DP1 2020-2021

Documento de Diseño del Sistema

Proyecto TEAcademy

URL repositorio: https://github.com/gii-is-DP1/dp1-2020-g1-01.git

Miembros en orden alfabético por apellidos:

- Alonso Martín, Fernando
- Álvarez García, Gonzalo
- Martínez Fernández, Javier
- Ramos Blanco, María Isabel
- Vilariño Mayo, Javier
- Yugsi Yugsi, Evelyn Gisele

Tutor: Manuel Resinas

GRUPO G1-1

Versión 1.0

08 de enero de 2021

Historial de versiones

Fecha	Versión	Descripción de los cambios	Sprint
26/12/2020	V1	• Creación del documento. • Añadido diagrama de dominio. • Añadidas decisiones de diseño.	3
08/01/2021	V2	• Añadidas decisiones de diseño. • Añadido diagrama de capas. • Terminados puntos restantes (introducción y patrones utilizados).	3
08/02/2021	V3	• Añadidas decisiones de diseño. • Modificación de diagramas de capas y de dominio.	4

Contents

Introducción

T.E.A es una academia de inglés sevillana, la cual se dedica tanto a la certificación de niveles oficiales de Cambridge, Trinity y Oxford, como a la realización de actividades extraescolares.

Se pretende que con la implementación del nuevo software se facilite y simplifique la gestión de alumnos, inscripciones, pagos, actividades extraescolares y otros aspectos de gestión. Por otra parte, se pretende que la disposición de material para los alumnos sea mucho más accesible y la visibilidad de la academia en cuanto a repercusión se aumente.

Tras la realización del proyecto, resaltamos la implementación de las historias relacionadas con el Wall of Fame, las del calendario y la de material ya que presentan un mayor aspecto visual y dificultad a la hora de desarrollarlo.

Diagrama(s) UML:

Diagrama de Dominio/Diseño

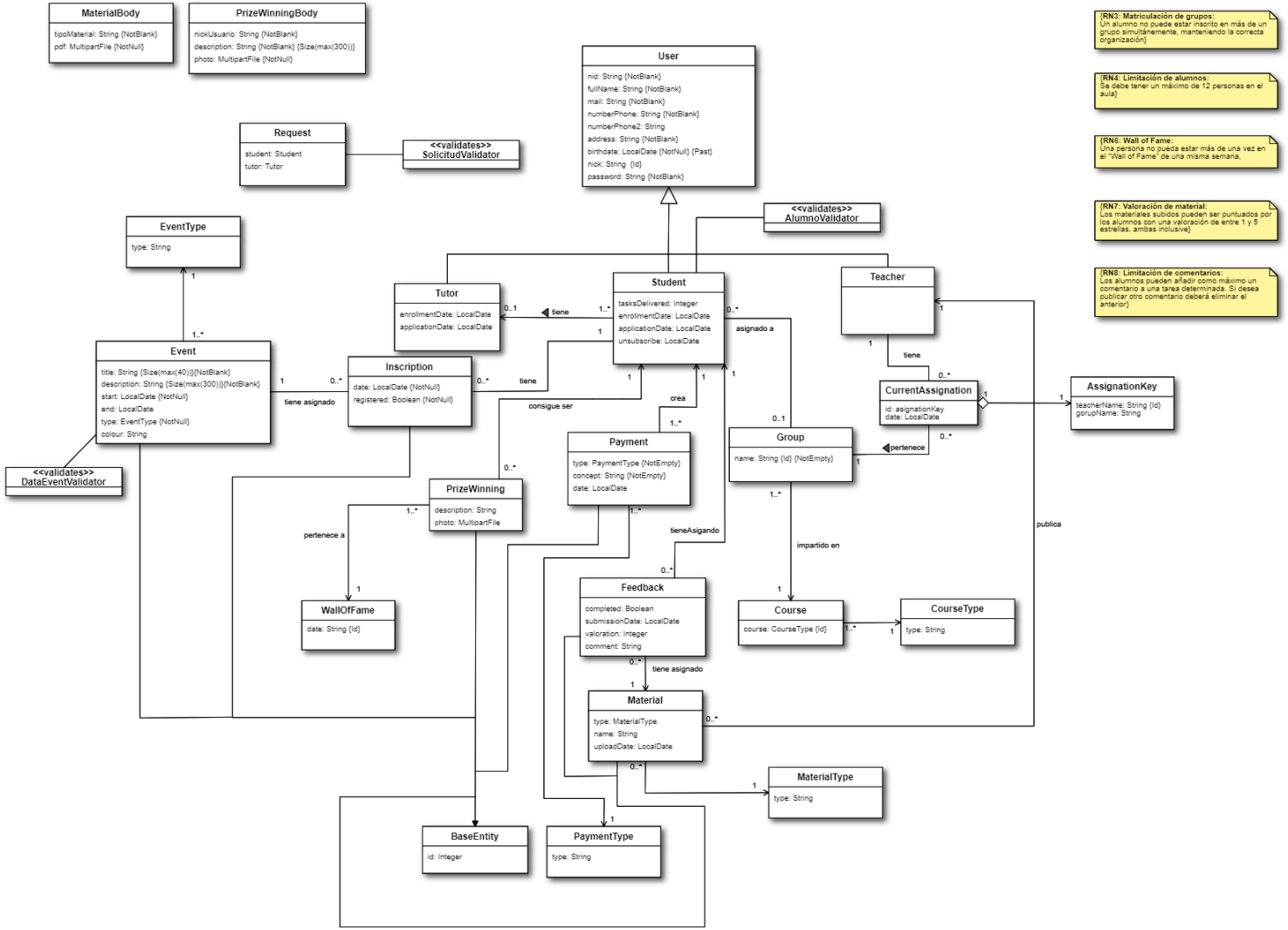
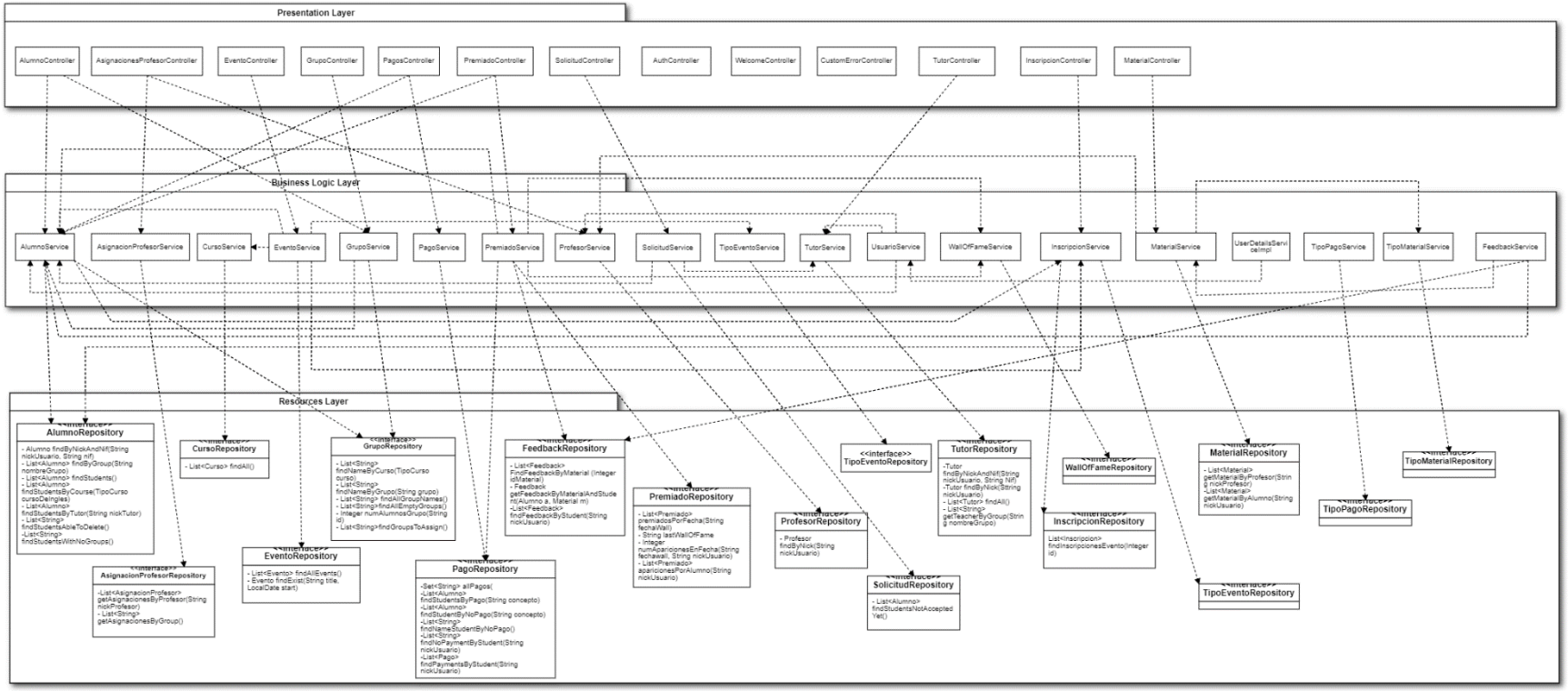


Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)



Patrones de diseño y arquitectónicos aplicados

Patrón: MVC

Tipo: Arquitectónico | de Diseño

Contexto de Aplicación

Hemos utilizado dicho patrón en toda nuestra aplicación, en concreto, las vistas y controladores se referirán a la capa de presentación del estilo arquitectónico por capas. Los servicios, las entidades de la parte del modelo del patrón se referirá a la capa de la lógica de negocio. Por último, los repositorios, también incluidos en el modelo MVC, se referirán a la capa de datos.

Para aplicar dicho patrón hemos utilizado frameworks de Spring y para las vistas hemos utilizado la tecnología de React, así como sus variantes, por ejemplo, Hooks, Redux, etc.

La interacción entre React y Spring es mediante llamadas desde las vistas en React a la API REST implementada en Spring, siendo React y Spring dos servidores distintos.

Clases o paquetes creados

Para poder interactuar con el frontend hemos decidido ubicar React concretamente en *dp1-2020-g1-01/src/main/resources/static/frontend*. Dentro de la carpeta *src/componentes* se encuentran los distintos componentes referentes a las vistas creados, necesarios para trabajar con React. Es importante destacar que el componente padre de todos se encuentra en la carpeta src y se llama App.js

En Spring los paquetes que siguen el patrón arquitectónico MVC se encuentran en la ruta *dp1-2020-g1-01/src/main/java/*org.springframework.samples.petclinic. Aquí se encuentran las distintas clases, los repositorios, los servicios y controladores asociados a dichas clases.

Ventajas alcanzadas al aplicar el patrón

Las distintas ventajas que hemos encontrado al realizar el proyecto con este patrón arquitectónico son:

- Alta cohesión
- Bajo acoplamiento
- Múltiples vistas
- Separación de responsabilidades

Decisiones de diseño de alto nivel

En esta sección describiremos las decisiones de diseño que se han tomado a lo largo del desarrollo de la aplicación que vayan más allá de la mera aplicación de patrones de diseño o arquitectónicos.

Decisión 1

Descripción del problema:

Como grupo se nos presentó la ocasión de hacer uso de Single Page Application (SPA), en lugar de hacer uso del ciclo de vida de página tradicional, ya que de esta forma conseguiríamos tener una única página HTML en el servidor que no se recargaría. El problema es que no conocíamos ninguna de las tecnologías de tipo SPA.

Alternativas de solución evaluadas:

Alternativa 1.a: Hacer uso de jsp.

Ventajas:

- Es el lenguaje visto en la asignatura, por lo que tendríamos bastantes menos problemas.
- Curva de aprendizaje menos pronunciada.

Inconvenientes:

- Supone un menor reto en la asignatura.

Alternativa 1.b: Hacer uso de Angular.

Ventajas:

- Tecnología muy usada y requerida en trabajos actualmente.
- Podemos llevar el frontend del proyecto más allá que con jsp.

Inconvenientes:

- Curva de aprendizaje más pronunciada.
- Aprendizaje externo a lo estudiado en la asignatura.

Alternativa 1.c: Hacer uso de React.

Ventajas:

- Tecnología muy usada y requerida en trabajos actualmente.
- Aprendizaje menos costoso que con Angular.
- Podemos llevar de nuevo el frontend más allá que con jsp.

Inconvenientes:

- La curva de aprendizaje sigue siendo muy costosa.
- Aprendizaje autodidacta.

Justificación de la solución adoptada

Finalmente, decidimos hacer uso de React para implementar el frontend de la aplicación ya que de esta forma añadimos complejidad al proyecto y además, aprendemos el funcionamiento de esta librería.

Decisión 2

Descripción del problema:

Tras encontrarnos numerosos problemas con el modelado de datos a la hora de integrarlo con Spring debido a bucles creados entre las entidades, decidimos hacer modificaciones en el mismo.

Alternativas de solución evaluadas:

Alternativa 2.a: Mantener el modelo y simplificar el sistema.

Ventajas:

- Más sencillez a la hora de desarrollar el proyecto.

Inconvenientes:

- Creación de bucles que dificultaría a largo plazo la realización de ciertas acciones.

Alternativa 2.b: Cambiar el modelo de datos.

Ventajas:

- No se crearían bucles.

Inconvenientes:

- Mucha más complicación a la hora de desarrollar el proyecto.

Justificación de la solución adoptada

Decidimos realizar modificaciones en el modelo de datos ya que queríamos que el proyecto fuese lo más cercano posible a un proyecto real y, sin que se creen bucles que dificulten las futuras operaciones.

Decisión 3

Descripción del problema:

Al intentar ampliar la seguridad de nuestro sistema, nos encontramos con varias soluciones de autentificación de usuarios.

Alternativas de solución evaluadas:

Alternativa 3.a: Utilizar Spring Security con JWT

Ventajas:

- Permite firme cifrado de datos.
- Uso de tokens.

Inconvenientes:

- Desarrollo más costoso

- Gran tamaño de los JSON Web Tokens (JWT)

Alternativa 3.b: Hacer uso de sesiones e implementarlo a mano.

Ventajas:

- Menor complejidad.

Inconvenientes:

- No se hace uso de Spring Security.

Justificación de la solución adoptada

Inicialmente decidimos implementar el auth haciendo uso de sesiones e implementando los métodos necesarios a mano. Pero, en la mitad de la implementación, nos surge una tercera alternativa:

Alternativa 3.c: Utilizar Spring Security con Basic auth

Ventajas:

- Mejor sinergia con nuestro proyecto.
- Utilización de lo aprendido en clase.
- Opción a obtener mayor calificación en la nota final.

Inconvenientes:

- Modificar código ya existente y mayor complejidad de integrarlo con lo que ya tenemos.

Decidimos por tanto hacer uso de Basic auth debido a que era la opción correcta ya que se haría uso lo aprendido en la asignatura y con ello, optaríamos a una mejor calificación en la nota final.

Decisión 4

Descripción del problema:

Al realizar el diseño de las vistas, nos encontramos que al implementar el frontend con React teníamos que escribir todo el código desde cero haciendo uso de JSX. Para facilitar este problema y tras repetidas búsquedas, nos encontramos con la existencia de una librería externa conocida como “Prime React”.

Alternativas de solución evaluadas:

Alternativa 4.a: Escribir todo el código en JSX.

Ventajas:

- No se haría uso de librerías externas.

Inconvenientes:

- Se tendrían que hacer todas las vistas desde cero.

Alternativa 4.b: Hacer uso de PrimeReact

Ventajas:

- Librería que contiene Componentes ya existentes.
- Incluye CSS y fácil integración.
- Menor dificultad y tiempo a la hora de hacer los componentes.

Inconvenientes:

- Aprendizaje del uso de la librería PrimeReact.

Justificación de la solución adoptada

Decidimos optar por la segunda alternativa, pues esta librería nos facilita el desarrollo de los distintos componentes.

Decisión 5

Descripción del problema:

Al realizar el diseño de los componentes, debíamos pasar diferentes estados entre ellos y, para ello tendríamos que pasar extensos estados desde el componente padre hacia el hijo correspondiente. Para facilitar este proceso y para no sobrecargar el componente padre, nos encontramos con la existencia de la librería "Redux"

Alternativas de solución evaluadas:

Alternativa 4.a: Escribir los estados en el componente padre.

Ventajas:

- No se haría uso de librerías externas.

Inconvenientes:

- Se sobrecargaría el componente padre de código inncesario.

Alternativa 4.b: Hacer uso de Redux

Ventajas:

- Librería que facilita el traspaso de estados entre componentes.
- Permite no sobrecargar el componente padre.
- Fácil de usar.

Inconvenientes:

- Aprendizaje del uso de la librería Redux.

Justificación de la solución adoptada

Decidimos optar por la segunda alternativa, pues esta librería nos facilita el intercambio de estado entre distintos componentes y permite no añadir código de más al componente padre

Decisiones de diseño de bajo nivel

Decisión 1

Descripción del problema:

Al encontrarnos con una gran carga de datos hemos decidido implementar paginación para la trata de estos.

Alternativa 1.a: Dejarlo como se mostraba por defecto

Ventajas:

- Menos costoso.
- Menos complejidad.

Inconvenientes:

- Mucha carga de datos y dificultad para la visualización

Alternativa 1.b: Realizar consultas just in time de cada página

Ventajas:

- Uso eficiente de recursos en el servidor.
- Minimiza el tráfico de red.
- Datos actualizados cada vez que se cambia de página.

Inconvenientes:

- Consulta SQL complicada y no estándar.
- Si los datos no se actualizan mucho, podría ejecutarse la misma consulta innecesariamente.

Alternativa 1.c: Paginar manteniendo el resultado en el navegador

Ventajas:

- Sólo se accede a la BD una vez.
- Los cambios de página son muy rápidos.

Inconvenientes:

- Se tarda mucho en cargar la página (el usuario podría cerrar la ventana del navegador)
- Se genera mucho tráfico de red.
- Se pueden mostrar datos obsoletos.

Alternativa 1.d: Paginar manteniendo el resultado en la sesión del usuario

Ventajas

- Solo se accede a la base de datos una vez

Inconvenientes

- Se tarda en procesar la consulta y pasar los datos desde la BD al servidor web.
- Consume mucha memoria del servidor durante mucho tiempo.
- Se desperdician recursos (al usuario sólo le suelen interesar los primeros resultados)
- Se pueden mostrar datos obsoletos.

Justificación de la solución adoptada

Finalmente decidimos optar por la Alternativa 1.c debido al poco impacto en el rendimiento de la aplicación y la utilidad que proporciona a la misma.

Decisión 2

Descripción del problema:

Nos encontramos con el problema de no poder realizar consultas necesarias para acceder directamente a datos requeridos por las historias de usuario.

Alternativa 2.a: Rediseño complejo del sistema

Ventajas:

- Se evitan las relaciones bidireccionales que podrían causar problemas en el rendimiento al crearse ocasionalmente bucles infinitos.

Inconvenientes:

- Impacto en la aplicación.

Alternativa 2.b: Añadir relaciones bidireccionales

Ventajas:

- Menor impacto en la aplicación.
- Acceso directo a los datos.

Inconvenientes:

- Puede acarrear problemas de rendimiento.

Justificación de la solución adoptada

Finalmente decidimos optar por la Alternativa 2.b debido a ser la solución más viable en cuanto a impacto general en el proyecto.

Decisión 3

Descripción del problema:

Para comprobar el correcto funcionamiento de los métodos de los repositorios, se deben realizar pruebas unitarias de estos. A través de ello, nos surgen dos posibles alternativas:

Alternativa 2.a: No mockear los repositorios en los dobles de pruebas en los servicios

Ventajas:

- No habría que realizar test únicamente de repositorios al estar probándolos en los servicios.

Inconvenientes:

- No queda el servicio completamente aislado.

Alternativa 2.b: Probar los repositorios de forma independiente

Ventajas:

- Servicios totalmente aislados.

Inconvenientes:

- Habría que realizar test de repositorios de forma independiente.

Justificación de la solución adoptada

Finalmente decidimos optar por la Alternativa 2.b debido a que queríamos tener menor acoplamiento y mayor cohesión en los test.