KVALITETA KODA

Statička analiza koda

SonarQube & SonarLint

Java/Spring | .NET | React | Angular | Vue.js

Što je statička analiza koda?

DEFINICIJA

Automatska provjera izvornog koda **bez izvršavanja programa** – pronalazi greške, sigurnosne propuste i loše prakse.

💡 Analogija

Kao spell-checker za kod – ali umjesto pravopisa, provjerava logiku, sigurnost i stil.

DETEKTIRA

Bugove – null reference, memory leaks

Sigurnosne rupe – SQL injection, XSS

Code smells – duplicirani kod, predugi metodi

Stil – nekonzistentno imenovanje

Zašto je statička analiza važna?

85%

bugova pronađeno prije produkcije

10x

jeftinije popraviti bug u razvoju

30%

manje vremena na code review

STVARNI PRIMJERI KATASTROFA

🚀 **Ariane 5 (1996)**

Integer overflow – raketa eksplodirala 37 sekundi nakon lansiranja. Šteta: $370 milijuna.

💳 **Equifax (2017)**

Neažurirana biblioteka s poznatom ranjivošću – procurili podaci 147 milijuna korisnika.

🔐 **Heartbleed (2014)**

Buffer over-read u OpenSSL – mogao se otkriti statičkom analizom.

SonarQube – Centralizirana platforma

🎯 Što je SonarQube?

Open-source platforma za kontinuiranu inspekciju kvalitete koda. Pokreće se na serveru i analizira cijeli projekt.

PODRŽANI JEZICI

Java | C# | JavaScript | TypeScript | Python | Go | PHP | +20 drugih

KLJUČNE ZNAČAJKE

Dashboard
Vizualni pregled metrika projekta

Quality Gates
Pass/Fail kriteriji za deploy

CI/CD Integracija
GitHub Actions, GitLab CI, Jenkins

Povijesni trendovi
Praćenje kvalitete kroz vrijeme

SonarLint – Analiza u IDE-u

Što je SonarLint?

Besplatni IDE plugin koji analizira kod **dok pišeš**. Kao spell-checker – crveno podcrtava probleme u realnom vremenu.

PODRŽANI IDE-OVI

VS Code | IntelliJ IDEA | Visual Studio | Eclipse

Connected Mode

Spoji s SonarQube serverom za sinkronizaciju pravila!

PREDNOSTI

Instant feedback

Vidiš greške dok pišeš, ne čekaj CI

Edukativne poruke

Objašnjava zašto je nešto problem

Quick fixes

Automatski popravci jednim klikom

Besplatno

Open source, bez licenciranja

Quality Gates – Čuvar produkcije

🚦 Što je Quality Gate?

Skup uvjeta koji kod mora zadovoljiti prije deploya. Ako ne prođe – build pada!

TIPIČNI UVJETI

- ✓ Coverage ≥ 80%
- ✓ 0 kritičnih bugova
- ✓ 0 sigurnosnih ranjivosti
- ✓ Duplicirani kod < 3%

✓ **PASSED**
Spreman za deploy

A
Reliability

✗ **FAILED**
Popravi prije deploya

D
Security

💡 **Pro tip**

Koristi "Sonar way" kao početni Quality Gate – balansiran je za većinu projekata.

JAVA

Loš kod – SQL Injection

✕ CRITICAL

```
// ❌ NIKADA ovako!
public User findByUsername(String username) {
String sql = "SELECT * FROM users WHERE "
+ "username = '" + username + "'";
return jdbc.queryForObject(sql, mapper);
}
```

🔓 Problem: Input ' OR '1'='1 vidi sve korisnike

⚠ S3649: SQL parameters should be sanitized

JAVA Ispravan kod – Parametrizirani upit ✓ SECURE

```java
// ✅ Parametrizirani upit - sigurno!
public User findByUsername(String username) {
// Placeholder ? sprječava injection
String sql = "SELECT * FROM users WHERE username = ?";
return jdbc.queryForObject(sql, mapper, username);
}
```

✓ Placeholder ? automatski escapea opasne znakove

💡 Alternative: Spring Data JPA, Hibernate HQL

.NET Loš kod – Nesiguran password hash

✕ CRITICAL

```
// ❌ Nikada ne koristi MD5 za lozinke!
public string HashPassword(string password) {
// 🚨 MD5 je BROKEN!
using var md5 = MD5.Create();
var hash = md5.ComputeHash(Encoding.UTF8.GetBytes(password));
return Convert.ToBase64String(hash);
}
```

🔓 MD5 rainbow tables cracku lozinke u sekundama

⚠ S4790: Weak cryptographic algorithm

.NET Ispravan kod – BCrypt hashing ✓ SECURE

```
// ✅ BCrypt - industrijski standard
using BCrypt.Net;
public string HashPassword(string password) {
return BCrypt.HashPassword(password, workFactor: 12);
}
public bool Verify(string pw, string hash) {
return BCrypt.Verify(pw, hash);
}
```

✓ Auto salt, adaptive cost, dizajniran da bude spor

💡 Alternative: ASP.NET Identity, Argon2, PBKDF2

REACT

Loš kod – XSS ranjivost

✕ CRITICAL

```
// ❌ dangerouslySetInnerHTML bez sanitizacije!
const Comment = ({ userComment }) => {
return (
<div
{/* 🚨 XSS RANJIVOST! */}
dangerouslySetInnerHTML={{ __html: userComment }}
/>
);
};
```

🔓 Napadač unese: <script>stealCookies()</script>

⚠ S6845: dangerouslySetInnerHTML should be avoided

REACT

Ispravan kod – DOMPurify sanitizacija

✓ SECURE

```
// ✅ Opcija 1: React auto-escape (preporučeno)
const Comment = ({ text }) => <div>{text}</div>;

// ✅ Opcija 2: Ako MORAŠ koristiti HTML - sanitiziraj!
import DOMPurify from 'dompurify';
const RichComment = ({ html }) => {
const clean = DOMPurify.sanitize(html);
return <div dangerouslySetInnerHTML={{ __html: clean }} />;
};
```

✓ React escapea tekst. Za HTML koristi DOMPurify

💡 Izbjegavaj dangerouslySetInnerHTML!

ANGULAR

Loš kod – Memory leak

✕ BUG

```
// ❌ Subscription bez unsubscribe = memory leak!
@Component({ selector: 'app-user' })
export class UserComponent implements OnInit {
user: User;
ngOnInit() {
// 🚨 Nikad nije otkazana!
this.userService.getUser().subscribe(u => this.user = u);
}
// ❌ Nema ngOnDestroy!
}
```

🦠 Subscription ostaje aktivna, app postaje sporija

⚠ S6544: Subscription should be unsubscribed

ANGULAR

Ispravan kod – takeUntilDestroyed

✓ SECURE

```
// ✅ Angular 16+ - takeUntilDestroyed
import { takeUntilDestroyed } from '@angular/core/rxjs-interop';
@Component({ selector: 'app-user' })
export class UserComponent {
private destroyRef = inject(DestroyRef);
constructor(private svc: UserService) {
svc.getUser().pipe(takeUntilDestroyed(this.destroyRef))
.subscribe(u => this.user = u);
} // ✅ Auto cleanup!
}
```

✓ Auto otkazuje subscription kad se komponenta uništi

💡 Alt: async pipe, Subject+takeUntil, ngOnDestroy

VUE.JS

v-for bez :key – Česta greška

✗ Loše

```
<template>
<ul>
<li v-for="user in users">
{{ user.name }}
</li>
</ul>
</template>
```

⚠ S6481: v-for requires :key

✓ Dobro

```
<template>
<ul>
<li
v-for="user in users"
:key="user.id">
{{ user.name }}
</li>
</ul>
</template>
```

✓ :key osigurava praćenje DOM elemenata

CI/CD Integracija

.github/workflows/sonar.yml

```
name: SonarQube Analysis
on: [push, pull_request]
jobs:
sonarqube:
runs-on: ubuntu-latest
steps:
uses: actions/checkout@v4
uses: sonarsource/sonarqube-scan-action@v2
env:
SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
SONAR_HOST_URL: ${{ vars.SONAR_URL }}
```

PODRŽANE PLATFORME

- GitHub Actions
- GitLab CI/CD
- Azure DevOps
- Jenkins

Workflow

Push → CI pokreće analizu → Quality Gate → Pass/Fail → Deploy ili blokiraj

Best Practices

1 Instaliraj SonarLint

Vidiš probleme dok pišeš kod – ne čekaj CI pipeline da ti javi greške!

2 Ne ignoriraj warninge

Svaki warning je potencijalni bug. Popravi ih odmah ili dokumentiraj zašto ih ignoriraš.

3 Definiraj Quality Gate

Blokiraj merge ako ne prođe Quality Gate. Zaštiti main branch!

4 Koristi Connected Mode

Spoji SonarLint sa SonarQube serverom – ista pravila u IDE-u i CI-u!

5 Fokusiraj se na novi kod

"Clean as You Code" – ne trebaš popraviti sve odjednom. Fokus na novi kod!

6 Educiraj tim

SonarLint poruke su edukativne – čitaj ih i uči iz grešaka!

Zaključak

- ✓ **SonarLint** = instant feedback dok pišeš kod
- ✓ **SonarQube** = centralizirani pregled kvalitete
- ✓ **Quality Gates** = automatska zaštita produkcije

SLJEDEĆI KORAK

Instalirati SonarLint u svoj IDE

sonarqube.org | sonarlint.org | docs.sonarqube.org

Projektni zadatak

- [x] Instalirati SonarList ili SonarQube plugin
- [x] Skenirati kod, napraviti screenshot problema
- [x] Ispraviti sve prijavljene probleme u svom dijelu koda (I8 – 2 boda)