# coding -rules

### Regels clean-coding

1. Geef betekenisvolle namen aan variabelen, functies en klassen. Gebruik namen die de intentie en het doel van de code duidelijk weergeven.
2. Houd je code modulair en goed georganiseerd. Gebruik functies en klassen om logische eenheden van functionaliteit te scheiden.
3. Schrijf code die gemakkelijk te begrijpen is. Voeg commentaar toe wanneer dat nodig is, maar zorg ervoor dat de code zelf al goed leesbaar is.
4. Houd de lengte van functies en methoden(1 methode) beperkt. Lange functies zijn moeilijker te begrijpen en te onderhouden. Overweeg om complexe functies op te splitsen in kleinere, meer beheersbare delen.
5. Volg de principes van DRY (Don't Repeat Yourself) en KISS (Keep It Simple, Stupid). Herhaal geen code als het vermeden kan worden en houd de code eenvoudig en duidelijk.
6. Gebruik consistente opmaak en inspringing. Dit maakt de code gemakkelijker leesbaar en begrijpelijk voor anderen.
7. Zorg voor goede error handling en foutafhandeling. Anticipeer op mogelijke fouten en zorg ervoor dat je code robuust is en correct omgaat met uitzonderlijke situaties.
8. Vermijd het gebruik van magische getallen en harde codering van waarden. Gebruik betekenisvolle constanten of variabelen om de intentie van de code duidelijk te maken.
9. Voer regelmatig code reviews uit en wees bereid om feedback te ontvangen en te geven. Het samenwerken met andere ontwikkelaars kan leiden tot verbeteringen en het delen van best practices.
10. Blijf continu leren en verbeteren. Houd jezelf op de hoogte van nieuwe ontwikkelingen en technieken in de softwareontwikkeling en pas deze toe in je code.

### Regels best practices

1. Planning en Requirements:

Zorg voor een duidelijke en volledige definitie van de vereisten voordat je aan de ontwikkeling begint.

Maak gebruik van gestructureerde planningstechnieken, zoals user stories of use cases, om de functionaliteit en het gedrag van de software te beschrijven.

1. Versiebeheer:

Gebruik een versiebeheersysteem, zoals Git, om wijzigingen in de code bij te houden en samen te werken met andere ontwikkelaars.

Maak regelmatig commits met betekenisvolle berichten om de geschiedenis en de intentie van de wijzigingen te documenteren.

1. Codekwaliteit:

Schrijf leesbare, goed gestructureerde code volgens de clean coding-richtlijnen die ik eerder heb genoemd.

Voer regelmatig code reviews uit om bugs, potentiële problemen en verbeterpunten te identificeren.

Maak gebruik van statische code-analysehulpmiddelen om codekwaliteit te controleren en mogelijke problemen op te sporen.

1. Testen:

Implementeer een teststrategie die unit tests, integratietests en systeemtests omvat.

Schrijf testbare code en automatiseer tests om regressiebugs te voorkomen en de kwaliteit van de software te waarborgen.

1. Schaalbaarheid en prestaties:

Ontwerp de software met het oog op schaalbaarheid, zodat het kan omgaan met toenemende gebruikers, gegevens en verkeer.

Optimaliseer kritieke delen van de code om de prestaties te verbeteren en vermijd inefficiënte algoritmen of databasequeries.

1. Beveiliging:

Implementeer beveiligingsmaatregelen om de software te beschermen tegen bekende bedreigingen, zoals authenticatie, autorisatie en gegevensversleuteling.

Volg best practices voor veilige code, zoals het vermijden van kwetsbaarheden zoals SQL-injectie of cross-site scripting (XSS).

1. Documentatie:

Documenteer de code, inclusief functies, klassen en modules, om andere ontwikkelaars te helpen de code te begrijpen en er effectief mee samen te werken.

Documenteer ook de installatie, configuratie en gebruiksinstructies van de software om het voor gebruikers gemakkelijk te maken.

1. Continuous Integration en Deployment:

Maak gebruik van Continuous Integration (CI) en Continuous Deployment (CD) om wijzigingen in de code automatisch te integreren, te testen en te implementeren.

Automatiseer zoveel mogelijk processen om menselijke fouten te verminderen en een snelle en efficiënte ontwikkelingsworkflow te bevorderen.