# G2-8 Design & Testing II

## Documentación refactor realizadas:

**Refactor PacienteController.**

**Remove Dispensables:**

En el controllador de Pacientes podemos encontrar varios dispensables. Para refactorizar PacienteController y que la deuda técnica sea reducida debemos de eliminar este código que no aporta nada a nuestro proyecto.

- Dead Code:

Eliminación de código comentado:





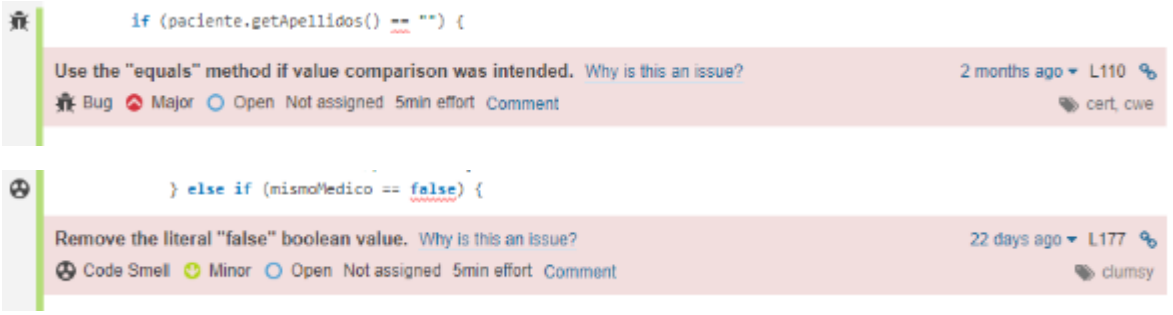
- Imports dispensables:

```
import org.hibernate.TypeMismatchException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.samples.petclinic.model.Cita;
import org.springframework.samples.petclinic.model.Paciente;
import org.springframework.samples.petclinic.service.AuthoritiesService;
import org.springframework.samples.petclinic.service.CitaService;
import org.springframework.samples.petclinic.service.HistoriaClinicaService;
import org.springframework.samples.petclinic.service.MedicoService;
import org.springframework.samples.petclinic.service.PacienteService;
import org.springframework.samples.petclinic.service.UserService;
import org.springframework.samples.petclinic.util.DniValidator;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
```

- Declaración de servicio el cual no es usado:


- Eliminar variables y declaraciones no utilizadas:


- Eliminar comprobaciones de boolean:






- Código duplicado

A continuación, se muestra el código duplicado el cual será declarado en variables.


Esta son las variables asignadas que arreglan estos malos olores:

```
private static final String VIEWS_PACIENTE_CREATE_OR_UPDATE_FORM = "pacientes/createOrUpdatePacientesForm";
private static final String REDIRECT_PACIENTES = "redirect:/pacientes/";
private static final String PACIENTE_STRING = "paciente";
private static final String MEDICO_LIST_URL = "medicoList";
private static final String IS_NEW_PACIENTE_LABEL = "isNewPaciente";
```

**Bloaters:**

**Long Method:**

En método borrarPaciente, como podemos ver en la imagen inferior, es bastante largo.

```java
@RequestMapping(value = "/pacientes/{pacienteId}/delete")
public String borrarPaciente(@PathVariable("pacienteId") final int pacienteId, final ModelMap modelMap) {
    String view = "/pacientes";

    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    Optional<Paciente> paciente = this.pacienteService.findPacienteById(pacienteId);

    if (paciente.isPresent()) {
        Collection<Cita> citas = this.citaService.findAllByPaciente(paciente.get());

        boolean puedeBorrarse = citas.isEmpty();
        if (!citas.isEmpty()) {
            LocalDate ultimaCita = citas.stream().map(Cita::getFecha).max(LocalDate::compareTo).get();
            LocalDate hoy = LocalDate.now();
            puedeBorrarse = hoy.compareTo(ultimaCita) >= 6
                    || hoy.compareTo(ultimaCita) == 5 && hoy.getDayOfYear() > ultimaCita.getDayOfYear();
        }

        boolean mismoMedico = paciente.get().getMedico().equals(this.userService.getCurrentMedico());
        boolean medicoEnabled = this.medicoService.getMedicoById(this.userService.getCurrentMedico().getId())
                .getUser().isEnabled();

        puedeBorrarse = puedeBorrarse && mismoMedico && medicoEnabled;

        if (puedeBorrarse) {
            this.pacienteService.deletePacienteByMedico(pacienteId, this.userService.getCurrentMedico().getId());
            modelMap.addAttribute("message", "Paciente borrado exitosamiente");
            view = "redirect:/pacientes";
        } else if (mismoMedico == false) {
            modelMap.addAttribute("message", "No tiene acceso para borrar a este paciente");
            view = "redirect:/pacientes/" + pacienteId;
        } else {
            modelMap.addAttribute("message", "Paciente no puede borrarse");
            view = "redirect:/pacientes/" + pacienteId;
        }
    } else {
        modelMap.addAttribute("message", "Paciente no encontrado");
        view = "accessNotAuthorized";
    }
    return view;
}
```

Para refactorizar este método hemos dividido las comprobaciones en varios métodos que también podrán ser reutilizados por el resto de métodos. A continuación, se muestra como quedaría el método borrarPaciente.

```java
@GetMapping(value = "/pacientes/{pacienteId}/delete")
public String borrarPaciente(@PathVariable("pacienteId") final int pacienteId, final ModelMap modelMap) {
    String view;
    Optional<Paciente> optionalPaciente = this.pacienteService.findPacienteById(pacienteId);
    String labelMessage = "message";

    if (optionalPaciente.isPresent()) {
        Paciente paciente = optionalPaciente.get();
        boolean puedeBorrarse = checkDeleteforCitas(paciente) && sameMedico(paciente)
                && paciente.getMedico().getUser().isEnabled();

        if (puedeBorrarse) {
            this.pacienteService.deletePacienteByMedico(pacienteId, this.userService.getCurrentMedico().getId());
            modelMap.addAttribute(labelMessage, "Paciente borrado exitosamiente");
            view = "redirect:/pacientes";
        } else if (!sameMedico(paciente)) {
            modelMap.addAttribute(labelMessage, "No tiene acceso para borrar a este paciente");
            view = REDIRECT_PACIENTES + pacienteId;
        } else {
            modelMap.addAttribute(labelMessage, "Paciente no puede borrarse");
            view = REDIRECT_PACIENTES + pacienteId;
        }
    } else {
        modelMap.addAttribute(labelMessage, "Paciente no encontrado");
        view = "accessNotAuthorized";
    }
    return view;
}
```

Estos son los métodos auxiliares creados que podrán ser reutilizados por otros métodos del controlador.

```
public Paciente checkPacienteIsPresent(int pacienteId) {
    return this.pacienteService.findPacienteById(pacienteId).orElse(null);
}

public boolean sameMedico(Paciente paciente) {
    return paciente.getMedico().equals(this.userService.getCurrentMedico());
}

public boolean checkDeleteforCitas(Paciente paciente) {
    Collection<Cita> citas = this.citaService.findAllByPaciente(paciente);
    if (!citas.isEmpty()) {
        LocalDate ultimaCita = citas.stream().filter(c -> c.getFecha() != null).map(Cita::getFecha)
                .max(LocalDate::compareTo).orElse(null);
        if (ultimaCita != null) {
            LocalDate hoy = LocalDate.now();

            int compareDate = hoy.compareTo(ultimaCita);

            return compareDate >= 6 || compareDate >= 5 && hoy.getDayOfYear() > ultimaCita.getDayOfYear();
        } else {
            return true;
        }
    } else {
        return true;
    }
}

public boolean hasContact(Paciente paciente) {
    return paciente.getN_telefono() == null && paciente.getDomicilio().isEmpty() && paciente.getEmail().isEmpty();
}

public boolean telefonoOk(Paciente paciente) {
    if ((paciente.getN_telefono() == null)) {
        return true;
    } else {
        return paciente.getN_telefono().toString().length() == 9;
    }
}
```

**Arreglo de bugs:**

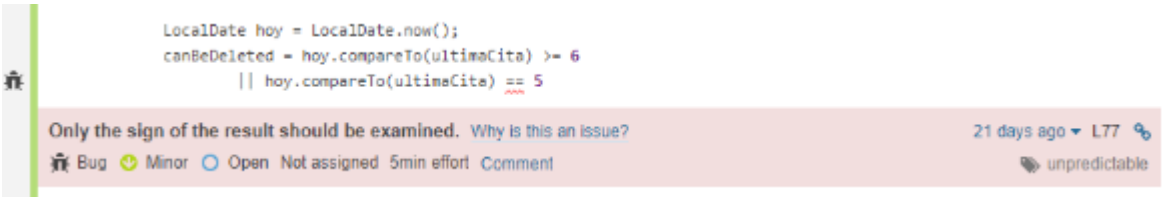
Sonar nos señala ciertos bugs, controlados por otras clases. A continuación, se muestran los bugs y como han sido fixeados.

El siguiente error se nos muestra que no comprobamos como se nos devuelve un Optional<Paciente>.


Para arreglarlo, hemos creado un método auxiliar que nos comprueba esto y nos devuelve el Paciente.

```
public Paciente checkPacienteIsPresent(int pacienteId) {
    return this.pacienteService.findPacienteById(pacienteId).orElse(null);
}
```

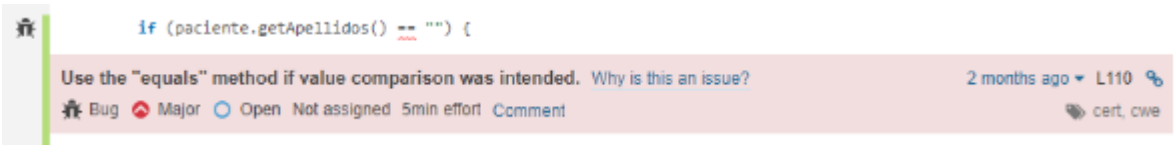
Se nos muestra el siguiente error, el cual comprueba que la última cita del Paciente sea mayor a 5 años desde el día actual.



Este error es fácilmente fixeado cambiando la igualdad por un comparador de mayor o igual.

```
return compareDate >= 6 || compareDate >= 5 && hoy.getDayOfYear() > ultimaCita.getDayOfYear();
```

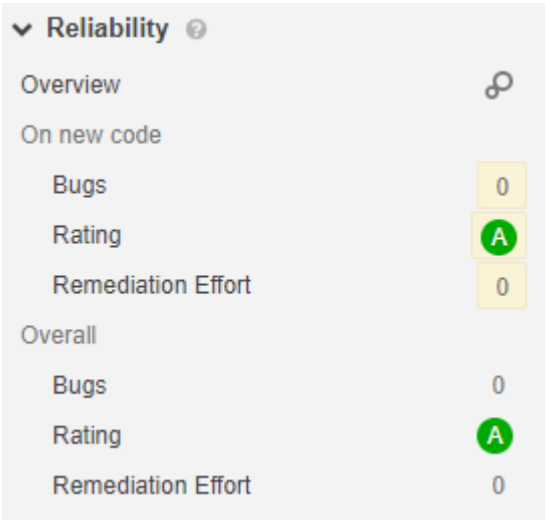
El siguiente equals es mostrado como error.

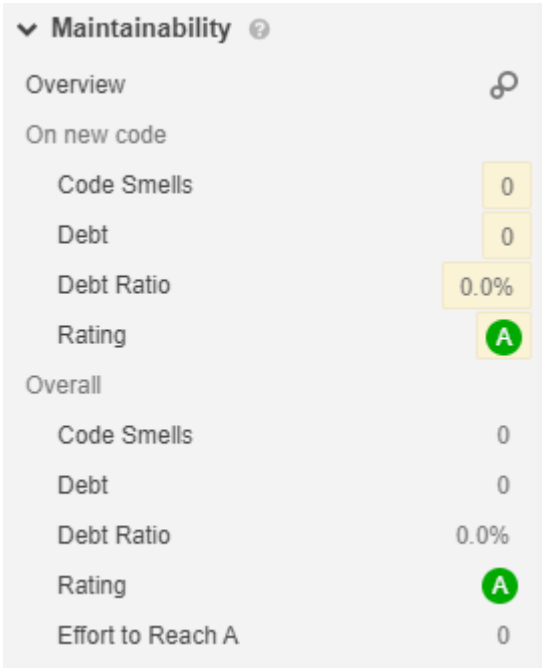



Para arreglarlo cambiamos el comparador de igualdad por un equals.

```
if (paciente.getApellidos().equals("")) {
```

Una vez hemos refactorizado la clase PacienteController, podemos observar como la deuda técnica y los bugs se reducen al 0%.

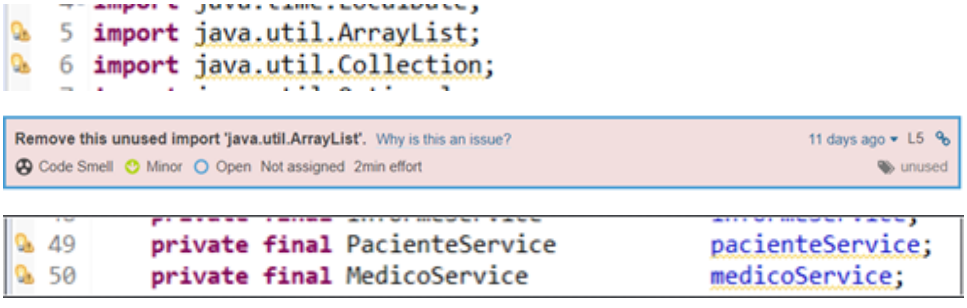



**RefactorInformeController:**

**Disposables:**

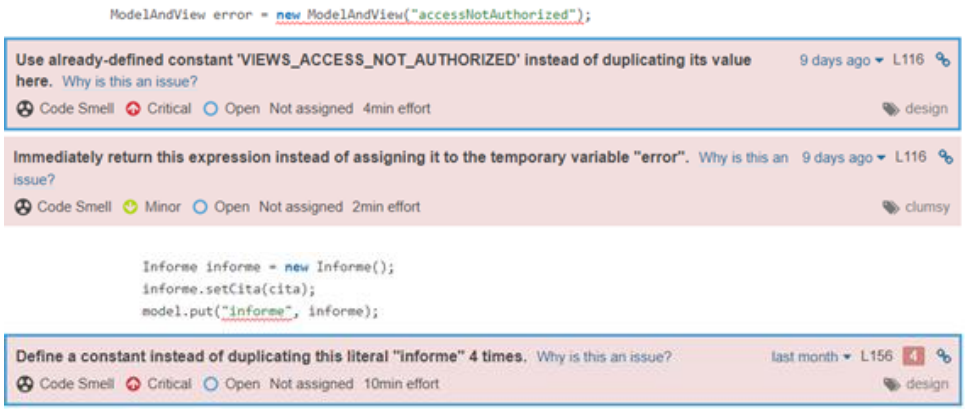
Eliminación de Imports Innecesarios.




Eliminación de código comentado:


Eliminación de Excepciones Innecesarias:


Código Duplicado:




Estas serían las variables estáticas tras la refactorización de estos disposables:

```
private static final String            VIEWS_INFORME_CREATE_OR_UPDATE_FORM = "informes/createOrUpdateInformeForm";
private static final String            VIEWS_ACCESS_NOT_AUTHORIZED         = "accessNotAuthorized";
private static final String            REDIRECTION_TO_CITA = "redirect:/citas/";
private static final String            INFORME_MODELO = "informe";
private final InformeService           informeService;
private final UserService              userService;
private final CitaService              citaService;
private final HistoriaClinicaService   historiaClinicaService;
private final TratamientoService       tratamientoService;
```

Eliminar comprobaciones de Boolean:


Para arreglar este tipo de errores, se ha realizado dos cosas:

En caso de tener que reutilizar la condición en otro método, crear un método auxiliar que lo haga.

```
public boolean hasCitaInforme(Cita cita){
    return this.informeService.citaHasInforme(cita);
}
```

En caso de ser una condición usada una sola vez, se ha añadido en el propio if:

```
if (!sameMedico(informe) && informe.getHistoriaClinica() == null) {
    return new ModelAndView(VIEWS_ACCESS_NOT_AUTHORIZED);
```

**Bloaters:**

**Método Largo:**

Métodos como ShowInforme eran muy largos antes de realizar una refactorización:

```java
@GetMapping(value = "/informes/{informeId}")
public ModelAndView showInforme(@PathVariable("informeId") final int informeId) {

    ModelAndView mav = new ModelAndView("informes/informeDetails");
    Informe informe = this.informeService.findInformeById(informeId).get();
    Medico medicoactual = this.userService.getCurrentMedico();
    Medico medicocorrecto = informe.getCita().getPaciente().getMedico();
    Boolean noasociado = informe.getHistoriaClinica() == null;

    if (!medicoactual.equals(medicocorrecto) && noasociado) {
        ModelAndView error = new ModelAndView("accessNotAuthorized");
        return error;
    } else {

        mav.addObject(informe);

        Cita cita = this.citaService.findCitaById(informe.getCita().getId()).get();
        Paciente paciente = cita.getPaciente();
        mav.addObject(paciente);

        Boolean canBeDeleted = informe.getCita().getFecha().plusDays(1).equals(LocalDate.now().plusDays(1)) && informe.getHistoriaClir
        mav.getModel().put("cannotbedeleted", !canBeDeleted);

        Collection<Tratamiento> tratamientos = this.tratamientoService.findTratamientosByInforme(informe);
        mav.getModel().put("editTratamientoOk", cita.getFecha().equals(LocalDate.now()));
        mav.getModel().put("tratamientos", tratamientos);

        Boolean canBeEdited = informe.getCita().getFecha().equals(LocalDate.now());
        mav.getModel().put("canbeedited", canBeEdited);

        return mav;
    }
}
```

Tras realizar diversas refactorizaciones, incluyendo una paginación para los tratamientos a raíz del tercer Profiling Realizado, el método dado queda así:

```java
@GetMapping(value = "/informes/{informeId}")
public ModelAndView showInforme(@PathVariable("informeId") final int informeId,
@PageableDefault(value = 5, page= 0) Pageable pageable){

    ModelAndView mav = new ModelAndView("informes/informeDetails");
    Informe informe = checkInformeIsPresent(informeId);

    if (!sameMedico(informe) && informe.getHistoriaClinica() == null) {
        return new ModelAndView(VIEWS_ACCESS_NOT_AUTHORIZED);
    } else {

        mav.addObject(informe);

        Cita cita = findCita(informe.getCita().getId());
        Paciente paciente = cita.getPaciente();
        mav.addObject(paciente);

        Boolean canBeDeleted = informe.getCita().getFecha().plusDays(1).equals(LocalDate.now().plusDays(1)) && informe.getHi
        mav.getModel().put("cannotbedeleted", !canBeDeleted);

        if(informe.getTratamientos() != null){
        Page<Tratamiento> tratamientos = this.tratamientoService.findTrata(informeId, pageable);

        mav.getModel().put("editTratamientoOk", cita.getFecha().equals(LocalDate.now()));
        mav.getModel().put("tratamientos", tratamientos.getContent());
        mav.getModel().put("tratapages", tratamientos.getTotalPages()-1);
            }

        Boolean canBeEdited = informe.getCita().getFecha().equals(LocalDate.now());
        mav.getModel().put("canbeedited", canBeEdited);

        return mav;
    }
```

Como podemos observar se han eliminado muchas declaraciones innecesarias y varias de las condiciones han sido movidas hacia métodos auxiliares. Cabe destacar también la inclusión de la paginación para los tratamientos tal. La refactorización realizada para la paginación está completamente desarrollada en el documento de “Profiling”.

Los métodos auxiliares que han descargado este bloater son los siguientes:

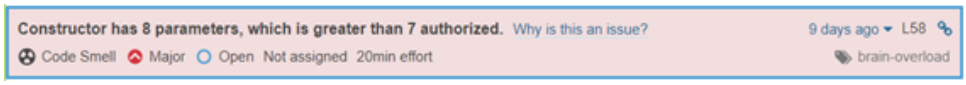
```java
public boolean sameMedico(Informe informe){
    return informe.getCita().getPaciente().getMedico().equals(this.userService.getCurrentMedico());
    }
```

```java
public Informe checkInformeIsPresent(int informeId){
    return this.informeService.findInformeById(informeId).orElse(null);
}
```

```java
@ModelAttribute("cita")
public Cita findCita(@PathVariable("citaId") final int citaId) {
    return this.citaService.findCitaById(citaId).orElse(null);
}
```

**Constructor largo:**

```java
@Autowired
public InformeController(final PacienteService pacienteService, final MedicoService medicoService, final UserServic
    final HistoriaClinicaService historiaClinicaService, final InformeService informeService, final TratamientoServ
    this.informeService = informeService;
    this.pacienteService = pacienteService;
    this.medicoService = medicoService;
    this.userService = userService;
    this.citaService = citaService;
    this.historiaClinicaService = historiaClinicaService;
    this.tratamientoService = tratamientoService;
}
```




Dado que el constructor estaba recibiendo demasiados parámetros (Y algunos de ellos completamente innecesarios, por lo que también hablamos de un Disposable), se han eliminado aquellos que no eran usados:

```java
@Autowired
public InformeController(final UserService userService, final AuthoritiesService authoritiesService,
    final CitaService citaService,final HistoriaClinicaService historiaClinicaService, final InformeService informeService,
    final TratamientoService tratamientoService) {
    this.informeService = informeService;
    this.userService = userService;
    this.citaService = citaService;
    this.historiaClinicaService = historiaClinicaService;
    this.tratamientoService = tratamientoService;
}    alvalfric, 2 months ago • Fixed errors after spring 2
```

**Arreglo de Bugs:**

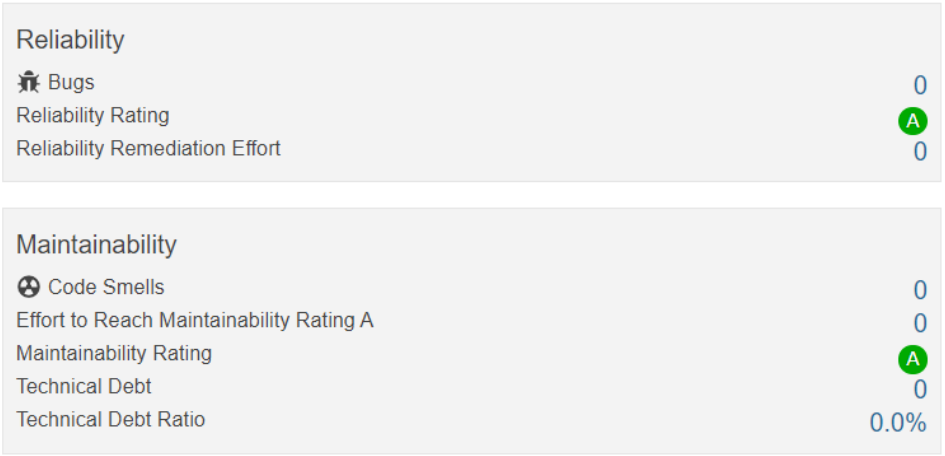
Dado el uso del Tipo Optional, hemos optado por aplicar la misma solución que en Paciente Controller y comprobar si el objeto está presente antes de asignarlo en una variable:

```java
public Informe checkInformeIsPresent(int informeId){
    return this.informeService.findInformeById(informeId).orElse(null);
}
```

**Conclusiones de la refactorización de Informe Controller:**

Aunque se han refactorizado la clase Informe Controller completa, sólo figuran aquí los ejemplos a destacar.

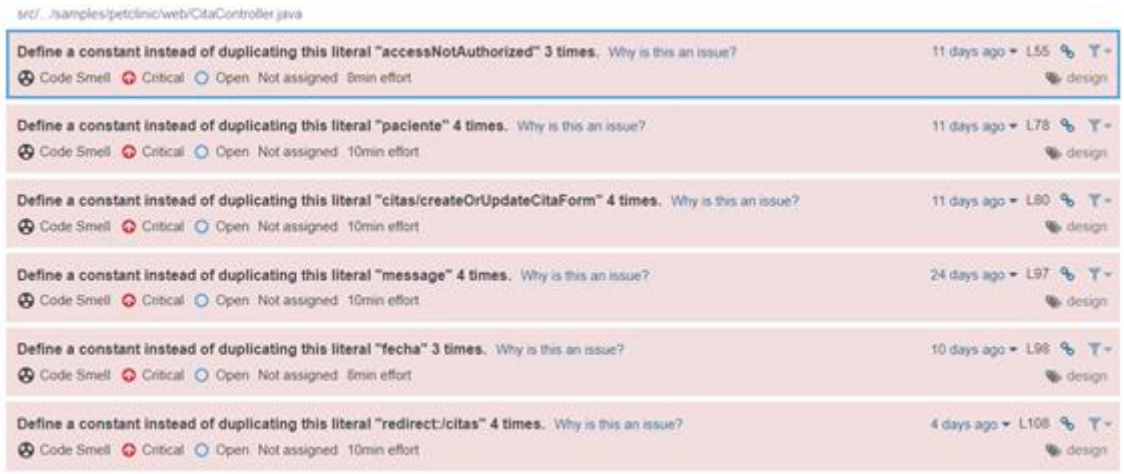
Tras los cambios realizados está es la Fiabilidad y Mantenibilidad de la clase Refactorizada:




Quedando una deuda técnica de 0.

**Refactor CitaController**

Una vez ejecutado SonarCloud observamos que teníamos 17 malos olores de nivel crítico, así que decidimos centrarnos en su refactorización correspondiente. De este modo, el análisis llevado a cabo por SonarCloud nos mostraba que la clase CitaController poseía casi un tercio de estos malos olores críticos tal como podemos ver en la siguiente imagen, por lo que decidimos solucionarlos.




Como podemos observar, estos malos olores críticos son en su totalidad de literales duplicados que se invocan en múltiples ocasiones, por lo que resulta una buena práctica convertirlos en constantes individuales que se invoquen cuando sea necesario. Se trata de un mal olor de tipo “Código duplicado”. Así pues, observemos un ejemplo otorgado por SonarCloud donde se aprecian estos literales duplicados:



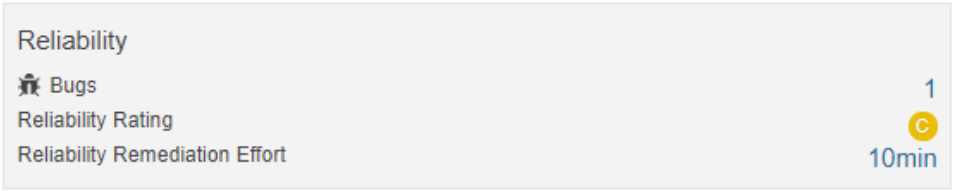

En estas imágenes, SonarCloud nos hace observar la múltiple repetición de un literal que podría ser declarado como constante. Este tipo de mal olor crítico se repite hasta un total de seis veces en esta clase, por lo que en pos de refactorizar nuestro código, procedemos a solucionar estos seis malos olores declarando los parámetros indicados como constantes. De este modo, una vez declaradas las constantes que producen estos malos olores, el código añadido es el siguiente:

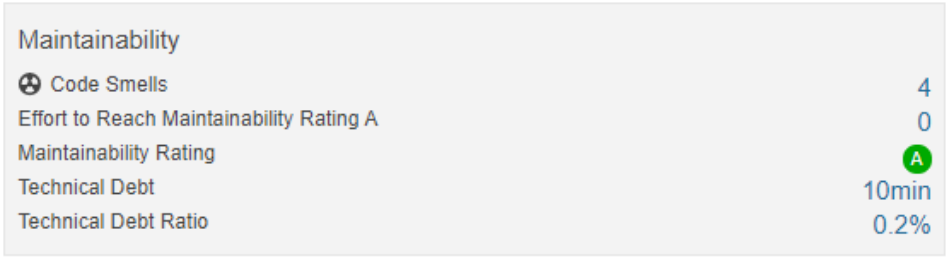
```
public class CitaController {

    private static final String accessNotAuthorized = "accessNotAuthorized";
    private static final String pacienteModelName = "paciente";
    private static final String VIEW_CREATE_OR_UPDATE_FORM = "citas/createOrUpdateCitaForm";
    private static final String message = "message";
    private static final String fecha = "fecha";
    private static final String VIEW_CITAS = "redirect:/citas";
```

Con este fragmento añadido y la correspondiente sustitución de los métodos que usan estos parámetros, los malos olores críticos de esta clase deberían desaparecer. Para comprobarlo, volvemos a ejecutar SonarCloud y el resultado es el siguiente:


Efectivamente, observamos que los malos olores críticos de CitaController han desaparecido después de la refactorización del código, con lo que podemos concluir que la refactorización ha sido realizada de forma efectiva.

Tras realizar este refactor, los datos proporcionados por SonarCloud sobre fiabilidad y mantenibilidad son los siguientes:



**Conclusiones finales:**

Acorde a la información proporcionada por SonarCloud, se resolvieron un total de 160 code smells y bugs:


Se puede concluir que las refactorizaciones detalladas en este documento han sido efectivos quedando finalmente los siguientes datos de mantenibilidad: