CRAFTING STABLE

Presented by:
Filipe Sousa-114196/Team Manager/DevOps Master/Dev
Gonçalo Calvo-120131/ QA Engineer/Dev
Daniel Simbe-110235/Product Owner/Dev

Product Concept

PROBLEM

- High cost of tool ownership with low usage frequency
- Underutilization of privately owned equipment
- Lack of a centralized tool-sharing platform
- Inefficient and informal rental processes
- Absence of real-time availability and scheduling
- Trust and safety issues in peer-to-peer rentals

SOLUTION

- Centralized peer-to-peer rental marketplace
- Structured digital catalog with availability management
- Reservation and scheduling with conflict prevention
- Secure and integrated payment system
- Reputation and rating mechanisms
- Scalable, modular, and extensible platform architecture

SCENARIOS

1. The owner lists a tool by defining availability, pricing, and item details.
2. A renter searches, compares options, and reserves a tool for a specific period.
3. The platform processes payment, confirms the rental, and collects post-rental reviews.

EPICS

1 – ITEM DISCOVERY & SEARCH
BROWSE AND FILTER TOOLS BY CATEGORY, LOCATION, AND PRICE.

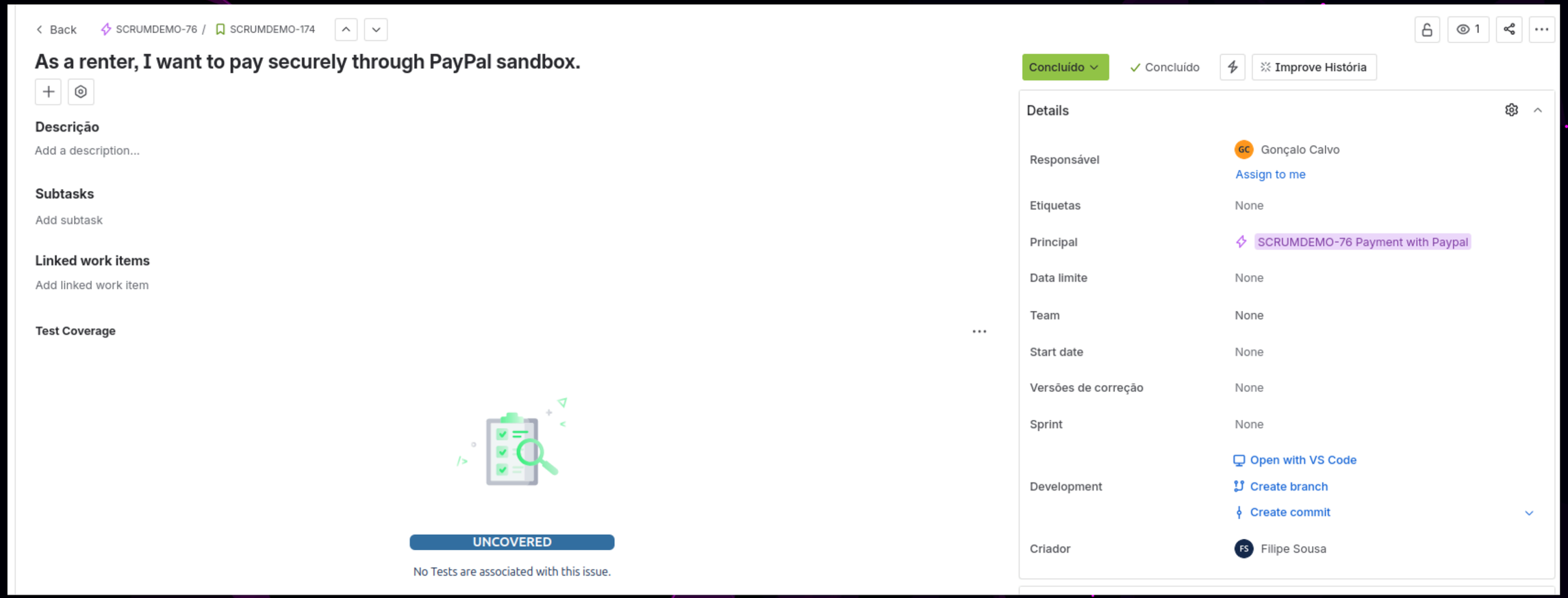
2 – BOOKING & SCHEDULING
MANAGE RESERVATIONS AND PREVENT BOOKING CONFLICTS.

3 – PAYMENTS & TRANSACTIONS
HANDLE SECURE PAYMENTS, REFUNDS, AND PAYOUTS.

4 – USER PROFILES & REPUTATION
MAINTAIN USER HISTORY, RATINGS, AND REVIEWS.

5 – PLATFORM MANAGEMENT & ANALYTICS
MONITOR PLATFORM ACTIVITY, TRANSACTIONS, AND SYSTEM METRICS.

User Stories



Software Architecture

FRONTEND

- React
- Vite
- Tailwind CSS

POSTGRESQL

- Persistante Data With Reservations and Chargers

BACKEND

- SpringBoot

NGINX

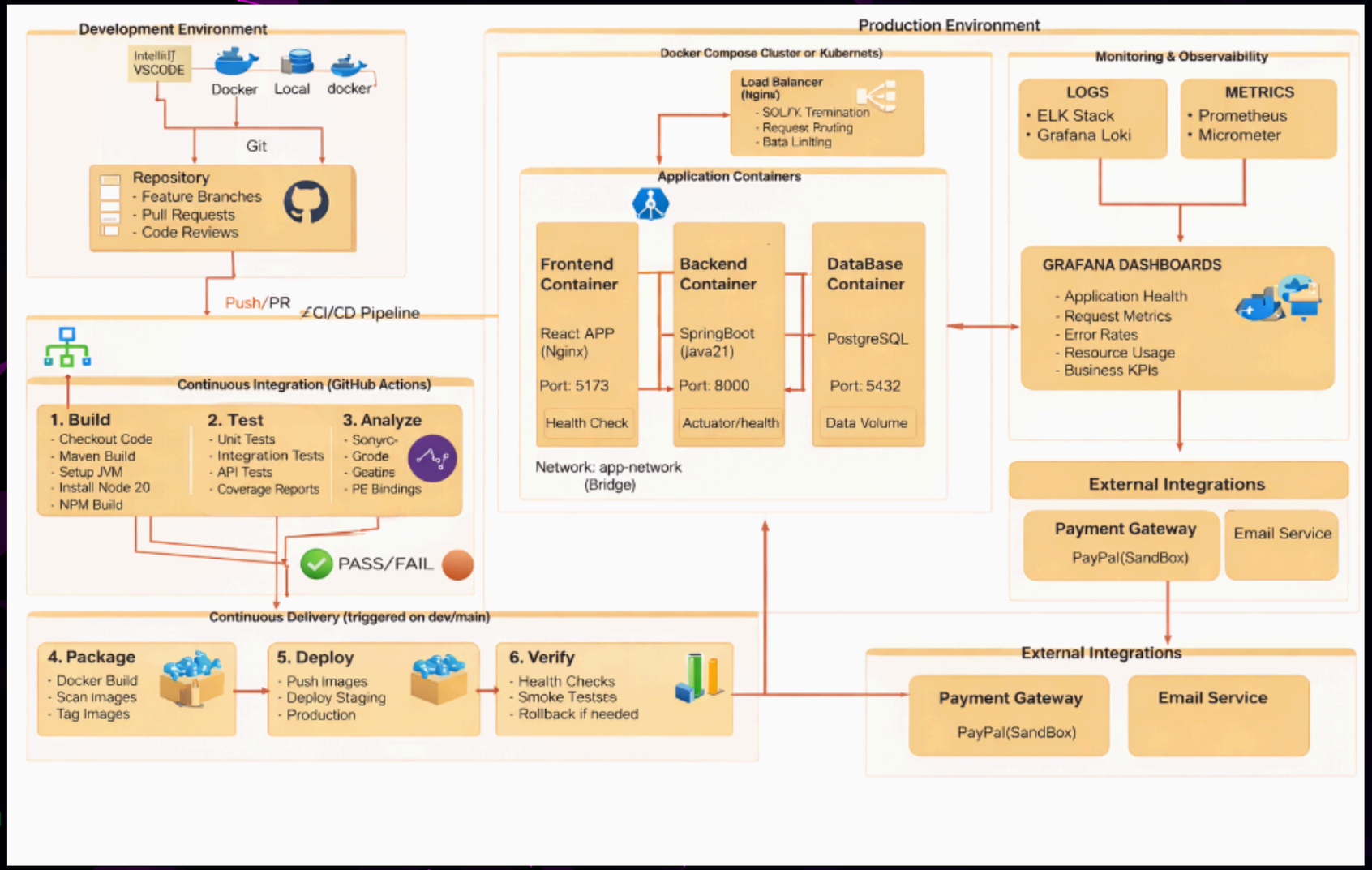
- Proxy to improve security on authentication

DOCKER

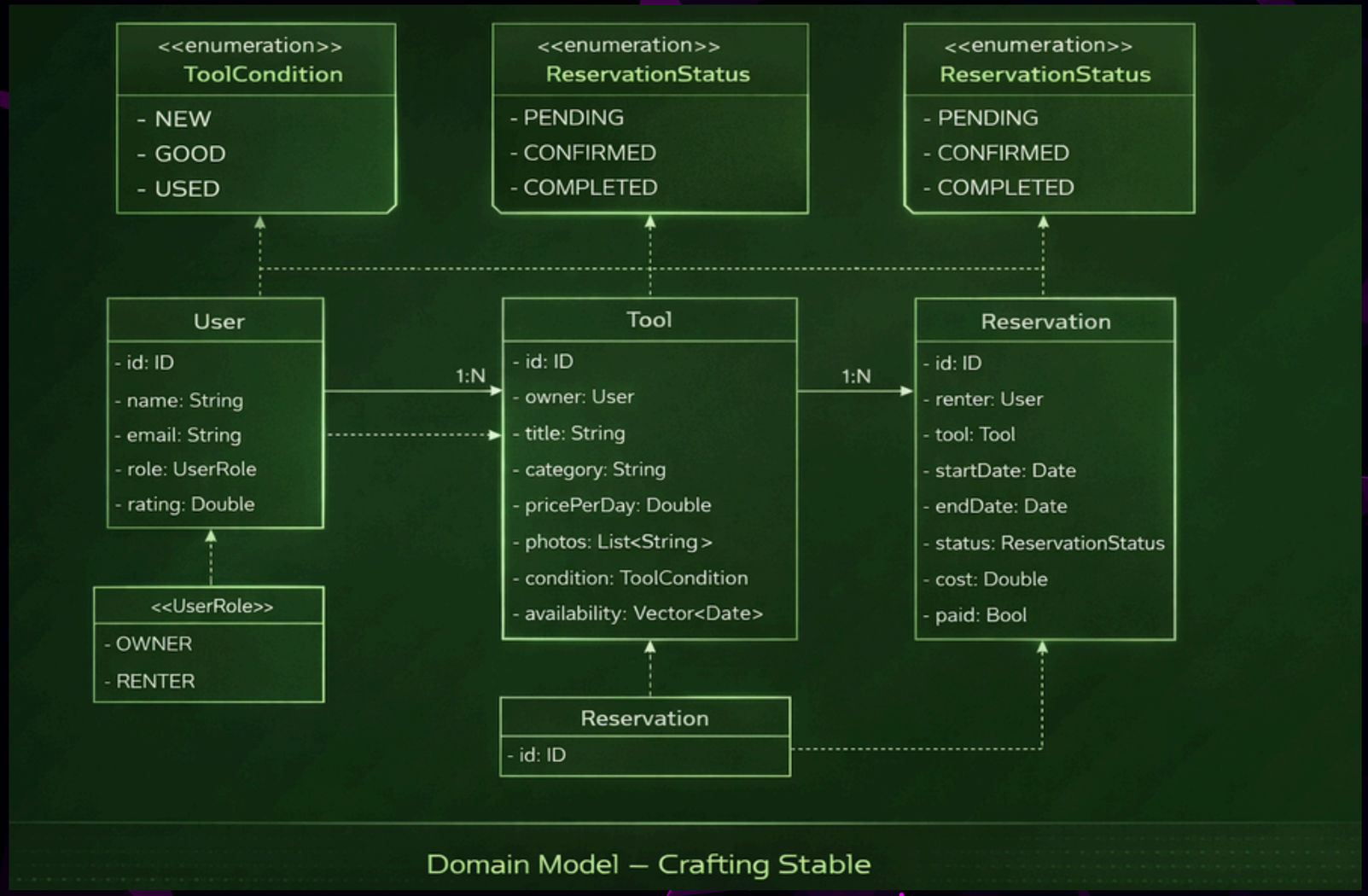
- All components containeraze

MONITOR

- Actuator=>Prometheus=>Grafana



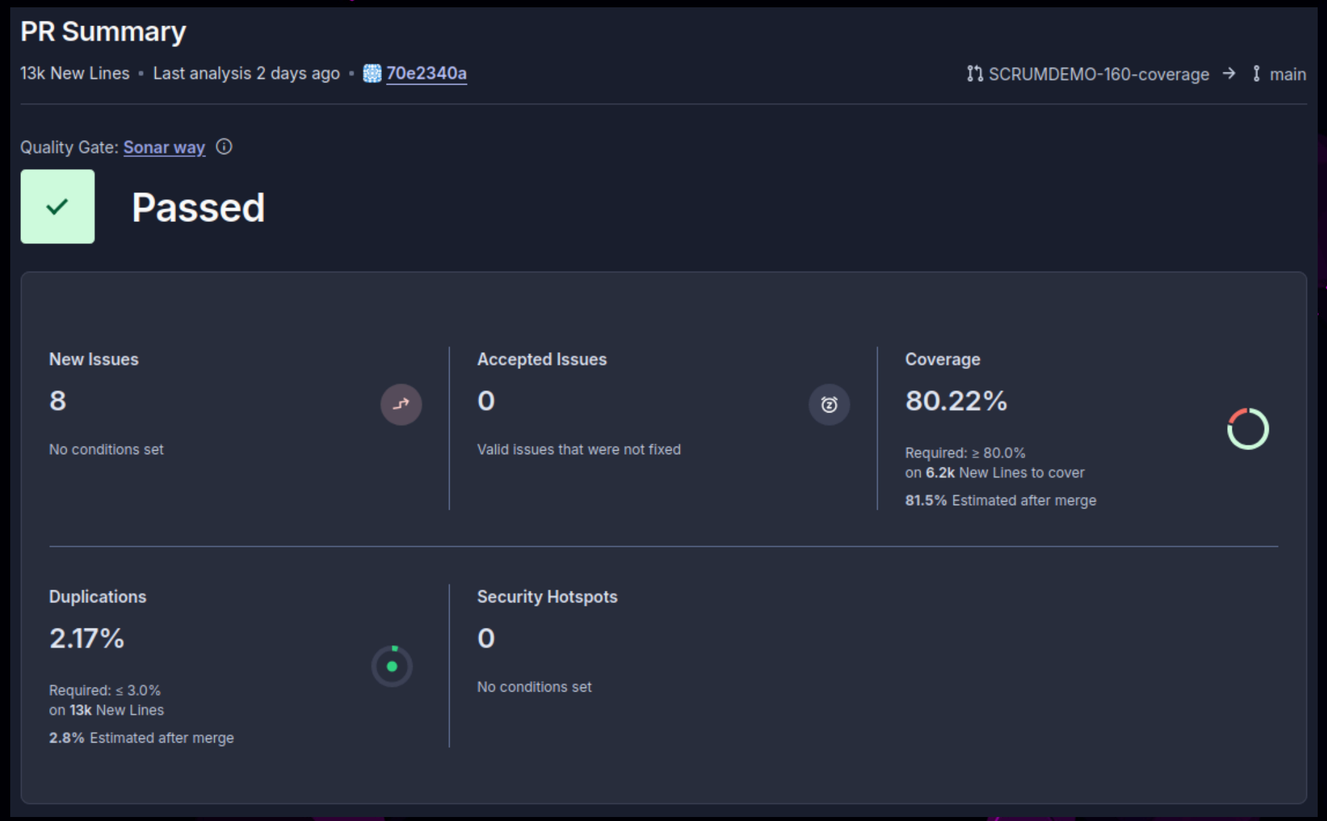
Domain Model



Domain Model — Crafting Stable

Code & Quality Management

SonarQube



Coding Style

- ESLint
  - JavaScript
  - TypeScript
- CheckStyle
  - Java
  - Google Style Guide

QA Stack

Automated Tests

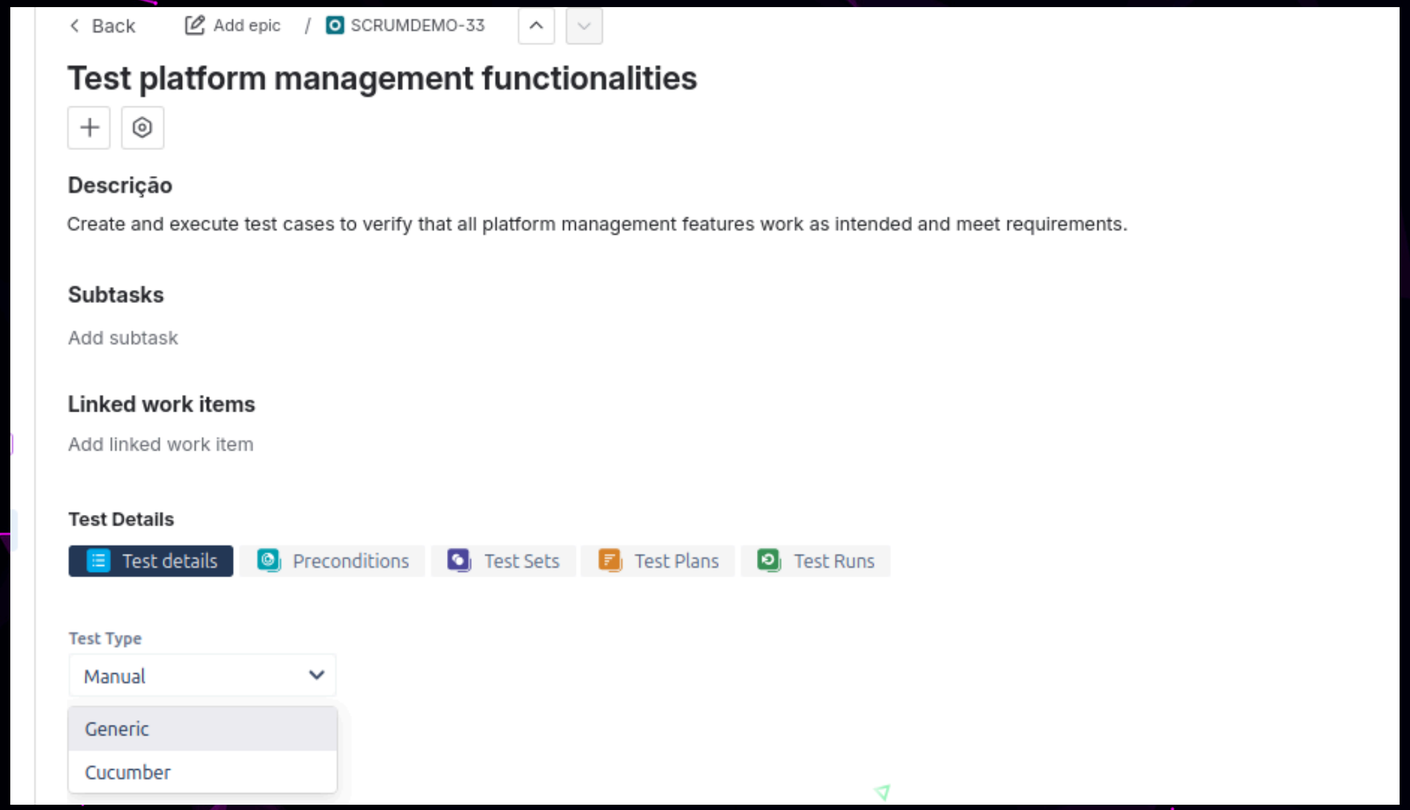
- Unit Tests: Junit[Backend>=80% coverage]
- Integration:Testcontainers[PostGreSQL]
- E2E:Cucumber+Playwright[Frontend]

Traceability

- User Stories==>Tests[Xray+Jira]
- Requirements Coverage: 100% MVP

Static Analysis

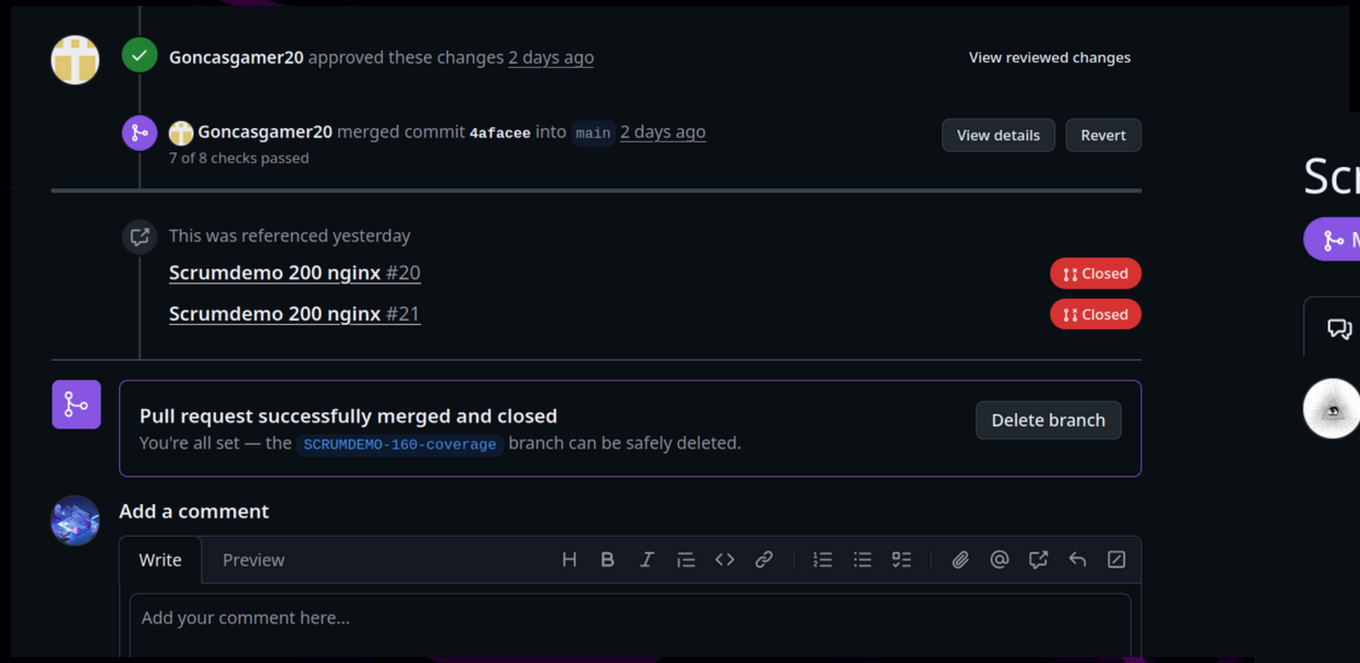
- SonaQube Quality Gate:
- Coverage>=80%
- Duplication<=3%



CI Pipeline


Pull Request
Test workflow
✓ JUnit · Cucumber
SonarCloud
✓ Quality Gate
Merge para main
Deploy workflow (deploy.yml)
Docker Compose
Self-hosted runner
Aplicação em produção

Code Review


Goncasgamer20 approved these changes 2 days ago
View reviewed changes
Goncasgamer20 merged commit 4afacee into main 2 days ago
7 of 8 checks passed
View details
Revert
This was referenced yesterday
Scrumdemo 200 nginx #20
Closed
Scrumdemo 200 nginx #21
Closed
Pull request successfully merged and closed
You're all set — the SCRUMDEMO-160-coverage branch can be safely deleted.
Delete branch
Add a comment
Write
Preview
Add your comment here...


Scrumdemo 160 coverage #11
Merged
Goncasgamer20 merged 91 commits into main from SCRUMDEMO-160-coverage 2 days ago
Conversation 2
Commits 91
Checks 7
Files changed 298
FilipePinaSousa commented 2 days ago • edited by coderabbitai bot
This Pull Request aims to address pending code coverage issues and enhance the project's quality met
SonarQube, ensuring that new or modified code meets the minimum threshold defined in the Quality
Quality Improvements (SonarQube Impact)
This change directly impacts the following SonarQube metrics:
Increased Coverage: Introduces new unit and/or integration tests to cover previously untested code bl
Line Coverage and Branch Coverage percentages.
Quality Gate Approval: Ensures the project passes the Quality Gate check by resolving issues related tc
coverage on new code (if applicable).
False Positive Suppression: (If applicable) Includes the suppression of false positives (such as the detec
in non-credential UI strings) using // NOSONAR to focus analysis only on genuine vulnerabilities.

Functional Tests

```java

    @Given("the application is running")
    public void applicationIsRunning() {
        assertNotNull(restTemplate, message: "TestRestTemplate should be initialized");
        assertNotNull(userRepository, message: "UserRepository should be initialized");
    }

    @And("the database is clean")
    public void databaseIsClean() {
        userRepository.deleteAll();
    }

    @Given("a user exists with email {string} and password {string}")
    public void userExists(String email, String password) {
        User user = TestDataFactory.createUser(email, password, UserRole.CUSTOMER);
        userRepository.save(user);
        sharedState.lastCreatedUser = user;
    }

    @When("the user logs in with email {string} and password {string}")
    public void userLogsIn(String email, String password) {
        AuthRequestDTO request = new AuthRequestDTO();
        request.setEmail(email);
        request.setPassword(password);

        sharedState.latestResponse = restTemplate.exchange(
            baseUrl + "/api/auth/login",
            org.springframework.http.HttpMethod.POST,
            new HttpEntity<>(request, sharedState.headers),
            String.class
        );
    }

    @Then("the login should be successful")
    public void loginSuccessful() {
        assertTrue(sharedState.latestResponse.getStatusCode().is2xxSuccessful(),
            "Login response should be 2xx, got: " + sharedState.latestResponse.getStatusCode());
        assertNotNull(sharedState.latestResponse.getBody());
    }

    @Then("the login response status is {int}")
    public void loginResponseStatusIs(int expectedStatus) {
        assertEquals(expectedStatus, sharedState.latestResponse.getStatusCode().value());
    }
```

```gherkin
Feature: Authentication and User Management
  As a user
  I want to authenticate and manage my account
  So that I can access the application securely

  Background:
    Given the application is running
    And the database is clean

  Scenario: User login with valid credentials
    Given a user exists with email "user@example.com" and password "password123"
    When the user logs in with email "user@example.com" and password "password123"
    Then the login response status is 200
    And the response contains a valid JWT token

  Scenario: User login with invalid credentials
    Given a user exists with email "user@example.com" and password "password123"
    When the user logs in with email "user@example.com" and password "wrongpassword"
    Then the login response status is 401

  Scenario: User login with non-existent email
    When the user logs in with email "nonexistent@example.com" and password "password123"
    Then the login response status is 401

  Scenario: User registration with valid credentials
    When the user registers with email "newuser@example.com" password "password123" and role "CUSTOMER"
    Then the registration response status is 201
    And the user is created in the system with email "newuser@example.com"

  Scenario: User registration with existing email
    Given a user exists with email "user@example.com" and password "password123"
    When the user registers with email "user@example.com" password "password123" and role "CUSTOMER"
    Then the registration response status is 400

  Scenario: Get authenticated user details
    Given a user exists with email "user@example.com" and password "password123"
    And the user is logged in with email "user@example.com"
    When the user requests their own details
    Then the response status is 200
    And the response contains the user email "user@example.com"

  Scenario: Access protected endpoint without authentication
    When an unauthenticated request is made to GET "/api/users"
    Then the response status is 403

```

Unit & Integration Tests

Integration Tests

```java
@Test
void quandoLoginValido_retornaTokenE200() throws Exception {
    String token = "token-abc-123";

    UserDetails user = new org.springframework.security.core.userdetails.User(
            "test@example.com",
            "password",
            List.of(new SimpleGrantedAuthority("ROLE_USER"))
    );

    when(authManager.authenticate(any(type: UsernamePasswordAuthenticationToken.class)))
            .thenReturn(new UsernamePasswordAuthenticationToken(user.getUsername(), user.getPassword(), user.getA

    when(userDetailsService.loadUserByUsername(eq(value: "test@example.com"))).thenReturn(user);
    when(jwtUtil.generateToken(eq(value: "test@example.com"), anyString())).thenReturn(token);
    when(userDetailsService.getUserId(any(type: UserDetails.class))).thenReturn(value: 1L);
    when(userDetailsService.getUserName(any(type: UserDetails.class))).thenReturn(value: "Test User");

    Map<String, String> payload = Map.of(
            "email", "test@example.com",
            "password", "password"
    );

    mvc.perform(post("/api/auth/login")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(mapper.writeValueAsString(payload)))
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$.token").value(token));
}
```

```java
@WebMvcTest(AuthController.class)
@AutoConfigureMockMvc(addFilters = false)
public class AuthControllerTest {

    @Autowired
    private MockMvc mvc;

    @Autowired
    private AuthenticationManager authManager;

    @Autowired
    private UserDetailsServiceImpl userDetailsService;

    @Autowired
    private JwtUtil jwtUtil;

    private final ObjectMapper mapper = new ObjectMapper();
```

Unit Tests

```java
@Test
void whenValidUsername_thenUserShouldBeFound() {
    User user = new User();
    user.setEmail(email: "test@user.com");
    user.setPassword(password: "password");
    user.setType(UserRole.CUSTOMER);

    when(userRepository.findByEmail(username: "test@user.com")).thenReturn(Optional.of(user));

    UserDetails userDetails = userDetailsService.loadUserByUsername(username: "test@user.com");
    assertEquals(expected: "test@user.com", userDetails.getUsername());
    assertTrue(userDetails.getAuthorities().stream().anyMatch(a -> a.getAuthority().equals("ROLE_CUSTOMER")));
}
```

```java
@ExtendWith(MockitoExtension.class)
class UserDetailsServiceImplTest {

    @Mock
    private UserRepository userRepository;

    @Mock
    private PasswordEncoder passwordEncoder;

    @InjectMocks
    private UserDetailsServiceImpl userDetailsService;
```

```java
@Test
void whenInvalidUsername_thenThrowUsernameNotFoundException() {
    when(userRepository.findByEmail(username: "nonexistent@user.com")).thenReturn(Optional.empty());

    assertThrows(expectedType: UsernameNotFoundException.class, () -> {
        userDetailsService.loadUserByUsername(username: "nonexistent@user.com");
    });
}
```

Product tour - DEMO

PMI Board

Plus	Minus	Interesting
Almost 90% code coverage	Started treating tests at the middle of development	Leading with data coming from external API's to take the most advantage of them
Robust CI/CD pipelines	Cannot book external [from maps API] charger slots	Sonarqube integrated in the pipeline
Good workflow and communication within the team	Selenium Testing [used Playwright instead]	Integration of Xray with Jira and communicate with the team "without communicating"

THANK YOU!!

Questions?

https://github.com/Crafting-Stable/CraftingStable