DP1 2020-2021

Documento de Diseño del Sistema

Proyecto QPLAN

[Enlace al repositorio de GitHub](https://github.com/gii-is-DP1/dp1-2020-g3-13.git)

Miembros:

* Baños González, Alejandro
* Flores Rodríguez, Román
* Gutiérrez Ceballos, Pablo
* Ibáñez Montero, Julia
* Meca Sánchez, Andrea
* Pérez Sosa, Ezequiel

Tutor: García Rodríguez, José María

GRUPO G3-13

Versión 1.0

03/01/2021

# Historial de versiones

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Versión | Descripción de los cambios | Sprint |
| 10/01/2021 | V1.0 | * Implementación del 70% de las HU * Confección de validadores * Implementación de 2 reglas de negocio * Restricciones simples * Modificación del CSS * Tests | 3 |

Contenido

[Historial de versiones 2](#_heading=h.gjdgxs)

[Introducción](#_heading=h.30j0zll) 3

[Diagrama(s) UML: 4](#_heading=h.1fob9te)

[Diagrama de Dominio/Diseño 4](#_heading=h.3znysh7)

[Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)](#_heading=h.2et92p0) 4

[Patrones de diseño y arquitectónicos aplicados](#_heading=h.tyjcwt) 4

[Decisiones de diseño](#_heading=h.3dy6vkm) 6

[Decisión 1 6](#_heading=h.1t3h5sf)

[Descripción del problema: 6](#_heading=h.4d34og8)

[Alternativas de solución evaluadas: 6](#_heading=h.2s8eyo1)

[Justificación de la solución adoptada 6](#_heading=h.17dp8vu)

[Decisión 2](#_heading=h.1t3h5sf) 7

[Descripción del problema:](#_heading=h.4d34og8) 7

[Alternativas de solución evaluadas:](#_heading=h.2s8eyo1) 7

[Justificación de la solución adoptada](#_heading=h.17dp8vu)

[Decisión 3](#_heading=h.1t3h5sf) 8

[Descripción del problema:](#_heading=h.4d34og8) 8

[Alternativas de solución evaluadas:](#_heading=h.2s8eyo1) 8

[Justificación de la solución adoptada](#_heading=h.17dp8vu)

[Decisión 4](#_heading=h.1t3h5sf) 9

[Descripción del problema:](#_heading=h.4d34og8) 9

[Alternativas de solución evaluadas:](#_heading=h.2s8eyo1) 9

[Justificación de la solución adoptada](#_heading=h.17dp8vu) 9

[Decisión 5](#_heading=h.1t3h5sf) 10

[Descripción del problema:](#_heading=h.4d34og8) 10

[Alternativas de solución evaluadas:](#_heading=h.2s8eyo1) 10

[Justificación de la solución adoptada](#_heading=h.17dp8vu) 10

# 

# 

# 

# 

# 

# Introducción

La idea del proyecto es hacer una página web desde la que se gestione la compra de entradas y organización de eventos culturales, así como proporcionar a aquel que visite la página toda la información de cada evento y su programación de una manera fácil e intuitiva.

Unas de las funcionalidades más destacadas es el sistema de peticiones, mediante el cual son los administradores del sistema los que deciden qué organizaciones pueden organizar eventos en su página, de esta manera se tiene un control también sobre los tipos de eventos que se pueden realizar a través de nuestra página. Esta historia de usuario se corresponde con: ¨Como usuario no registrado quiero hacer una petición para poder registrarme como organización¨.

Otra funcionalidad que consideramos interesante es la implementación de un carrito mediante el cual puedes ir almacenando las entradas que, como cliente, quieras comprar, tanto para el cliente registrado como para el asistente que él especifique. De esta manera el cliente podrá hacer un solo pago con todas las entradas que quiera en lugar de un pago cada vez que quiera comprar una entrada. Esto también es aplicable a los lugares de realización que quieran alquilar los organizadores de los eventos. La historia de usuario correspondiente sería:

“Como cliente quiero que las entradas se añadan a un carrito para poder finalizar mi compra con todas las entradas en un mismo pago.” y “Como organización quiero que los lugares que quiero alquilar se añadan a un carrito para poder finalizar mi compra con todos los lugares en un mismo pago.”

Con respecto al diseño cabe destacar el uso de JavaScript para el diseño de la página principal, de manera que se vean de forma intuitiva y rápida los futuros eventos disponibles pudiendo acceder a ellos de un solo vistazo.

# 

# Diagrama(s) UML:

## Diagrama de Dominio/Diseño

El diagrama estará en el siguiente enlace ya que por las dimensiones del mismo no se puede leer correctamente dentro de este documento.

Enlace: [Diagrama de Dominio/Diseño](https://drive.google.com/file/d/1EKbdYk9akRZcCFCcXFWvS2FOb_WVh_Rh/view?usp=sharing)

## Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)

El diagrama UML de clases que describe el conjunto de controladores, servicios, y repositorios implementados se adjunta en un archivo .jpg aparte [aquí](https://drive.google.com/file/d/1-ifoOYPsLpCbhyueY2rEgJZCA-4Zxwa2/view?usp=sharing) y en el zip ya que es excesivamente grande como para introducirlo aquí.

## Patrón: <Repository>

### Tipo: de Diseño

### Contexto de Aplicación

Para la implementación de las funcionalidades de tipo CRUD (create, read, update & delete), hemos implementado un repositorio heredado de la clase CRUDrepository, el cual nos dará las funcionalidades necesarias para guardar los cambios en la base de datos.

### Clases o paquetes creados

Para cada clase modelo hemos creado un repository, los cuales se crean dentro del paquete “repository”.

### Ventajas alcanzadas al aplicar el patrón

Este patrón nos ha permitido el acceso fácil a la base de datos y así poder tratar de manipular los datos de esta de una forma más sencilla.

## Patrón: <Service-Locator>

### Tipo: de Diseño

### Contexto de Aplicación

Para apoyar al patrón repositorio y poder crearle más funcionalidades a la hora de manipular los datos hemos creado unas clases service.

### Clases o paquetes creados

Para cada clase repositorio le hemos creado un service, dentro del paquete “service”.

### Ventajas alcanzadas al aplicar el patrón

Esto nos ha permitido un acceso más funcional a la base de datos, aportando nuevos métodos que necesitaban los acceso al repositorio.

## Patrón: <Arquitectura centrada en datos>

### Tipo: de Diseño

### Contexto de Aplicación

Para verse reflejados los cambios hemos creado un sql dónde están guardados de manera íntegra nuestros datos.

### Clases o paquetes creados

Hemos creado una clase “data.sql” la cual nos sirve de base de datos.

### Ventajas alcanzadas al aplicar el patrón

Esto nos ha permitido poder almacenar recursos y datos, y además poder acceder a ellos.

## Patrón: <Message Enricher>

### Tipo: de Diseño

### Contexto de Aplicación

Para mandar automáticamente un correo con el usuario y contraseña a las organizaciones cuyas peticiones han sido aceptadas.

### Clases o paquetes creados

Lo hemos usado en las clases de *Organización* y hemos creado *EnvioEmailService.*

### Ventajas alcanzadas al aplicar el patrón

Poder mandar automáticamente el email.

## Patrón: <Capas>

### Tipo: Arquitectónico

### Contexto de Aplicación

Para separar las funcionalidades y roles en service y model.

### Clases o paquetes creados

Hemos usado el patrón arquitectónico en capas en toda la página.

### Ventajas alcanzadas al aplicar el patrón

Mayor facilidad para organizar el código, para identificar errores y hacer bien las pruebas.

## Decisión 1: Creación de organización.

### Descripción del problema:

Como grupo nos gustaría poder crear organizaciones y validarlas.

Alternativas de solución evaluadas:

*Alternativa 1.a:* Enviar un pdf con los datos de usuario validados a la organización.

**Ventajas**:

* Se permite una validación más real.

**Inconvenientes**:

* Ralentiza el trabajo puesto que dependemos de métodos externos.
* Se han implementado métodos más complejos

*Alternativa 1.b:* Crear un validador para organización.

**Ventajas**

* Se permite una validación más rápida.

### **Inconvenientes**

* Se podrían crear demasiadas organizaciones falsas, puesto que podrían pasar los validadores.

### 

### Justificación de la solución adoptada

Como consideramos que la validación debe ser más fuerte y no queremos renunciar a un trabajo difícil durante el desarrollo de la aplicación, seleccionamos la alternativa de diseño 1.a.

## Decisión 2: Añadir lugar de realización al evento.

### Descripción del problema:

Como grupo nos gustaría poder dar a cada evento la opción de añadir un lugar de realización.

Alternativas de solución evaluadas:

*Alternativa 2.a:* Asociar a cada actividad del evento un lugar.

**Ventajas**:

* Se permite poder añadir varios lugares a un evento.
* Un mejor reparto y validación del aforo
* Funcionalidad más clara.

**Inconvenientes**:

* Habría que crear varias clases padres y puente.

*Alternativa 2.b:* Asociar a cada evento un lugar.

**Ventajas**

* El código para hacerlo sería más asequible

### **Inconvenientes**

* Solo podríamos asignar un lugar
* Mayor dificultad para el control de aforo,
* Menos óptimo.

### 

### Justificación de la solución adoptada

Como consideramos que el control de aforo y poder añadir varios lugares es importante, hemos decidido optar por la alternativa 2.a

## Decisión 3: Realización de pago entradas.

### Descripción del problema:

Como grupo nos gustaría poder implementar una pasarela que nos permita realizar pagos.

Alternativas de solución evaluadas:

*Alternativa 3.a:* Cada vez que se vaya a adquirir una entrada o lugar aparecería una pasarela de pago unitaria.

**Ventajas**:

* Menor dificultad de implementación.
* Pago seguro por acceso único a base de datos.

**Inconvenientes**:

* Poco funcional, incómodo para el usuario.

*Alternativa 2.b:* Añadir los elementos de pago a un carrito

**Ventajas**

* Más cómodo para el cliente.
* Más real.

### **Inconvenientes**

* Mayor dificultad de implementación.

### 

### Justificación de la solución adoptada

Como consideramos que el pago debe ser total hemos decidido optar por la alternativa 3.b

## Decisión 4: Tipo de entradas.

Descripción del problema:

Como grupo nos gustaría poder implementar la creación de diferentes tipos de entrada

Alternativas de solución evaluadas:

*Alternativa 4.a:* El tipo de entrada sería un string a modificar por la organización.

**Ventajas**:

* La organización podrá crear más tipos.

**Inconvenientes**:

* Poco real.
* Dificultad de implementación lógica.

*Alternativa 4.b:* El tipo de entrada sería un enum creado por nosotros

**Ventajas**

* Más fácil la implementación de las relaciones y sus métodos.
* Más real.

### **Inconvenientes**

* Mayor dificultad de implementación.

### 

### Justificación de la solución adoptada

Como consideramos que el tipo de entrada es más lógico que sea enum, hemos adoptado la solución 4b.

## Decisión 5: Autoridad de usuario.

### **Descripción del problema:** Como grupo queremos darle al usuario un tipo de autoridad (cliente, organización o admin)

**Alternativas de solución evaluadas:**

*Alternativa 5.a:* Implementación de autoridad mediante una List.

**Ventajas**:

* Creación más variada del tipo Autoridad.

**Inconvenientes**:

* Poco real.

*Alternativa 5.b:* El tipo de autoridad sería una relación creada.

**Ventajas**

* Asignación más lógica.
* Más real.

### **Inconvenientes**

* Elección predefinida.

### Justificación de la solución adoptada

Como consideramos que la autoridad esté elegida de forma predefinida, hemos adoptado la solución 5b.