-11

Miembros:

- Bueno Gómez, Manuel
- Calvo Durán, Fernando
- González Boza, Enrique
- Manzano Dorado, Alejandro
- Rodríguez Santiago, Javier
- Santos Ortiz, Pablo

Tutor: Muller Cejas, Carlos

GRUPO G3-11

Historial de versiones

Fecha	Versión	Descripción de los cambios	Sprint
13/12/2020	V1	● Creación del documento	2
13/12/2020	V2.1	● Añadido diagrama de dominio/diseño ● Completar campos requeridos	3
09/01/2021	V2.2	● Completar apartados necesarios ● Revisión de diagramas ● Revisión general del documento para la entrega	3

Contents

Introducción

Aplicación web para gestionar festivales.

Servirá mayoritariamente para llevar la gestión de un festival asociado a su administrador, donde podrá asociar artistas, recintos, conciertos, entradas…. Los usuarios, tienen la posibilidad de comprar entradas y los Sponsors, alquilar puestos de venta de productos en recintos disponibles del festival.

La aplicación se dividirá en módulos comunes para todos los festivales, como la sección de compra de entradas de diferentes tipos, las ofertas aplicables a cada festival, poder ver cada cartel y los horarios de los conciertos, entre otras muchas cosas.

Se pretende facilitar la organización de cualquiera de los festivales en diversos aspectos, simplificando algunos que, de no ser vía online, serían más complicados, como la gestión de los puestos por parte de los dueños de estos o la posibilidad de poder elegir ofertas con antelación y no tener que hacer cola fisica para comprar alguno de estos paquetes, una vez llegados al festival.

Diagrama(s) UML:

Diagrama de Dominio/Diseño

Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)

Los diagramas se adjuntan en la carpeta “Diagramas” para facilitar la visualización del mismo y evitar errores de formato al pegarlos en el documento.

Patrones de diseño y arquitectónicos aplicados

Patrón: Modelo-Vista-Controlador

Tipo: Arquitectónico

Contexto de Aplicación

La aplicación está separada en capas, que son: presentación, lógica de negocio y recursos. Cada una se encarga de una tarea y ayuda a las otras.

Clases o paquetes creados

Para cada modelo, se ha creado un controlador y una o varias vistas según haga falta.

Ventajas alcanzadas al aplicar el patrón

Usar este patrón nos ha sido de gran utilidad, ya que hemos podido separar las responsabilidades en distintas capas. Además, ha facilitado el mantenimiento del código, ya que están separados los

lenguajes de programación y ha hecho que todo esté ordenado. Por último, también nos ha permitido reutilizar código, lo que ha agilizado la creación de la aplicación.

Decisiones de diseño

Decisión 1: Exceso de historias de usuario

Descripción del problema:

Nuestro grupo ha tenido claro que quería un 10 en esta asignatura, por lo que cuando pensábamos en posibles funcionalidades de la aplicación, las apuntábamos, para hacerlas más adelante, pero teniendo en cuenta el número exigido para sacar la máxima nota. Sin embargo, un día nos dimos cuenta de que habíamos realizado muchas más historias de usuario de las que se pedían.

Alternativas de solución evaluadas:

Alternativa 1.a: Realizar todas las historias de usuario apuntadas.

Ventajas:

- Aplicación mucho más completa y útil.

Inconvenientes:

- Mucha carga de trabajo innecesaria.

Alternativa 1.b: Dejar de hacer historias de usuario y quedarnos con las que teníamos hechas.

Ventajas:

- Ya estaría acabada la aplicación, solo a falta de revisar.

Inconvenientes:

- Actores sin funcionalidades y entidades con poca utilidad.

Alternativa 1.c: Realizar las historias de usuario indispensables para darle sentido a todos los actores y entidades.

Ventajas:

- Aplicación completa y útil.

Inconvenientes:

- Carga de trabajo extra.

Justificación de la solución adoptada

Seleccionamos la alternativa de diseño 1.c, ya que era necesario que tuvieran sentido todas las entidades y los actores y, por otro lado, hacíamos una aplicación útil sin una carga de trabajo excesiva.

Decisión 2: Relación entrada usuario

Descripción del problema:

No sabíamos al principio cómo abordar esta relación, ya que por un lado queríamos que un usuario solo pudiera comprar una entrada para un festival, para que no hubiera opción a reventa, ya que se tendría en cuenta su DNI, pero por otro, pensamos que si alguien se equivocase con su DNI al registrarse y comprase la entrada por otra persona, esta otra no podría comprar la entrada.

Alternativas de solución evaluadas:

Alternativa 2.a: Una única entrada por DNI en un festival

Ventajas:

- No habría opción a reventa.

Inconvenientes:

- No se podrían hacer regalos de entradas.
- Posibles errores en el registro de usuario.

Alternativa 2.b: Se pueden comprar todas las entradas que se quieran

Ventajas:

- No implementación de esa regla de negocio (menos trabajo).
- Evasión de posibles errores a la hora de registrarse.
- Se pueden regalar entradas.

Inconvenientes:

- Posible reventa de entrada

Justificación de la solución adoptada

Seleccionamos la alternativa de diseño 2.b, ya que tiene muchas más ventajas que desventajas y consideramos importante poder regalar entradas.

Decisión 3: Limitación de funcionalidades referentes a Festival

Descripción del problema:

En nuestra aplicación, un administrador es quien posee y gestiona un festival.

Según hemos avanzado con la implementación nos hemos dado cuenta que no era buena idea dejar que un usuario se pudiese registrar como administrador y, acto seguido, fuera este quien registrase su festival en la página para poder gestionarlo.

Alternativas de solución evaluadas:

Alternativa 3.a: Dejar el registro igual y no limitar el acceso a esta funcionalidad para el usuario con autoridad de administrador.

Ventajas:

- Mantener la idea principal y poder trabajar en otras historias no realizadas más importantes.

Inconvenientes:

- Sistema de acceso a la aplicación poco seguro y con poco sentido y lógica.

Alternativa 3.b: El registro de administrador y de su festival lo realizan los superAdministradores (equipo de trabajo del proyecto) desde el código fuente.

Ventajas:

- Dotar de sentido real a la traza de esta funcionalidad
- Más comodidad para implementar la entidad y sus funcionalidades correspondientes.

Inconvenientes:

- Gasto de tiempo en implementar funcionalidades obsoletas.

Justificación de la solución adoptada

Seleccionamos la alternativa de diseño 1.c, ya que era necesario que tuvieran sentido todas las entidades y los actores y, por otro lado, hacíamos una aplicación útil sin una carga de trabajo excesiva.

Decisión 4: Regla de negocio con conciertos

Descripción del problema:

Como grupo, queríamos introducir una regla de negocio que no permitiese crear dos conciertos en el mismo tramo horario y escenario. El problema es que nos resultó difícil maniobrar con los objetos DateTime de Java.

Alternativas de solución evaluadas:

Alternativa 4.a: Cambiar esta regla de negocio por otra que no fuese tan complicada.

Ventajas:

- No tener que lidiar con objetos DateTime y por ende, que la regla de negocio no fuese tan ardua.
- Tendríamos el mismo número de reglas de negocio que con la idea original.

Inconvenientes:

- Seguiríamos teniendo una regla de negocio por hacer.

Alternativa 4.b: Eliminar directamente esta regla de negocio debido a que ya contábamos con todas las necesarias para la nota deseada.

Ventajas:

- No tendríamos más reglas de negocio a implementar, por ende, menos trabajo.

Inconvenientes:

- Tendríamos una regla de negocio menos, por lo tanto, menos funcionalidades en nuestra página. De todas formas esto no sería un inconveniente real (al menos de cara a la asignatura), ya que disponíamos de las reglas de negocio necesarias para alcanzar la nota que queríamos.

Justificación de la solución adoptada

Seleccionamos la alternativa de diseño 4.b, ya que, como hemos mencionado, no nos importaba eliminar una regla de negocio para ahorrarnos trabajo porque disponíamos ya de las suficientes.