Nombre: Alejandro Barranco Ledesma Grupo: G2-06

Informe individual de actividades del proyecto

Datos generales

URL del Repositorio de GitHub: https://github.com/gii-is-DP1/dp1-2020-g2-06
Nombre de usuario en GitHub: alebarled
Rama del usuario en Github: alebarled
Nota: en github puede aparecer como si no hubiera hecho aportación alguna al proyecto, pero filtrando los pull request por autoría de alebarled sí se pueden observar los commits hechos por mí.

Participación en el proyecto

Historias de usuario en las que he participado

He implementado **completas** las historias de usuario **HU-9, HU-10, HU-12, HU-13, HU-23, HU-24 y HU-37**. Además, he desarrollado junto a mi compañero **Juan Ramón Ostos Rubio** las historias de usuario **HU-1, HU-3, HU-6, HU-34, HU-35 y HU-36** y junto a mi compañero **Jesús Aparicio Ortiz** las historias de usuario **HU-3, HU-7, HU-8, HU-17 y HU-18**.

Funcionalidad implementada

- He implementado el controlador AlumnoController, NoticiaController, PublicacionController, EnvioController, ProblemaController y ApiRestController.
- He añadido varios métodos a los servicios AlumnoService, PublicacionService, NoticiaService, EnvioService, FileService y JudgeService.
- He creado la entidad Alumno, Noticia, Publicacion, Envio, Temporada y Auth.
- He creado las vistas alumnos/alumnoDetails, alumnos/alumnosList, alumnos/createOrUpdateAlumnoForm, envios/envioDetails, noticias/createOrUpdateNoticiaForm, noticias/noticiaDetails, noticias/noticiasList y publicaciones/publicacionesList, además de implementar la paginación usando AJAX en cliente para tutores/tutorDetails, problemas/problemasList y problemas/problemaDetails.
- He añadido a la clase Utils las funciones getActualSeason(), getActualYearOfSeason(), authLoggedIn() y getMonthName().
- He creado el custom validator EmailValidator y el correspondiente EmailConstraint.
- He modificado tags de petclinic para poder elegir type en los campos de un formulario (por ejemplo un text-area).
- He modificado las consultas SQL en el archivo SecurityConfiguration para poder logearse en la web como alumno, tutor, creador o administrador usando el email como usuario.

Esto hace un total de 20 clases implementadas por mí y 4 interfaces definidos.

Pruebas implementadas

Pruebas unitarias

He creado todos los tests unitarios para 3 servicios (EnvioService, JudgeService y AlumnoService), algunos test unitarios para 2 servicios (FileService, CreadorService y ComentarioService), todos los tests unitarios para 3 controladores (ProblemaController, EnvioController y ApiRestController). Eso hace un total de 8 clases de test unitarios con un total de 64 métodos anotados con @Test.

Pruebas de Controlador

He creado 1 casos de prueba positivo y 2 negativo de controlador para la **HU-7**, 1 caso positivo y 1 negativo para la **HU-10**, 1 caso positivo para la **HU-8**, 1 caso positivo para la **HU-32,** 1 casos positivo y 1 negativo para la **HU-9,** 1 casos positivo y 1 negativo para la **HU-37**. Otras pruebas de controlador comprueban restricciones de seguridad (restricciones según alumno, tutor, creador o administrador) y pruebas del controlador de la API REST que fue necesaria desarrollar para paginar con AJAX en cliente.

Ejemplos de funcionalidades implementadas

Entidades (máximo de dos ejemplos)

Envio.java

```
package org.springframework.samples.petclinic.model;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.time.LocalDateTime;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;

import org.springframework.format.annotation.DateTimeFormat;

import com.fasterxml.jackson.annotation.JsonIgnore;

import lombok.EqualsAndHashCode;
```

```
import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
@EqualsAndHashCode(callSuper=true)
@Entity
@Table(name="envios")
public class Envio extends BaseEntity{

	@Column(name = "id_judge")
	@NotNull
	private Integer idJudge;

	@NotNull
	@DateTimeFormat(pattern = "yyyy/MM/dd HH:mm:ss")
	private LocalDateTime fecha;

	@Column(name="codigo_path")
	@NotEmpty
	private String codigoPath;

	private String resolucion;

	@ManyToOne
	@JoinColumn(name="id_alumno")
	private Alumno alumno;


	@ManyToOne
	@JoinColumn(name="id_problema")
	private Problema problema;

	@ManyToOne
	@JoinColumn(name="id_season")
	private Temporada season;  /// redundante pero necesario para query

	@Column(name = "season_year")
	private Integer seasonYear;   /// redundante pero necesario para query


	@JsonIgnore
	@OneToMany(cascade = CascadeType.ALL, mappedBy = "envio")
	private List<Comentario> listaComentarios;

	public List<String> getCodigoString() throws IOException {
		return Files.readAllLines(Paths.get(codigoPath));
```

```
	}

	public String getFechaFormat() {
		String month = String.valueOf(fecha.getMonthValue());
		if(month.length()==1)
			month = 0 + month;

		return fecha.getDayOfMonth()+"/"+month+"/"+fecha.getYear()+"
"+fecha.getHour()+":"+fecha.getMinute();
	}

}
```

Justificación: Clase de la entidad Envio. Tiene las etiquetas necesarias para crear la entidad (además de los getters setters de Lombok y el nombre correspondiente de la tabla sql). Extiende de BaseEntity para utilizar el id ya implementado. En esta clase las etiquetas NotNull o NotEmpty no tienen demasiada importancia porque todos los valores de la entidad serán asignados de forma sistemática en el controlador, el usuario sólo envía un archivo a una url con el id del problema a resolver, por lo que los datos de esta entidad se generan a partir de esos dos elementos. Se han creado muchas relaciones para acceder en la vista de forma rápida y ágil a los datos del alumno que realiza el envío o del problema al que se refiere. Season y seasonYear son propiedades derivadas, son redudantes pero se guardan porque son datos muy útiles en el cálculo de las puntuaciones de los alumnos. Se trata de consultas sql complejas y filtrar si los envíos aceptados coinciden en temporada y año de temporada con el problema es de mucha utilidad para valorar si ese ”accepted” puntúa o no. Finalmente hay una función que puede ser llamada desde la vista (usando envio.fechaFormat) que muestra la fecha/hora en un formato concreto (dd/MM/YY HH:mm). Las etiquetas @JsonIgnore evitan bucles infinitos a la hora de generar algún json que contenga instancias de esta entidad por parte de la API para la paginación.

Servicio

JudgeService.java

```
package org.springframework.samples.petclinic.service;

import java.io.File;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.FileSystemResource;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.samples.petclinic.domjudge.Judgement;
import org.springframework.samples.petclinic.domjudge.ProblemResponse;
import org.springframework.samples.petclinic.repository.JudgeRepository;
```

```
import org.springframework.stereotype.Service;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.client.RestTemplate;

@Service
public class JudgeService implements JudgeRepository {

	@Autowired
	private RestTemplate restTemplate;

	private final String host = "http://localhost:12345";

	public Judgement getResult(int id, int contest) {
		//restTemplate = new RestTemplate();
		return
restTemplate.getForObject(host+"/api/v4/contests/"+contest+"/judgements/"+id+"?strict=f
alse", Judgement.class);
	}

	public Integer addSubmission(int contest,String filePath, String language, String
entryPoint, int problem) {
		//restTemplate = new RestTemplate();

		MultiValueMap<String, Object> parameters = new
LinkedMultiValueMap<String, Object>();

//		parameters.add("entry_point", entryPoint.chars());
		parameters.add("problem", String.valueOf(problem));
		parameters.add("language", language);
		parameters.add("code[]", new FileSystemResource(new File(filePath)));


		HttpHeaders headers = new HttpHeaders();
		headers.set("Content-Type", "multipart/form-data");
//		headers.set("Accept", "text/plain");
		HttpEntity<MultiValueMap<String, Object>> entity = new
HttpEntity<MultiValueMap<String, Object>>(parameters, headers);


		return
Integer.parseInt(restTemplate.postForObject(host+"/api/v4/contests/"+contest+"/submissi
ons", entity,  String.class));

	}
```

```
	public ProblemResponse addProblem(int contest,String filePath) {
		//restTemplate = new RestTemplate();

		MultiValueMap<String, Object> parameters = new
LinkedMultiValueMap<String, Object>();
		parameters.add("zip[]", new FileSystemResource(new File(filePath)));
		HttpHeaders headers = new HttpHeaders();
		headers.set("Content-Type", "multipart/form-data");
		headers.set("Accept", "text/plain");



		return
restTemplate.postForObject(host+"/api/v4/contests/"+contest+"/problems",
				new HttpEntity<MultiValueMap<String, Object>>(parameters,
headers),
			ProblemResponse.class);

	}



	public String judge(int contest, int runNumber) throws InterruptedException {

		Integer seconds = 25;
		String veredict = "Error inesperado, inténtelo de nuevo más tarde.";

		while(seconds>=0&& (!veredict.equals("AC") ||!veredict.equals("CE")
||!veredict.equals("WA")||!veredict.equals("TLE"))) {
		try {
			veredict = getResult(runNumber,contest).getJudgementTypeId();
			if(veredict==null)
				veredict="Error inesperado, inténtelo de nuevo más tarde.";
		}catch (Exception e) {

		}
		Thread.sleep(1000);
		seconds--;
		}
		return veredict;
	}

}
```

Justificación: Se ha elegido este servicio porque es más peculiar que los demás que hice, que son meras llamadas al repositorio correspondiente. JudgeService se encarga de las llamadas a la API del juez DOMJudge. Crea una instancia de un objeto RestTemplate para poder hacer las consultas http a la api, desplegada en un contenedor de Docker. En los métodos que hacen consultas POST se establecen parámetros como problema, lenguaje, array de bytes con el código del envío, array de bytes con el zip del problema… Uno de los métodos devuelve un objeto Judgement que se generó automáticamente con un json de prueba. RestTemplate lo parsea y ya se puede acceder a los elementos del objeto Java sin problema. La última función es peculiar porque, ya que el juez tarda un poco en dar el veredicto, realiza una espera activa consultando a la API hasta que el veredicto está disponible. Hace una consulta por segundo y tiene un time-out de 25 segundos (menor que el time-out de 30 segundos del navegador).

Controlador

EnvioController.java

```
package org.springframework.samples.petclinic.web;

import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.LocalDateTime;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.samples.petclinic.model.Comentario;
import org.springframework.samples.petclinic.model.Envio;
import org.springframework.samples.petclinic.model.Problema;
import org.springframework.samples.petclinic.service.AlumnoService;
import org.springframework.samples.petclinic.service.EnvioService;
import org.springframework.samples.petclinic.service.FileService;
import org.springframework.samples.petclinic.service.JudgeService;
import org.springframework.samples.petclinic.service.ProblemaService;
import org.springframework.samples.petclinic.util.Utils;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;

import lombok.extern.slf4j.Slf4j;
```

```
@Slf4j
@Controller
@RequestMapping("/envios")
public class EnvioController {

	private final Path rootCodes = Paths.get("codes");

	@Autowired
	private EnvioService envioService;

	@Autowired
	private AlumnoService alumnoService;

	@Autowired
	private ProblemaService problemaService;

	@Autowired
	private FileService fileService;

	@Autowired
	private JudgeService judgeService;

	@Autowired
	private ProblemaController problemaController;


	@GetMapping("/{id}")
	public String envioDetails(@PathVariable("id") int id, ModelMap model) throws
IOException {
		Optional<Envio> envio = envioService.findById(id);
		if(envio.isPresent()) {

	if(envio.get().getAlumno().getEmail().equals(SecurityContextHolder.getContext().get
Authentication().getName()) || Utils.authLoggedIn().equals("tutor")) {
				model.addAttribute("me",true);
			}else {
				model.addAttribute("me",false);
			}
			model.addAttribute("comentarioNuevo", new Comentario());
			model.addAttribute("comentarios",
envio.get().getListaComentarios());
			model.addAttribute("envio",envio.get());
			model.addAttribute("codigo",envio.get().getCodigoString());
			return "envios/envioDetails";
		}
		else {
```

```
                        model.addAttribute("message","No podemos encontrar el envío que
intenta visualizar");
                        return problemaController.listProblemas(model);
                }

        }

        @PostMapping("/send/{problema}")
        public String envioSend(@PathVariable("problema") int problema,
@RequestParam("archivo") MultipartFile archivo, ModelMap model) throws IOException,
InterruptedException {
                Problema problem = problemaService.findById(problema).get();
                if(!Utils.authLoggedIn().equals("alumno")) {
                        model.addAttribute("message","Sólo los alumnos pueden realizar
envíos");
                        log.warn("Un usuario esta intentando realizar un envio sin estar
registrado, con email
"+SecurityContextHolder.getContext().getAuthentication().getName());
                        return problemaController.problemaDetails(problema, model);
///redirect al problema
                }
                else if(problem.getSeasonYear().compareTo(Utils.getActualYearofSeason())>0
|| (problem.getSeasonYear().compareTo(Utils.getActualYearofSeason())==0 &&
problem.getSeason().getId().compareTo(Utils.getActualSeason().getId())>0)) {
                        model.addAttribute("message", "No puedes realizar el envio de este
problema");
                        log.warn("Un usuario esta intentado realizar un envio de un problema
no en vigencia aun con email
"+SecurityContextHolder.getContext().getAuthentication().getName());
                        return problemaController.listProblemas(model);
                }
                else if(archivo.getBytes().length/(1024*1024)>10){
                        model.addAttribute("message","Archivo demasiado grande");
                        return problemaController.problemaDetails(problema, model);
///redirect al problema
                }
                else {
                        String diferenciador = "";
                        String filename = archivo.getOriginalFilename();
                        String extension = "";
                        if(filename.endsWith("java")) {
                                extension = "java";
                                diferenciador = Utils.diferenciador(extension);
                        }
                        else if(filename.endsWith("c")) {
                                extension = "c";
                                diferenciador = Utils.diferenciador(extension);
```

```
                        }
                        else if(filename.endsWith("cpp")) {
                                extension = "cpp";
                                diferenciador = Utils.diferenciador(extension);
                        }
                        else {
                                model.addAttribute("message","Tipo de archivo incorrecto");
                                log.info("Un usuario ha intentado subir un archivo que no es
un .java o .c, con email
"+SecurityContextHolder.getContext().getAuthentication().getName());
                                return problemaController.problemaDetails(problema,
model); //redirect al problema
                        }
                        String email =
SecurityContextHolder.getContext().getAuthentication().getName();
                        Envio e = new Envio();
                        fileService.saveFile(archivo, rootCodes, diferenciador);
                        fileService.saveFile(archivo, rootCodes, filename);
                        log.info(archivo.toString());

                        e.setAlumno(alumnoService.findByEmail(email).get());
                        e.setFecha(LocalDateTime.now());
                        e.setProblema(problemaService.findById(problema).get());
                        e.setSeason(Utils.getActualSeason());
                        e.setSeasonYear(Utils.getActualYearofSeason());
                        e.setCodigoPath(rootCodes + "/" + diferenciador);
                        Integer idJudge;
                        String entryPoint = filename.substring(0, filename.length()-5);
                        log.info(entryPoint);

                        idJudge = judgeService.addSubmission(2, rootCodes + "/" + filename,
extension, entryPoint, problemaService.findById(problema).get().getIdJudge());
                        e.setIdJudge(idJudge);
                        fileService.delete(Paths.get(rootCodes + "/" + filename));
                        String veredict = judgeService.judge(2,idJudge);
                        e.setResolucion(veredict);
                        envioService.save(e);
                        log.info("Se ha utilizado la api de DomJudge satisfactoriamente");
                        return "redirect:/envios/"+e.getId();
                }
        }


}
```

Justificación: Este controlador gestiona las consultas GET y POST que corresponden a los envíos. Se crean al inicio instancias de servicios y repositorios necesarios (con la etiqueta @Autowired). Cada función mapea una consulta con una ruta específica. La primera de las funciones muestra los detalles de un envío. Comprueba que el envío que se está consultando existe, si no devuelve la lista de problemas con un mensaje de error. Comprueba si se está consultando un envío propio del alumno o si se es un tutor, para que después en la vista se contempla para mostrar el código o no. Se añaden atributos necesarios para mostrar la vista correctamente (lista de comentarios, comentario vacío para evitar errores en la vista, objeto envío y código del envío en formato String. En la segunda vista se gestiona la realización de un envío. Se hacen varias comprobaciones al principio, como si se es un alumno o no, si se puede realizar el envío porque está vigente o retirado o si el archivo del código supera un tamaño máximo. También se comprueba que sea un tipo de archivo válido. Para guardar el archivo se utiliza el servicio fileService, y se le asigna un nombre generado por la fecha y hora (incluyendo nanosegundos) para asegurarnos de que los nombres serán siempre distintos. Usando judgeService se envía el código al juez y se obtiene un veredicto tipo String. A partir de la ruta del archivo, el id dado por parámetro y el resultado del juez se crea un objeto Envio y se guarda utilizando envioService.save(). Una vez finalizado todo el proceso se reenvía a la página del nuevo envío para que el usuario pueda ver los detalles de su envío. Se guardan algunos detalles del proceso en el log con las líneas tipo log.xxx…

Repositorio

AlumnoRepository.java

```
package org.springframework.samples.petclinic.repository;

import java.util.Collection;
import java.util.Optional;

import org.springframework.dao.DataAccessException;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Slice;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.Repository;
import org.springframework.data.repository.query.Param;
import org.springframework.samples.petclinic.model.Alumno;
import org.springframework.samples.petclinic.model.Problema;

public interface AlumnoRepository extends Repository<Alumno, Integer>{

	Collection<Alumno> findAll() throws DataAccessException;

	Optional<Alumno> findById(int id) throws DataAccessException;


	void save(Alumno alumno) throws DataAccessException;
```

```
	@Query("SELECT DISTINCT p FROM Problema p JOIN p.envios e WHERE e.alumno.id
LIKE :id AND e.resolucion LIKE 'AC' " + "AND e.season.id LIKE :season AND e.seasonYear LIKE
:seasonyear AND "+ "p.season.id LIKE e.season.id AND p.seasonYear like e.seasonYear")
	public Collection<Problema> problemasResueltosBySeason(@Param("id") int
id,@Param("season")Integer season, @Param("seasonyear")Integer seasonyear);

	@Query("SELECT DISTINCT p FROM Problema p JOIN p.envios e WHERE e.alumno.id
LIKE :id AND e.resolucion LIKE 'AC' " + "AND p.season.id LIKE e.season.id AND p.seasonYear
like e.seasonYear")
	public Collection<Problema> problemasResueltos(@Param("id") int id);

	@Query("SELECT DISTINCT p FROM Problema p JOIN p.envios e WHERE e.alumno.id
LIKE :id AND e.resolucion LIKE 'AC' " + "AND p.season.id LIKE e.season.id AND p.seasonYear
LIKE e.seasonYear AND YEAR(e.fecha) LIKE :year")
	public Collection<Problema> problemasResueltosDateFilter(@Param("id") int
id,@Param("year")int year);

	@Query("SELECT DISTINCT a FROM Alumno a WHERE a.email LIKE :email")
	Optional<Alumno> findByEmail(@Param("email") String email);

	@Query("SELECT a FROM Alumno a WHERE a.enabled LIKE true")
	public Slice<Alumno> findAllPageable(Pageable pageable);


	@Query("SELECT DISTINCT a FROM Alumno a WHERE a.confirmation_token LIKE
:confirmation_token")
	Optional<Alumno> findByToken(@Param("confirmation_token") String
confirmation_token);

}
```

Justificación: Este repositorio contiene 3 funciones que el framework Spring interpreta de forma automática. Las 3 siguientes consultas sirven para calcular la puntuación de los alumnos según si se mira por temporada, por año o todos. Al ser una consulta muy compleja al final quedó en un Collection de problemas que tan sólo queda por sumar sus puntuaciones. Con el tiempo nos dimos cuenta que al devolver esto quizá el lugar correcto para implementarlo sería el ProblemaService, pero a esas alturas del proyecto suponía muchos cambios y se optó por dejarlo en este servicio. Las últimas funciones se refieren a encontrar un alumno por email, la consulta paginada y una consulta que realizó mi compañero David Brincau para encontrar un alumno por el token generado, ya que la necesitaba para desarrollar la confirmación de cuenta por correo electrónico.

Validador y anotación asociada (si aplica, máximo de dos ejemplos)

EmailValidator.java

```
package org.springframework.samples.constraint.validators;

import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.samples.constraint.EmailConstraint;
import org.springframework.samples.petclinic.service.AlumnoService;
import org.springframework.samples.petclinic.service.TutorService;

public class EmailValidator implements ConstraintValidator<EmailConstraint, String> {

	@Override
	public void initialize(EmailConstraint contactNumber) {
	}
	@Override
	public boolean isValid(String emailField, ConstraintValidatorContext cxt) {
	return emailField != null && (emailField.matches(".*@us.es$") ||
emailField.matches(".*@alum.us.es$"));
	}


}
```

Justificación: Este validador comprueba si el correo electrónico es del dominio de la universidad (@us.es o @alum.us.es). Para eso exige que no sea nulo y que haga cumpla las dos expresiones regulares que figuran en la función isValid.

EmailConstraint.java

```
package org.springframework.samples.constraint;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import javax.validation.Constraint;
```

```
import javax.validation.Payload;

import org.springframework.samples.constraint.validators.EmailValidator;

@Documented
@Constraint(validatedBy = EmailValidator.class)
@Target( { ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
public @interface EmailConstraint {
String message() default "Email incorrecto. Debe ser un email @us.es o @alum.us.es";
Class<?>[] groups() default {};
Class<? extends Payload>[] payload() default {};
}
```

Justificación: Creamos la anotación @EmailConstraint con esta clase. En la anotación @Contraint() se hace referencia al validador expuesto anteriormente. Como message() se estipula el mensaje que se mostrará en caso de no cumplirse la condición.

Ejemplos de pruebas implementadas

Pruebas unitarias (máximo de dos ejemplos)

AlumnoServiceTests - shouldSaveAlumno

```
@Test
		public void shouldSaveAlumno() {
			int id = alumnoService.findAll().size();
			Alumno alumno = new Alumno();
			alumno.setNombre("Carmen");
			alumno.setApellidos("Barra");
			alumno.setEmail("carbarmen@alum.us.es");

	alumno.setImagen("resources/images/alumnos/2020122315471487915720 0.jpg");
			alumno.setPass("pass1DD234%");
			alumno.setEnabled(true);
			alumnoService.save(alumno);
			String email = alumnoService.findById(id).get().getEmail();

			assertThat(alumno.getEmail()).isEqualTo(email);
		}
```

Justificación:

- Arrange: crear un alumno con datos válidos.
- Act: guardar el alumno en la base de datos.
- Assert: que este alumno se ha guardado, y por lo tanto, es accessible. Para calcular el id se ha calculado la cantidad de alumnos en base de datos antes de salvar. De esa

forma con el id de ese tamaño debemos recibir el alumno que acabamos de crear posteriormente.

JudgeServiceTests – shouldAddSubmission

```
@Test
	public void shouldaddSubmission() {
		Integer first =
judgeService.addSubmission(2,"codes/20211101942285800000.c","c","",1);
		Integer second =
judgeService.addSubmission(2,"codes/20211101942285800000.c","c","",1);

		assertThat(second).isEqualTo(first+1);
	}
```

Justificación:

- Arrange: datos válidos para envío al juez (id contest; siempre 2, id del problema, ruta del código, lenguaje de programación e id del problema).
- Act: realizar el envío
- Assert: Comprobar que el integer que devuelve el judgeService para el segundo envío es correlativo al primero y que, por lo tanto, se están realizando los envíos y el juez los está procesando.

Pruebas de controlador

2 pruebas de ProblemaControllerTests

```
package org.springframework.samples.petclinic.web;

import static org.mockito.BDDMockito.given;
import static
org.springframework.security.test.web.servlet.request.SecurityMockMvcRequestPostProcessors.csrf;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.model;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;

import java.io.IOException;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
```

```
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.mock.web.MockMultipartFile;
import org.springframework.samples.petclinic.domjudge.ProblemResponse;
import org.springframework.samples.petclinic.model.Creador;
import org.springframework.samples.petclinic.model.Problema;
import org.springframework.samples.petclinic.model.Temporada;
import org.springframework.samples.petclinic.service.AlumnoService;
import org.springframework.samples.petclinic.service.CreadorService;
import org.springframework.samples.petclinic.service.EnvioService;
import org.springframework.samples.petclinic.service.FileService;
import org.springframework.samples.petclinic.service.JudgeService;
import org.springframework.samples.petclinic.service.ProblemaService;
import org.springframework.security.test.context.support.WithMockUser;
import org.springframework.security.test.web.servlet.setup.SecurityMockMvcConfigurers;
import org.springframework.test.annotation.DirtiesContext;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.ui.Model;
import org.springframework.web.context.WebApplicationContext;

@ExtendWith(SpringExtension.class)
@SpringBootTest(webEnvironment=SpringBootTest.WebEnvironment.RANDOM_PORT)
@DirtiesContext
public class ProblemaControllerTests {

	@Autowired
  private WebApplicationContext context;

	private MockMvc mockMvc;

	@MockBean
	private ProblemaService problemaService;

	@MockBean
	private FileService fileService;

	@MockBean
	private EnvioService envioService;

	@MockBean
	private JudgeService judgeService;
```

```
@MockBean
private CreadorService creadorService;

@MockBean
private AlumnoService alumnoService;

@BeforeEach
void setup() throws InterruptedException, IOException {
        mockMvc = MockMvcBuilders
                .webAppContextSetup(context)
                .apply(SecurityMockMvcConfigurers.springSecurity())
                .build();

Temporada temporada = new Temporada();
temporada.setId(0);
temporada.setNombre("PRIMAVERA");

Problema problema  = new Problema();
problema.setId(6);
problema.setCreador(new Creador());
problema.setIdJudge(6);
problema.setName("Hello friend");
problema.setDificultad("6");
problema.setPuntuacion(6);
problema.setDescripcion("This is my description");
problema.setCasos_prueba("2 3 2 3");
problema.setSalida_esperada("Out");
problema.setFechaPublicacion(LocalDate.now());
problema.setSeason(temporada);
problema.setSeasonYear(2020);

//Consultado en uno de los controladores
problema.getCreador().setEmail("davbrican@us.es");
problema.getCreador().setId(4);

Creador cr = new Creador();
cr.setEmail("davbrican@us.es");
cr.setId(4);

Map<String, Long> resoluciones = new HashMap<String,Long>();
resoluciones.put("AC", 2L);

ProblemResponse pr = new ProblemResponse();
List<Integer> ls = new ArrayList<>();
ls.add(1);
pr.setProblemIds(ls);
```

```
	//Mocks de consultas a servicios
	given(this.envioService.resolucionProblema(Mockito.anyInt())).willReturn(resolucion
es);
	given(this.problemaService.findById(6)).willReturn(Optional.of(problema));
	given(this.problemaService.ProblemasVigentes()).willReturn(new ArrayList<>());
	given(this.creadorService.findByEmail(Mockito.anyString())).willReturn(Optional.of(c
r));
	given(this.judgeService.addProblem(Mockito.anyInt(),Mockito.anyString())).willRetur
n(pr);

	}


	@WithMockUser(username = "jesus@us.es", authorities="alumno")
	@Test
	void testShowProblemaDetails() throws Exception {

		//HU-10+E1 Visualización del envío realizado número 10

		mockMvc.perform(get("/problemas/6")).andExpect(status().isOk())
		.andExpect(view().name("problemas/problemaDetails"));
	}

	@WithMockUser(authorities="alumno")
	@Test
	void testDoNotShowNoExistingProblemaDetails() throws Exception {

		//HU-10-E1 Intentar ver problema que no existe (por id)

		mockMvc.perform(get("/problemas/1")).andExpect(status().isOk())
		.andExpect(view().name("problemas/problemasList"));
	}
```

Justificación: Se crean instancias @MockBean de los servicios que se van a mockear de forma que su comportamiento sea simulado. En @BeforeEach se crean objetos que se van a devolver cuando en el controlador se requieran datos de los servicios mockeados. Sólo se simulan con given los métodos que devuelven algún valor, los métodos void se obvian pues no afectan en nada para nuestras pruebas. En estas dos pruebas que se ponen como ejemplo, en una de ellas se requieren los detalles del problema 6. Como hay un given que, al pedir el problema 6 devuelve el objeto que hemos creado antes lo procesa y acaba en la vista de los detalles del problema. En el segundo ejemplo no esta moqueado el problema 1, por lo que se obtiene un collection vacío en el controlador, el cual está programado para llevarnos a la vista de lista de problemas en vez de la vista de detalles del problema. De esta forma comprobamos que en el caso positivo llega a una vista y en el caso negativo a otra (la que le corresponde). Esta comprobación es suficiente para saber que el controlador se comporta de la forma esperada en los dos escenarios.

Principales problemas encontrados

- Hasta que no empezamos a hacer debug era complicado identificar los problemas que hacían que no se mostrara correctamente una vista, por ejemplo.
- La comunicación con la aplicación del juez (DOMJudge) fue complicada de establecer. Se tuvieron que capturar con wireshark consultas desde la documentación de la API para simularlas en Java.
- El guardar archivos, con nombres distintos, también fue un reto, sobre todo para los envíos porque según la convención java el archivo debe llamarse igual que la clase principal. Esto nos dio problemas al renombrar antes de enviar al juez y éste no podía identificar el entrypoint correctamente.
- Cierta torpeza con el uso de github, no ha habido unas nociones básicas sobre su uso en clase, por lo que hemos tenido que aprender a base de errores.

Otros comentarios

Algunas otras tareas que he llevado a cabo en el proyecto son:

- Investigación y desarrollo a la hora de comunicarnos con la API REST del juez.
- Investigación y desarrollo para crear nuestra API REST con los datos paginados.
- Investigación y desarrollo a la hora de usar AJAX en las vistas para recibir y mostrar los datos paginados.
- Retoques de tags petclinic para poder elegir tipo de campo de entrada (ejemplo textArea).
- Consulta sql en archivo de seguridad y cambios en archivo de seguridad SecurityConfiguration.
- Función para printear fecha correctamente formateada en cada entidad que involucra fecha (envío, noticia, articulo)
- Retoque y pulido en la mayoría de las vistas.
- Ayuda a mis compañeros con cualquier problema que les surgiera en sus implementaciones.

Detailed report

08/01/2020 - 03/02/2021

Total: 121:47:00 Billable: 00:00:00 Amount: 0.00 USD

Date	Description	Duration	User
02/09/2021	Pruebas de funciones de servicios añadidas recientemente CodeUs G2-L6	02:00:00 11:00:00AM - 01:00:00PM	Alejandro Barranco
02/08/2021	Documento individual CodeUs G2-L6	06:00:00 05:00:00PM - 11:00:00PM	Alejandro Barranco
02/08/2021	Retoques en pruebas y vistas CodeUs G2-L6	05:00:00 12:00:00PM - 05:00:00PM	Alejandro Barranco
02/07/2021	Documentación - decisiones de diseño CodeUs G2-L6	03:00:00 06:00:00PM - 09:00:00PM	Alejandro Barranco
02/07/2021	Estilo y retoques en vistas CodeUs G2-L6	01:00:00 05:00:00PM - 06:00:00PM	Alejandro Barranco
02/07/2021	Comprobación de todos los tests CodeUs G2-L6	01:00:00 04:00:00PM - 05:00:00PM	Alejandro Barranco
02/07/2021	ApiRestController pruebas CodeUs G2-L6	03:30:00 10:30:00AM - 02:00:00PM	Alejandro Barranco
02/05/2021	imageCrop move to fileService y controller fixes CodeUs G2-L6	01:00:00 12:00:00AM - 01:00:00AM	Alejandro Barranco
02/03/2021	Rankings CodeUs G2-L6	03:00:00 06:00:00PM - 09:00:00PM	Alejandro Barranco
02/03/2021	ImageCrop function CodeUs G2-L6	02:00:00 04:00:00PM - 06:00:00PM	Alejandro Barranco
02/03/2021	ProblemaControllerTest y retoques en EnvioControllerTest CodeUs G2-L6	03:00:00 12:00:00PM - 03:00:00PM	Alejandro Barranco
02/02/2021	ProblemaControllerTest work in progress CodeUs G2-L6	06:00:00 07:00:00PM - 01:00:00AM	Alejandro Barranco
02/02/2021	paginacionEnvios en problemaDetails y paginacionProblemasNoVigentes CodeUs G2-L6	05:00:00 12:00:00PM - 05:00:00PM	Alejandro Barranco

02/01/2021	Paginacion AJAX / investigación y desarrollo CodeUs G2-L6	06:00:00 07:00:00PM - 01:00:00AM	Alejandro Barranco
02/01/2021	EnvioControllerTests CodeUs G2-L6	02:16:00 03:21:00PM - 05:37:00PM	Alejandro Barranco
02/01/2021	TutorController pruebas problems (MultipartFile y autowired) CodeUs G2-L6	02:21:00 12:00:00PM - 02:21:00PM	Alejandro Barranco
01/10/2021	Pruebas envioService, judgeService, creadorService y comentarioService CodeUs G2-L6	07:00:00 03:00:00PM - 10:00:00PM	Alejandro Barranco
01/10/2021	Publicaciones de alumno en foro CodeUs G2-L6	01:14:00 03:00:00AM - 04:14:00AM	Alejandro Barranco
01/10/2021	Cambios estructurales y estéticos CodeUs G2-L6	01:18:00 01:42:00AM - 03:00:00AM	Alejandro Barranco
01/10/2021	Filtrar botones en vistas según rol CodeUs G2-L6	01:42:00 12:00:00AM - 01:42:00AM	Alejandro Barranco
01/09/2021	Finalizar envío service / Añadir en problema controller y vista envío CodeUs G2-L6	06:00:00 06:00:00PM - 12:00:00AM	Alejandro Barranco
01/08/2021	Varios cambios en entidades, servicios y vistas CodeUs G2-L6	07:15:00 07:00:00PM - 02:15:00AM	Alejandro Barranco
01/08/2021	Utils for authLog and idLoged CodeUs G2-L6	02:55:00 05:30:00PM - 08:25:00PM	Alejandro Barranco
01/08/2021	Auth implementation + cambios necesarios CodeUs G2-L6	03:16:00 12:30:00PM - 03:46:00PM	Alejandro Barranco
01/07/2021	Manual despliegue DOMJudge con Docker CodeUs G2-L6	01:00:00 12:30:00PM - 01:30:00PM	Alejandro Barranco
01/05/2021	Troubleshoot Problema & Competicion (vista y controller) CodeUs G2-L6	03:00:00 12:00:00AM - 03:00:00AM	Alejandro Barranco
12/28/2020	Custom validator email CodeUs G2-L6	01:00:00 09:00:00PM - 10:00:00PM	Alejandro Barranco
12/28/2020	modificaciones archivos / entidad Temporada (enumerate) CodeUs G2-L6	02:30:00 06:00:00PM - 08:30:00PM	Alejandro Barranco

12/23/2020	Modificaciones por subida de archivos CodeUs G2-L6	03:00:00 01:00:00PM - 04:00:00PM	Alejandro Barranco
12/21/2020	Implementación API DOM Judge CodeUs G2-L6	05:00:00 02:00:00PM - 07:00:00PM	Alejandro Barranco
12/20/2020	Implementación API DOM Judge CodeUs G2-L6	02:00:00 07:00:00PM - 09:00:00PM	Alejandro Barranco
12/05/2020	Querys de puntuación para Alumno / modificaciones necesarias en Alumno y Envio CodeUs G2-L6	05:00:00 01:00:00PM - 06:00:00PM	Alejandro Barranco
11/29/2020	Envío - entidad CodeUs G2-L6	01:00:00 12:00:00AM - 01:00:00AM	Alejandro Barranco
11/28/2020	Alumno solucionar problemas StackOverflow CodeUs G2-L6	01:00:00 11:00:00PM - 12:00:00AM	Alejandro Barranco
11/28/2020	Alumno pruebas CodeUs G2-L6	01:00:00 09:00:00PM - 10:00:00PM	Alejandro Barranco
11/26/2020	Alumno - entidad y vistas CodeUs G2-L6	02:00:00 05:00:00PM - 07:00:00PM	Alejandro Barranco
11/25/2020	Modificación entidad / vista Noticia CodeUs G2-L6	01:00:00 02:00:00PM - 03:00:00PM	Alejandro Barranco
11/14/2020	Creación de mockups CodeUs G2-L6	03:10:00 12:00:00PM - 03:10:00PM	Alejandro Barranco
11/04/2020	Puesta en común y descripción general del proyecto. CodeUs G2-L6	01:30:00 01:00:00PM - 02:30:00PM	Alejandro Barranco
11/03/2020	Creación de Entidad / Controlador / Repositorio / Servicio / Vista de Noticia CodeUs G2-L6	02:00:00 11:00:00PM - 01:00:00AM	Alejandro Barranco
11/02/2020	Creación de Entidad / Controlador / Repositorio / Servicio / Vista de Noticia CodeUs G2-L6	02:00:00 10:00:00PM - 12:00:00AM	Alejandro Barranco
10/19/2020	Enumeración de entidades, roles e historias de usuario. CodeUs G2-L6	01:50:00 10:40:00AM - 12:30:00PM	Alejandro Barranco