DP1 2020-2021

Documento de Diseño del Sistema

Proyecto codeUS

https://github.com/gii-is-DP1/dp1-2020-g2-06

Miembros:

- Aparicio Ortiz, Jesús
- Barranco Ledesma, Alejandro
- Brincau Cano, David
- Granero Gil, Victor Javier
- Ostos Rubio, Juan Ramón

Tutor: Irene Bedilia Estrada Torres

GRUPO G2-06

Versión 2

10/01/2121

Historial de versiones

Fecha	Versión	Descripción de los cambios	Sprint
07/01/2021	V1	• Creación del documento	2

10/01/2021	V2	• Añadido diagrama de dominio/diseño • Explicación de la aplicación del patrón caché • Explicación de ciertas decisiones de diseño	3
09/02/2021	V3	• Explicación de todos los patrones de diseños y estilos arquitectónicos implementados. • Explicación de todas las decisiones de diseño	4

Contents

Introducción

En esta sección debes describir de manera general cual es la funcionalidad del proyecto a rasgos generales (puedes copiar el contenido del documento de análisis del sistema). Además puedes indicar las funcionalidades del sistema (a nivel de módulos o historias de usuario) que consideras más interesantes desde el punto de vista del diseño realizado.

CodeUS es un portal de programación competitiva con objeto de animar e introducir a los alumnos de la Universidad de Sevilla en este campo. La actividad principal de la página es resolver problemas de programación competitiva para conseguir puntos y así subir puntos en el ranking.

El alumnado encontrará en CodeUS problemas de programación de todo tipo (numéricos, grafos, recursión...) para poder enviar soluciones y comprobar su validez con el juez online. Cada problema tiene asociados unos puntos, los cuales serán sumados a los puntos del alumno cuando este realice un envío que el juez considere válido. De esta forma se anima al alumnado a mejorar sus habilidades de programación compitiendo entre ellos. También prepara a los alumnos para competiciones oficiales del ámbito, como Ada Byron, Las12Uvas, Google Code Jam...

El comportamiento de la página es similar a http://aceptaelreto.com pero añadiéndole un carácter más competitivo, tal y como hacen los videojuegos de moda actuales (rankings, puntos, rangos…).

También le añadimos valor educativo al incluir una figura de tutor, que ayuda y guía en la resolución de problemas. Además, publica noticias y artículos resolviendo problemas retirados para la consulta de los alumnos.

Diagrama(s) UML:

Diagrama de Dominio/Diseño

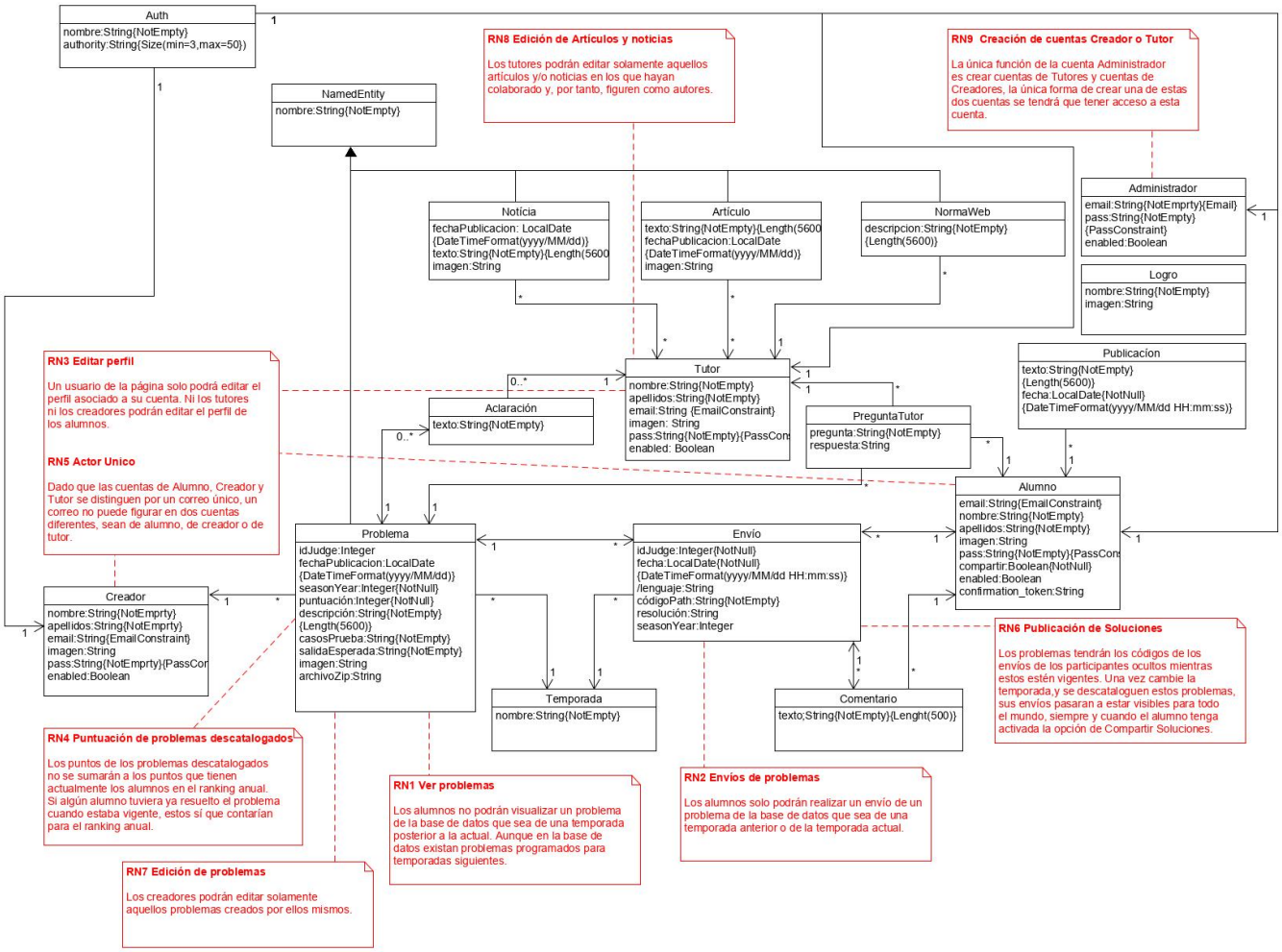
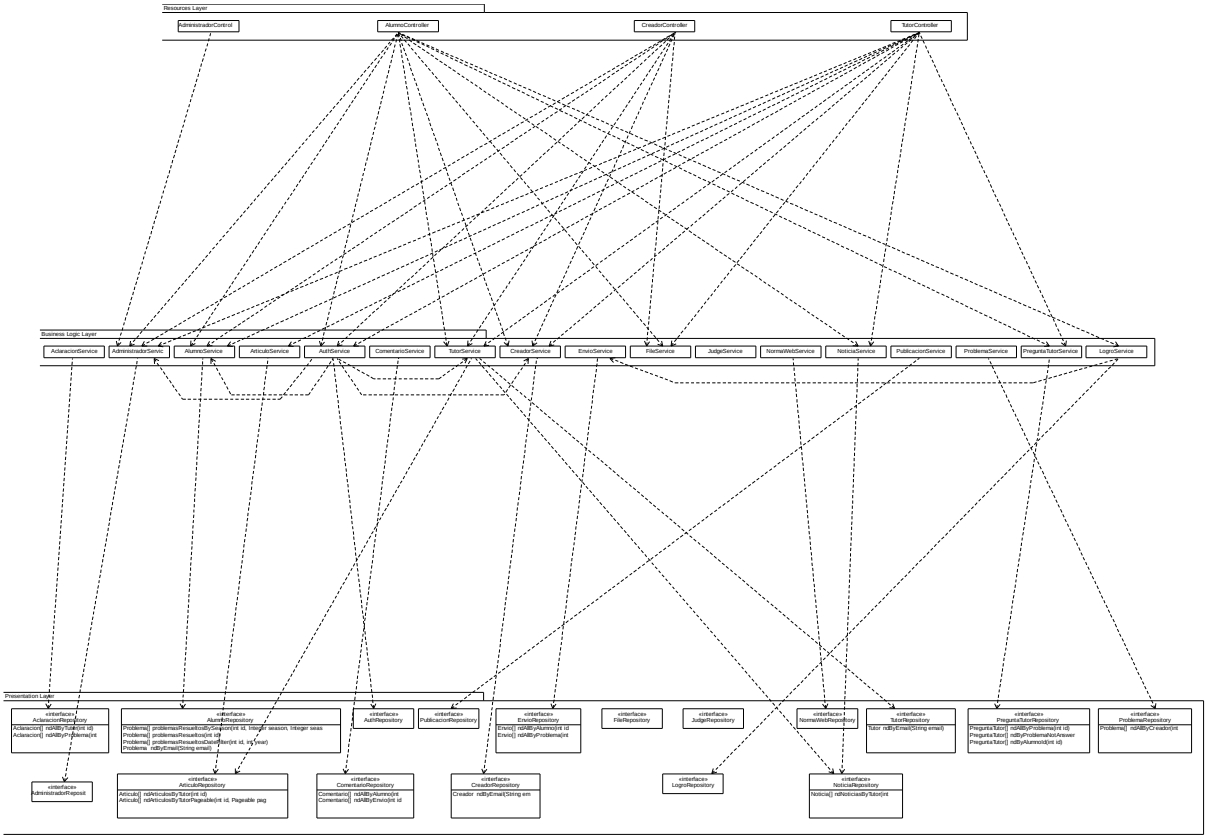
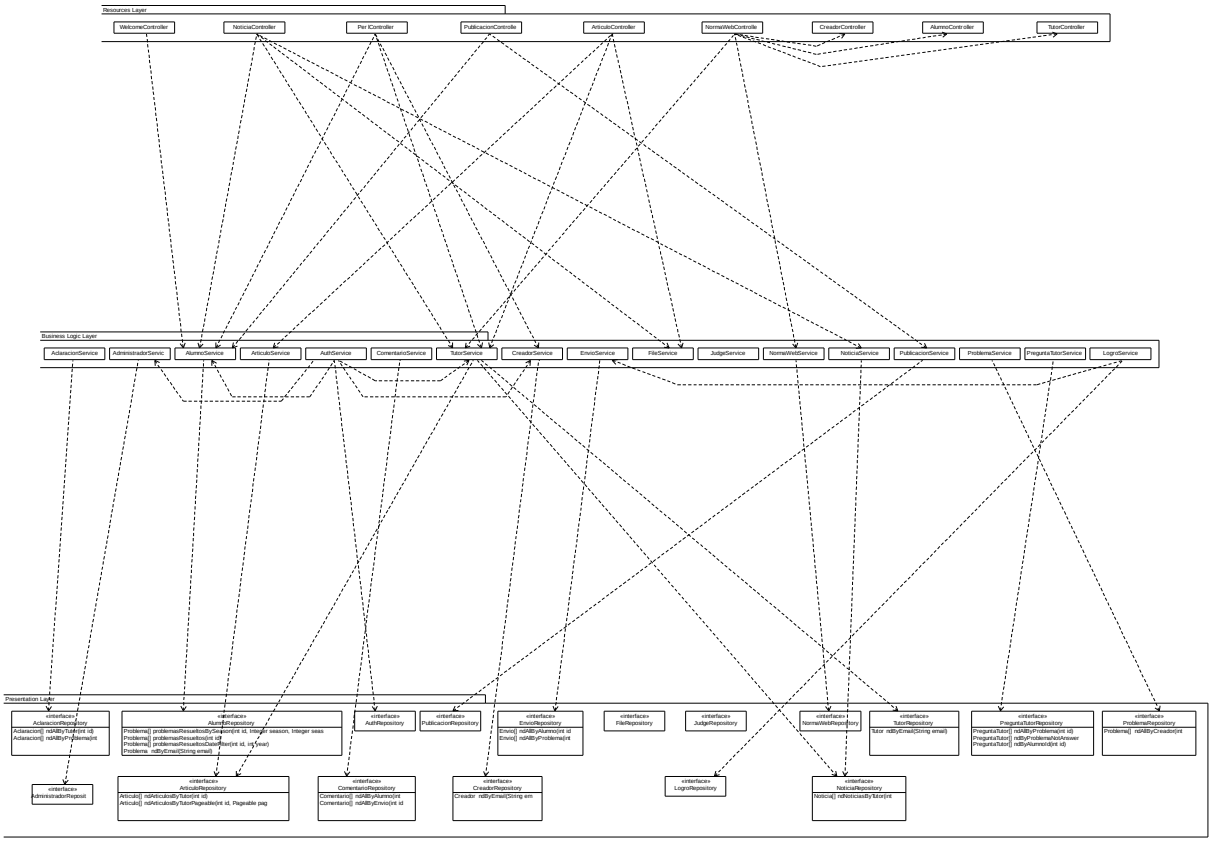
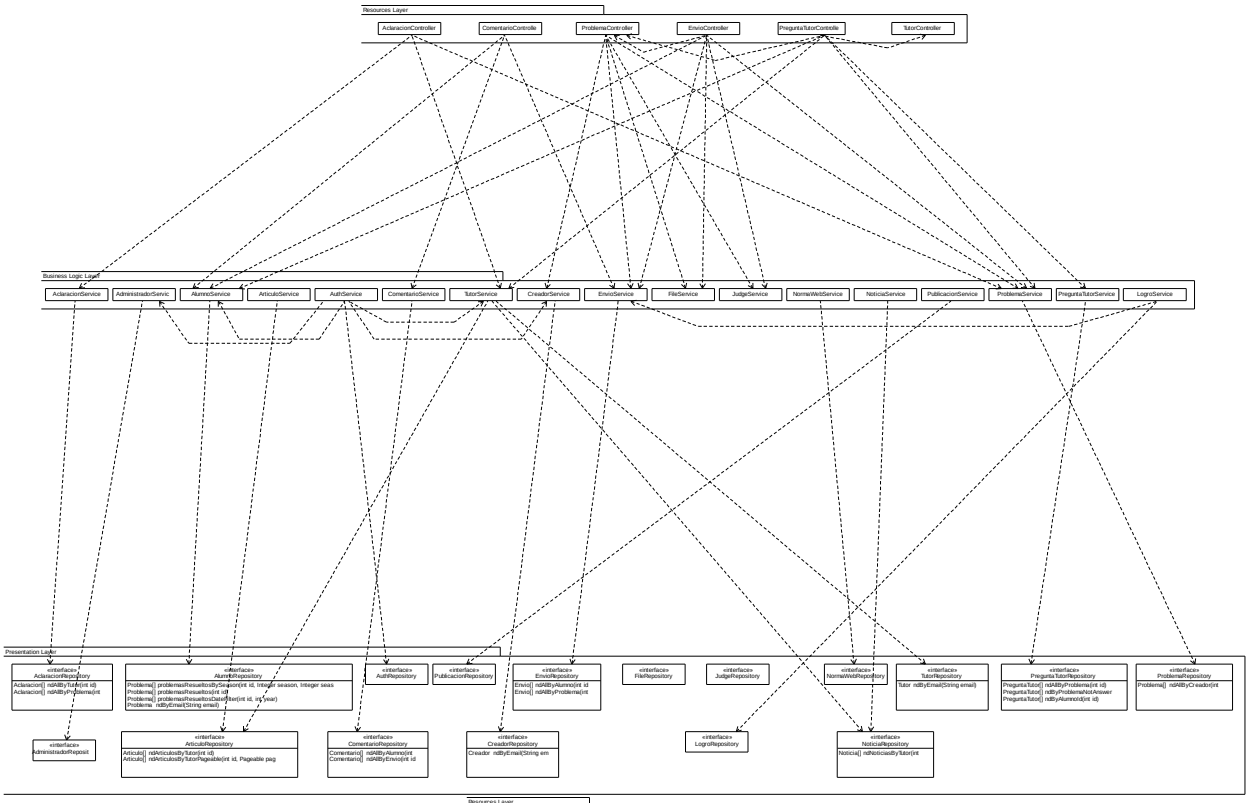
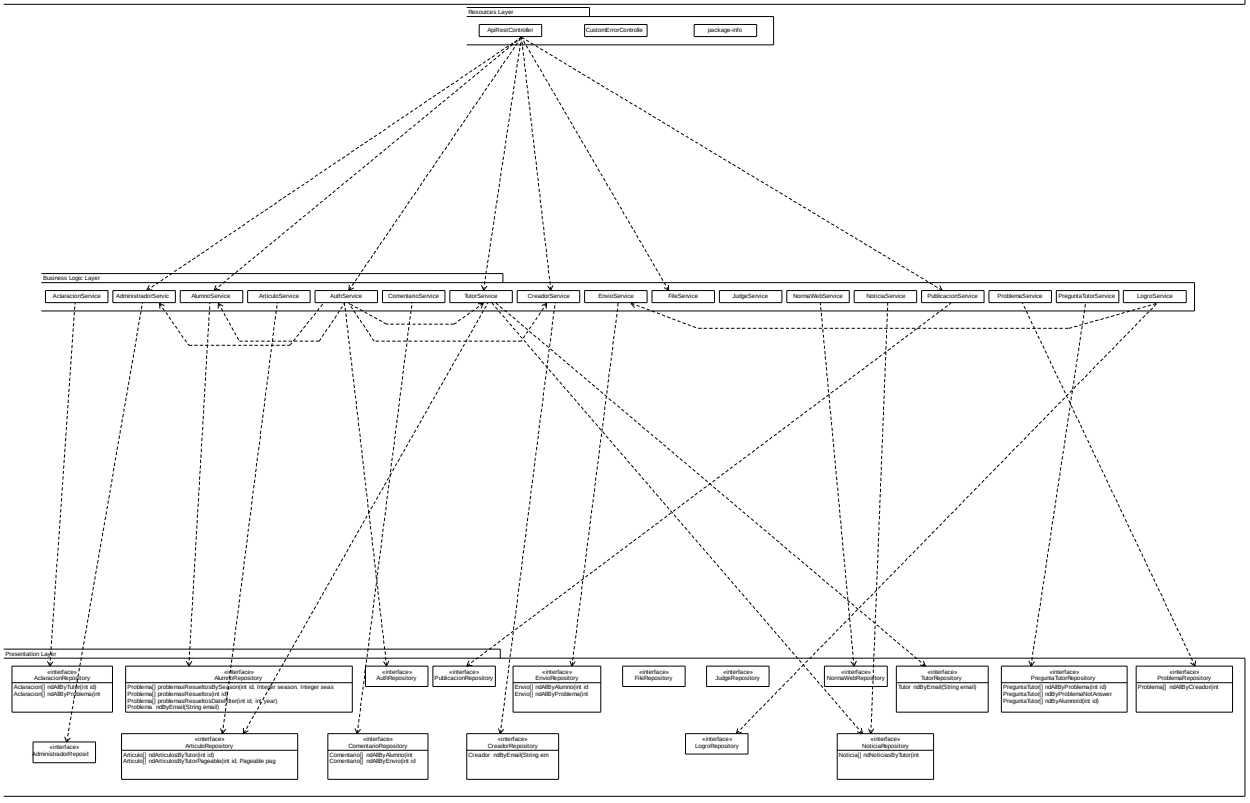


Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)




Resources Layer
WelcomeController
NoticiaController
PerXController
PublicacionController
ArticuloController
NormaWebController
CreadorController
AlumnoController
TutorController
Business Logic Layer
AclaracionService
AdministradorServic
AlumnoService
ArticuloService
AuthService
ComentarioService
TutorService
CreadorService
EnvioService
FileService
JudgeService
NormaWebService
NoticiaService
PublicacionService
ProblemaService
PreguntaTutorService
LogrosService
Presentation Layer
«interface» AclaracionRepository
Aclaracion[] ndABByTutor(int id)
Aclaracion[] ndABByProblema(int
«interface» AlumnRepository
Problema[] problemasResueltosBySeason(int id, Integer season, Integer seas
Problema[] problemasResueltos(int id)
Problema[] problemasResueltosDate(Integer id, int year)
Problema ndByEmail(String email)
«interface» AuthRepository
«interface» PublicacionRepository
«interface» EnvioRepository
Envio[] ndABByAlumno(int id
Envio[] ndABByProblema(int
«interface» FileRepository
«interface» JudgeRepository
«interface» NormaWebRepository
«interface» TutorRepository
Tutor ndByEmail(String email)
«interface» PreguntaTutorRepository
PreguntaTutor[] ndABByProblema(int id)
PreguntaTutor[] ndByProblemaAndAnswer
PreguntaTutor[] ndByAlumno(int id)
«interface» ProblemaRepository
Problema[] ndABByCreador(int
«interface» AdministradorReposit
«interface» ArticuloRepository
Articulo[] ndArticulosByTutor(int id)
Articulo[] ndArticulosByTutorPageable(int id, Pageable pag
«interface» ComentarioRepository
Comentario[] ndABByAlumno(int
Comentario[] ndABByEnvio(int id
«interface» CreadorRepository
Creador ndByEmail(String em
«interface» LogrosRepository
«interface» NoticiaRepository
Noticia[] ndNoticiasByTutor(int


Resources Layer
AclaracionController
ComentarioController
ProblemaController
EnvioController
PreguntaTutorController
TutorController
Business Logic Layer
AclaracionService
AdministradorServic
AlumnoService
ArticuloService
AuthService
ComentarioService
TutorService
CreadorService
EnvioService
FileService
JudgeService
NormaWebService
NoticiaService
PublicacionService
ProblemaService
PreguntaTutorService
LogroService
Presentation Layer
«interface» AclaracionRepository
«interface» AlumnoRepository
«interface» AuthRepository
«interface» PublicacionRepository
«interface» EnvioRepository
«interface» FileRepository
«interface» JudgeRepository
«interface» NormaWebRepository
«interface» TutorRepository
«interface» PreguntaTutorRepository
«interface» ProblemaRepository
«interface» AdministradorReposit
«interface» ArticuloRepository
«interface» ComentarioRepository
«interface» CreadorRepository
«interface» LogroRepository
«interface» NoticiaRepository


Resources Layer
ApiRestController
CustomErrorControlle
package-info
Business Logic Layer
AclaracionService
AdministradorServic
AlumnoService
ArticuloService
AuthService
ComentarioService
TutorService
CreadorService
EnvioService
FileService
JudgeService
NormaWebService
NoticiaService
PublicacionService
ProblemaService
PreguntaTutorService
LogroService
Presentation Layer
«interface» AclaracionRepository
«interface» AlumnoRepository
«interface» AuthRepository
«interface» PublicacionRepository
«interface» EnvioRepository
«interface» FileRepository
«interface» JudgeRepository
«interface» NormaWebRepository
«interface» TutorRepository
«interface» PreguntaTutorRepository
«interface» ProblemaRepository
«interface» AdministradorReposit
«interface» ArticuloRepository
«interface» ComentarioRepository
«interface» CreadorRepository
«interface» LogroRepository
«interface» NoticiaRepository

Patrones de diseño y arquitectónicos aplicados

Estilo Arquitectónico: Capas

Tipo: Arquitectónico

Contexto de Aplicación

Este estilo arquitectónico se ha aplicado a toda la aplicación para darle una estructura base. De esta forma la capa de presentación contendrá todas las vistas y controladores de la aplicación, la capa de lógica de negocio contiene los servicios y modelos y la capa de recursos contiene los repositorios que se comunican con la base de datos.

Clases o paquetes creados

Se crean los distintos paquetes:

- org.springframework.samples.petclinic.model
- org.springframework.samples.petclinic.service
- org.springframework.samples.constraint
- org.springframework.samples.constraint.validators
- org.springframework.samples.petclinic.repository
- org.springframework.samples.petclinic.domjudge
- org.springframework.samples.petclinic.utils
- org.springframework.samples.petclinic.web

*El paquete “domjudge” contiene todo lo relacionado con el modelo y funcionalidades del juez que valida los envíos de los problemas. La clase “Utils.java” dentro del paquete “utils” contiene funcionalidades. Los paquetes “constraint” y “constraint.validators” contienen validaciones para los modelos.

Para la capa de presentación se crean los distintos “.jsp” dentro de la carpeta “jsp”, además de los “.less” para dar estilo a la página.

Ventajas alcanzadas al aplicar el patrón

- Desacoplar las distintas partes de la aplicación, de esta manera se puede modularizar más y con ello agrupar en cada capa por una funcionalidad más específica.
- Identificación más rápida de errores, pues fallaría una capa y no el sistema entero.

Estilo Arquitectónico: REST API

Tipo: Arquitectónico

Contexto de Aplicación

Dicha API es un servicio cuya interfaz la provee el servidor usando REST. Este estilo se utiliza dentro de la aplicación en la clase ApiRestController, donde se utiliza una API de manera interna para paginar las

distintas vistas. Además, se utiliza en todo lo relacionado con DOMjudge, de modo que nos comunicamos con el juez a través de una API externa.

Clases o paquetes creados

Se ha creado la clase:

- ApiRestController

Se ha creado el paquete:

- org.springframework.samples.petclinic.domjudge

Ventajas alcanzadas al aplicar el patrón

- Destaca la escalabilidad, ya que por la separación entre el cliente y el servidor, el producto puede escalar sin que ello represente mucha dificultad.
- Gran flexibilidad y portabilidad.
- Independencia a la hora de desarrollar distintos entornos de desarrollo.

Patrón diseño: MVC

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado en toda la aplicación a excepción del tema de seguridad que se gestiona aparte.

Clases o paquetes creados

No se han implementado clases adicionales.

Ventajas alcanzadas al aplicar el patrón

- Distribuir los datos, la metodología y la interfaz gráfica en distintos componentes.
- Cada uno de estos componentes se encarga específicamente de una tarea: los datos de una aplicación (Modelo), la interfaz del usuario (Vista) y la lógica de control (Controlador).
- Alta cohesión y bajo acoplamiento entre los distintos componentes del sistema, además de la escalabilidad del sistema al separarlo en distintos componentes.
- Los errores se pueden solucionar de una manera más sencilla, ya que al haber bajo acoplamiento se pueden solucionar los errores en el componente que contenga el error y no en todo el sistema.

Patrón diseño: Front Controller

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado a la hora de implementar una serie de controladores que intercepten las peticiones de los usuarios y gestiones dichas peticiones para hacer de intermediario entre el servicio y la vista y modificarlos en caso de que sea necesario.

Clases o paquetes creados

Los controladores se han creado en el paquete:

- org.springframework.samples.petclinic.web

Ventajas alcanzadas al aplicar el patrón

- Este patrón de diseño está implementado ya en el framework de Spring y simplemente tenemos nosotros que crear dichos controladores.
- Es muy flexible, ya que permite decidir qué método debe manejar cada petición.
- Al utilizar controladores tenemos un control centralizado y esto favorece al seguimiento de los usuarios y la seguridad de la aplicación.
- El controlador puede aportar funcionalidades adicionales como el procesamiento de peticiones para la validación de los datos y su correspondiente transformación en caso de que fuera necesario.

Patrón diseño: Dependency Injection

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado a la hora de inyectar dependencias entre los distintos componentes de la aplicación mediante el Application Context. Todo ello a través de las etiquetas: “@Component”, “@Repository”, “@Controller”, “@Service”, “@Configuration”.

Clases o paquetes creados

No se han implementado clases adicionales.

Ventajas alcanzadas al aplicar el patrón

- Este patrón de diseño está implementado ya en el framework de Spring, por lo que únicamente tenemos que utilizar las etiquetas anteriormente mencionadas para que el Application Context inyecte las dependencias. -
- Con este patrón se crea una instancia y solo las clases que la necesiten harán uso de ella. Aplicando este patrón utilizamos a la vez el patrón de diseño “Proxy Pattern”.

Patrón diseño: Proxy Pattern

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado en función de aplicar el patrón de diseño “Dependency Injection”.

Clases o paquetes creados

No se han implementado clases adicionales.

Ventajas alcanzadas al aplicar el patrón

- Este patrón de diseño es totalmente transparente.
- El proxy provee un sustituto a otro objeto para controlar su acceso.

Patrón diseño: Domain model

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado a la hora de utilizar un modelo para cada entidad de la aplicación, de este modo dicho modelo contiene los atributos de la propia entidad y las relaciones entre entidades

Clases o paquetes creados

Los modelos se han creado en el paquete:

- Org.springframework.samples.petclinic.model

La lógica de negocio correspondiente con las entidades como las relaciones entre ellas y sus restricciones simples se incluyen dentro de dichos modelos.

Ventajas alcanzadas al aplicar el patrón

- Este patrón es soportado por la gran mayoría de frameworks hoy día, entre ellos Spring.
- Además, permite implementar lógica compleja y además es soportado

Patrón diseño: Service Layer

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado a la hora de utilizar una capa de servicio que se encuentra entre el modelo y la vista para que estas dos interactúen entre sí.

Clases o paquetes creados

Los servicios se han creado en el paquete:

- Org.springframework.samples.petclinic.service
- Org.springframework.samples.petclinic.web**********

Ventajas alcanzadas al aplicar el patrón

- Separa la lógica del dominio (en el Domain Model) y la lógica de la aplicación (en la capa de servicio).
- Incluye control de transacciones a la hora de comunicarse con la base de datos.

- Realiza verificaciones de seguridad.
- Coordina una gran cantidad de entidades del dominio.

Patrón diseño: Data Mapper

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado a la hora de mover datos entre los objetos y la base de datos mediante una transferencia bidireccional de datos, todo ello mientras se mantienen independientes entre sí y del propio mapeador de datos.

Clases o paquetes creados

Los repositorios se han creado en el paquete:

- Org.springframework.samples.petclinic.repository

Ventajas alcanzadas al aplicar el patrón

- Los objetos de dominio no necesitan saber nada sobre cómo son almacenados en la base de datos.

Patrón diseño: Identity Field

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado a la hora de añadir un identificador a los modelos de las entidades, de modo que todos los objetos de una entidad sean distintos. Todos ellos se han implementado a través de BaseEntity, la cual genera el id automáticamente al crear un objeto.

Clases o paquetes creados

- Todas las entidades heredan de BaseEntity, la cual contiene el identificador.

Ventajas alcanzadas al aplicar el patrón

- Todos los objetos que se creen de una entidad serán distintos.
- El identificador actuará como clave primaria.
- Todas las entidades tendrán el mismo tipo de clave primaria.
- El identificador se incrementa automáticamente.

Patrón diseño: Layer supertype

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado a la hora de que todas las entidades hereden de una clase abstracta(BaseEntity) la cual contiene el identificador.

Clases o paquetes creados

- La clase BaseEntity

Ventajas alcanzadas al aplicar el patrón

- Se utiliza una clase abstracta(BaseEntity) para todas las entidades la cual contiene el identificador de los objetos.
- No hace falta repetir el mismo atributo en todas las clases.
- En caso de tener atributos iguales en distintas entidades se puede crear una clase supertipo con los atributos que tienen en común las entidades que hereden de dicha clase.

Patrón diseño: Repository

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado a la hora de encapsular toda la lógica requerida para acceder a la base de datos en los repositorios.

Clases o paquetes creados

Los repositorios se han creado en el paquete:

- Org.springframework.samples.petclinic.repository

Ventajas alcanzadas al aplicar el patrón

- Da la sensación de estar accediendo a una colección de objetos en memoria en vez de a una base de datos.
- Encapsula toda la lógica requerida para a acceder a los datos.
- Provee una arquitectura flexible.
- Centraliza la lógica de acceso a datos, de modo que la mantenibilidad es más fácil.

Patrón diseño: Template View

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado en todas las vistas de nuestra página, ya que están escritas en JSP/JSTL sobre HTML.

Clases o paquetes creados

Todas las vistas de la aplicación se encuentran en:

- src/webapp/WEB-INF/jsp/

Ventajas alcanzadas al aplicar el patrón

- Al estar basado en JAVA, contiene información dinámica.
- Sus etiquetas simplifican código.
- La ejecución es más rápida comparado con otros lenguajes dinámicos.

Patrón diseño: Eager/Lazy loading

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado en las entidades cuando utilizábamos las etiquetas: @OneToMany, @OneToOne, @ManyToOne, @ManyToMany, en los diferentes atributos de las entidades, este patrón designa la forma de inicializar los datos, en el último momento(Lazy) o al principio de todo(Eager).

Clases o paquetes creados

No se han implementado clases adicionales.

Ventajas alcanzadas al aplicar el patrón

- Inicializar clases cuando realmente es necesario.
- La eficiencia es mucho mayor.

Patrón diseño: Pagination

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado en las entidades: Alumno, Envío, Tutor, Creador, Administrador, Noticia, Articulo y Problema; a través de una API interna, la cual implementa una paginación, que se trata de la numeración en páginas de los distintos datos en partes iguales.

Clases o paquetes creados

La clase ApiRestController donde están todas las paginaciones de la página.

Ventajas alcanzadas al aplicar el patrón

- Un mejor manejo de datos
- A nivel de usuario, los datos son expuestos de una forma más clara y ordenada

Patrón diseño: Strategy

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado en todas las entidades ya que es la forma de determinar cómo se intercambian los datos entre los diferentes objetos para resolver una determinada tarea en tiempo de ejecución.

Clases o paquetes creados

No se han implementado clases adicionales

Ventajas alcanzadas al aplicar el patrón

- Previene los estados condicionales (if, while, etc) simplificando el código.
- Promueve un bajo acoplamiento y una alta cohesión.
- Promueve la extensibilidad de los algoritmos soportados.

Patrón diseño: Chain of responsibility

Tipo: Diseño

Contexto de Aplicación

El patrón se ha aplicado en todas las entidades, se trata de un patrón de comportamiento que hace que las solicitudes pasen a lo largo de una cadena de manejadores, los cuales deciden manejar la petición o pasarla al siguiente controlador de la cadena.

Clases o paquetes creados

No se han implementado clases adicionales

Ventajas alcanzadas al aplicar el patrón

- Promueve el bajo acoplamiento
- Simplifica los objetos ya que no necesita saber la estructura de la cadena ni tampoco mantener referencias directas con los miembros.
- Permite añadir o eliminar responsabilidades dinámicamente cambiando los miembros o el orden de la cadena.

Decisiones de diseño

Decisión 1: Importación de datos reales para demostración

Descripción del problema:

Como grupo nos gustaría poder hacer pruebas con un conjunto de datos reales suficientes, porque resulta más motivador. El problema es al incluir todos esos datos como parte del script de inicialización de la base de datos, el arranque del sistema para desarrollo y pruebas resulta muy tedioso.

Alternativas de solución evaluadas:

Alternativa 1.a: Incluir los datos en el propio script de inicialización de la BD (data.sql).

Ventajas:

- Simple, no require nada más que escribir el SQL que genere los datos.

Inconvenientes:

- Ralentiza todo el trabajo con el sistema para el desarrollo.
- Tenemos que buscar nosotros los datos reales

Alternativa 1.b: Crear un script con los datos adicionales a incluir (extra-data.sql) y un controlador que se encargue de leerlo y lanzar las consultas a petición cuando queramos tener más datos para mostrar.

Ventajas:

- Podemos reutilizar parte de los datos que ya tenemos especificados en (data.sql).
- No afecta al trabajo diario de desarrollo y pruebas de la aplicación

Inconvenientes:

- Puede suponer saltarnos hasta cierto punto la división en capas si no creamos un servicio de carga de datos.
- Tenemos que buscar nosotros los datos reales adicionales

Alternativa 1.c: Crear un controlador que llame a un servicio de importación de datos, que a su vez invoca a un cliente REST de la API de datos oficiales de XXXX para traerse los datos, procesarlos y poder grabarlos desde el servicio de importación.

Ventajas:

- No necesitamos inventarnos ni buscar nosotros lo datos.
- Cumple 100% con la división en capas de la aplicación.
- No afecta al trabajo diario de desarrollo y pruebas de la aplicación

Inconvenientes:

- Supone mucho más trabajo.
- Añade cierta complejidad al proyecto

Justificación de la solución adoptada

Como consideramos que la carga inicial de datos no es excesivamente grande y se puede manejar bastante bien manualmente, seleccionamos la alternativa de diseño 1.a.

Decisión 2: Almacenar fotos

Descripción del problema:

Como la aplicación será utilizada tanto por alumnos, tutores, como creadores, entonces necesitamos almacenar dichas fotos de alguna manera en el sistema. Además, necesitamos guardar otras fotos para distintas entidades como problema, articulo o noticia.

Alternativas de solución evaluadas:

Alternativa 1.a: Guardar la foto como un tipo File en el modelo de cada entidad

Ventajas:

- Sería lo más rápido a implementar, además de lo más sencillo.

Inconvenientes:

- La base de datos no soporta objetos de tipo File.

Alternativa 1.b: Guardar la foto como un array de byte en el modelo de cada entidad

Ventajas:

- Sería una solución bastante rápida.

Inconvenientes:

- Se tendría que realizar la conversión de dicha imagen a bytes cada vez que se quiera guardar y volver a convertirla a imagen cuando se quiera mostrar en la vista.
- Tener un array de byte en vez de un path referenciando a la imagen es poco práctico.

Alternativa 1.c: Almacenar las imágenes por carpetas en función de las entidades dentro de la carpeta “static/resources/images” , de este modo las imágenes se guardarán de manera organizada y si hubiera un error se detectaría rápidamente en qué lugar surge ese error. Además, para referenciar las imágenes, se guardará el path donde están guardadas las imágenes en la entidad correspondiente, de este modo se podrá obtener dicho archivo a través del path.

Ventajas:

- Mayor organización.
- Mayor Facilidad a la hora de mostrar las imágenes en la vista, ya que las imágenes están almacenadas en la carpeta “static”.

Inconvenientes:

- Más difícil de gestionar-
- Añade cierta complejidad al proyecto.

Alternativa 1.d: Almacenar las imágenes con una url y no guardar ninguna imagen como archivo.

Ventajas:

- Sencillez a la hora de guardar las imágenes.

Inconvenientes:

- No se puede colgar cualquier imagen.

Justificación de la solución adoptada

Como trabajar de manera más organizada y tener las imágenes divididas en carpetas lo consideramos interesante para el futuro desarrollo de la aplicación ya que proporciona mayor escalabilidad al sistema, seleccionamos la alternativa de diseño 1.c.

Decisión 3: Nombre de las imágenes

Descripción del problema:

Las imágenes que se suban a la aplicación no pueden estar repetidas

Alternativas de solución evaluadas:

Alternativa 1.a: Dejar el nombre original del archivo

Ventajas:

- Simple, no requiere de cambios.

Inconvenientes:

- Si dos personas suben una imagen con el mismo nombre salta una excepción.

Alternativa 1.b: Poner como nombre de las imágenes el valor de un contador que va aumentando 1 cada vez que se suba una imagen

Ventajas:

- Sencillo, solo se requiere de un contador.

Inconvenientes:

- Se pueden dar inconsistencias de datos.

Alternativa 1.c: Utilizar de nombre de la imagen un identificador que se compone de: “AñoActual+MesActual+DíaActual+HoraActual+MinutoActual+SegundoActual+NanoSegundoActual”

Ventajas:

- Se genera un identificador de manera simple.
- Prácticamente imposible de que dos imágenes tengan el mismo identificador.

Inconvenientes:

- El identificador es bastante largo.
- En un caso muy remoto puede llegar a saltar una excepción debido a que dos personas han hecho un envío al mismo tiempo exactamente.

Justificación de la solución adoptada

Como consideramos que el hecho de que no existan inconsistencias en el sistema es algo fundamental, hemos considerado que la solución 1.c. es la mejor a implementar.

Decisión 4: Puntuación de los alumnos

Descripción del problema:

Como vamos a calcular la puntuación de los alumnos necesitamos obtener la suma individual, en diferentes periodos de tiempo.

Alternativas de solución evaluadas:

Alternativa 1.a: Añadir atributos de puntuación en la entidad alumno

Ventajas:

- Simplificación a la hora de realizar las consultas a la base de datos.

Inconvenientes:

- Almacenamiento de datos innecesarios.
- Intervalos de tiempo fijos, debido a la acumulación en una propiedad.
- Un error de escritura en la base de datos puede causar inconsistencia en los datos (ej: los puntos que tiene un alumno no corresponden realmente a la suma de sus envíos aceptados).

Alternativa 1.b: Obtener los puntos a través de una consulta un tanto más complicada.

Ventajas:

- No se necesita almacenar nada más en la entidad alumno.
- Pueden realizarse consultas para cualquier intervalo de tiempo.
- Asegura que la puntuación es correcta en ese justo momento.

Inconvenientes:

- Se necesita realizar una consulta a la base de datos muy compleja.

Justificación de la solución adoptada

Ya que queremos realizar todo de la manera más correcta y siguiendo las buenas prácticas, preferimos no almacenar cosas innecesarias en la base de datos, ya que se pueden obtener dichos puntos a través de una consulta un tanto más compleja, por ellos hemos seleccionado la alternativa 1.b

Decisión 5: Declaración de validadores

Descripción del problema:

Como nos gustaría poder realizar validaciones semánticas a determinados atributos de algunas entidades, que no llega a cubrir las diferentes etiquetas de validación sintáctica como @NotEmpty, @NonNull, etc...

Alternativas de solución evaluadas:

Alternativa 1.a: Crear nuestros propios validadores.

Ventajas:

- Es un método que nos permite, fácilmente, implementar etiquetas en varios modelos.
- Permite realizar validaciones sintácticas y semánticas.

Inconvenientes:

- Puede llegar a ser más complejo.

Alternativa 1.b: Gestionarlo a través de excepciones.

Ventajas:

- Permite realizar validaciones sintácticas y semánticas.
- Mayor facilidad para expresar mensajes a los usuarios de la aplicación.

Inconvenientes:

- Solo una excepción es lanzada al mismo tiempo.
- Código sea más complejo.

Justificación de la solución adoptada

Como las validaciones que queremos realizar son comunes para la mayoría de las entidades hemos decidido hacer nuestros propios validadores ya que nos será más útil a la hora de rápidamente aplicarlo a las diferentes entidades 1.a.

Decisión 6: Aplicación externa para juzgar los envíos

Descripción del problema:

Cada envío debe evaluarse para conseguir el veredicto y saber si ha sido aceptado o por el contrario presenta algún problema de límite de tiempo, compilación, error en tiempo de ejecución...

Alternativas de solución evaluadas:

Alternativa 1.a: Evaluar en nuestra propia aplicación cada script.

Ventajas:

- No se necesita usar aplicaciones externas ni desplegarlas.

Inconvenientes:

- El diseño de un juez es algo más complejo de lo que parece, sobre todo para evaluar correctamente los tiempos de ejecución.

Alternativa 1.b: Utilizar Mooshak como juez.

Ventajas:

- Desplegando un solo contenedor docker podemos usar este juez a través de su API.

Inconvenientes:

- Presenta problemas al interpretar código C (en la versión para docker).
- No cuenta con llamadas a la API para añadir nuevos problemas.

Alternativa 1.c: Utilizar DOMJudge como juez.

Ventajas:

- API muy completa, posibilita subir problemas. Diseñado de una forma modular (base de datos, server y jueces) de forma que es totalmente escalable si se necesitan más recursos en momentos de uso masivo.

Inconvenientes:

- Es necesario desplegar al menos 3 contenedores de docker.

Justificación de la solución adoptada

Ya que necesitamos subir problemas y que el juez de Mooshak presenta problemas en algunos lenguajes decidimos usar DOMJudge. 1.c

Decisión 7: Tener estadísticas de nuestro problema(graficas)

Descripción del problema:

Al realizar un envió, pueden ocurrir varias cosas, tanto que el envió tiene una solución correcta, como que puede dar un Time Limit Error, como un Wrong Answer, por lo que vimos interesante poder ver unas estadísticas que reflejaran todas estas posibles respuestas.

Alternativas de solución evaluadas:

Alternativa 1.a: Utilizar la librería de JS “Morris.js”

Ventajas:

- Una librería simple y fácil de implementar.

Inconvenientes:

- Es una librería que, aunque es muy fácil de implementar es bastante escasa en cuanto a recursos

Alternativa 1.b: Utilizar la Api JFreeChart

Ventajas:

- API muy completa, con una gran variedad de gráficas, animaciones, etc...

Inconvenientes:

- Al contrario que la primera alternativa es mucho más compleja de implementar y poner en funcionamiento

Justificación de la solución adoptada

Ya que no necesitamos tener unas graficas potentes y que solo queremos gestionar los resultados de los envíos de un determinado problema, hemos decidido hacer uso de la librería JS “Morris.js”, ya que nos permite hacer lo que estábamos buscando de una forma más sencilla y eficaz. 1.a

Decisión 8: Realizar paginación de diferentes entidades

Descripción del problema:

En nuestra página se manejan muchos datos, tanto de alumnos, tutores, envíos, etc.Por ello necesitamos tener un sistema para poder mostrar al usuario todos estos datos, de una forma cómoda y sencilla, por lo que vemos conveniente establecer una paginación para todas estas entidades.

Alternativas de solución evaluadas:

Alternativa 1.a: Realizar la paginación de forma que se recargue la página completa modificando parámetros en la URI para pasar de página.

Ventajas:

- Sencillo de implementar.

Inconvenientes:

- Empeora la experiencia del usuario en alguna de las vistas de la página, teniendo que recargar el DOM entero.
-

Alternativa 1.b: Cambiar de página de forma dinámica con Ajax

Ventajas:

- Mejora la experiencia de usuario al conseguir pasar de página de forma rápida y cómoda.
- Libera de carga al servidor tener que manejar los datos de cada página y tratarlos en la vista, se consigue un JSON y se gestiona del lado del cliente.
- Comportamiento más personalizable.

Inconvenientes:

- La implementación es más compleja (necesidad de una API REST para obtener los datos desde el navegador del cliente utilizando Ajax, código complejo de escribir a la hora de actualizar los datos del div de cada página utilizando Ajax y JQuery).

Justificación de la solución adoptada

Hemos decidido paginar de forma dinámica con Ajax (solución 1.b), ya que conseguimos una paginación más elegante y rápida, además de más personalizable (algunas paginaciones vuelven a la cabecera de la web y otras se mantienen en la misma posición de scroll). También libera de carga al servidor. Todas estas ventajas (además de ser motivo de **A++**) nos condujo a adoptar esta decisión.

Decisión 9: Contraseñas

Descripción del problema:

En nuestra página los usuarios pueden adoptar distintos roles dentro de la misma, ya puede ser tutores, alumnos, creadores o administradores. Por ello es importante que cada uno de ellos tengan una forma de autenticarse para que puedan desempeñar su papel, para ello los usuarios deben identificarse con un email y contraseña. Sobre como guardar las contraseñas en nuestra base de datos, se nos ocurrieron dos formas.

Alternativas de solución evaluadas:

Alternativa 1.a: Almacenar las contraseñas en texto plano

Ventajas:

- Es sencillo de implementar ya que se guardara directamente la contraseña sin modificar en la base de datos.

Inconvenientes:

- Es poco seguro, ya que podrían acceder a la base de datos y tener todas las contraseñas de todos los usuarios.

Alternativa 1.b: Encriptar las contraseñas con la librería Bcrypt

Ventajas:

- Es mucho más seguro, ya que, aunque accedan a la base de datos las contraseñas estarán encriptadas y el conseguir la contraseña original será prácticamente imposible.

Inconvenientes:

- Se requiere encriptar la contraseña mediante dicha librería.

Justificación de la solución adoptada

Hemos decidido encriptar las contraseñas de nuestros usuarios (alternativa 1.b), ya que es la opción más segura para poder almacenar todas las contraseñas de nuestros usuarios.

Decisión 10: Confirmación de correo electrónico

Descripción del problema:

Como ya se ha comentado anteriormente, los usuarios pueden tener distintos roles dentro de la misma página, y todos ellos se identifican mediante un email y contraseña. En nuestra página solo podrán acceder aquellos que tengan un correo de la universidad de Sevilla.

Alternativas de solución evaluadas:

Alternativa 1.a: No verificar correo

Ventajas:

- Es sencillo, simplemente con el validador ya existente vemos que el correo pertenece a la universidad de Sevilla.

Inconvenientes:

- Un mismo usuario podría inventarse muchos correos, sin que realmente existan.

- Es una práctica que es poca segura, ya que podrían hacer ataques a la página aprovechando el hecho de que no se verifica dicho correo.

Alternativa 1.b: Verificar correo

Ventajas:

- Nos aseguramos de que existe el correo de dicho usuario
- Es más seguro, ya que hasta que no se verifica dicho correo, el usuario no puede cometer ningún tipo de acción relacionada con la cuenta creada.

Inconvenientes:

- Más complejo de implementar, ya que hay que enviar un correo de confirmación al usuario mediante el protocolo SMTP.

Justificación de la solución adoptada

Como queremos asegurarnos de que los usuarios que estén en la página tengan un correo perteneciente a la universidad de Sevilla, hemos decidido verificar los correos, de este modo nos aseguramos de que el correo realmente pertenece al usuario. 1.b

Decisión 11: Política de logs

Descripción del problema:

Una de las buenas practicas es tener una buena política de logs. Y como equipo barajamos distintas políticas al respecto

Alternativas de solución evaluadas:

Alternativa 1.a: No hacer uso de logs

Ventajas:

Inconvenientes:

- Es una mala practica
- En términos de organización es malo no tener una buena política de logs

Alternativa 1.b: Hacer uso de logs

Ventajas:

- Mejora la gestión y el control de la información
- Detectamos amenazas procedentes de los usuarios
- Previene fugas de información, así como comportamientos inadecuados

Inconvenientes:

- Requiere una implementación en toda la página y requiere una gran cantidad de tiempo

Justificación de la solución adoptada

En nuestro proyecto queremos trabajar de una forma adecuada y haciendo buenas practicas, por eso decidimos hacer la política de logs teniendo en cuenta los siguientes parámetros:

-INFO:

- Que versión de la aplicación está corriendo y en que entorno
- API, servicios externos
- Operaciones importantes que lance el usuario

-WARNING:

- Cuando un usuario mete datos incorrectos
- Cuando un usuario intenta acciones que no puede hacer
- Cuando usamos un servicio externo, pero funciona la aplicación

-ERROR

- Cuando usamos un servicio externo y no funciona la aplicación

-DEBUG//TRACE:

- Parámetros importantes útiles en el diagnóstico del sistema
- Asociado a la sesión de usuario

Propuestas de A+

Uso de DomJudge:

- Utilizamos una aplicación externa (a través de su API) para evaluar los envíos de los distintos alumnos a los problemas planteados

API propia:

- Se ha realizado una API propia, necesaria para poder paginar con AJAX del lado del cliente

Paginación con AJAX:

- Se ha realizado a partir de la API propia, explicada en el apartado anterior, la paginación de las distintas vistas, mediante AJAX obteniendo los datos en JSON de la api implementada

Verificación de correo electrónico por SMTP:

- Se ha realizado la verificación del correo electrónico de los alumnos que se registran en la aplicación, mediante el protocolo SMTP y la API mail de java

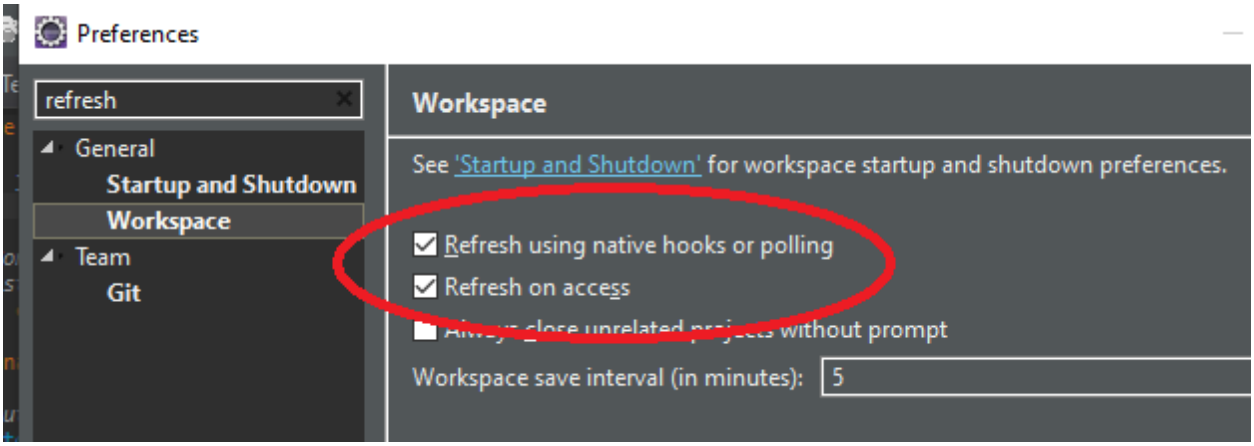
Uso de librería JavaScript Morris.js:

- Se ha hecho uso de la librería JavaScript "Morris.js" una librería de libre uso que nos ha permitido hacer el apartado de estadísticas, en los problemas, con la inserción de graficas personalizables

Requisitos de despliegue

- Para poder ver los logs de una forma correcta en nuestro proyecto deberá:
  Una vez dentro de eclipse seleccionamos la pestaña "help" -> "Install New Software" -> en el apartado "Work with" poner la siguiente URL: http://www.mihai-nita.net/eclipse -> una vez escrita pulsamos el botón "Add" -> seleccionamos la Ansi Console -> Next -> Next -> Accept License…. -> reiniciamos Eclipse y ya podemos ver los logs.

- Para que al subir un archivo en nuestra plataforma esté disponible en el workspace, se deben activar en -> Preferencias -> General -> Workspace se deben activar las dos pestañas que se muestran a continuación:



- Para poder usar el juez (DomJudge) y por lo tanto que toda la aplicación y todas las pruebas funcionen correctamente debe seguir los pasos descritos en un archivo pdf de la carpeta docs del repositorio (DESPLIEGUE_DE_DOMJUDGE_EN_LOCAL.pdf).

- Al estar encriptada las contraseñas no se pueden consultar en la consola de la base de datos, por lo que se incluye un archivo con credenciales, para todos los roles, en la carpeta docs del repositorio (passwordlist.txt).