DP2

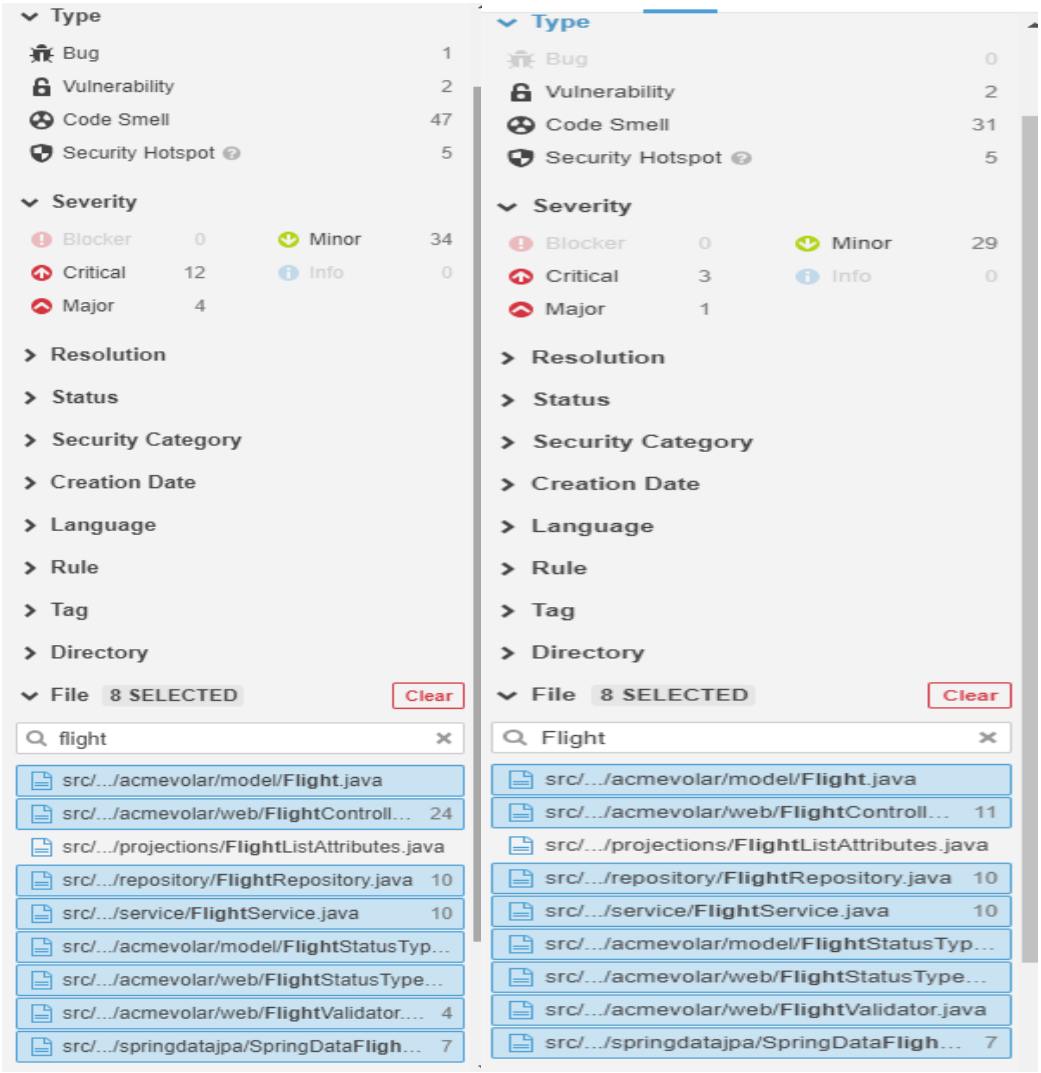
REFACTORING

Grupo - G2-1
Miembros del grupo
DANIEL ARELLANO MARTÍNEZ
EDUARDO MIGUEL BOTÍA DOMINGO
JOSE MARTÍN SÁNCHEZ
JUAN NOGUEROL TIRADO
JOSÉ MANUEL SÁNCHEZ RUIZ
JAVIER VÁZQUEZ ZAMBRANO

ÍNDICE

Flight

Para esta entidad, nos hemos centrado en solventar malos olores y un bug, localizados específicamente en el controlador y un validador. Antes de pasar a enumerar cambios, mostramos una captura de pantalla de un análisis anterior y otro posterior a los cambios y refactorizaciones que hemos llevado a cabo.



A continuación, analizamos el bug que hemos encontrado, que podría haber causado un error de tipo *NullPointerException*.

```
    if (flight.getPublished() || flight.getAirline().getName().equals(airline.getName())) {
    if (flight.getPublished() || airline != null &&
flight.getAirline().getName().equals(airline.getName())) {
```

Por otra parte, hemos solucionado malos olores, de los que destacamos los que mostramos a continuación.

- Reutilización de código

En diversas ocasiones, reutilizábamos el nombre de las variables o cadenas de texto con el mismo contenido una y otra vez, y gracias al análisis de Sonar, nos hemos percatado de ello y hemos realizado un reajuste definiendo cadenas de texto *final*, como el que mostramos a continuación.

```
  private static final String FLIGHTS_WORD                = "flights";
   private static final String FLIGHT_LIST_URI            = "flights/flightList";
   private static final String FLIGHT                     = "flight";
   private static final String REFERENCE                  = "reference";
   private static final String DEPARTES                   = "departes";
   private static final String LANDS                      = "lands";
   private static final String AIRPORTFULLOFPLANES        = "airportFullOfPlanes";
   private static final String AIRPORTFULLOFPLANES_MESSAGE = "This airport is
full of planes this day";
```

A continuación, mostramos un ejemplo de un punto del código en el que hemos efectuado dicha refactorización.

```
model.put("flights", flights);
  return "flights/flightList";
model.put(FLIGHTS_WORD, flights);
return FLIGHT_LIST_URI;
```

- No respetar convenciones de estilo de código

Por otra parte, ha habido una variable que no hemos nombrado siguiendo las convenciones y estándares de nombrado, en concreto el siguiente ejemplo:

```
List<Boolean> opciones_publicao = new ArrayList<>();
List<Boolean> opcionesPublicao = new ArrayList<>();
```

- Comentarios de código

Por otra parte, también hemos eliminado fragmentos de código que no estaban en uso, como el siguiente ejemplo.

```
// flight.getPlane().addFlight(flight);
```

Book

Se han resuelto 3 tipos de Code Smells:

- Literales repetidos:
  Define a constant instead of duplicating this literal "flight".

```
model.put("flight", flight);
```

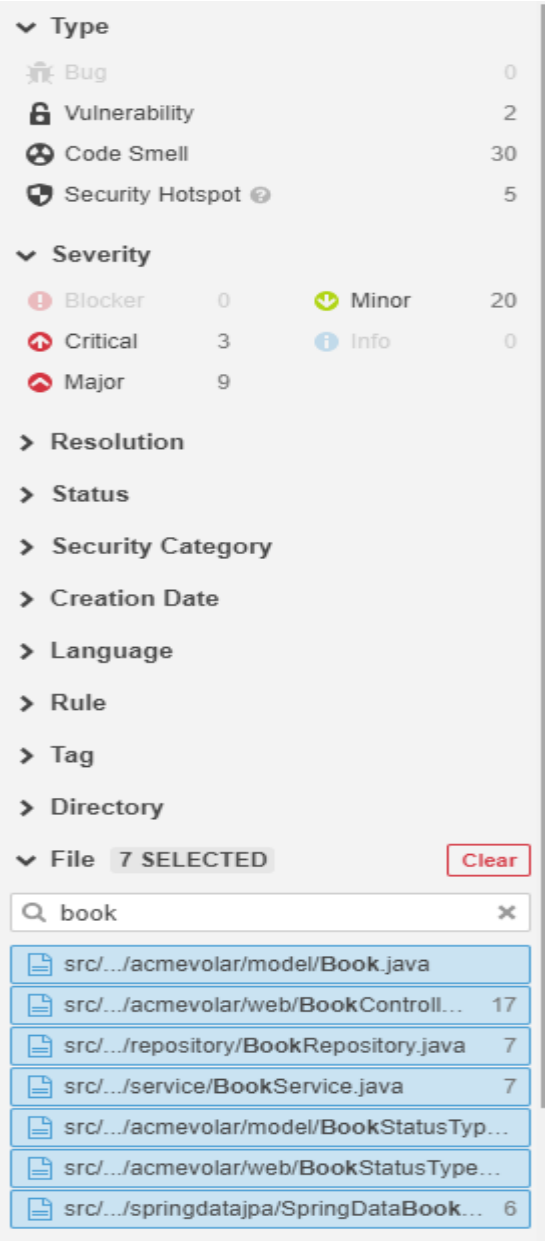
```
private static final String FLIGHT_ATTRIBUTTE = "flight";
model.put(FLIGHT_ATTRIBUTTE, flight);
```

- Comentarios de código:
  This block of commented-out lines of code should be removed.

```
//     Client client = SecurityContextHolder.getContext().getAuthentication();
```

- Repositorios no usados:
  Remove this unused private field.

```
private BookRepository                  bookRepository;
private AirlineRepository               airlineRepository;
private SpringDataPlaneRepository   springPlaneRepository;

@Autowired
public BookService(final BookRepository bookRepository, final
SpringDataPlaneRepository springPlaneRepository, final AirlineRepository
airlineRepository) {
        this.bookRepository = bookRepository;
        this.springPlaneRepository = springPlaneRepository;
        this.airlineRepository = airlineRepository;
}
```



Airline

Se han resuelto 3 tipos de Code Smells (pongo un ejemplo de cada tipo):

- Aserciones incompletas:
  Complete the assertion.

```
assertThat(airline.getName().equals("Sevilla Este Airways"));
```

```
assertThat(airline.getName().equals("Sevilla Este Airways")).isTrue();
```

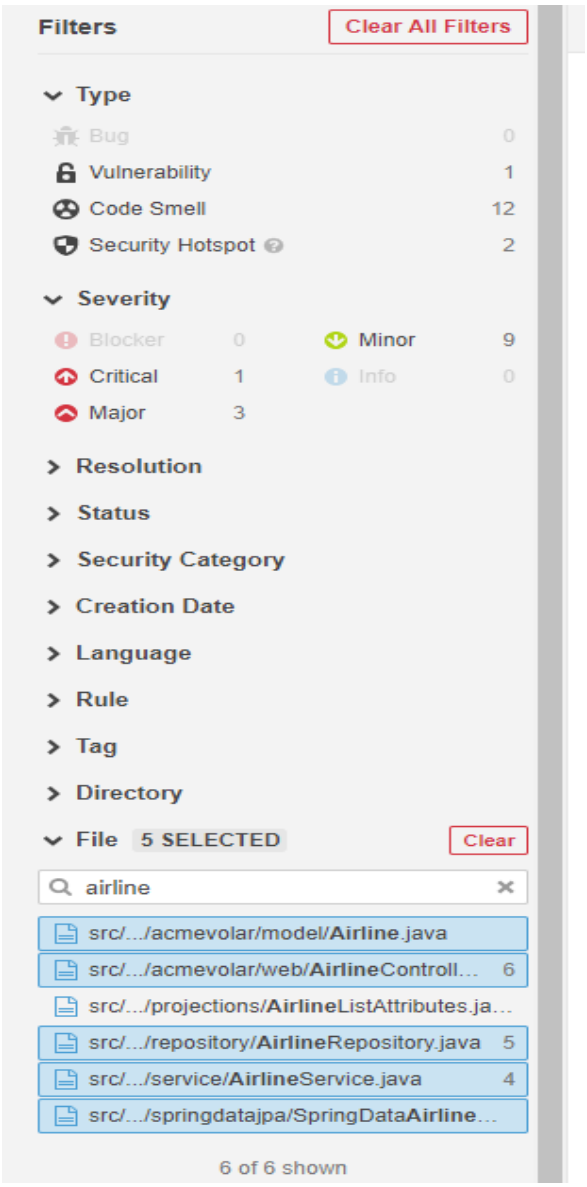
- Comentarios de código:
  This block of commented-out lines of code should be removed.

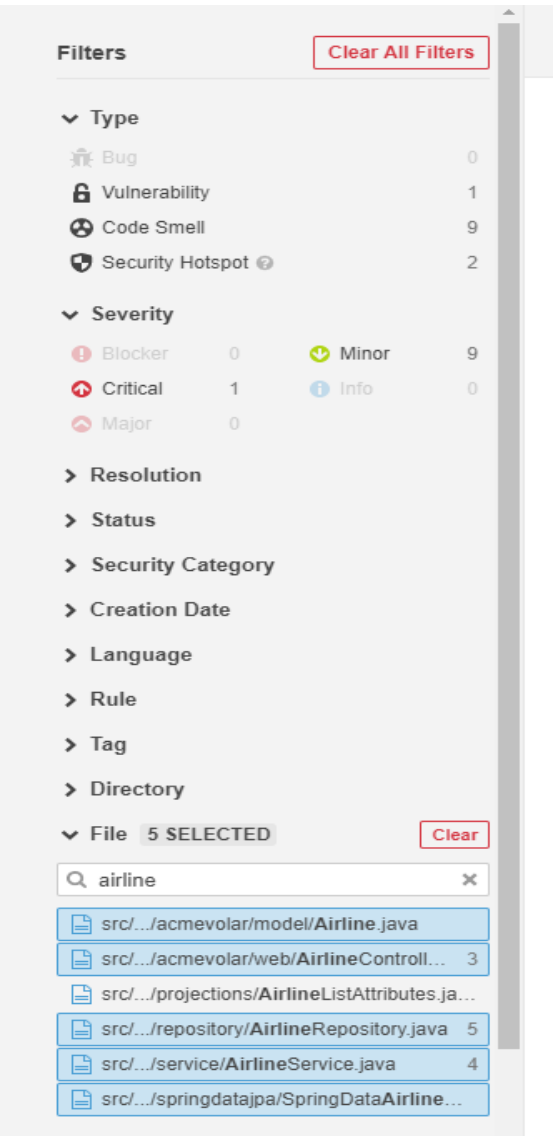
```
//     Client client = SecurityContextHolder.getContext().getAuthentication();
```

Aquí se pueden observar el cambio realizado:

Client

Se ha resuelto 1 tipo de Code Smells:



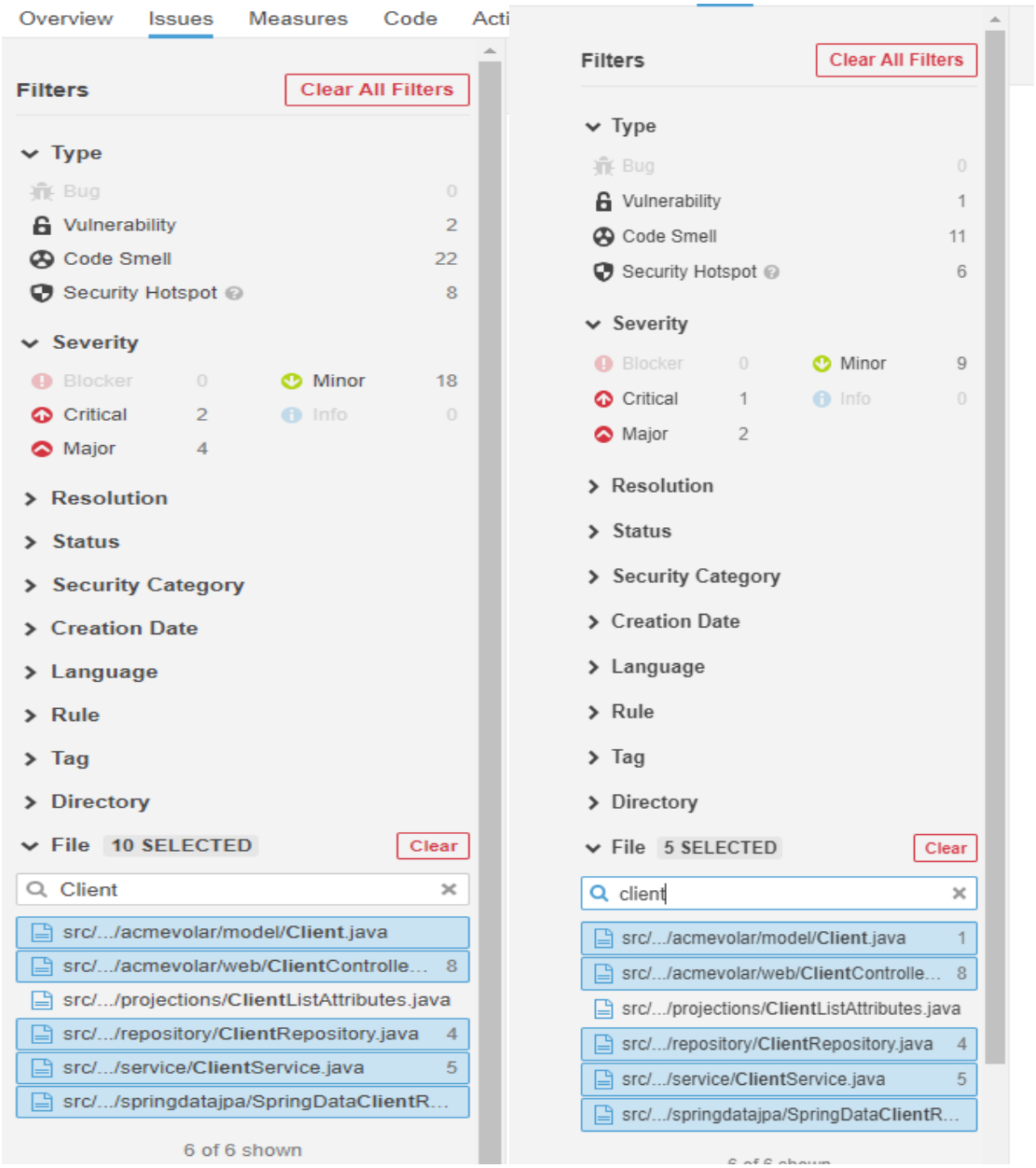


- Aserciones incompletas:
  Complete the assertion.

```
assertThat(client1.getName().length()>0);
```

```
assertThat(client1.getName().length()>0).isTrue();
```

Aquí se pueden observar el cambio realizado:



Plane

Se han resuelto 4 tipos de Code Smells:

- Aserciones incompletas:
  Define a constant instead of duplicating this literal "flight".

```
assertThat(nActualAviones.equals(nPrevioAviones));
```

```
assertThat(nActualAviones.equals(nPrevioAviones)).isTrue();
```

- Comentarios de código:
  This block of commented-out lines of code should be removed.

```
//private static final String REQUIRED = "required";
```

- Literales repetidos:

  Define a constant instead of duplicating this literal "plane" 3 times

```
private String  planeString = "plane";
```

```
model.put("plane", plane);
model put(planeString, plane)
```

- Sentencias if mal colocadas

  Move this "if" to a new line or add the missing "else"

```
} if(plane.getLastMaintenance()==null) {
} else if (plane.getLastMaintenance() == null) {
```

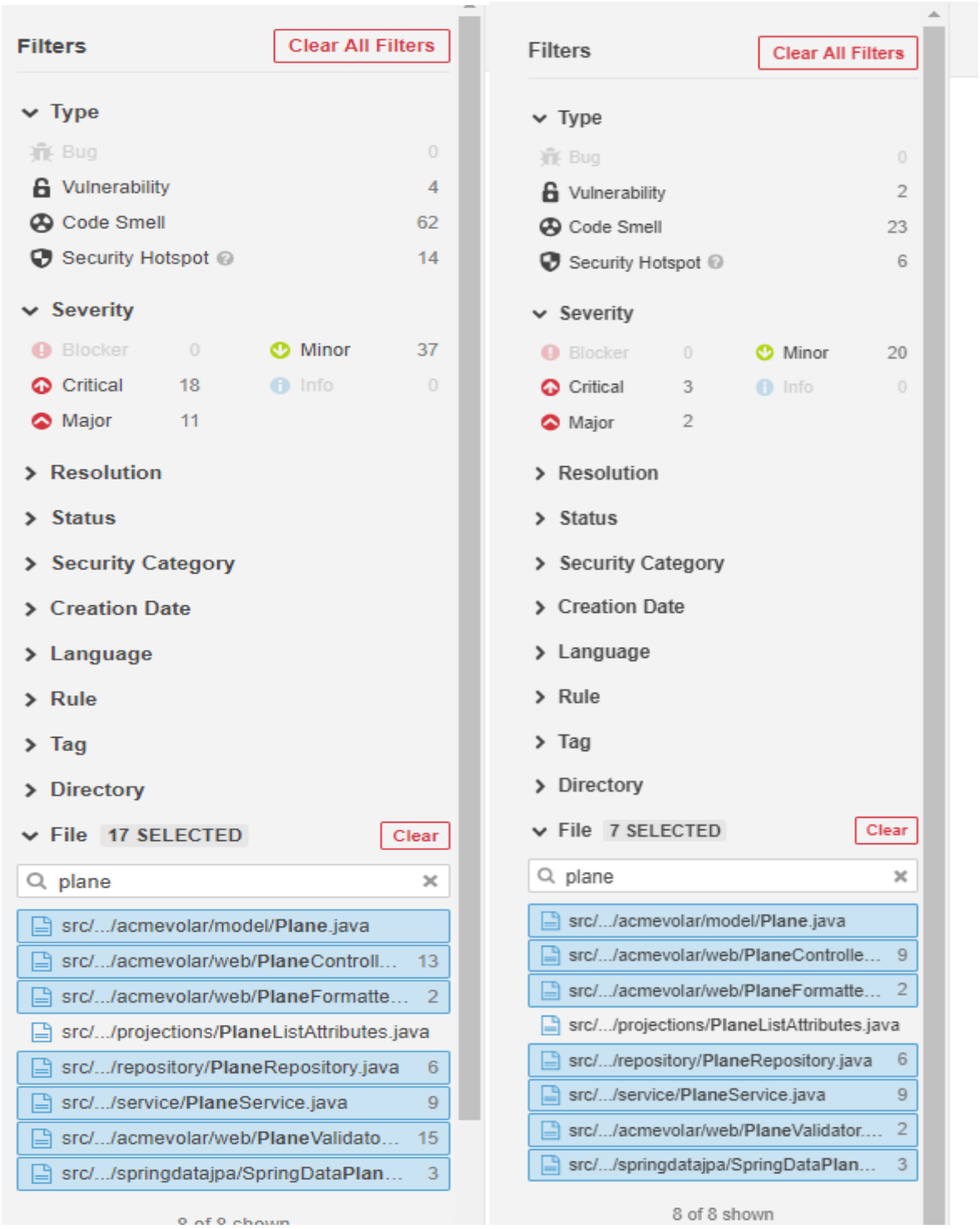
- Excepciones generales:

  Define and throw a dedicated exception instead of using a generic one.

```
if (!planes.contains(plane)) {
        throw new Exception("No está autorizado para modificar un vuelo que
        no es suyo.");
}
```

```
if (!planes.contains(plane)) {
        throw new AccessDeniedException("No está autorizado para
        modificar un vuelo que no es suyo.");
}
```

Aquí se pueden observar el cambio realizado:



Security Configuration

Se ha resuelto 1 tipo de Code Smells:

- Literales repetidos:

Define a constant instead of duplicating this literal "plane" 3 times

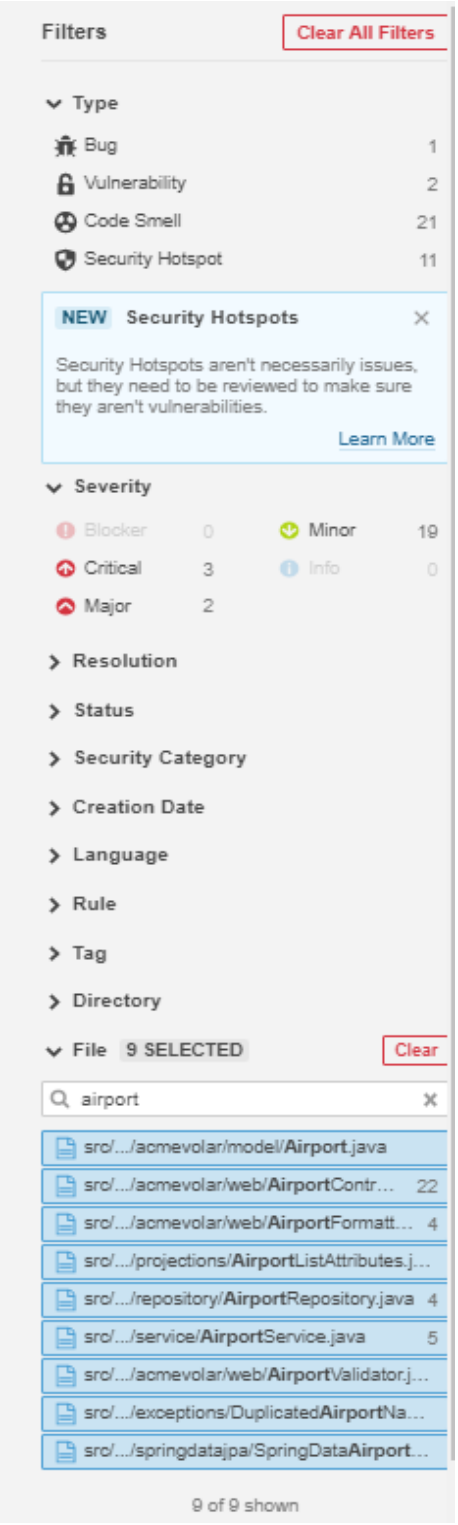
```
private String  airlineString    = "airline";
private String  clientString     = "client";
```

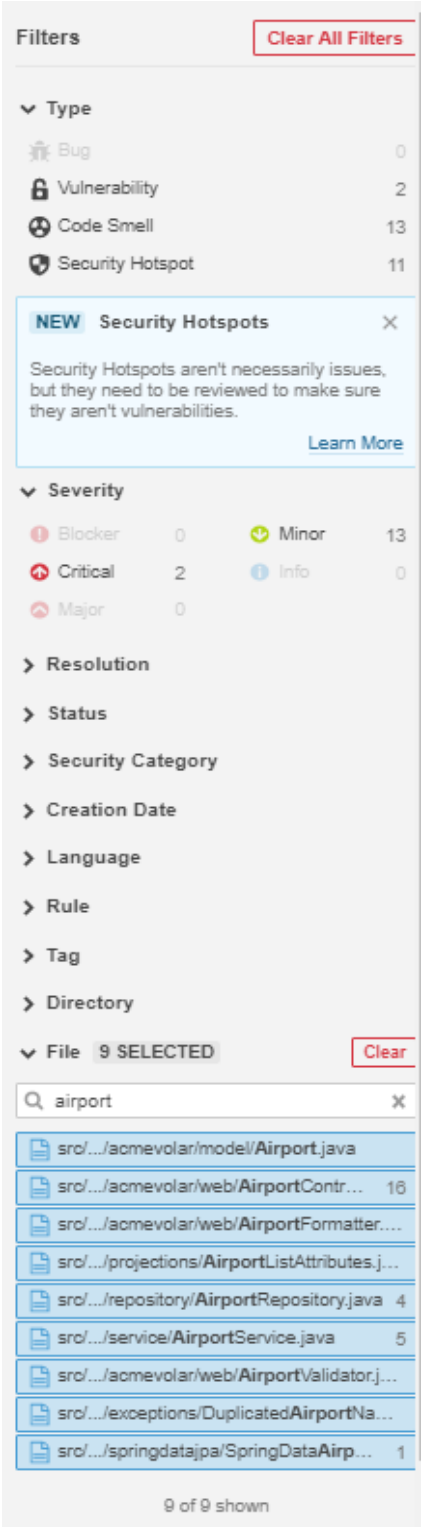
```
.antMatchers("/my_planes").hasAuthority("airline").antMatchers("/planes/new").hasAu
thority("airline").antMatchers("/planes/{^[\\d]$}").hasAnyAuthority("client",
"airline").antMatchers("/planes/{^[\\d]$}/edit").hasAuthority("airline")
```

```
.antMatchers("/my_planes").hasAuthority(this.airlineString).antMatchers("/planes/new
").hasAuthority(this.airlineString).antMatchers("/planes/{^[\\d]$}").hasAnyAuthority(thi
s.clientString,
this.airlineString).antMatchers("/planes/{^[\\d]$}/edit").hasAuthority(this.airlineString)
```

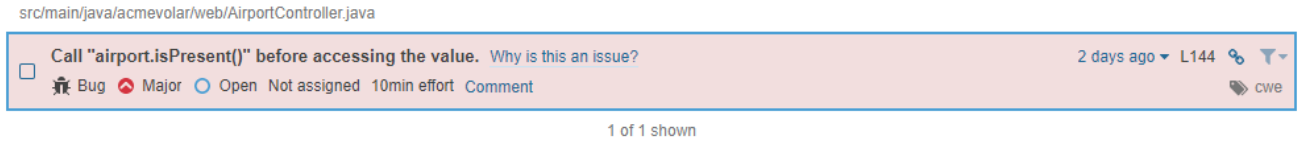
Airport

Para esta entidad, hemos resuelto varios malos olores y un bug. Aquí se presentan dos capturas de pantalla. La primera muestra el análisis de SonarCloud antes de la refactorización. La segunda muestra el análisis después de esta.





A continuación, se muestra una imagen con los detalles del bug.



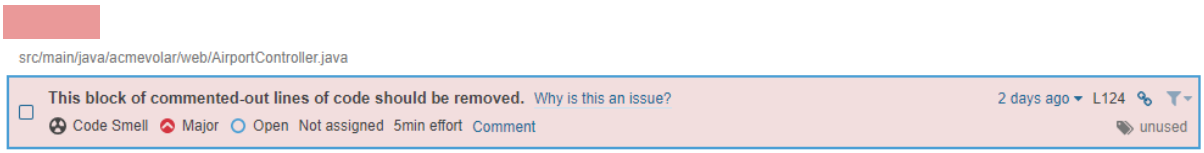
Este se encontraba en la clase AirportController y era causado por el código:

```
@PreAuthorize("hasAuthority('airline')")
@GetMapping(value = "/airports/{airportId}/delete")
public String deleteAirport(@PathVariable("airportId") final int airportId) throws
NonDeletableException {

        Optional<Airport> airport = this.airportService.findById(airportId);

        boolean deletable = this.flightService.findFlights().stream().noneMatch(f-
        >f.getDepartes().getAirport().equals(airport.get()) ||
        f.getLands().getAirport().equals(airport.get()));

        if (deletable) {
                this.airportService.deleteAirport(airport.get());
        } else {
                throw new NonDeletableException();
        }
        return "redirect:/airports";

    }
```

Este bug indicaba un acceso a un objeto Optional sin verificar antes que este objeto estuviese presente, es decir, que no fuese nulo. Esto se consigue con .isPresent(), que devuelve true si el objeto no es nulo. Así, se añadió airport.isPresent() a la sentencia if:

```
        if (deletable && airport.isPresent()) {
                this.airportService.deleteAirport(airport.get());
        } else {
                throw new NonDeletableException();
        }
        return "redirect:/airports";
```

Una vez resuelto el bug, también se prestó especial atención a un mal olor catalogado como de “Major Severity”. Aquí se presenta una captura de pantalla de este mal olor:



Indicaba que existía un comentario que incluía código, el cuál fue eliminado:

```
//Airport airportToUpdate = this.airportService.findAirportById(airportId);
```

El resto de elementos solucionados fueron malos olores, de los cuales algunos se presentan a continuación, junto a sus respectivas soluciones:

- Diamond operator (“<>”)

```
Collection<AirportListAttributes> airports = new ArrayList<AirportListAttributes>();
```

```
Collection<AirportListAttributes> airports = new ArrayList<>();
```

- Definir constante en lugar de repetir “airport”

```
private static final String AIRPORT_CONSTANT = "airport";
```

```
model.put(“airport”, airport);
```

```
model.put(AirportController.AIRPORT_CONSTANT, airport);
```

- Usar .isEmpty() para comprobar si una lista está vacía

```
airport.getName().size() != 0
```

```
!airport.getName().isEmpty()
```

- Sustituir una negación junto a .anyMatch() por .noneMatch

```
boolean deletable = !this.flightService.findFlights().stream().anyMatch(f->f.getDepartes().getAirport().equals(airport.get())||f.getLands().getAirport().equals(airport.get()));
```

```
boolean deletable = this.flightService.findFlights().stream().noneMatch(f->f.getDepartes().getAirport().equals(airport.get())||f.getLands().getAirport().equals(airport.get()));
```

- Eliminar imports innecesarios (Minor)

Runway

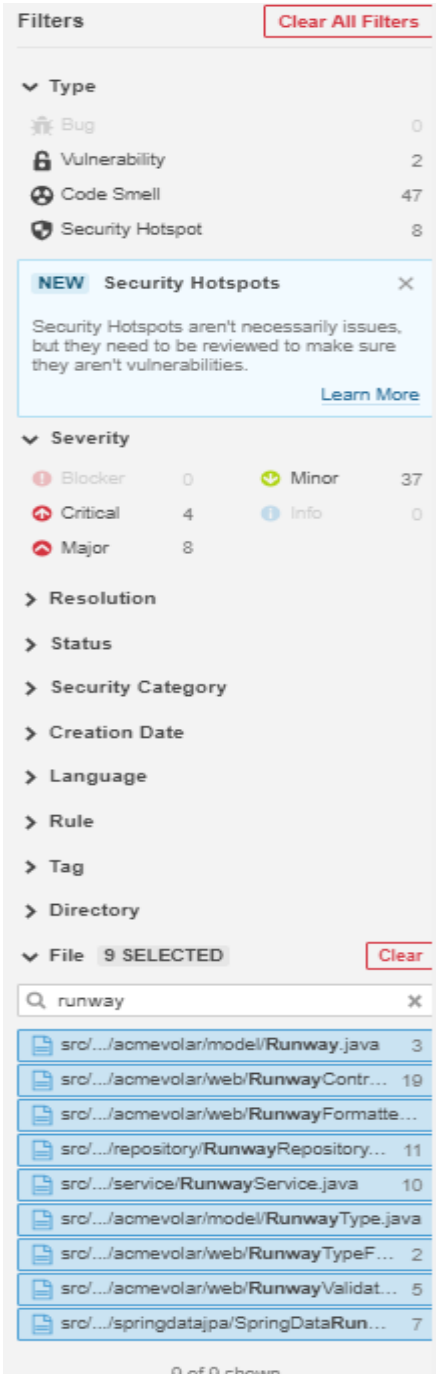
Para esta entidad, el análisis de SonarCloud no ha detectado ningún bug potencial, pero ha indicado la existencia de varios malos olores. Nos hemos centrado en solucionar estos. Estas

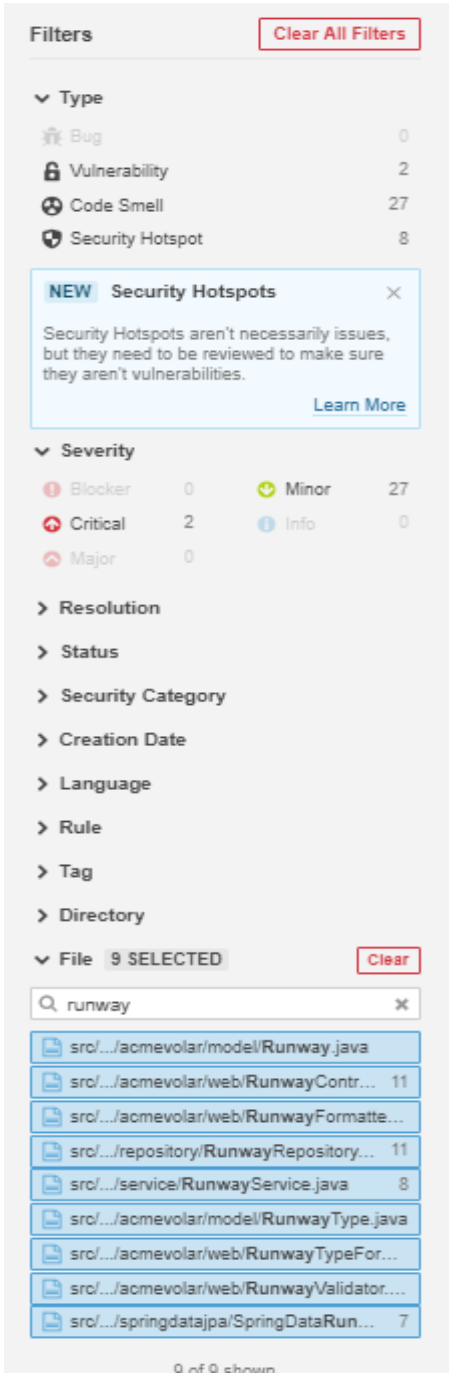
capturas de pantalla muestran los resultados de los análisis deSonarCloud antes y después de la refactorización:

Aquí se enumeran algunos de los malos olores y sus soluciones:

- Definir constante en lugar de repetir “runway” (Critical)

```
private static final String RUNWAY_CONSTANT = "runway";
```





```
model.put(“runway”, runway);
```

```
model.put(RunwayController.RUNWAY_CONSTANT, runway);
```

- Eliminar comentarios que contienen código:

```
//Optional<Airport> airport = this.airportService.findById(airportId);
```

- Usar .isEmpty() para comprobar si una lista está vacía

```
runway.getName().size() != 0
```

```
!runway.getName().isEmpty()
```

- Eliminar imports innecesarios

- Eliminar argumentos que no se estén usando en algún método

```
public void insertData(final Map<String, Object> model,
@PathVariable("airportId") int airportId) {

        List<RunwayType> runwayTypes =
        this.runwayService.findRunwaysTypes();

        model.put("runwayTypes", runwayTypes);
    }
```

```
public void insertData(final Map<String, Object> model,
@PathVariable("airportId") int airportId) {

        List<RunwayType> runwayTypes =
        this.runwayService.findRunwaysTypes();

        model.put("runwayTypes", runwayTypes);
    }
```

Todos los tests

Se ha resulto un tipo de Code Smells para todos los tests en general:

- Clases públicas:

Remove this 'public' modifier.

```
public class ClientServiceTests {
class ClientServiceTests {
```