Software Review (Based on Software Review Article):

Vi startede med at gennemgå den funktionelle del af vores kode, hvor vi sikrede, at den opfyldte de fastsatte krav og udførte de nødvendige opgaver, herunder CRUD-funktionerne. Derefter fokuserede vi på de ikke-funktionelle aspekter, hvor vi evaluerede koden med hensyn til ydeevne, vedligeholdelse og testbarhed. Ved hjælp af load tests vurderede vi performance, og vi sørgede for, at koden var modulær og nem at teste i overensstemmelse med designprincipper som Single Responsibility Principle (SRP). Endelig vurderede vi, om best practices var blevet fulgt, herunder korrekt kodestruktur, overholdelse af designprincipper, hensigtsmæssig navngivning, dokumentation og fejlhåndtering med passende statuskoder og tests.

Fund

Ved gennemgang af koden identificerede vi flere områder med forbedringspotentiale, men også nogle positive aspekter. Vores projekt klarede overraskende godt i load tests, hvilket er tilfredsstillende, da projektet er relativt lille. Dog bemærkede vi flere steder i koden, hvor navngivningen var uklar, hvilket gjorde logikken svær at følge. Dokumentationen var begrænset til Swagger, hvilket kan blive problematisk i takt med, at systemet bliver mere komplekst. Derudover manglede der tests for visse edge cases, hvilket kan påvirke både testdækningen og kvalitetssikringen.

Forbedringer

For at adressere disse udfordringer implementerede vi en række forbedringer. Først refaktorerede vi dele af koden for at gøre den mere enkel og læsbar. Vi forbedrede også navngivningen for at gøre logikken lettere at forstå og tilføjede flere kommentarer i både koden og tests. Derudover skrev vi de manglende tests for at øge testdækningen og sikre, at alle edge cases blev håndteret. Disse ændringer gjorde koden mere robust og nemmere at arbejde med fremadrettet. For dokumentationen overvejede vi muligheden for at tilføje XML-kommentarer til controller- og metodebeskrivelser, så Swagger automatisk kunne inkludere dem. Dette er noget, vi kan implementere i fremtiden for at forbedre dokumentationen yderligere. I denne omgang har vi dog endnu ikke tilføjet XML-kommentarer.

Reflection on Testing and Code Quality

Da vi kodede i C#, brugte vi FxCop, hvilket gjorde det muligt for os at opdage problemer i vores kodebase, som ville have været vanskelige at identificere manuelt, såsom inkonsistente navngivningskonventioner og ineffektive loops. Dette sparede os tid og hjalp os med at fange fejl tidligt i udviklingsprocessen. Det understøttede også overholdelsen af best practices på tværs af hele kodebasen.

Mocking var essentielt for at sikre, at vores unit tests var isolerede og fokuserede udelukkende på komponenternes logik uden at være afhængige af eksterne ressourcer som databaser. Ved at bruge mocks blev testene udført langt hurtigere, da vi undgik netværkskald og databaseforespørgsler, der kunne forsinke testprocessen. Selvom det på nuværende tidspunkt ikke var en stor faktor, ville det i takt med projektets skalering gøre en betydelig forskel. Dette forbedrede vores testdækning og gav os større tillid til kodens robusthed, da vi kunne teste forskellige scenarier effektivt.

Equivalence Partitioning og Boundary Value Analysis kunne have haft en væsentlig indflydelse på vores teststrategi ved at sikre, at vi dækkede alle relevante inputområder. Ved at fokusere på edge cases opdagede vi potentielle problemer ved grænsetilfælde, hvor fejl ofte opstår. Selvom vi ikke brugte disse strategier tidligt i udviklingsprocessen, kunne vi efterfølgende se de klare fordele ved at implementere dem i vores tests.

Både software reviews og code reviews gav os værdifuld feedback. Code reviews hjalp med at sikre den funktionelle korrekthed af koden, mens software reviews adresserede ikke-funktionelle aspekter som ydeevne og vedligeholdelse. Dette gav os en dybere forståelse af kvalitetskravene og hjalp os med at identificere problemer. Havde vi indført disse processer tidligere i udviklingsforløbet, kunne vi have reduceret fejl, især gentagne fejl, og sparet tid på længere sigt.