Refactoring

- Durante todo el desarrollo de la aplicación, por experiencias pasadas con otras asignaturas, siempre hemos tenido en cuenta la calidad de nuestro código, es decir, siempre hemos intentado seguir un standard a la hora de desarrollar tanto la aplicación como sus pruebas y siempre hemos intentado que sea claro y fácil de entender en numerosas revisiones. Por ello, hay pocas refactorizaciones que hayamos podido llevar a cabo.

    - La mayoría de malos olores detectados tienen como solución la extracción de métodos compuestos y simplificación de condicionales:

```
@PreAuthorize("hasAuthority('doctor')")
@Transactional(readOnly = true)
public Doctor findCurrentDoctor() throws DataAccessException {

    Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    UserDetails user = (UserDetails) principal;
    String username = user.getUsername();


    return this.doctorRepository.findDoctorByUsername(username);
}
```

- Extraemos ese fragmento de código relacionado con identificar a un usuario a un método aparte, quedando de la siguiente forma:

```
private String getUsernameCurrentDoctor() {
    Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    UserDetails user = (UserDetails) principal;
    return user.getUsername();
}

@PreAuthorize("hasAuthority('doctor')")
@Transactional(readOnly = true)
public Doctor findCurrentDoctor() throws DataAccessException {
    return this.doctorRepository.findDoctorByUsername(getUsernameCurrentDoctor());
}
```

```
@Transactional
        return "/patient/updatePatientForm";

    } else {

        BeanUtils.copyProperties(patientForm.getPatient(), patientToUpdate, "id", "password",
                "username", "enabled");
        if (patientForm.getNewPassword() != null
                && !StringUtils.isEmpty(patientForm.getNewPassword())) {
            patientToUpdate.setPassword(patientForm.getNewPassword());
        }

        this.patientService.save(patientToUpdate);

        if (!oldUsername.equals(patientToUpdate.getUsername())) {
            return "redirect:/logout";
        } else {
            return "redirect:/patient";
        }

    }

}
```

- También encontramos esta condición un poco compleja de entender, pudiendo simplificarla, de forma que quede más sencilla:

✓ Solución

```
private boolean checkNewPassword(final PatientForm patientForm) {
    return patientForm.getNewPassword() != null
        && !StringUtils.isEmpty(patientForm.getNewPassword());
}

@PostMapping(value = "/patient/edit")
public String processUpdatePatientForm(@Valid final PatientForm patientForm,
        final BindingResult result) {

    Patient patientToUpdate = this.patientService.findCurrentPatient();
    String oldUsername = String.valueOf(patientToUpdate.getUsername());


    if (result.hasErrors()) {

        return "/patient/updatePatientForm";

    } else {

        BeanUtils.copyProperties(patientForm.getPatient(), patientToUpdate, "id", "password",
                "username", "enabled");
        if (this.checkNewPassword(patientForm)) {
            patientToUpdate.setPassword(patientForm.getNewPassword());
        }

        this.patientService.save(patientToUpdate);

        if (!oldUsername.equals(patientToUpdate.getUsername())) {
            return "redirect:/logout";
        } else {
            return "redirect:/patient";
        }
```

- Como podemos observar, el predicado para comprobar si la contraseña es nula o está vacía (líneas 102 y 103) es excesivamente extenso. Para simplificarlo, hemos creado un método privado que realiza la misma función. Dicho método devuelve un booleano, por lo que se introducirá en la condición del bloque *if* directamente:

```
85      @PostMapping(value = "/patient/edit")
86      public String processUpdatePatientForm(@Valid final PatientForm patientForm,
87              final BindingResult result) {
88
89          Patient patientToUpdate = this.patientService.findCurrentPatient();
90          String oldUsername = String.valueOf(patientToUpdate.getUsername());
91
92          System.out.println(result.getAllErrors());
93
94          if (result.hasErrors()) {
95
96              return "/patient/updatePatientForm";
97
98          } else {
99
100             BeanUtils.copyProperties(patientForm.getPatient(), patientToUpdate, "id", "password",
101                     "username", "enabled");
102             if (patientForm.getNewPassword() != null
103                     && !StringUtils.isEmpty(patientForm.getNewPassword())) {
104                 patientToUpdate.setPassword(patientForm.getNewPassword());
105             }
106
107             this.patientService.save(patientToUpdate);
108
109             if (!oldUsername.equals(patientToUpdate.getUsername())) {
110                 return "redirect:/logout";
111             } else {
112                 return "redirect:/patient";
113             }
114
115         }
116
117     }
```

✓ Solución

```
private boolean passwordNotNull(PatientForm patientForm) {

    return patientForm.getNewPassword() != null
        && !StringUtils.isEmpty(patientForm.getNewPassword());
}

@PostMapping(value = "/patient/edit")
public String processUpdatePatientForm(@Valid final PatientForm patientForm,
        final BindingResult result) {

    Patient patientToUpdate = this.patientService.findCurrentPatient();
    String oldUsername = String.valueOf(patientToUpdate.getUsername());

    System.out.println(result.getAllErrors());

    if (result.hasErrors()) {

        return "/patient/updatePatientForm";

    } else {

        BeanUtils.copyProperties(patientForm.getPatient(), patientToUpdate, "id", "password",
                "username", "enabled");
        if (passwordNotNull(patientForm)) {
            patientToUpdate.setPassword(patientForm.getNewPassword());
        }

        this.patientService.save(patientToUpdate);

        if (!oldUsername.equals(patientToUpdate.getUsername())) {
            return "redirect:/logout";
        } else {
            return "redirect:/patient";
        }

    }

}
```

- Encontramos la siguiente condición en el validador de Constant:

```
Consultation consultation = this.consultationService.findConsultationById(this.consultationId);
if (consultation != null && consultation.getConstants().stream().anyMatch(c -> constant.getConstantType().equals(c.getConstantType()) && constant.getId() != c.getId())) {
    errors.rejectValue("constantType", "alreadyExistingConstantType", "Este tipo de constante ya ha sido registrada en la consulta");
}
```

- Para solucionar la condición excesivamente larga, extraemos la condición a una función independiente que evalúe la condición:

```
@Override
public void validate(final Object target, final Errors errors) {
    Constant constant = (Constant) target;

    if (constant.getConstantType() == null) {
        errors.rejectValue("constantType", "requiredConstantType", "Este campo es obligatorio");
    } else {

        Consultation consultation = this.consultationService.findConsultationById(this.consultationId);
        if (existsConstantTypeInConsultation(constant, consultation)) {
            errors.rejectValue("constantType", "alreadyExistingConstantType", "Este tipo de constante ya ha sido registrada en la consulta");
        }
    }

    if (constant.getValue() < 0f) {
        errors.rejectValue("value", "negativeValue", "El valor de la constante no debe ser negativo");
    }
}

protected boolean existsConstantTypeInConsultation(Constant constant,
        Consultation consultation) {
    return consultation != null && consultation.getConstants().stream().anyMatch(c -> constant.getConstantType().equals(c.getConstantType()) && constant.getId() != c.getId());
}
```

Al realizar este cambio aumenta la facilidad para comprender el objetivo de la condición y su significado.

- Por otra parte, encontramos también otra condición excesivamente larga en el validador de Consultation:

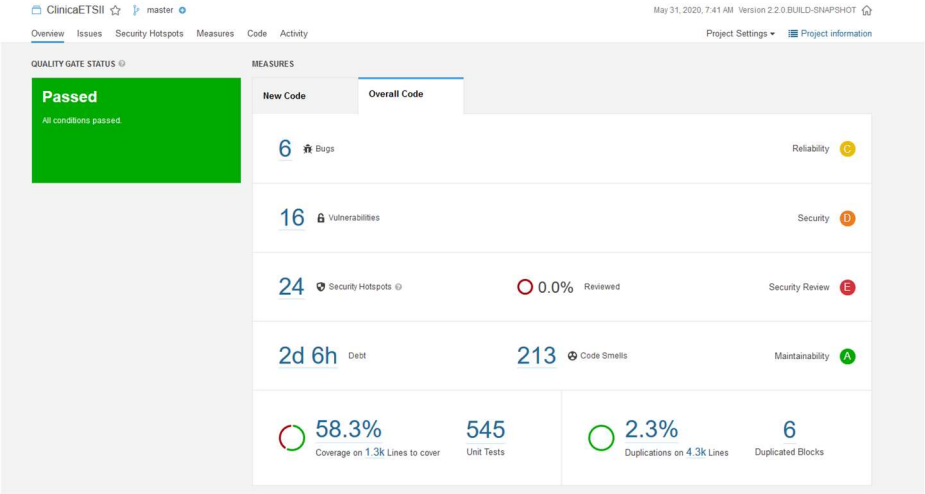
```
if (consultationToUpdate.getDischargeType() != null && !cosultation.isNew()
        && (cosultation.getExaminations() == null
                || cosultation.getExaminations().isEmpty())) {
    errors.rejectValue("dischargeType", "emptyExplorations",
            "No es posible dar de alta una consulta sin exploraciones");
}
```

- Para solucionar la condición excesivamente larga, extraemos la condición a una función independiente que evalúe la condición:

```
@Override
public void validate(final Object target, final Errors errors) {
    Appointment appointment = this.appointmentService.findAppointmentById(this.appointmentId);

    Consultation cosultation = new Consultation();
    if (this.consultationId != null) {
        cosultation = this.consultationService.findFullConsultationById(this.consultationId);
    }

    Consultation consultationToUpdate = (Consultation) target;

    if (consultationToUpdate.getStartTime() == null) {
        errors.rejectValue("startTime", "requiredStartTime",
                "La fecha de inicio es obligatoria");
    } else if (appointment.getStartTime().isAfter(consultationToUpdate.getStartTime())) {
        errors.rejectValue("startTime", "invalidStartTime",
                "La fecha de inicio de la consulta debe ser igual o posterior al de la cita");
    }

    if (invalidConsultationToUpdate(consultationToUpdate, cosultation)) {
        errors.rejectValue("dischargeType", "emptyExplorations",
                "No es posible dar de alta una consulta sin exploraciones");
    }
}

protected boolean invalidConsultationToUpdate(Consultation consultationToUpdate,
        Consultation cosultation) {
    return consultationToUpdate.getDischargeType() != null && !cosultation.isNew()
            && (cosultation.getExaminations() == null
                    || cosultation.getExaminations().isEmpty());
}
```

En este caso se ha utilizado un nombre más genérico al declarar la función puesto que en dicha condición se evalúa más de un atributo, aun así, el factor común es que esta condición solo se evalúa si ya existe la consulta con anterioridad y se trata de una actualización.

Para facilitar la búsqueda de Bad Smells se ha recurrido al uso de la herramienta SonarQube.



A partir de este análisis se ha obtenido un resultado considerablemente bueno, debido a las siguientes razones:

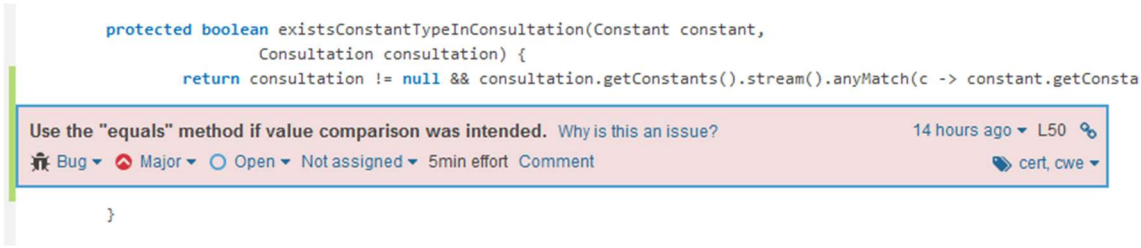
- Se han encontrado 6 bugs de los cuales, aquellos marcados con una cruz azul, pueden ser considerados falsos positivos debido a la implementación del código.

- A continuación, se procederá a corregir los dos bugs presentes en el código de la siguiente imagen, encontrado en SecurityConfiguration.

✓ Solución

```
@Bean
public RestTemplate restTemplate(final RestTemplateBuilder builder) {

    return builder.setConnectTimeout(Duration.ofMillis(10 * 1000l))
            .setReadTimeout(Duration.ofMillis(10 * 1000l)).build();
}
```

Otro de los bugs es la recomendación del uso de equals en la siguiente condición de ConstantValidator:




```
:tConstantType()) && constant.getId() != c.getId());
```

✓ Solución

```
:ConstantType()) && !constant.getId().equals(c.getId()));
```

- Se han encontrado 16 vulnerabilidades, las cuales se deben principalmente al uso de objetos persistentes con la anotación @Entity como argumento de un método con @RequestMapping. De esto podemos deducir que el número de vulnerabilidades no se va a reducir, porque forma parte del patrón de programación utilizado a lo largo del proyecto.

- Se han encontrado 24 Hotspots, de los cuales 11 se deben al uso de anotaciones @PreAuthorize y @PostAuthorize y 12 al uso de patterns.

- Se han encontrado 213 Code Smells, en las cuales hay 5 Blockers, 19 Critical, 25 Mayors, 162 Minors y 2 Info.

Comenzaremos resolviendo los Info:

Se trata de dos comentarios TODO que describen 2 funcionalidades que habían sido planeadas para el código. Como estas ya han sido implementadas, se va a proceder borrando dichos comentarios.

- Resolvemos uno de los Code Smells de severidad Minor:

El primero de los code smells señalados, situado en SecurityConfiguration, se resuelve devolviendo directamente el valor, en lugar de asignarlo a encoder, como se muestra en la siguiente imagen:

- Algunos de uno de los Code Smells de severidad Blocker son:

Los cuales se deben a que faltan algunos asserts o se encuentran incompletos. Su corrección es rápida, puesto que muchos de ellos se encuentras ausentes debido a despistes.

- Algunos de uno de los Code Smells de severidad Critical son:

```
@Transactional(readOnly = true)
public ConsultaFiltro findViasDeAdministracionByNmobre(final String nombre,
                final Integer pagina) throws RestClientException {
        return this.restTemplate.getForObject(
                        CIMAService.MAESTRAS_URL + 1 "?maestra={maestra}&nombre={nombre}&pagina={pagina}",
```

Define a constant instead of duplicating this literal "?maestra={maestra}&nombre={nombre}& pagina={pagina}" 4 times. Why is this an issue?
Code Smell Critical Open Not assigned 10min effort Comment
2 months ago L48
design

```
                        ConsultaFiltro.class, 4, nombre, pagina);
}

@Transactional(readOnly = true)
public ConsultaFiltro findLaboratoriosByNombre(final String nombre,
                final Integer pagina) throws RestClientException {
        return this.restTemplate.getForObject(
                        CIMAService.MAESTRAS_URL + 2 "?maestra={maestra}&nombre={nombre}&pagina={pagina}",
                        ConsultaFiltro.class, 6, nombre, pagina);
}

@Transactional(readOnly = true)
public ConsultaFiltro findFormasFarmaceuticasByNombre(final String nombre,
                final Integer pagina) throws RestClientException {
        return this.restTemplate.getForObject(
                        CIMAService.MAESTRAS_URL + 3 "?maestra={maestra}&nombre={nombre}&pagina={pagina}",
                        ConsultaFiltro.class, 3, nombre, pagina);
}

@Transactional(readOnly = true)
public ConsultaFiltro findPrincipiosActivosByNombre(final String nombre,
                final Integer pagina) throws RestClientException {
        return this.restTemplate.getForObject(
                        CIMAService.MAESTRAS_URL + 4 "?maestra={maestra}&nombre={nombre}&pagina={pagina}",
                        ConsultaFiltro.class, 1, nombre, pagina);
}
```

✓ Solución

```
private static final String MAESTRAS_URL_WITH_QUERIES = CIMAService.MAESTRAS_URL + "?maestra={maestra}&nombre={nombre}&pagina={pagina}";


@Transactional(readOnly = true)
public ConsultaFiltro findViasDeAdministracionByNmobre(final String nombre,
        final Integer pagina) throws RestClientException {
    return this.restTemplate.getForObject(
            MAESTRAS_URL_WITH_QUERIES,
            ConsultaFiltro.class, 4, nombre, pagina);
}

@Transactional(readOnly = true)
public ConsultaFiltro findLaboratoriosByNombre(final String nombre,
        final Integer pagina) throws RestClientException {
    return this.restTemplate.getForObject(
            MAESTRAS_URL_WITH_QUERIES,
            ConsultaFiltro.class, 6, nombre, pagina);
}

@Transactional(readOnly = true)
public ConsultaFiltro findFormasFarmaceuticasByNombre(final String nombre,
        final Integer pagina) throws RestClientException {
    return this.restTemplate.getForObject(
            MAESTRAS_URL_WITH_QUERIES,
            ConsultaFiltro.class, 3, nombre, pagina);
}

@Transactional(readOnly = true)
public ConsultaFiltro findPrincipiosActivosByNombre(final String nombre,
        final Integer pagina) throws RestClientException {
    return this.restTemplate.getForObject(
            MAESTRAS_URL_WITH_QUERIES,
            ConsultaFiltro.class, 1, nombre, pagina);
}
```

- En DoctorFormValidator, se ha encontrado:

```
if (doctor.getName() == null || StringUtils.isEmpty(doctor.getName())) {
        errors.rejectValue("doctor.name", "requiredName", 1 "Este campo es obligatorio");
```

Define a constant instead of duplicating this literal "Este campo es obligatorio" 7 times. 2 months ago ▾ L44

Why is this an issue?

Code Smell ▾ Critical ▾ Open ▾ Not assigned ▾ 16min effort Comment design ▾

```
}
```

✓ Solución:

```
private static final String REQUIRED = "Este campo es obligatorio";
```

```
if (doctor.getName() == null || StringUtils.isEmpty(doctor.getName())) {
    errors.rejectValue("doctor.name", "requiredName", REQUIRED);
}
```

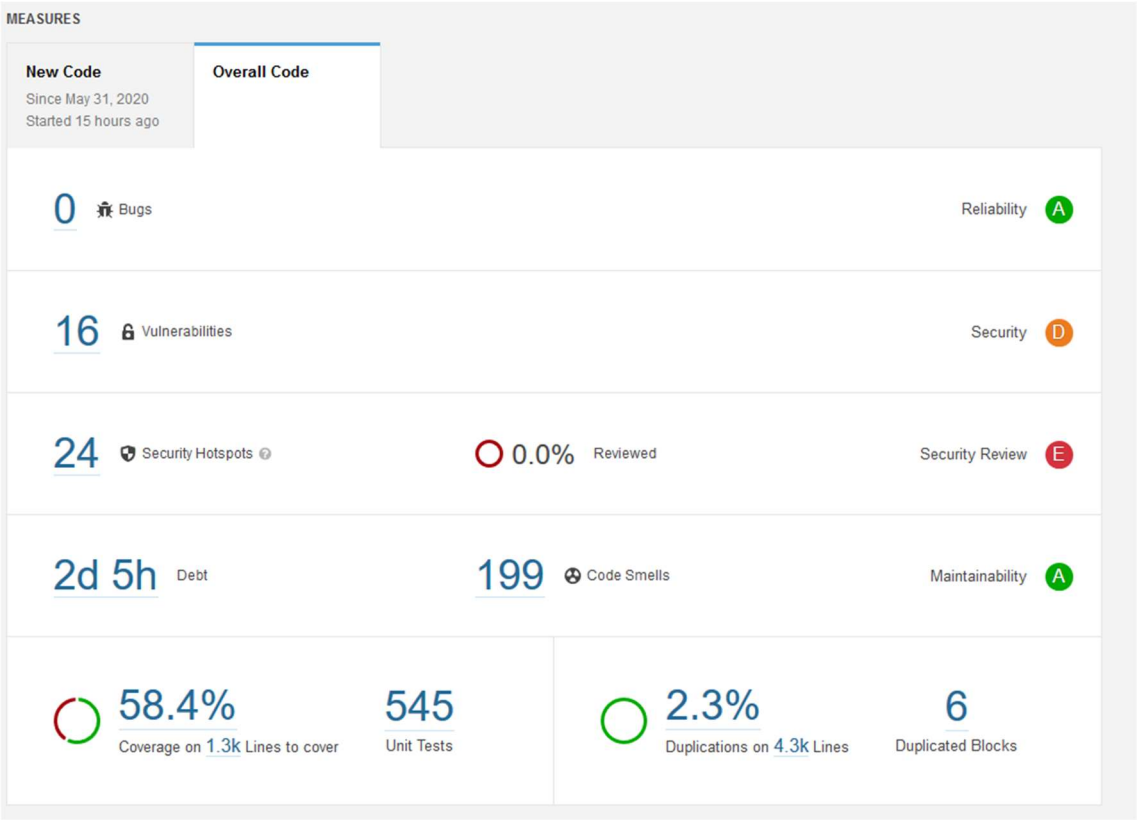
- Algunos de uno de los Code Smells de severidad Critical son:

```
public class CIMAService {

	private RestTemplate restTemplate;

	private static final String BASE_URL = "https://cima.aemps.es/cima/rest";
	private static final String MEDICAMENTO_URL = CIMAService.BASE_URL + "/medicamento";
	private static final String MEDICAMENTOS_URL = CIMAService.BASE_URL + "/medicamentos";
	private static final String BUSQUEDA_URL = CIMAService.BASE_URL + "/buscarEnFichaTecnica";
```

Remove this unused "BUSQUEDA_URL" private field. Why is this an issue? 2 months ago L19
Code Smell Major Open Not assigned 5min effort Comment unused

Se soluciona borrando la constante no utilizada.

- Tras la resolución de algunos de los Code Smells encontrados, los resultados obtenidos del análisis son:



Tal como se esperaba, el número de vulnerabilidades se mantiene, pero se han reducido los bugs y el número de Code Smells a pasado de 213 a 199.