Testning av ListLife

Kurs: YSYS-TES

Klass: SYSM8

Termin och år: VT 2025

Författare: Nän Ekman

Lärare: Benjamin Berglund

Sammanfattning

Denna rapport beskriver testningen av applikationen ListLife, utvecklad av mig, Alexander Larsson och Farishta Sulaiman.

Vid utförande av mina tester har jag använt mig av youtube-videos, google, Chat-GPT och diskuterat en del med mina klasskompisar. Jag och Farishta har diskuterat en del för att hjälpa varandra förstå och ta oss förbi vissa barriärer som vi fastnat på vid testningens gång.

Testningen fokuserade på att säkerställa applikationens funktionalitet och användarvänlighet. Genom både utforskande och funktionella tester identifierades flera brister, såsom otillräcklig validering vid inloggning och registrering samt problem med produktmängder i listorna.

Testning med mockade databaser och isolerade enhetstester visade på olika metoder för att testa applikationens funktionalitet utan externa beroenden. Trots några utmaningar i testprocessen, som behovet av mer specifika Gherkin-syntax och integration med externa resurser, kunde flera förbättringsområden identifieras, bland annat hantering av användarkonton och produktkategorisering.

Innehållsförteckning

Inledning

Syftet med denna rapport är att visa hur jag testat mitt projekt ListLife. Rapporten fokuserar på applikationens testbarhet.

Varför är testning viktigt?

Testning är en avgörande del av mjukvaruutveckling eftersom den säkerställer att applikationen fungerar enligt förväntning, är användarvänligt och fri från buggar. Genom testning kan man också få en möjlighet att förbättra kvaliteten på projektet.

Rapportens kapitel

Planering

Innan jag började testa min applikation ville jag få en känsla för hur allt fungerar, både genom att testa lite på egen hand och samtidigt kolla att funktionerna faktiskt gör det de ska. Jag började med att klicka runt i gränssnittet för att se om det fanns något som var otydligt, rörigt eller om något bara inte funkade som det skulle. Jag gjorde med andra ord ett slags utforskande testande, där jag försökte se applikationen med "nya ögon". Samtidigt gjorde jag också funktionella tester, inklusive E2E-tester med Playwright och Gherkin/SpecFlow, för att försäkra mig om att de viktigaste funktionerna som att skapa listor, lägga till produkter och så vidare, fungerade som tänkt.

En annan aspekt av planeringen var att analysera koden för att identifiera eventuella brister i logik och struktur. Jag fokuserade på att förstå hur varje metod fungerade och om den var testbar. Denna reflektion ledde till identifiering av områden för förbättring, både vad gäller kodkvalitet och funktionalitet i applikationen.

Min grundinställning var att ständigt ifrågasätta hur vi skulle kunna ha tänkt annorlunda när vi byggde applikationen, vilket påminner om att mjukvara alltid kan förbättras.

Utmaningar

Jag började med att skriva all Gherkin-syntax för de tester jag ville göra. I efterhand insåg jag att det kanske inte var det smartaste att lägga så mycket tid på det direkt, eftersom jag ändå behövde gå tillbaka och justera flera av dem. Jag hade missat vissa steg här och där, och dessutom hade jag problem med att SpecFlow inte kunde matcha vissa steg, särskilt när de hade väldigt liknande namn.

Ett annat moment som ställde till det var att man behövde vara inloggad för att köra varje test. För den första sidan jag skrev tester för (CreateNewShoppinglist), använde jag Background: Given I am logged in, vilket kändes smidigt till en början. Men tyvärr funkade det inte så bra när jag började skriva tester för andra sidor, då jag inte riktigt förstod hur, eller om, jag kunde återanvända Background-steget från CreateNewShoppinglist i andra feature- och steps-filer. Jag testade att lägga till Scope

för CreateNewShoppinglist-steget i hopp om att isolera just de stegen till den feature-filen, så att de inte skulle krocka med andra. Det fungerade till viss del, men det blev också tydligt att jag behövde ha bättre struktur på vilka steg som var generella och vilka som var specifika för en viss sida eller funktion.

Till slut fick jag istället skriva mer specifika Scenario-steg och sätta in inloggning direkt i varje scenario där det behövdes, så att SpecFlow kunde köra testerna utan att trigga en error på gemensamma delar.

Förutom utmaningar vid Gherkin och SpecFlow testningen stötte jag även på utmaningingar vid enhetstestningen. Efter bankomat-uppgiften hade jag uppfattningen att jag skulle kunna nå min kod i ListLife från testprojektet. Det kunde jag inte. Vart jag än letade och vem jag än diskuterade med i klassen var svaret att använda en mockad databas, förslagsvis In-Memory-Database. Jag genomförde två enhetstester med In-Memory-Database. Jag använde mig då även av Moq för att skapa en mockad version av UserManager för att hantera användarrelaterade operationer utan att behöva en faktisk användarapplikation.

När jag gjort mina två enhetstester kunde jag inte släppa tanken på att jag kanske inte hade genomfört riktiga enhetstester, eftersom detta även kan räknas som integrationstestning, pga. Databaskopplingen (även om den är mockad). En renodlad enhetstestning bör enligt min uppfattning vara helt isolerad, det vill säga utan koppling till externa resurser som en databas. Jag frågade även Chat-GPT om detta, men fick samma svar om att använda en mockad databas.

Till slut hittade jag ett sätt att helt isolera mina tester genom att mocka nödvändig data och skriva tester som inte var beroende av externa resurser. Detta gjorde det möjligt att testa specifika metoder och funktioner utan att påverkas av databasens tillstånd eller andra externa faktorer.

I detta enhetstest simulerade jag och testade skapandet av en ny shoppinglista och hanteringen av dubbla produkter med samma namn, utan att behöva interagera med en faktisk databas. Istället ärvde jag från page CreateNewShoppinglist i ListLife. Jag behövde även refaktorera genom att ändra en metod till virtual i ListLife-projektet för att kunna ärva och överskrida den metoden i mitt enhetstest.

Under testningen av applikationen använde jag XPath för att navigera i och hitta dolda HTML-element. XPath användes när jag behövde fylla i ett e-post för att dela en shoppinglista. Eftersom formuläret var doldt och jag inte kunde finna input fältet med hjälp av selektorn, använde jag XPath för att hitta rätt formulär och identifiera det specifika e-postfältet baserat på innehållet i listnamnet.

Testningen avslöjade flera utmaningar i applikationen, där vissa områden behövde ytterligare arbete. Ett exempel på detta var problem med inloggning och registrering där

det saknades validering, vilket möjliggjorde skapandet av "låtsasanvändare" och hinder för användare att återställa lösenord. Vidare uppstod användarvänlighetsproblem där registreringsformuläret inte gav tillräcklig information om vad som behövde fyllas i förrän användaren klickade i en input-box. Dessa problem ledde till svårigheter i att säkerställa en korrekt och smidig användarupplevelse.

En annan utmaning var testningen av möjligheten att skapa en ny shoppinglista. Testen visade på svårigheter att hantera produktmängder för dubbla produkter med samma namn, vilket ledde till att de inte sammanfördes som förväntat. Detta behöver åtgärdas för att ge användaren en bättre upplevelse.

Framtida förbättringar

Framtida förbättringar kan inkludera att implementera striktare validering för produktkategorier och säkerställa att endast godkända produkter kan läggas till i specifika kategorier. Detta skulle minska risken för att användare väljer fel kategori för produkter som inte passar in.

Även validering för att inte kunna lägga till produkt med negativ mängd hade varit värdefullt att implementera, eftersom det både kan skapa förvirring och påverka logiken i applikationen.

Dessutom, när det gäller hantering av produktmängder, skulle en funktion som sammanför produkter med samma namn under en enda rad och summerar deras mängder förbättra användarupplevelsen och eliminera överflöd i listorna.

Hade jag haft mer tid hade det varit givande att refaktorera delar av koden baserat på resultatet jag fick under testningen. Både för att förbättra struktur och läsbarhet, men också för att skapa en ännu mer stabil och användarvänlig upplevelse i ListLife.

Slutsatser och avslutande diskussion

Testningen av applikationen ListLife har visat att applikationen har en stabil grundfunktionalitet, men det finns flera områden som kan förbättras både när det gäller användargränssnitt och kodstruktur.

En viktig aspekt av testningen var att jag jämförde två olika metoder för enhetstestning: enhetstestning med mockad databas och helt isolerad enhetstestning. Med enhetstestning som använder en mockad databas kan jag testa hur min kod fungerar i en mer realistisk miljö i relation till applikationen. Här interagerar metoderna med en "simulerad" databas som hanterar användare, utan att behöva en fullständig databas eller användarhantering i produktion. Detta ger möjlighet att testa applikationen i en produktionsliknande miljö där de vanliga beroendena till externa tjänster eller databaser är representerade, vilket gjorde att jag kunde validera applikationens funktionalitet i en mer realistisk miljö.

Men om man istället gör ett mer avskilt enhetstest, så testar man bara en specifik metod eller funktion, helt för sig själv. Det vill säga att man inte kopplar upp sig mot till exempel en databas. Istället används mockad data, så att man kan fokusera på att se om just den lilla biten av koden fungerar som den ska. Genom denna typ av testmetod kan man med högre säkerhet kontrollera om varje funktion fungerar korrekt på egen hand, men den ger inte lika realistisk förståelse gällande hur metoderna interagerar med andra delar av applikationen.

För att sammanfatta dessa två tillvägagångssätt, för enhetstestning, tycker jag att testning med en mockad databas passar bra när man vill efterlikna verkliga scenarier där samspel med andra delar, som databasen, är viktigt. Isolerad enhetstestning däremot, är mer användbar när man vill fokusera helt på om en specifik funktion gör det den ska, utan att påverkas av annat runt omkring. Båda metoderna har sina styrkor, men att börja med isolerade tester tror jag kan vara ett smart sätt att snabbt upptäcka fel i logiken när man bygger ett nytt projekt.

Vidare visade testerna på ett antal andra områden för förbättring. Ett specifikt problem var hanteringen av produktmängder. När två produkter med samma namn lades till i en lista, visades de på separata rader istället för att slås samman till en produkt med summerad mängd. En potentiell förbättring skulle vara att implementera funktionalitet som slår samman produkter med samma namn och summerar deras mängd.

Sammanfattningsvis har testningen visat att ListLife fungerar bra i sina grundläggande funktioner, men att det finns viktiga områden för förbättring. Genom testning av applikationen kunde man se brister, till exempel att vi hade behövt implementera striktare validering, förbättra produktkategorisering och optimera hanteringen av användarkonton. På så sätt hade applikationen kunnat göras både mer användarvänlig och stabil.

Något jag har lärt mig från kursen "Testning" är hur effektivt man kan identifiera brister och samtidigt höja kvaliteten i ett projekt genom att kombinera olika typer av tester under utvecklingsprocessens gång. Genom att integrera tester tidigt i utvecklingen kan man kontinuerligt verifiera funktionalitet och användarupplevelse, vilket både förbättrar applikationen och minimerar risken för mänskliga fel.