Testning

Alexander Larsson

SYSM8

Inledning

Vi har byggt en webbapplikation som heter **ListLife**, en inköpslista-app främst fokuserad på matvaror. I ListLife kan användare skapa, redigera och ta bort inköpslistor, samt dela listor med andra användare. Målet är att ListLife på sikt ska kunna hantera fler typer av listor, såsom att göra-listor, utgiftslistor eller andra strukturerade anteckningar, en app för att organisera livet med hjälp av listor.

I detta projekt har fokus legat på att testa och säkerställa kvaliteten i applikationen genom en kombination av unit tester och automatiserade UI-tester. I testningen kontrolleras appens funktionalitet och testbarhet.

Testbarhet

Testbarheten i appen har varit varierande. Det mesta har gått att testa men strukturen har mycket att önska för att underlätta både testning och förståelse av koden.

Ett tydligt exempel är att vi inte hade några services utan större delen av logiken är direkt kopplad till sidorna eller hanteras i back enden för respektive vy. Detta försvårar att isolera logiken för unit testning.

Vi har all kod fördelad på huvudsakligen 2 sidor, My page och Create new shoppinglist vilket gör det rörigt och svårläst. Hade jag haft tiden till att strukturera om det hade jag jobbat med fler sidor och partial pages.

Mycket av dessa tillkortakommanden kommer ifrån bristande planering av oss som grupp när vi började med projektet. Vi delade upp uppgifterna men satte aldrig en tydlig struktur för hur projektets olika delar skulle organiseras.

Testning

När jag skulle börja med testningen började jag med att, steg för steg, gå igenom exakt vad vår app gjorde och hur. Därefter la jag upp en planering för vilken ordning testerna skulle utföras och kom fram till att unit testerna kändes mest logiskt att börja med för funkar de så vet jag att det inte är dem det är fel på när jag sedan kör specflow/playwright testerna.

Då hela vår apps syfte är spara och visa data från databasen blev det snabbt tydligt att jag var tvungen att sätta upp en In-memory database för simulera listor under

kontrollerade former. Eftersom de flesta unit tester använde sig av den så skapade jag en helper jag kunde kalla på istället för att upprepa kod.

I början förstod jag det som att detta inte var renodlade unit tester eftersom en databas användes. Men min förståelse för vad som är ett integrationstest har ändrats flera gånger under arbetets gång, det är inte alltid tydligt när man försöker läsa sig till svaret. Men jag har nu landat i att även om jag simulerar en databas så är det isolerat till det specifika testet med mockad data och räknas därför inte som integrations test då det inte testar relationer till någonting annat i projektet.

Jag insåg snabbt att man skulle kunna skriva hur många tester som helst om man ville testa alla tänkbara scenarion. Samtidigt kände jag att det blir mycket upprepning av kod som inte tillförde förståelse för något nytt. Därför jag gick över till BDD testningen när jag kände att jag täckt de flesta väsentliga bitarna i unit testningen.

Eftersom vår app inte är så stor och inte har så många funktioner behövdes det inte särskilt mycket planering för vad testerna skulle göra. Jag skapade ett scenario outline test för inloggning där jag testade alla olika scenarios jag kunde komma på. Därefter gjorde jag ett test som testar hela flödet - från inloggning till att skapa en lista med produkter från alla kategorier och sedan kontrollera att den sparats korrekt under “mina listor”.

Resultat och förbättringar

Under testandets gång har ett par förändringar gjorts. Bland annat bröt jag ut koden för delning av listor och lade den i en egen service för enklare testning göra den återanvändbar när fler listtyper tillkommer. Denna typ av uppdelning hade kunnat göras på fler ställen i appen, men eftersom det huvudsakliga syftet var testning ville jag inte lägga allt för mycket tid på detta och riskera att appen slutar fungera vilket hade kostat ännu mer tid.

Jag upptäckte också ett fel där man kunde spara negativa antal till en produkt i listan. Detta åtgärdades snabbt genom att lägga till en validering i produktmodellen som inte tillåter negativa värden.

Det finns många delar i appen med förbättringspotential. Ett exempel är att jag gärna hade haft en addProduct metod istället för att lägga till produkter direkt i en `foreach`-loop i sidan. En sådan metod hade förenklat testning, förbättrat underhåll och gjort koden mer återanvändbar när nya funktioner läggs till.

Jag skulle även vilja ändra de långa javascript stycken vi har till c# i den mån det går då javascript är svår testat och dela upp logiken. Just nu sköter showListDetails att hämta data, rendera HTML och lagra checkboxstatus något som helst skulle separeras i flera delar.

Reflektion

Arbetet med att testa vår applikation har gett mig en helt annan förståelse till varför man ska hålla koden så konkret och tydlig så möjligt, det blir enklare att skriva tester och förstå vad som ska testas.

Något som jag verkligen har fått jobba med är att inte bara skriva tester som bekräftar det jag redan vet. Det är lätt att att det blir så när det är du som själv har byggt appen och vet hur allt fungerar, då får man tänka som en utomstående användare som kanske inte gör allt precis som jag har tänkt att man ska göra det.

Testningen har också fått mig att se vikten av att skriva kod med testbarhet i åtanke från början. Många av de förändringar jag gjorde under arbetets gång – som att bryta ut logik till en separat service – var egentligen sådant jag gärna hade haft från början. Det är en lärdom jag tar med mig till framtida projekt.