

**ESCUOLA COLOMBIANA DE INGENIERIA
JULIO GARAVITO**

**IT SECURITY AND PRIVACY
GROUP 1L**

LABORATORY 10

**SUBMITTED BY:
JUAN PABLO FERNANDEZ GONZALES
MARIA VALENTINA TORRES MONSALVE**

1

**SUBMITTED TO:
Eng. DANIEL ESTEBAN VELA LOPEZ**

**BOGOTÁ D.C.
DATE:
31/03/2025**

Introduction

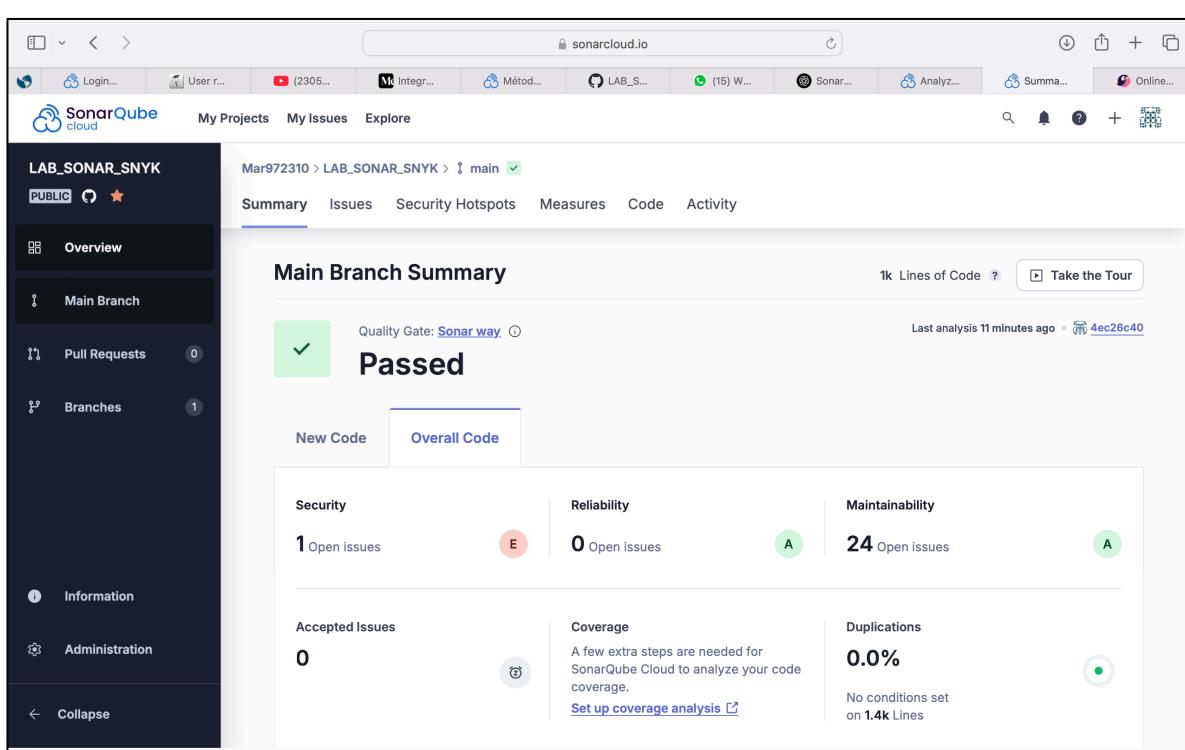
This report analyzes the source code of a real estate property management system to identify security vulnerabilities and ensure clean, maintainable code. The project implements a **CRUD (Create, Read, Update, Delete) system** using a **backend in Spring Boot** and a **frontend developed with HTML, CSS and JavaScript**. The data is stored in a **MySQL database**.

To assess code security and quality, **SonarCloud** is used to detect code smells, bugs, and vulnerabilities, and **Snyk** to identify security risks in dependencies and configurations. The goal is to improve project reliability, performance, and safety, addressing potential weaknesses and following development best practices.

The application we will be using for the analysis is in the repository:
<https://github.com/Mar972310/LAB SONAR SNYK.git>

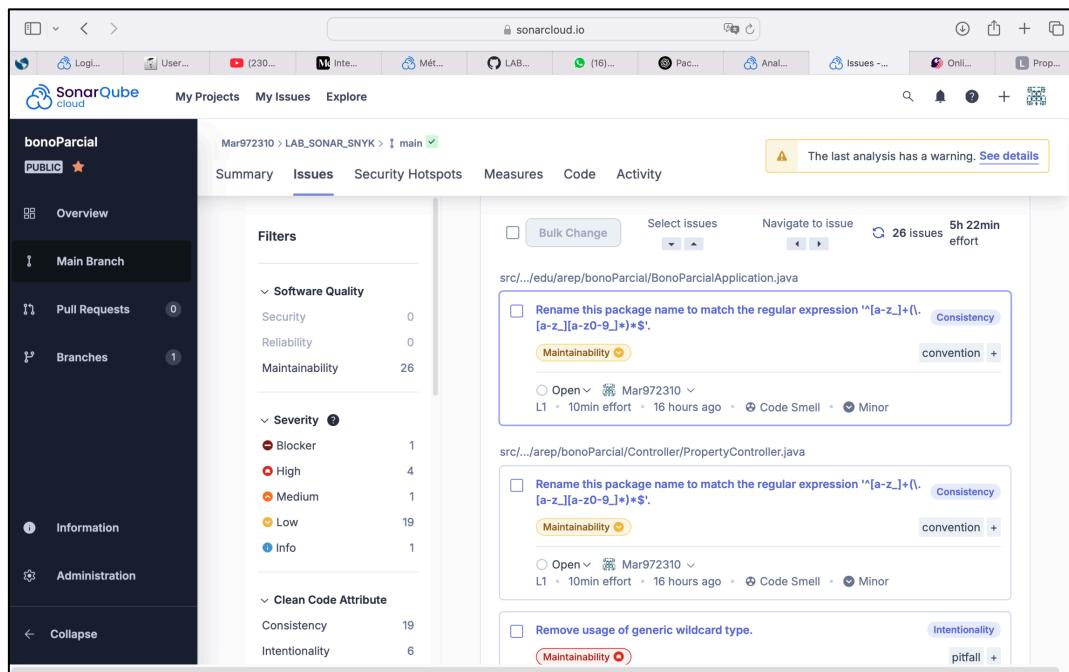
SonarCloud Analysis

The analysis carried out with SonarCloud to the Mar972310/LAB SONAR SNYK repository, gave us as an answer after having analyzed the code of the application that 24 maintainability issues and one security issue.



- Correcting the naming of packages according to Java conventions

Among the maintainability issues we have that 9 of those refer to ***Rename this package name to match the regular expression '^/[a-z_]+(\.[a-z_]/[a-z0-9_]*\$'.*** This issue indicates that the name of the packets does not comply with the convention set in the regular expression $^/[a-z_]+(\.[a-z_]/[a-z0-9_]*$$, which requires package names to begin with lowercase letters or underscores and that sub-packages, separated by periods, also begin with a lowercase letter or underscore, allowing only lowercase letters, numbers, and underscores in the rest of the name. No capitalization, hyphens (-), or spaces are allowed. For example, com. ExampleApp is incorrect because it uses all caps, whereas com.example_app would be a valid version.



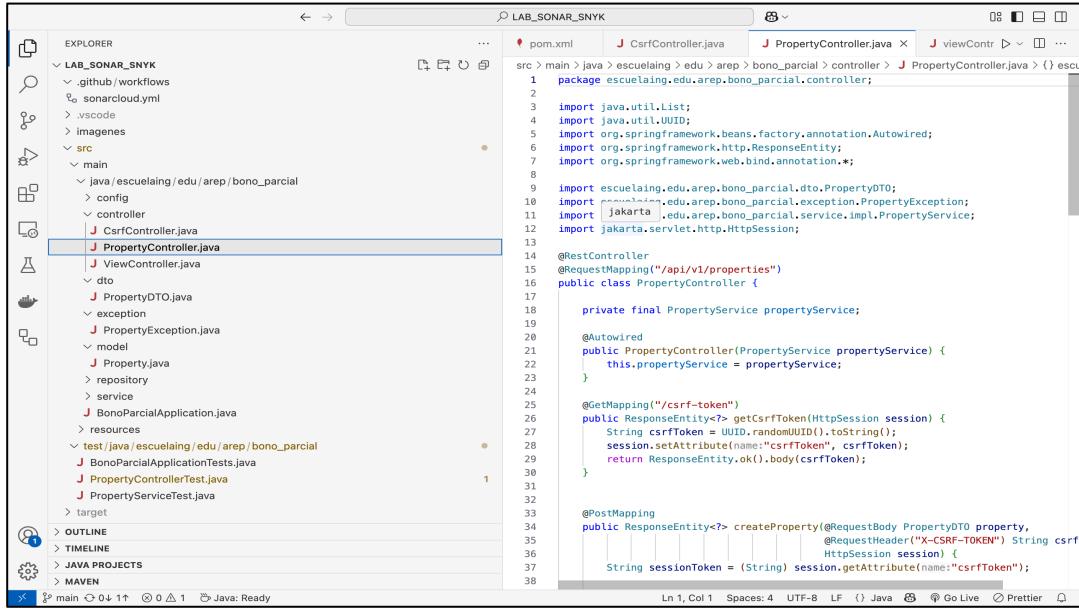
The screenshot shows the SonarCloud interface for the project 'bonoParcial'. The main navigation bar includes links for Logins, User, Issues (230...), Metrics, LAB..., (16)..., Pac..., Anal..., Issues..., Onlin..., and Prop... The 'Issues' tab is selected. A summary bar indicates 26 issues and 5h 22min effort. The left sidebar shows the project structure with Main Branch, Pull Requests (0), Branches (1), Information, and Administration. The main content area displays maintainability issues under the 'Filters' section. One specific issue is highlighted: 'Rename this package name to match the regular expression '^/[a-z_]+(\.[a-z_]/[a-z0-9_]*\$'. It is categorized as 'Maintainability' and 'Consistency'. Other issues listed include 'Remove usage of generic wildcard type.' and 'Open Mar972310'.

3

In the case of our application we have that the names of the packages are as follows:

- escuelaing.edu.arep.bonoPartial
- escuelaing.edu.arep.bonoPartial.Controller
- escuelaing.edu.arep.bonoPartial.DTO
- escuelaing.edu.arep.bonoPartial.Exception
- escuelaing.edu.arep.bonoPartial.Repository
- escuelaing.edu.arep.bonoPartial.ServiceImpl
- escuelaing.edu.arep.bonoPartial.Service
- escuelaing.edu.arep.bonoPartial.model

What we will do is correct **PartialBonus** for **bono_parcial**, the subpackages that start with a capital letter, for example, Controller, DTO, Exception, Repository, Service and Impl we will leave them as follows: controller, dto, exception, repository, service and impl



```

1 package escuelaing.edu.arep.bono_parcial.controller;
2
3 import java.util.List;
4 import java.util.UUID;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.ResponseEntity;
7 import org.springframework.web.bind.annotation.*;
8
9 import escuelaing.edu.arep.bono_parcial.dto.PropertyDTO;
10 import escuelaing.edu.arep.bono_parcial.exception.PropertyException;
11 import jakarta.edu.arep.bono_parcial.service.impl.PropertyService;
12 import jakarta.servlet.http.HttpSession;
13
14 @RestController
15 @RequestMapping("/api/v1/properties")
16 public class PropertyController {
17
18     private final PropertyService propertyService;
19
20     @Autowired
21     public PropertyController(PropertyService propertyService) {
22         this.propertyService = propertyService;
23     }
24
25     @GetMapping("/csrf-token")
26     public ResponseEntity<?> getCsrfToken(HttpServletRequest session) {
27         String csrfToken = UUID.randomUUID().toString();
28         session.setAttribute(name:"csrfToken", csrfToken);
29         return ResponseEntity.ok().body(csrfToken);
30     }
31
32     @PostMapping
33     public ResponseEntity<?> createProperty(@RequestBody PropertyDTO property,
34                                                 @RequestHeader("X-CSRF-TOKEN") String csrf
35                                                 HttpSession session) {
36         String sessionToken = (String) session.getAttribute(name:"csrfToken");
37     }
38

```

4

- Correction of naming in classes according to Java conventions

Another problem related to the convention established by the regular expression `^[_a-zA-Z]+[._][a-zA-Z_]*$` is that the `viewController.java` class does not comply with Java best practices, since its name does not start with a capital letter. According to naming conventions in Java, class names should start with an uppercase letter, so the proper correction would be `ViewController.java`.



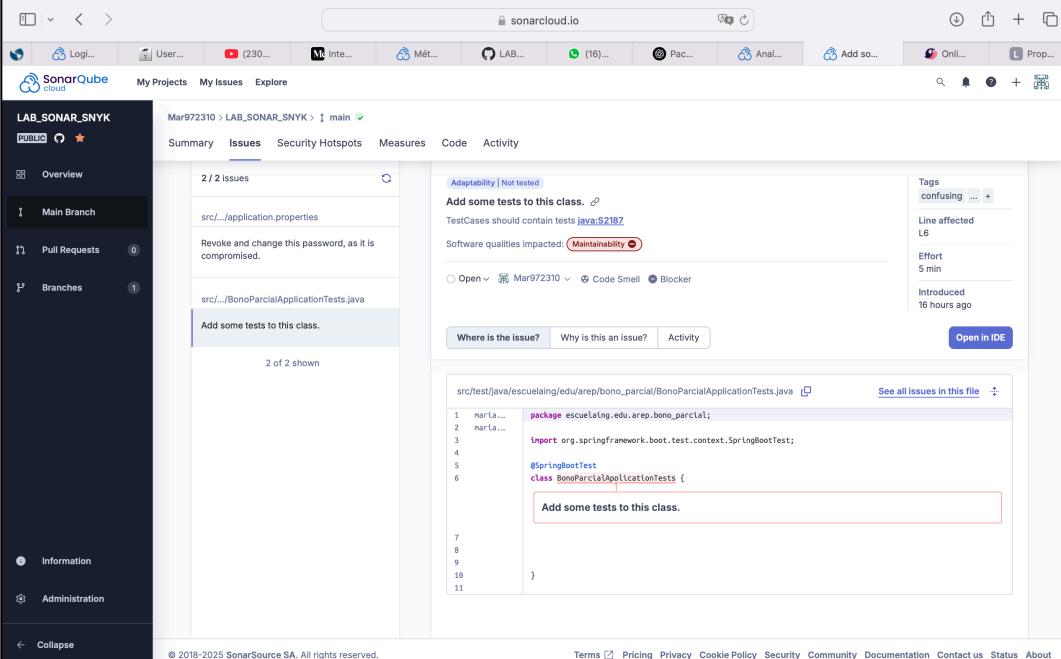
```

1 package escuelaing.edu.arep.bono_parcial.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5
6 /**
7 * Controller to handle view navigation.
8 */
9 @Controller
10 public class ViewController {
11
12     /**
13      * Maps the "/home" endpoint to return the "inmobiliaria" view.
14      *
15      * @return The name of the view template to be rendered.
16      */
17     @GetMapping("/home")
18     public String home() {
19         return "inmobiliaria";
20     }
21 }

```

- Deleting classes without code or functionality

In the test package of the application we have a class in which we are not implemented and there is only the empty class.



The screenshot shows the SonarCloud interface for the project 'LAB SONAR SNYK'. It displays a 'Issues' tab with 2 / 2 issues found. One issue is highlighted in the center: 'src.../BonoParcialApplicationTests.java' with the message 'Add some tests to this class.' A red box highlights this message. The code editor shows the Java file content:

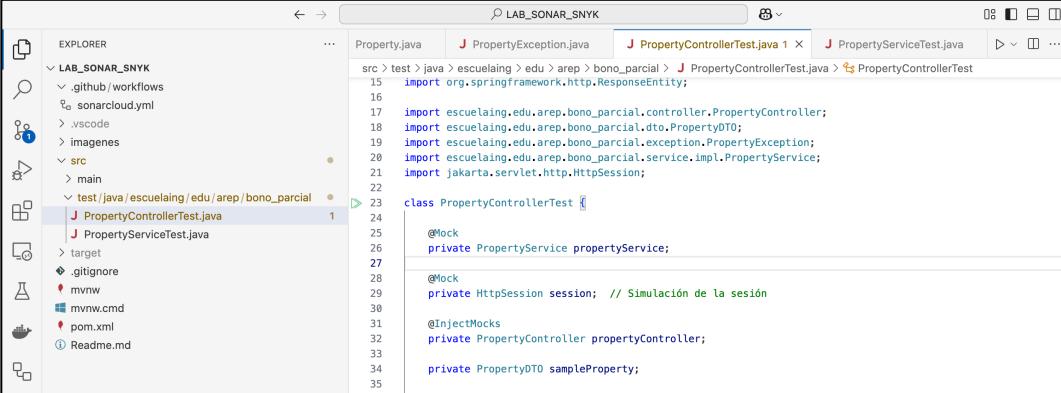
```

1 maria...
2 maria...
3
4
5
6 package escuelaling.edu.arep.bono_parcial;
import org.springframework.boot.test.context.SpringBootTest;
@SpringBootTest
class BonoParcialApplicationTests {
}

```

5

As the tests were implemented in other classes, we can eliminate this one and avoid this issue and a more orderly project structure with what is necessary.



The screenshot shows the VS Code IDE with the project 'LAB SONAR SNYK' open. The 'EXPLORER' sidebar shows files like '.github/workflows', 'sonarcloud.yml', 'src/main/test/java/escuelaling/edu/arep/bono_parcial/PropertyControllerTest.java', and 'pom.xml'. The code editor shows the content of 'PropertyControllerTest.java':

```

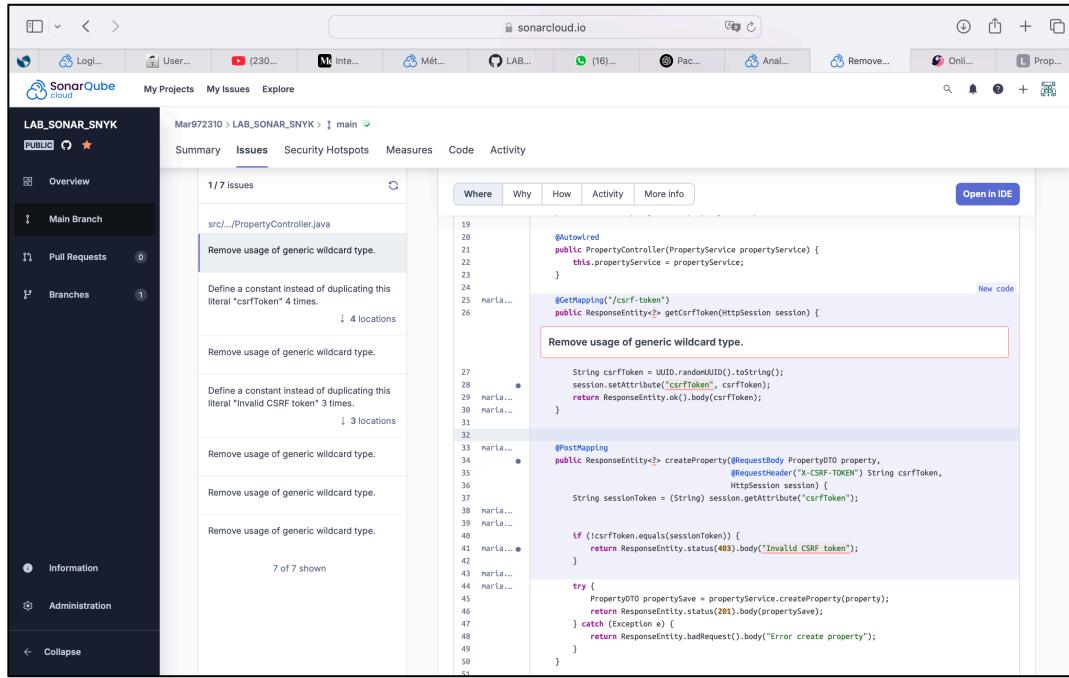
src > test > java > escuelaling > edu > arep > bono_parcial > J PropertyControllerTest.java 1 X J PropertyServiceTest.java
15 import org.springframework.http.ResponseEntity;
16
17 import escuelaling.edu.arep.bono_parcial.controller.PropertyController;
18 import escuelaling.edu.arep.bono_parcial.dto.PropertyDTO;
19 import escuelaling.edu.arep.bono_parcial.exception.PropertyException;
20 import escuelaling.edu.arep.bono_parcial.service.impl.PropertyService;
21 import jakarta.servlet.http.HttpSession;
22
23 class PropertyControllerTest {
24
25     @Mock
26     private PropertyService propertyService;
27
28     @Mock
29     private HttpSession session; // Simulación de la sesión
30
31     @InjectMocks
32     private PropertyController propertyController;
33
34     private PropertyDTO sampleProperty;
35
36     @BeforeEach

```

- Remove usage of generic wildcard type.

The problem lies in the use of wildcards (?) in the return type of a Java method, which prevents the type from being properly constrained in any context. This is due to the invariance of generic parameters in Java, which means that they cannot behave either covariantly or contravariantly when used in methods with input and output. As a

result, a wildcard return does not allow safe manipulation of the returned type, indicating that the developer's intent was likely different and should be corrected by using a more specific type.



The screenshot shows the SonarCloud interface for a Java project named 'LAB SONAR SNYK'. The 'Issues' tab is selected, displaying 1/7 issues. One issue is highlighted with a red border:

```

src/.../PropertyController.java
Remove usage of generic wildcard type.

Define a constant instead of duplicating this literal "csrfToken" 4 times.
    1 4 locations

Remove usage of generic wildcard type.

Define a constant instead of duplicating this literal "Invalid CSRF token" 3 times.
    1 3 locations

Remove usage of generic wildcard type.

    7 of 7 shown

```

The code snippet for the highlighted issue is:

```

@.Autowired
public PropertyController(PropertyService propertyService) {
    this.propertyService = propertyService;
}

@GetMapping("/csrf-token")
public ResponseEntity<?> getCsrfToken(HttpServletRequest session) {
    String csrfToken = UUID.randomUUID().toString();
    session.setAttribute("csrfToken", csrfToken);
    return ResponseEntity.ok().body(csrfToken);
}

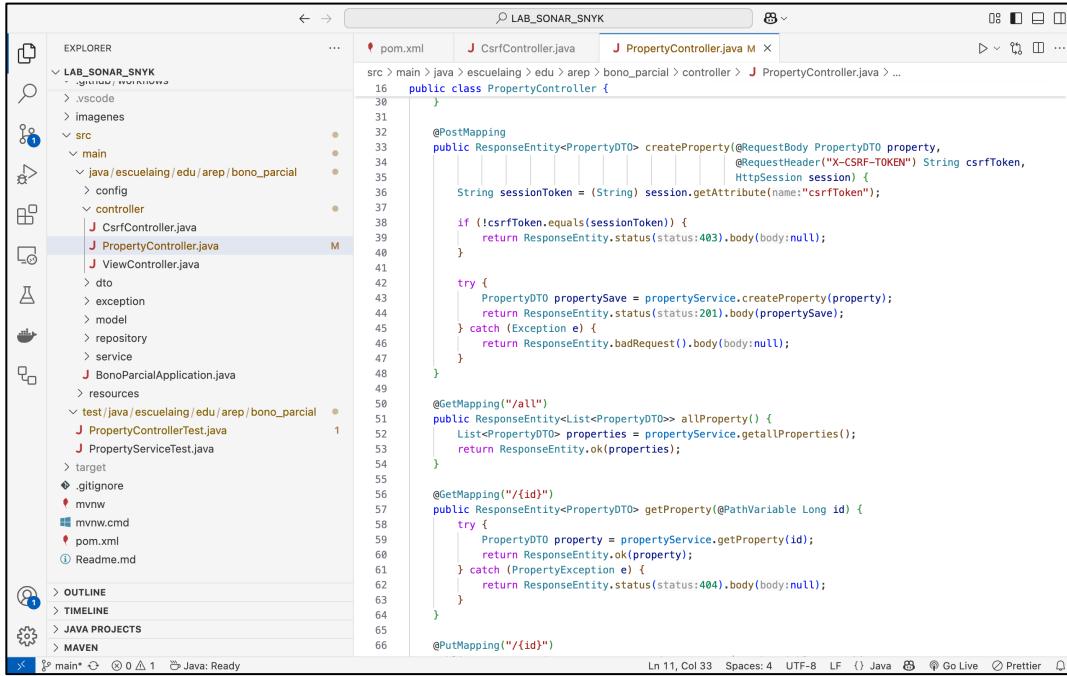
@PostMapping
public ResponseEntity<PropertyTO> createProperty(@RequestBody PropertyTO property,
    @RequestHeader("x-CSRF-TOKEN") String csrfToken,
    HttpSession session) {
    String sessionToken = (String) session.getAttribute("csrfToken");

    if (!csrfToken.equals(sessionToken))
        return ResponseEntity.status(403).body("Invalid CSRF token");
}

try {
    PropertyTO propertySave = propertyService.createProperty(property);
    return ResponseEntity.status(201).body(propertySave);
} catch (Exception e) {
    return ResponseEntity.badRequest().body("Error create property");
}
}

```

In class ***PropertyController.java***, we initially used wildcards in return types due to uncertainty about the specific values we needed to return on each endpoint. However, now that development is complete and we have clarity on the types of data that should be returned by API endpoints, we can replace wildcards with accurate and definitive return types. This improves readability, makes code easier to maintain, and avoids issues related to type variance in Java.



```

LAB SONAR SNYK
pom.xml J CsrfController.java J PropertyController.java M ...
src > main > java > escuelaing > edu > arep > bono_parcial > controller > J PropertyController.java > ...
16 public class PropertyController {
30     }
31
32     @PostMapping
33     public ResponseEntity<PropertyDTO> createProperty(@RequestBody PropertyDTO property,
34             @RequestHeader("X-CSRF-TOKEN") String csrfToken,
35             HttpSession session) {
36         String sessionToken = (String) session.getAttribute(name:"csrfToken");
37
38         if (!csrfToken.equals(sessionToken)) {
39             return ResponseEntity.status(status:403).body(body:null);
40         }
41
42         try {
43             PropertyDTO propertySave = propertyService.createProperty(property);
44             return ResponseEntity.status(status:201).body(propertySave);
45         } catch (Exception e) {
46             return ResponseEntity.badRequest().body(body:null);
47         }
48
49         @GetMapping("/all")
50         public ResponseEntity<List<PropertyDTO>> allProperty() {
51             List<PropertyDTO> properties = propertyService.getAllProperties();
52             return ResponseEntity.ok(properties);
53         }
54
55         @GetMapping("/{id}")
56         public ResponseEntity<PropertyDTO> getProperty(@PathVariable Long id) {
57             try {
58                 PropertyTO property = propertyService.getProperty(id);
59                 return ResponseEntity.ok(property);
60             } catch (PropertyException e) {
61                 return ResponseEntity.status(status:404).body(body:null);
62             }
63         }
64
65         @PutMapping("/{id}")
66     }

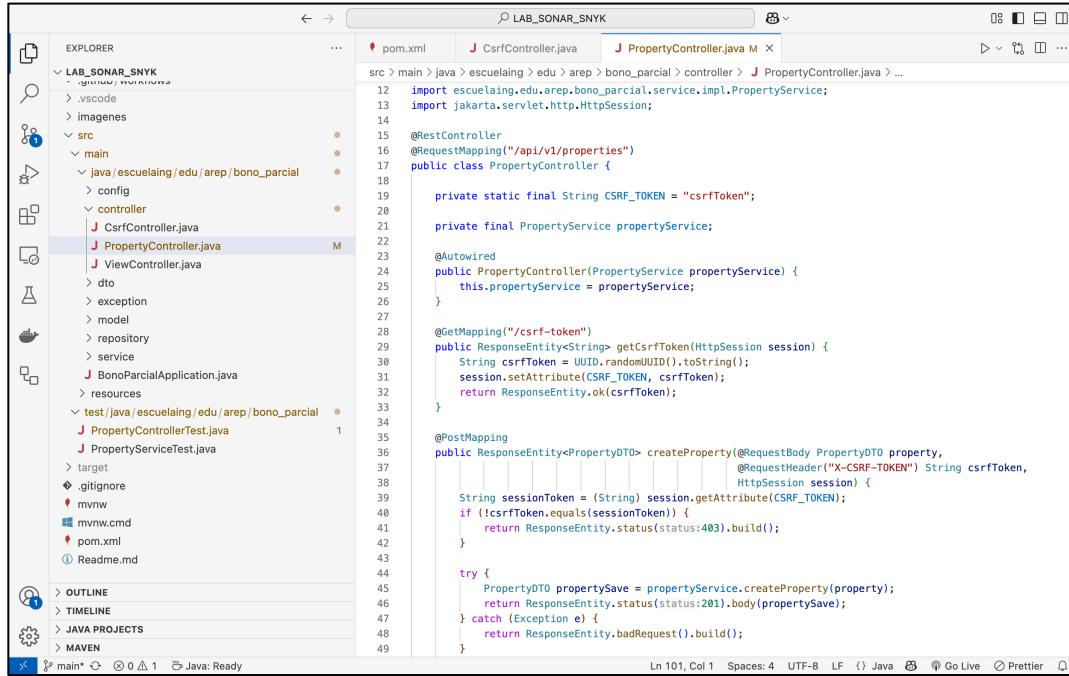
```

Ln 11, Col 33 Spaces: 4 UTF-8 LF () Java ⚡ Go Live ⌂ Prettier ⌂

- Define a constant instead of duplicating this literal "csrfToken" 4 times.

This issue in class **PropertyController.java** is the duplication of the "csrfToken" literal in four different places within the code. Repeating the same value in multiple parts of the code can lead to maintenance issues and increase the risk of errors in future modifications. The best practice to avoid this repetition is to define a constant with a descriptive name and use it instead of the literal value.

To work around this issue, we've declared a `CSRF_TOKEN` constant of type static final String, assigning it the value "csrfToken". Then, we replace all occurrences of the repeated string with the constant. In this way, if at any time it is necessary to modify the name of the key used to store the CSRF token in the session, it will only be necessary to change it in a single place, improving the readability and maintainability of the code.



```

src > main > java > escuelaing > edu > arep > bono_parcial > controller > J PropertyController.java ...
12 import escuelaing.edu.arep.bono_parcial.service.impl.PropertyService;
13 import jakarta.servlet.http.HttpSession;
14
15 @RestController
16 @RequestMapping("/api/v1/properties")
17 public class PropertyController {
18
19     private static final String CSRF_TOKEN = "csrfToken";
20
21     private final PropertyService propertyService;
22
23     @Autowired
24     public PropertyController(PropertyService propertyService) {
25         this.propertyService = propertyService;
26     }
27
28     @GetMapping("/csrf-token")
29     public ResponseEntity<String> getCsrfToken(HttpSession session) {
30         String csrfToken = UUID.randomUUID().toString();
31         session.setAttribute(CSRF_TOKEN, csrfToken);
32         return ResponseEntity.ok(csrfToken);
33     }
34
35     @PostMapping
36     public ResponseEntity<PropertyDTO> createProperty(@RequestBody PropertyDTO property,
37             @RequestHeader("X-CSRF-TOKEN") String csrfToken,
38             HttpSession session) {
39         String sessionToken = (String) session.getAttribute(CSRF_TOKEN);
40         if (!csrfToken.equals(sessionToken)) {
41             return ResponseEntity.status(status:403).build();
42         }
43
44         try {
45             PropertyDTO propertySave = propertyService.createProperty(property);
46             return ResponseEntity.status(status:201).body(propertySave);
47         } catch (Exception e) {
48             return ResponseEntity.badRequest().build();
49         }
50     }
51
52     /**
53      * Retrieves all properties.
54      *
55      * @return A list of PropertyDTOs.
56      */
57     @Override
58     public List<PropertyDTO> getAllProperties() {
59         List<Property> properties = propertyRepository.findAll();
60         return properties.stream().map(this::toDTO).collect(Collectors.toList());
61     }
62 }

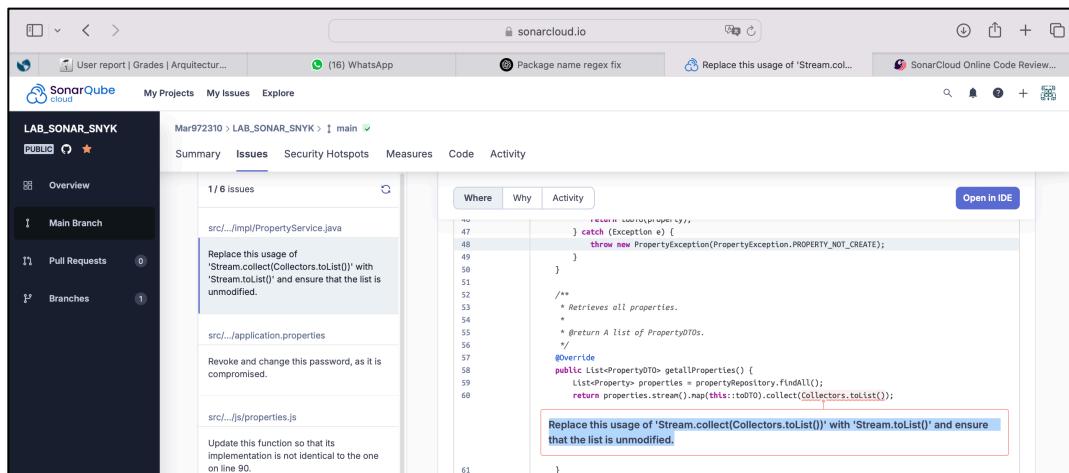
```

Ln 101, Col 1 Spaces: 4 UTF-8 LF () Java ⚡ Go Live ⌂ Prettier

- Replace this usage of 'Stream.collect(Collectors.toList())' with 'Stream.toList()' and ensure that the list is unmodified.

8

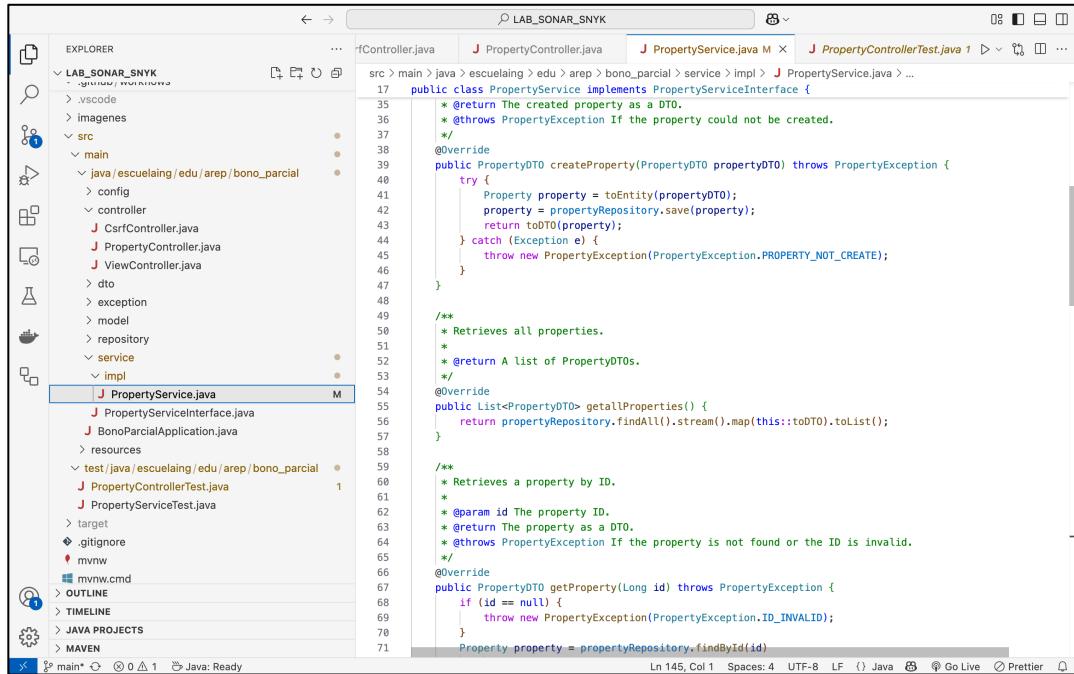
The issue identified is related to the use of Stream.collect(Collectors.toList()), which is an older, more verbose way to convert a Stream to a list. Since Java version 16, the Stream.toList() method has been introduced, which offers a more concise and efficient alternative. This method returns an immutable list, ensuring that it cannot be modified after it is created.



Replace this usage of 'Stream.collect(Collectors.toList())' with 'Stream.toList()' and ensure that the list is unmodified.

Replace this usage of 'Stream.collect(Collectors.toList())' with 'Stream.toList()' and ensure that the list is unmodified.

To fix this issue, we'll replace all instances of Stream.collect(Collectors.toList()) with Stream.toList(). This in addition to reducing the complexity of the code. It will also improve security by preventing unexpected modifications to the resulting list.

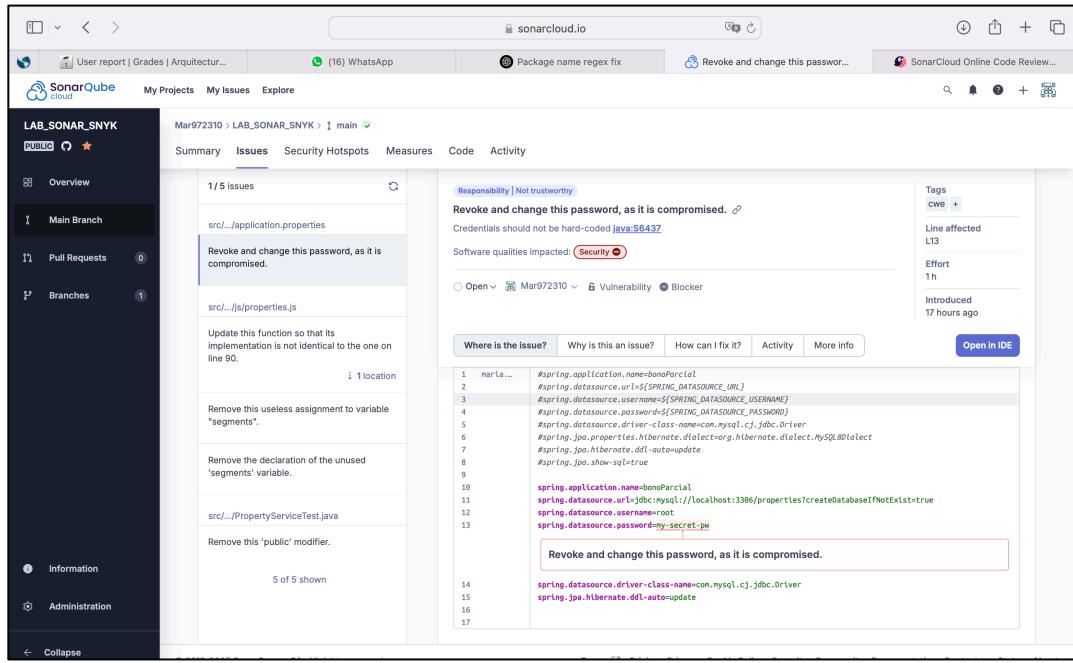


```

17  public class PropertyService implements PropertyServiceInterface {
18      /**
19       * @return The created property as a DTO.
20       * @throws PropertyException If the property could not be created.
21      */
22      @Override
23      public PropertyDTO createProperty(PropertyDTO propertyDTO) throws PropertyException {
24          try {
25              Property property = toEntity(propertyDTO);
26              property = propertyRepository.save(property);
27              return toDTO(property);
28          } catch (Exception e) {
29              throw new PropertyException(PropertyException.PROPERTY_NOT_CREATE);
30          }
31      }
32
33      /**
34       * Retrieves all properties.
35       *
36       * @return A list of PropertyDTOs.
37      */
38      @Override
39      public List<PropertyDTO> getAllProperties() {
40          return propertyRepository.findAll().stream().map(this::toDTO).toList();
41      }
42
43      /**
44       * Retrieves a property by ID.
45       *
46       * @param id The property ID.
47       * @return The property as a DTO.
48       * @throws PropertyException If the property is not found or the ID is invalid.
49      */
50      @Override
51      public PropertyDTO getProperty(Long id) throws PropertyException {
52          if (id == null) {
53              throw new PropertyException(PropertyException.ID_INVALID);
54          }
55          Property property = propertyRepository.findById(id);
56
57          return property != null ? toDTO(property) : null;
58      }
59
60
61
62
63
64
65
66
67
68
69
70
71
    
```

Ln 145, Col 1 Spaces: 4 UTF-8 LF ⚡ Java ⚡ Go Live ⚡ Prettier

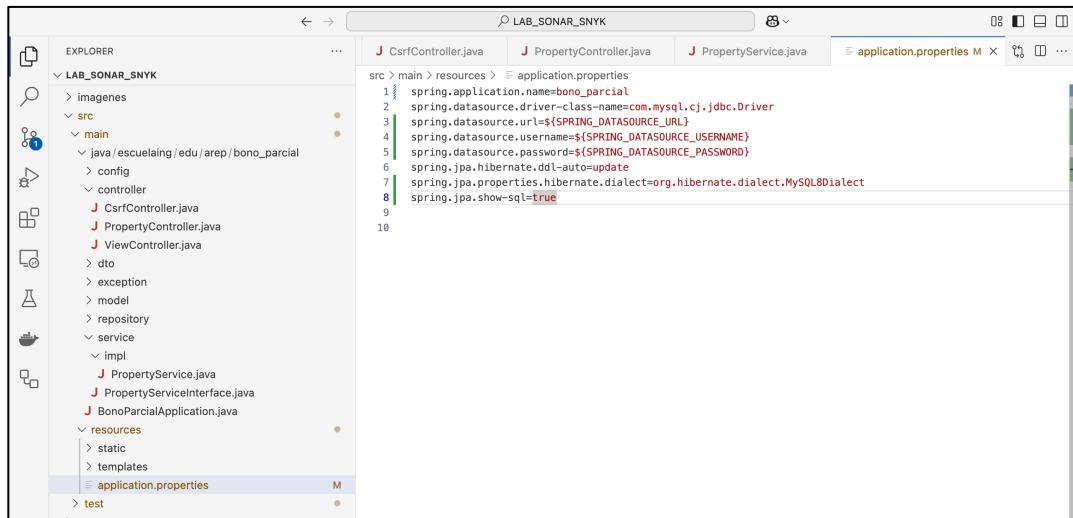
- Currently, we are performing the database configuration in the application includes credentials and connection details in the application.properties file. This poses a security risk, as credentials can be exposed if the code is shared or stored in a public repository. In addition, this rigid configuration makes it difficult to port the application between different environments (development, test, and production), as any changes to the database credentials or URL require directly modifying the configuration file.



The screenshot shows the SonarCloud interface for the project 'LAB SONAR SNYK'. The main view displays a summary of 1/5 issues, with one critical issue highlighted: 'Revoke and change this password, as it is compromised.' This issue is categorized under 'Security' and is marked as a 'Blocker'. The code snippet in question is from the file 'application.properties' at line 13, which contains the hard-coded password 'my_secret_pw'. A red box surrounds this specific line of code.

To work around this problem, we're going to replace the plaintext values with environment variables \${SPRING_DATASOURCE_URL}, \${SPRING_DATASOURCE_USERNAME}, \${SPRING_DATASOURCE_PASSWORD}), so that your app dynamically gets the settings based on the environment it's running in. This will allow us to improve security by not exposing credentials in the code and will make the deployment more flexible and scalable.

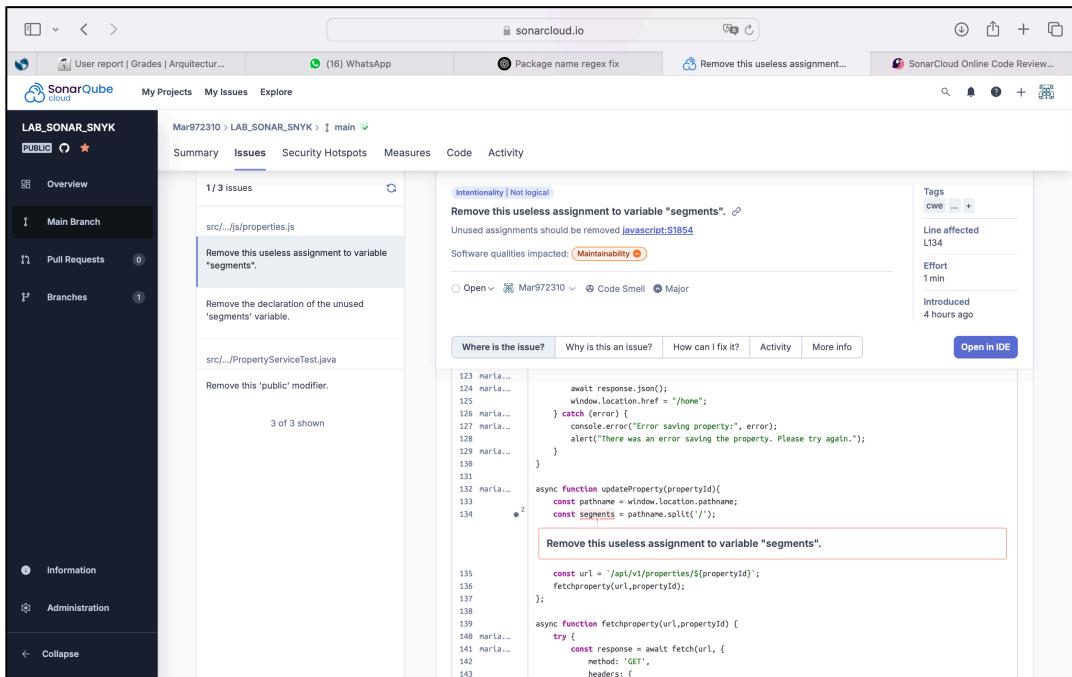
10



The screenshot shows the Eclipse IDE interface with the project 'LAB SONAR SNYK'. The code editor displays the 'application.properties' file, which now contains the environment variable \${SPRING_DATASOURCE_PASSWORD} instead of the hardcoded value 'bono_parcial'. The code also includes imports for 'com.mysql.cj.jdbc.Driver' and 'org.hibernate.dialect.MySQL8Dialect'. The rest of the code remains identical to the previous version.

- Remove this useless assignment to variable "segments"

SonarCloud has identified an unnecessary mapping to the segments variable, indicating that its value is not used anywhere in the code. These types of unnecessary mappings affect the clarity and efficiency of your code, as they introduce elements that don't provide functionality and can lead to confusion in future revisions. In addition, maintaining redundant code can negatively impact the maintainability and scalability of the project.



The screenshot shows the SonarCloud interface for the project 'LAB SONAR SNYK'. The 'Issues' tab is selected, displaying 1/3 issues. One issue is highlighted: 'Remove this useless assignment to variable "segments".' It points to a line of code in 'src/main/java/PropertyServiceTest.java':

```

123 maria...
124 maria...
125
126 maria...
127 maria...
128
129 maria...
130
131
132 maria...
133
134
    await response.json();
    window.location.href = "/home";
} catch (error) {
    console.error("Error saving property:", error);
    alert("There was an error saving the property. Please try again.");
}

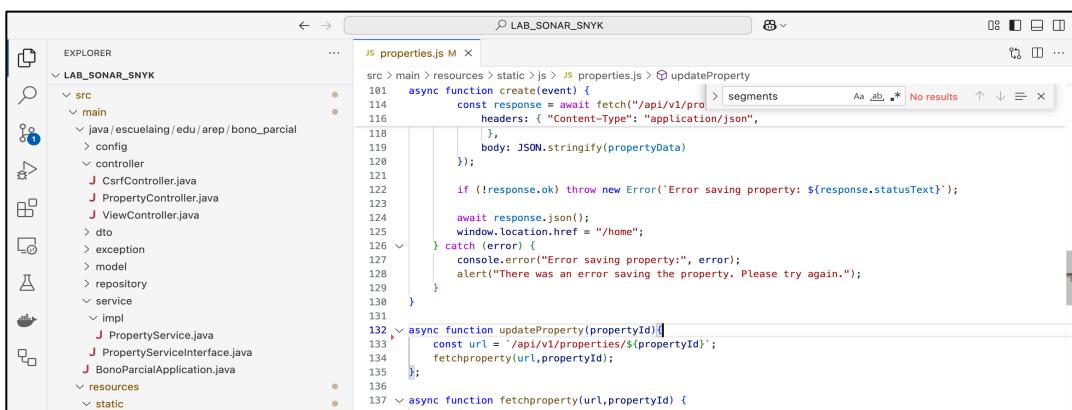
async function updateProperty(propertyId){
    const pathname = window.location.pathname;
    const segments = pathname.split('/');

```

A red box highlights the line 'const segments = pathname.split('/');

11

To resolve this issue, we'll remove the mapping of the segments variable if it's not used elsewhere in the code. This fix allows us to maintain cleaner code, reduce unnecessary complexity, and improve the overall quality of our codebase, aligning with the best practices recommended by SonarCloud.



The screenshot shows the IntelliJ IDEA code editor with the file 'properties.js' open. The variable 'segments' is highlighted in red. The code editor shows the following snippet:

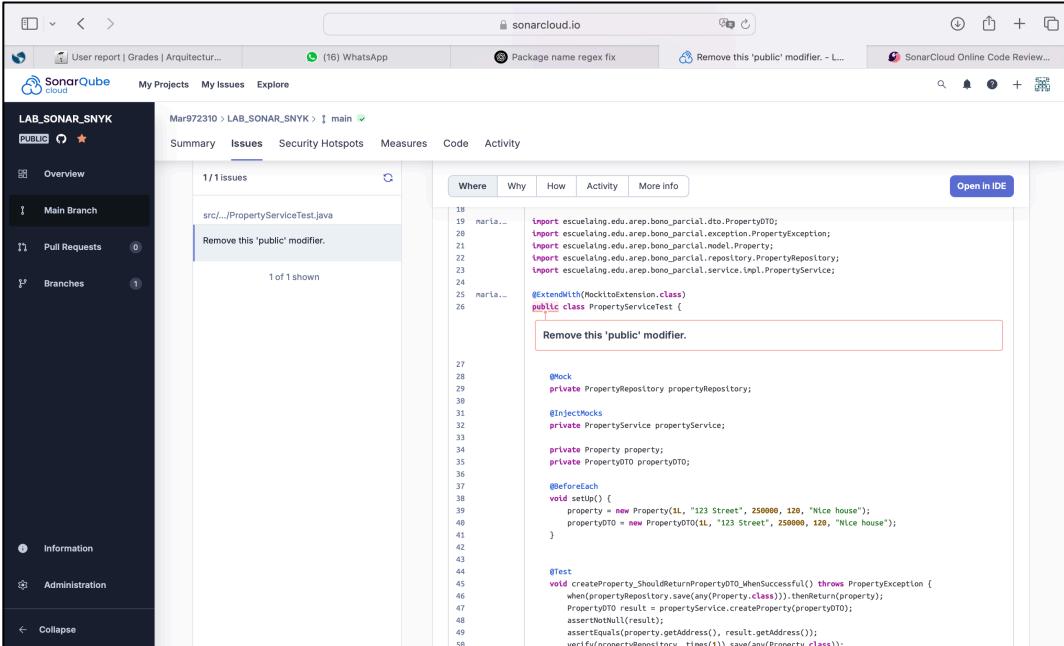
```

101  async function create(event) {
114      const response = await fetch("/api/v1/pro");
116      headers: { "Content-Type": "application/json",
117      },
118      body: JSON.stringify(propertyData)
119  );
120
121
122  if (!response.ok) throw new Error(`Error saving property: ${response.statusText}`);
123
124  await response.json();
125  window.location.href = "/home";
126  } catch (error) {
127      console.error("Error saving property:", error);
128      alert("There was an error saving the property. Please try again.");
129  }
130
131
132  async function updateProperty(propertyId){
133      const url = `/api/v1/properties/${propertyId}`;
134      fetchproperty(url,propertyId);
135  };
136
137  async function fetchproperty(url,propertyId) {

```

- Remove this 'public' modifier.

In JUnit 5, the classes and test methods do not need to be public, unlike in JUnit 4. Keeping them public can affect the readability of the code and does not follow recommended conventions. In addition, in the case of private methods, JUnit 5 ignores them without warning, which can lead to reliability issues in the tests.

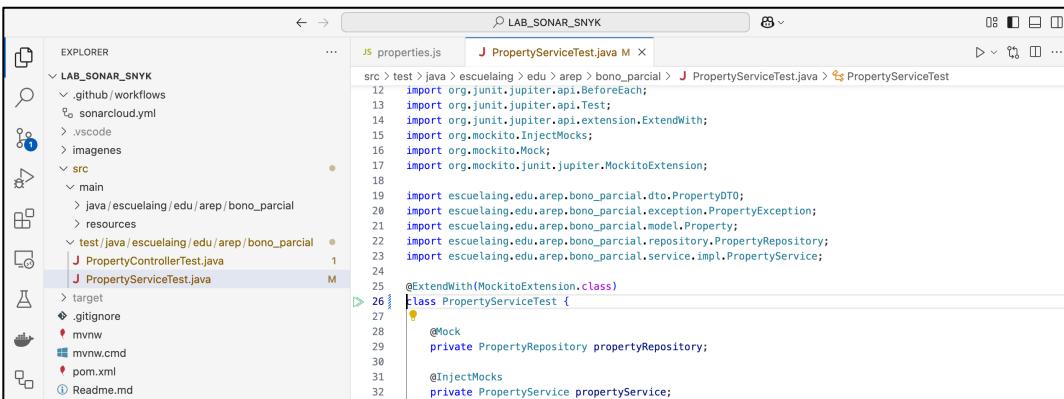


The screenshot shows the SonarCloud interface for a Java project named 'LAB SONAR SNYK'. The 'Issues' tab is selected, showing 1/1 issue in the file 'src/main/java/com/escuelaling/edu/arep/bono_parcial/PropertyServiceTest.java'. The issue is a warning about the 'public' modifier on the class definition. The code snippet shows:

```

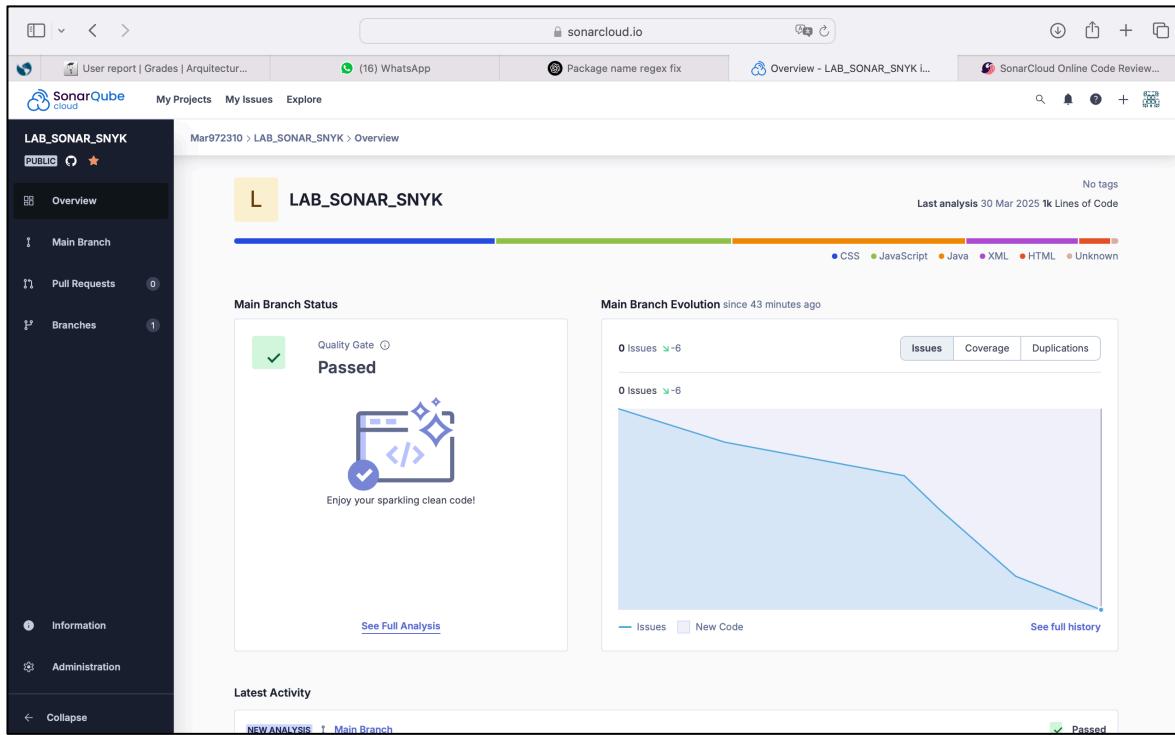
18 import com.escuelaling.edu.arep.bono_parcial.dto.PropertyDTO;
19 import com.escuelaling.edu.arep.bono_parcial.exception.PropertyException;
20 import com.escuelaling.edu.arep.bono_parcial.model.Property;
21 import com.escuelaling.edu.arep.bono_parcial.repository.PropertyRepository;
22 import com.escuelaling.edu.arep.bono_parcial.service.impl.PropertyService;
23
24 @ExtendWith(MockitoExtension.class)
25 public class PropertyServiceTest {
26
27     @Mock
28     private PropertyRepository propertyRepository;
29
30     @InjectMocks
31     private PropertyService propertyService;
32
33     private Property property;
34     private PropertyDTO propertyDTO;
35
36     @BeforeEach
37     void setup() {
38         property = new Property(1L, "123 Street", 250000, 120, "Nice house");
39         propertyDTO = new PropertyDTO(1L, "123 Street", 250000, 120, "Nice house");
40     }
41
42     @Test
43     void createProperty_ShouldReturnPropertyDTO_whenSuccessful() throws PropertyException {
44         when(propertyRepository.save(any(Property.class))).thenReturn(property);
45         PropertyDTO result = propertyService.createProperty(propertyDTO);
46         assertNotNull(result);
47         assertEquals(property.getAddress(), result.getAddress());
48         verify(propertyRepository, times(1)).save(any(Property.class));
49     }
50
51 }
```

To improve readability and follow best practices, we must remove the public modifier from our classes and test methods, unless there is a technical reason to keep it. Using the default visibility (package-private) is enough for JUnit 5 to recognize and execute tests correctly.



The screenshot shows the IntelliJ IDEA interface with the Java file 'PropertyServiceTest.java' open. The code is identical to the one shown in the SonarCloud screenshot. A red box highlights the line '@ExtendWith(MockitoExtension.class)' in the code editor, with a callout box containing the text 'Remove this "public" modifier.'

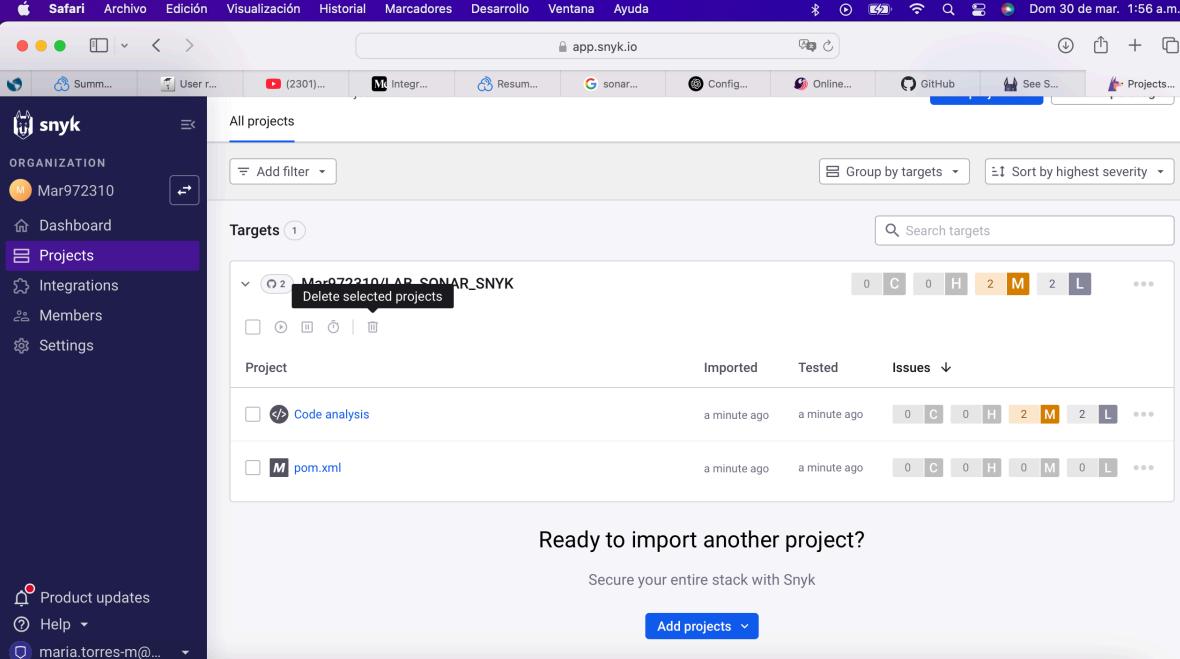
After having corrected all these issues, we will be able to see that the code complies with all good code practices.



13

Snyk Analysis

The analysis carried out with Snyk to the Mar972310/LAB SONAR SNYK repository, gave us as an answer after having analyzed the application code that we have two Medium risk vulnerabilities and two low risk vulnerabilities.



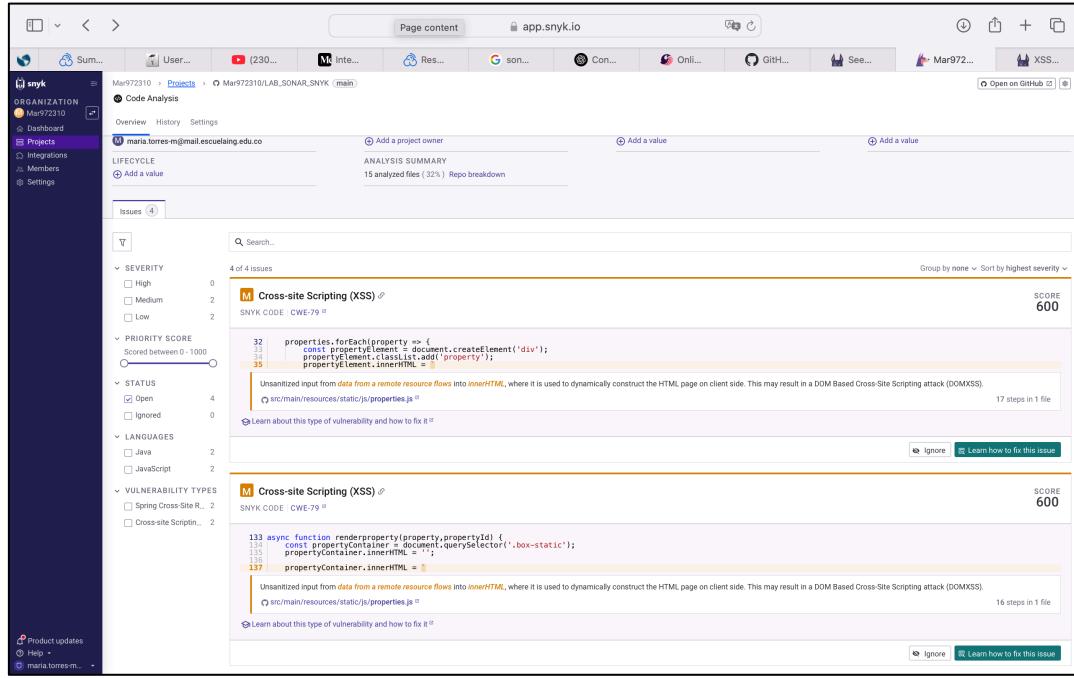
The screenshot shows the Snyk web interface with the following details:

- Organization:** Mar972310
- Project:** Mar972310/LAP_SONAR_SNYK
- Targets:** 1 (Mar972310/LAP_SONAR_SNYK)
- Issues:** 2 (M)
- Code analysis:** pom.xml (Imported a minute ago, Tested a minute ago, Issues: 0 C, 0 H, 2 M, 2 L)

Below we will be analyzing each of the issues and making the corresponding corrections.

o DOM-BASED CROSS-Site Scripting (DOM XSS)

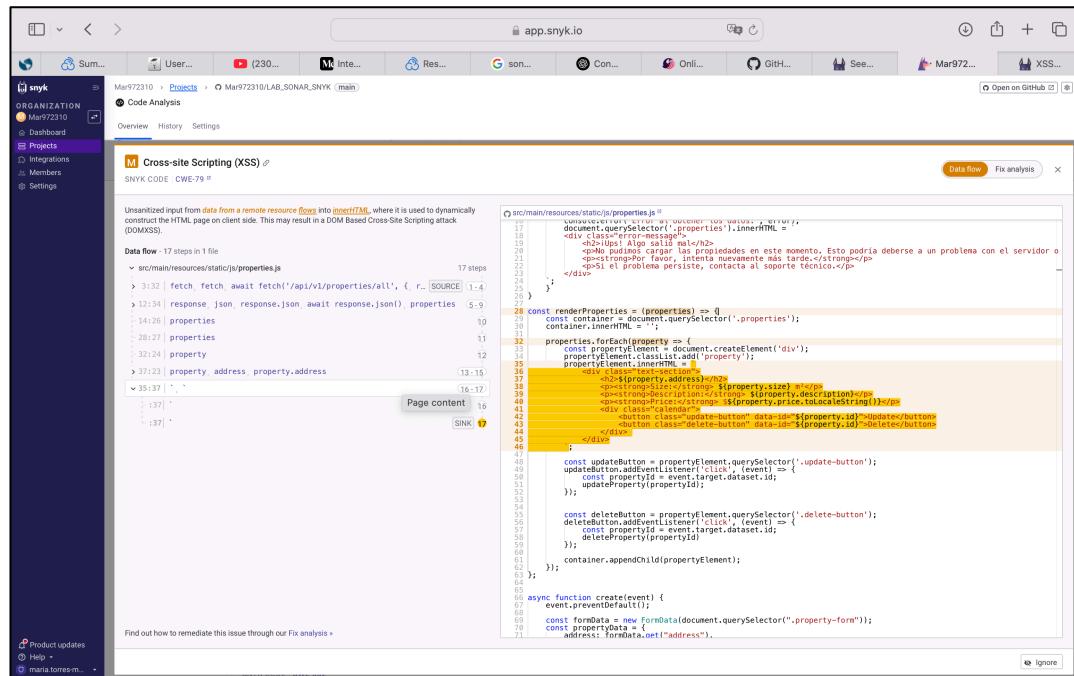
Medium-risk issues are caused by receiving data from a remote resource flow, such as responses from an external server and APIs, and we are injecting them directly into innerHTML without performing any validation or sanitization of all the data that was received, opening up the possibility of a **DOM-based Cross-Site Scripting (DOM XSS)** attack. This type of attack is orchestrated entirely on the client side, i.e., in the user's browser, without being detected by the server. To work around this vulnerability, untrusted data should not be inserted directly into innerHTML. Instead, it's best to use textContent, which inserts content as text without interpreting HTML.



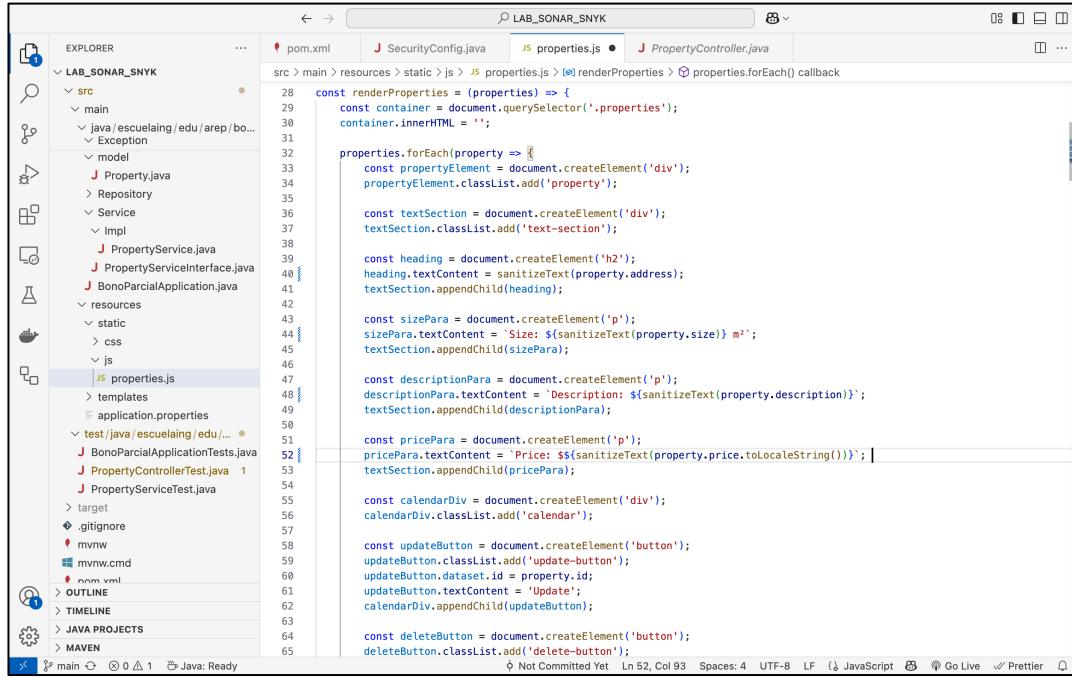
The screenshot shows the Snyk Code Analysis interface for a project named 'Mar972310'. It displays two XSS vulnerabilities. The first issue is at line 32 of 'src/main/resources/static/js/properties.js' with a score of 600. The second issue is at line 133 of the same file with a score of 600. Both issues involve unsanitized input from a remote resource flowing into innerHTML, which can lead to a DOM Based Cross-Site Scripting attack (DOMXSS). The interface includes filters for severity, priority score, status, languages, and vulnerability types.

1. RenderProperties function, this is one of the functions that is being used to render the list of properties that are registered in the application's database, the information is being obtained from an endpoint of the properties API.

15



The screenshot shows the Snyk Code Analysis interface with the 'Data flow' tab selected for a XSS vulnerability. The code flow highlights the 'src/main/resources/static/js/properties.js' file, specifically the 'RenderProperties' function. The code snippet shows unsanitized input from a remote resource flowing into innerHTML, which can lead to a DOM Based Cross-Site Scripting attack (DOMXSS). The interface includes a 'Fix analysis' button and a note to find out how to remediate the issue.

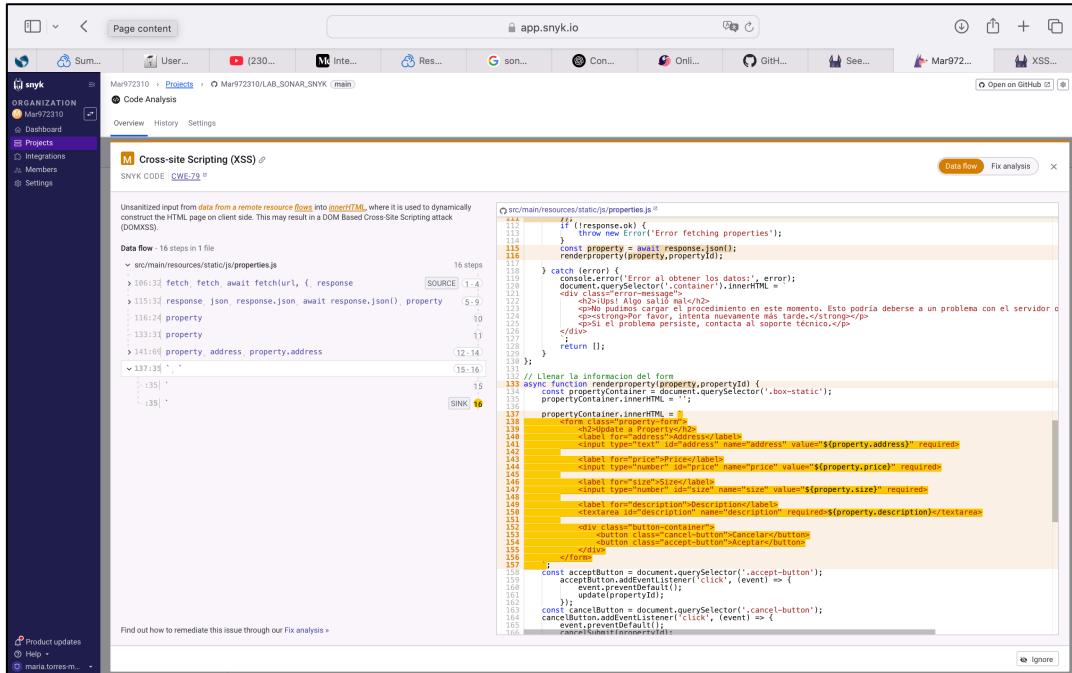


```

src > main > resources > static > js > JS properties.js > (e) renderProperties > (S) properties.forEach() callback
  28 const renderProperties = (properties) => {
  29   const container = document.querySelector('.properties');
  30   container.innerHTML = '';
  31
  32   properties.forEach(property => {
  33     const propertyElement = document.createElement('div');
  34     propertyElement.classList.add('property');
  35
  36     const textSection = document.createElement('div');
  37     textSection.classList.add('text-section');
  38
  39     const heading = document.createElement('h2');
  40     heading.textContent = sanitizeText(property.address);
  41     textSection.appendChild(heading);
  42
  43     const sizePara = document.createElement('p');
  44     sizePara.textContent = `Size: ${sanitizeText(property.size)} m2`;
  45     textSection.appendChild(sizePara);
  46
  47     const descriptionPara = document.createElement('p');
  48     descriptionPara.textContent = `Description: ${sanitizeText(property.description)}`;
  49     textSection.appendChild(descriptionPara);
  50
  51     const pricePara = document.createElement('p');
  52     pricePara.textContent = `Price: ${$sanitizeText(property.price.toLocaleString())}`;
  53     textSection.appendChild(pricePara);
  54
  55     const calendarDiv = document.createElement('div');
  56     calendarDiv.classList.add('calendar');
  57
  58     const updateButton = document.createElement('button');
  59     updateButton.classList.add('update-button');
  60     updateButton.dataset.id = property.id;
  61     updateButton.textContent = 'Update';
  62     calendarDiv.appendChild(updateButton);
  63
  64     const deleteButton = document.createElement('button');
  65     deleteButton.classList.add('delete-button');
  
```

Not Committed Yet | Ln 52, Col 93 | Spaces: 4 | UTF-8 | LF | ↵ JavaScript | ⚡ Go Live | ✨ Prettier

2. RenderProperty function, this is a function that is being used to render a specific property that is registered in the application's database, like the previous one, the information is being obtained from an endpoint of the properties API.



Cross-site Scripting (XSS)

SNYK CODE [CWE-79](#)

Unsanitized input from [data from a remote resource flows](#) into `innerHTML`, where it is used to dynamically construct the HTML page on client side. This may result in a DOM Based Cross-Site Scripting attack (DOMXSS).

Data flow - 16 steps in 1 file

src/main/resources/static/js/properties.js

```

  16 steps
  1 SOURCE: 1-4
  2 > 106:32 fetch await fetch(url, { response
  3 > 115:32 response.json response.json await response.json() property
  4 > 116:24 property
  5 > 133:31 property
  6 > 141:6 property.address property.address
  7 > 137:35 , '
  8 > 135: '
  9 > 135: '
  10 > 135: '
  11 > 135: '
  12 > 135: '
  13 > 135: '
  14 > 135: '
  15 > 135: '
  16 > 135: '
  17 > 135: '
  18 > 135: '
  19 > 135: '
  20 > 135: '
  21 > 135: '
  22 > 135: '
  23 > 135: '
  24 > 135: '
  25 > 135: '
  26 > 135: '
  27 > 135: '
  28 > 135: '
  29 > 135: '
  30 > 135: '
  31 > 135: '
  32 > 135: '
  33 > 135: '
  34 > 135: '
  35 > 135: '
  36 > 135: '
  37 > 135: '
  38 > 135: '
  39 > 135: '
  40 > 135: '
  41 > 135: '
  42 > 135: '
  43 > 135: '
  44 > 135: '
  45 > 135: '
  46 > 135: '
  47 > 135: '
  48 > 135: '
  49 > 135: '
  50 > 135: '
  51 > 135: '
  52 > 135: '
  53 > 135: '
  54 > 135: '
  55 > 135: '
  56 > 135: '
  57 > 135: '
  58 > 135: '
  59 > 135: '
  60 > 135: '
  61 > 135: '
  62 > 135: '
  63 > 135: '
  64 > 135: '
  65 > 135: '
  66 > 135: '
  67 > 135: '
  68 > 135: '
  69 > 135: '
  70 > 135: '
  71 > 135: '
  72 > 135: '
  73 > 135: '
  74 > 135: '
  75 > 135: '
  76 > 135: '
  77 > 135: '
  78 > 135: '
  79 > 135: '
  80 > 135: '
  81 > 135: '
  82 > 135: '
  83 > 135: '
  84 > 135: '
  85 > 135: '
  86 > 135: '
  87 > 135: '
  88 > 135: '
  89 > 135: '
  90 > 135: '
  91 > 135: '
  92 > 135: '
  93 > 135: '
  94 > 135: '
  95 > 135: '
  96 > 135: '
  97 > 135: '
  98 > 135: '
  99 > 135: '
  100 > 135: '
  101 > 135: '
  102 > 135: '
  103 > 135: '
  104 > 135: '
  105 > 135: '
  106 > 135: '
  107 > 135: '
  108 > 135: '
  109 > 135: '
  110 > 135: '
  111 > 135: '
  112 > 135: '
  113 > 135: '
  114 > 135: '
  115 > 135: '
  116 > 135: '
  117 > 135: '
  118 > 135: '
  119 > 135: '
  120 > 135: '
  121 > 135: '
  122 > 135: '
  123 > 135: '
  124 > 135: '
  125 > 135: '
  126 > 135: '
  127 > 135: '
  128 > 135: '
  129 > 135: '
  130 > 135: '
  131 > 135: '
  132 > 135: '
  133 > 135: '
  134 > 135: '
  135 > 135: '
  136 > 135: '
  137 > 135: '
  138 > 135: '
  139 > 135: '
  140 > 135: '
  141 > 135: '
  142 > 135: '
  143 > 135: '
  144 > 135: '
  145 > 135: '
  146 > 135: '
  147 > 135: '
  148 > 135: '
  149 > 135: '
  150 > 135: '
  151 > 135: '
  152 > 135: '
  153 > 135: '
  154 > 135: '
  155 > 135: '
  156 > 135: '
  157 > 135: '
  158 > 135: '
  159 > 135: '
  160 > 135: '
  161 > 135: '
  162 > 135: '
  163 > 135: '
  164 > 135: '
  165 > 135: '
  
```

Find out how to remediate this issue through our Fix analysis

Ignore

The screenshot shows a Java IDE interface with the following details:

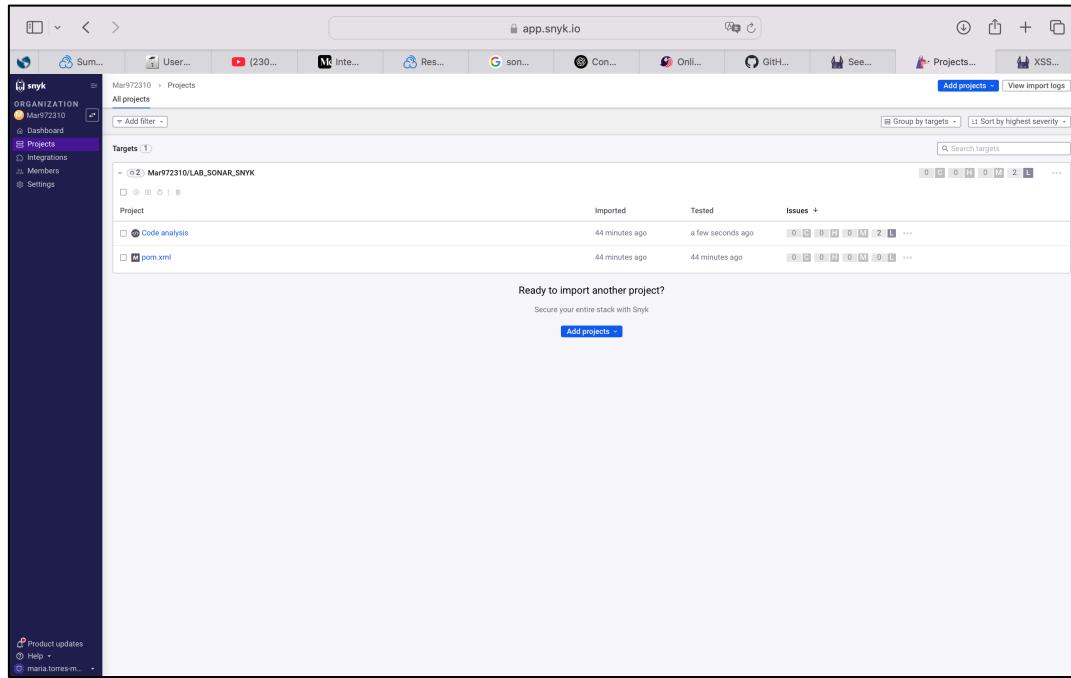
- Toolbar:** Includes icons for back, forward, search, and file operations.
- Header:** Shows the project name "LAB SONAR SNYK" and tabs for "pom.xml", "SecurityConfig.java", "properties.js", and "PropertyController.java".
- Sidebar (EXPLORER):** Lists the project structure:
 - src
 - main
 - java / escuelaing / edu / arep / bo...
 - Exception
 - model
 - J Property.java
 - Repository
 - Service
 - Impl
 - J PropertyService.java
 - J PropertyServiceInterface.java
 - J BonоАcialApplication.java
 - resources
 - static
 - > css
 - > js
 - JS properties.js
 - templates
 - application.properties
 - test / java / escuelaing / edu / ...
 - J BonоАcialApplicationTests.java
 - J PropertyControllerTest.java
 - J PropertyServiceTest.java
 - target
 - .gitignore
 - mvnw
 - mvnw.cmd
 - root vml
 - OUTLINE
 - TIMELINE
 - JAVA PROJECTS
 - MAVEN
- Central Area:** Displays the content of the "properties.js" file. The code uses template literals and the `document.createElement` method to dynamically generate HTML elements for address, price, and size inputs.
- Status Bar:** Shows the date ("Mar 97 23:10 (2 hours ago)"), line count ("Ln 333, Col 4"), spaces setting ("Spaces: 4"), and other system information.

With these fixes we already mitigate the vulnerability of DOM-based Cross-Site Scripting, after uploading the changes to the repository we can see how SNYK performs the analysis again and we no longer have those issues.

17

○ Cross-Site Request Forgery (CSRF)

The next vulnerabilities that the analysis shows us are two ***Cross-Site Request Forgery (CSRF)*** vulnerabilities that can generate a type of attack in which an attacker tricks an authenticated user into performing unwanted actions in a web application without their consent. This occurs because the user's browser automatically sends cookies and authentication headers with each request to a website on which they are authenticated, which an attacker can exploit to execute malicious actions on the user's behalf.



In a Spring application, if CSRF protection is not implemented, an attacker could, for example, send a POST request from a malicious website to change a user's password or make a bank transfer without their authorization. Since the user is already authenticated, the application would process the request as legitimate.

18

In our application, before making the changes, since we do not have an authentication system, anyone could send data to the server and make malicious insertions to the databases. For this we will have to add the security dependency provided by the Spring-boot framework.

1. In the pom.xml we will add this dependency.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

2. We created a class to configure everything related to security restrictions and access permissions, since adding this dependency in the pom will enable us to have a username and password to perform the authentications by default.
3. To the Property Controller to the POST, PUT and DELETE requests we will have to add parameters to receive the CSRF token and the session, with this we can control if an attacker tries to send malicious requests on behalf of the user without

having the correct token, the checks will fail and the request will be rejected.

This means that any user who logs into the browser will receive a unique CSRF token and will need to submit it on each protected request. Even if there is no authentication, this verification prevents attacks where an external site tries to send malicious requests on behalf of the user who has an active session in our application.

```
PropertyController.java M ● J SecurityConfig.java J CsrfController.java JS properties.js
src > main > java > esquealaing > edu > arep > bonoParcial > Controller > J PropertyController.java > PropertiesController > updateProperty(Long, PropertyDTO, String, HttpSession)
15 public class PropertyController {
16
17     private final PropertyService propertyService;
18
19     @Autowired
20     public PropertyController(PropertyService propertyService) {
21         this.propertyService = propertyService;
22     }
23
24     @GetMapping("/csrf-token")
25     public ResponseEntity<?> getCsrfToken(HttpServletRequest session) {
26         String csrfToken = UUID.randomUUID().toString();
27         session.setAttribute(name:"csrfToken", csrfToken);
28         return ResponseEntity.ok().body(csrfToken);
29     }
30
31     @PostMapping
32     public ResponseEntity<?> createProperty(@RequestBody PropertyDTO property,
33                                         @RequestHeader("X-CSRF-TOKEN") String csrfToken,
34                                         HttpSession session) {
35         String sessionToken = (String) session.getAttribute(name:"csrfToken");
36
37         if (!csrfToken.equals(sessionToken)) {
38             return ResponseEntity.status(status:403).body(body:"Invalid CSRF token");
39         }
40
41         try {
42             PropertyDTO propertySave = propertyService.createProperty(property);
43             return ResponseEntity.status(status:201).body(propertySave);
44         } catch (Exception e) {
45             return ResponseEntity.badRequest().body(body:"Error create property");
46         }
47
48     }
49
50
51     @GetMapping("/all")
52     public ResponseEntity<List<PropertyDTO>> allProperties() {
53 }
```

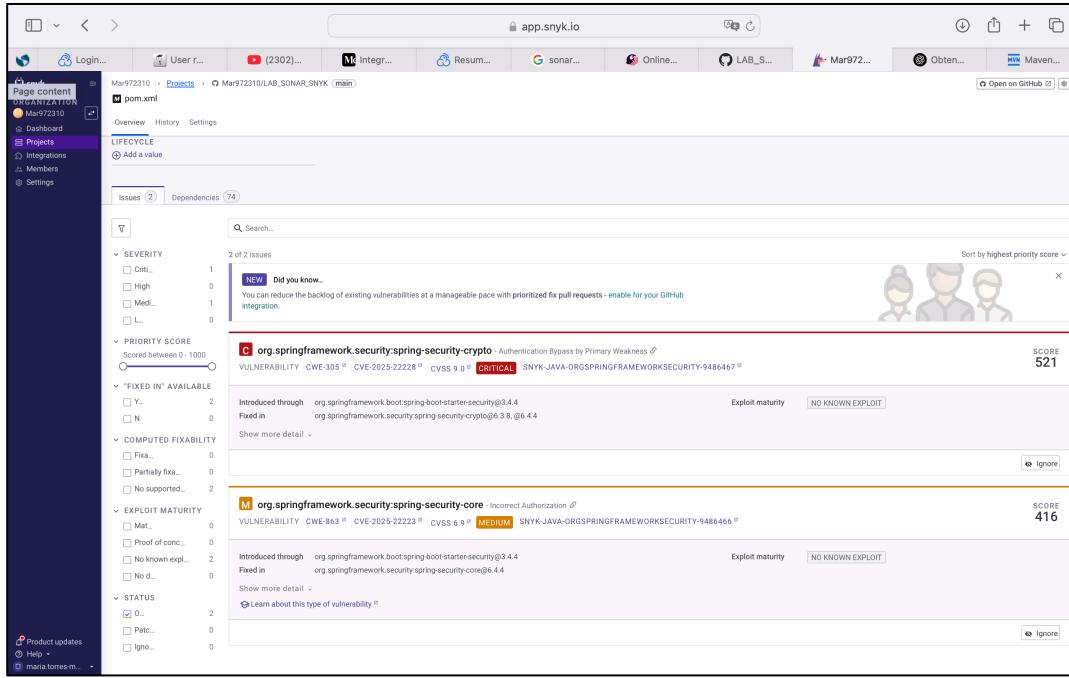


```
J PropertyController.java M ● J SecurityConfig.java J CsrfController.java JS properties.js

src > main > java > esquelting > edu > arep > bonoParcial > Controller > J PropertyController.java > updateProperty(Long, PropertyDTO, String, HttpSession)
15 public class PropertyController {
16     public ResponseEntity<?> getProperty(@PathVariable Long id) {
17         ...
18     }
19
20     @PutMapping("/{id}")
21     public ResponseEntity<?> updateProperty(@PathVariable Long id, @RequestBody PropertyDTO property,
22         @RequestHeader("X-CSRF-TOKEN") String csrfToken, HttpSession session) {
23         if (!csrfToken.equals(session.getAttribute(name:"csrfToken"))) {
24             return ResponseEntity.status(status:403).body(body:"Invalid CSRF token");
25         }
26         try {
27             PropertyDTO propertyUpdate = propertyService.updateProperty(id, property);
28             return ResponseEntity.ok(propertyUpdate);
29         } catch (PropertyException e) {
30             String error = e.getMessage();
31             if (error.equals(PropertyException.PROPERTY_NOT_UPDATE)) {
32                 return ResponseEntity.status(status:400).body(e.getMessage());
33             } else {
34                 return ResponseEntity.status(status:404).body(e.getMessage());
35             }
36         }
37     }
38 }
```

Going back to analyze the platform, it throws us two new issues in the pom.xml, the first vulnerability in org.springframework.security:spring-security-crypto since it is affecting the BCryptPasswordEncoder.matches() function, which only considers the first 72 characters of a password when comparing it with its hash. This allows

different passwords with the same first 72 characters to be accepted as valid, facilitating brute force attacks and possible authentication bypasses.



The screenshot shows the Snyk.io dashboard for a project named 'Mar972310'. The 'Dependencies' tab is selected, displaying 74 issues. Two specific vulnerabilities are highlighted:

- org.springframework.security:spring-security-crypto**: Authentication Bypass by Primary Weakness (CVE-2025-22228) - Score: 521. This vulnerability is marked as 'NEW' and 'CRITICAL'. It was introduced through org.springframework.boot:spring-boot-starter-security@3.4.4 and fixed in org.springframework.security:spring-security-crypto@6.3.8, @6.4.4. Exploit maturity: NO KNOWN EXPLOIT.
- org.springframework.security:spring-security-core**: Incorrect Authorization (CVE-2025-22223) - Score: 416. This vulnerability is marked as 'MEDIUM'. It was introduced through org.springframework.boot:spring-boot-starter-security@3.4.4 and fixed in org.springframework.security:spring-security-core@6.3.8, @6.4.4. Exploit maturity: NO KNOWN EXPLOIT.

Additionally, it threw another vulnerability in `org.springframework.security:spring-security-core` can affect authorization in our application if we use `@EnableMethodSecurity` in parameterized methods or types. The issue arises because the `findClosestMethodAnnotations()` function does not correctly search for security annotations in legacy methods, interfaces, or parameterized superclasses, which could allow certain methods to bypass security restrictions. However, our application would not be vulnerable if we do not use `@EnableMethodSecurity`, if we do not protect parameterized methods or if we always place the annotations directly in the methods we want to secure.

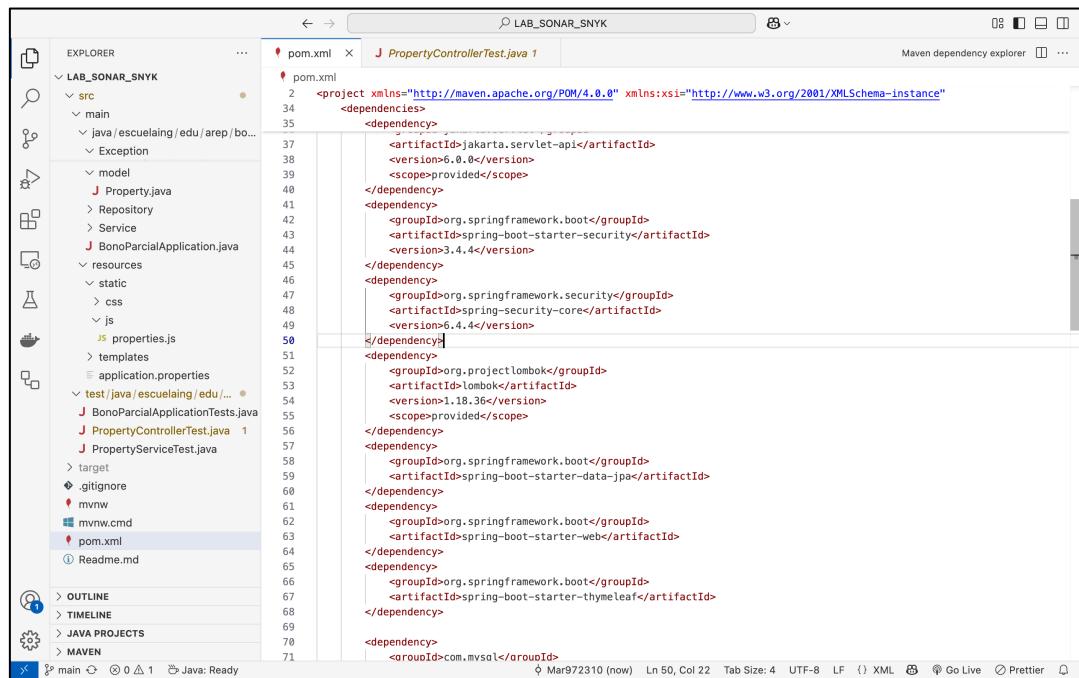
To mitigate both vulnerabilities found in ***spring-security-crypto*** and ***spring-security-core***, we have added the updated dependencies that fix these issues, ensuring that our application is protected even if we do not implement user authentication and access is free. Even if we do not handle credentials, it is crucial to prevent possible authorization bypasses and failures in password validation that could compromise the security of the system. With these updates, we strengthen the integrity of our application, avoiding risks associated with vulnerable versions of Spring Security and ensuring a more secure environment for its execution.

- `spring-security-crypto`:

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-crypto</artifactId>
    <version>6.4.4</version>
</dependency>
```

- spring-security-core:

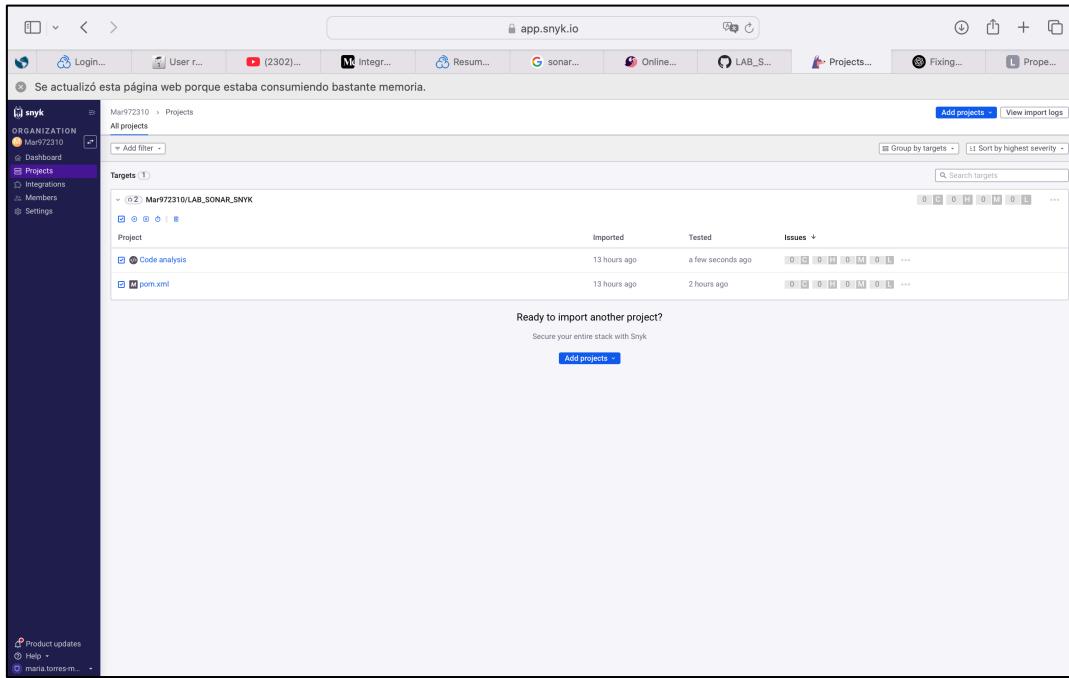
```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>6.4.4</version>
</dependency>
```



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure under "LAB SONAR SNYK". It includes a "src" folder containing "main" (with "java", "model", "resources", "static", and "test/java" subfolders), "test" (with "java" and "resources" subfolders), and "target", ".gitignore", "mvnw", "mvnw.cmd", "pom.xml", and "Readme.md" files.
- Maven Dependency Explorer:** An open view showing the Maven dependency tree. The XML code in the editor pane includes:


```
<dependency>
        <groupId>jakarta.servlet-api</groupId>
        <version>6.0.0</version>
        <scope>provided</scope>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
        <version>3.4.4</version>
      </dependency>
      <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-core</artifactId>
        <version>6.4.4</version>
      </dependency>
      <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.36</version>
        <scope>provided</scope>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
      </dependency>
      <dependency>
        <groupId>com.mysol</groupId>
```
- Status Bar:** Shows the current Java version as "Java: Ready", the date and time as "Mar972310 (now)", and other developer information.



The screenshot shows the Snyk web interface with the URL app.snyk.io in the address bar. The page displays a project named 'Mar972310' under the 'Projects' section. It shows one target, '(0)Z_Mar972310/LAB SONAR SNYK'. Under this target, there is one project named 'Project'. This project has two files listed: 'Code analysis' (Imported 13 hours ago, Tested a few seconds ago, Issues 0) and 'pom.xml' (Imported 13 hours ago, Tested 2 hours ago, Issues 0). A message at the bottom says 'Ready to import another project? Secure your entire stack with Snyk'.

Conclusions

This lab made it possible to significantly improve the security and maintainability of the real estate property management system code. By using SonarCloud and Snyk, issues related to nomenclature, redundancy, security, and good development practices were identified and fixed. These improvements strengthen project security and ensure a more efficient and maintainable code.