Inlämningsuppgift:
Tester av Webbaplikationer med Selenium och Cucumber.

Inom ramen för den här kursen har vi förstått hur **Cucumber** och **Selenium** verkar inom kontexten automatiserad testning för hemsidor. Cucumber och Selenium är välanvända verktyg för automatiserad testning med särskilt inriktning mot webbapplikationer. Vi simulerar en användare som registrerar ett konto på en webbplats. Cucumber är ett verktyg för beteendedriven utveckling (BDD) som gör det möjligt att skriva testfall i ett lättläst format med Gherkin-syntax.Test-scenarion skrivs i .feature-filer med Given-When-Then-steg. Det betyder att varje steg i .feature-filen motsvarar en Java-metod i steg definitionerna och det är där som själva automatiseringslogiken implementeras. Selenium interagerar med olika element på en webbsida (med hjälp olika ID eller xpath) så som en riktig användare skulle göra—genom att klicka på knappar, fylla i formulär, och validera svar. Selenium utför instruktionerna från Cucumber, öppnar webbläsaren, navigerar till sidor och genomför handlingar som att mata in text eller skicka formulär. Den metod som vi använt under kursens gång bygger på Behavior-Driven Development (BDD) som är en agil mjukvaruutvecklings genom att fokusera på den önskade beteenden hos applikationen. Som tilllägg till testningen mappar man användarhistorier som beskriver funktionaliteten ur slutanvändarens perspektiv. Dessa användarhistorier följer ofta en standardiserad struktur: Titel: En tydlig titel. Berättelse: En kort introduktion med följande struktur: Som en [roll] (användare, admin etc) vill jag att [funktion] gör denna syssla. Acceptanskriterier: En beskrivning av varje specifikt scenario med följande struktur: Given (initiala sammanhanghet) när en viss händelse inträffar och då ges ett förväntat resultat. Denna struktur, känd som Gherkin-språket, möjliggör en gemensam förståelse mellan tekniska och icke-tekniska teammedlemmar. (Beteendedriven utveckling i praktiken" från Göteborgs universitet). Assert har en binär struktur som bygger på logik och olika villkor. Om en assertion misslyckas break;ar loopen omedelbart och markeras som misslyckat och printas i en textfil. För ett registreringsformulär används assert för att bekräfta att ett bekräftelsemeddelande visas efter att formuläret skickats in typ såhär: Assert.assertTrue(condition) som godkänns om vilkoret är sant.

Kontinuerlig integration är en mjukvaruutvecklingspraxis där utvecklare ofta integrerar sina kodändringar i ett delat arkiv ofta flera gånger om dagen. Det är viktigt att skriva mycket kommentarer som människor förstår och att följa det ”meta” som varje företag har. I en proffesionell miljö måste varje integration byggas upp och testas sedan automatiskt, vilket gör att problem kan upptäckas tidigt och så att man vet exakt var problemet sitter. Det betyder att man måste dela in koden i små delar så att man kan förstå exakt var problemet uppstår. En fördel med detta är att mer frekventa integrationer innebär att man kan upptäcka konflikter, buggar eller integrationsfel. Regelbunden sammanslagning av kod hjälper till att upprätthålla en delad kodbas, vilket minskar problemet med "det fungerar på min maskin". Det är viktig inom ramen för snabb återkoppling så att man kan återgärda problem (gärna upp kvällstid direkt efter launch. Det är viktigt med förbättrad kodkvalitet men ett problem som man ofta inte förstår inom industrin är att du måst välja mellan att få arbetet utfört snabb eller ha en kod med högre kvalitet. Det handlar inte bara om hur bra en programmerare är, även om det säkert hjälper utom att det finns en hel drös med oväntade problem. Det finns problem med CI och det är att det tar tid att konfigurera servrar, skript, och processer. Frekventa byggen och tester kan förbruka avsevärda datorresurser och ta mycket tid. Mycket vänta på att builden ska bli klar. Det är ett jättstort problem om man jobbar själv. Det är viktigt att kvaliteten på de automatiserade testerna är bra. Problem kan uppstå med Dåliga tester eller ofullständiga tester då det kan inge en falsk trygghet. Testautomatisering innebär att man använder specialiserade verktyg för att automatiskt köra tester på mjukvaran. Detta ersätter eller minskar behovet av manuell testning och säkerställer att tester körs konsekvent och snabbt. Automatiserade tester integreras i **CI-pipelinen** så att varje kodändring testas automatiskt och säkerställer att nya ändringar inte bryter befintlig funktionalitet. Automatiserade tester ger kontinuerlig, realtidsfeedback om kodkvaliteten, vilket möjliggör snabba korrigeringar och förbättringar. Inom CD (kontinuerlig leverans/utveckling) är godkända automatiserade tester ofta en förutsättning för att koden ska kunna distribueras, vilket fungerar som en säkerhetsbarriär mot att släppa defekt programvara. Man kan köra automatiska tester i github med github actions vilket ofta är standard. GitHub-arkiv som innehåller projektets källkod och automatiserade tester. Skapa en katalog i ditt arkiv med namnet .github/workflows/ YAML-fil i denna katalog (t.ex. ci.yml) för att definiera arbetsflödet. Skapa triggers för när arbetsflödet ska köras. Vanliga triggers inkluderar push, pull_request eller att köras enligt ett schema.

2025-02-05