Lönesystem

Uppgift i Test och Kvalitetssäkring

Lars Strömberg - 2 February 2020

Introduktion

Bygga ett lönesystem. Under utvecklingens gång skall tester skapas och köra tester mot systemet. Vilka tester som användes och varför skall dokumenteras.

Krav

Testerna ska gå igenom, unit-tester + 4 andra tester.
Egna krav:

* Ett krav som inte finns med i uppgiften är att kunna logga ut vilket jag lägger till för funktionaliteten.
* Användarnamn och lösenord måste bestå av 4 bokstäver och 2 siffror samt ha en min-/max-längd på 6 tecken bortsett från de tvingande admin kontot (krav).
* Admin kan godkänna/neka alla ändringar med 2 egna metoder.

Utöver det så finns kraven nedan under unit-testerna.

Tester

Tester som fokuseras på är **unit-tester** samt **integrationstester**. **Unit-tester** tester för individuella metoder samt mindre och större integrationstester för att säkerställa funktionalitet. Utöver dessa kommer även **Stress-**, **Smoke-, System-, Sanity-, Monkey-** och **Regression-testing** implementeras.

✅ Unit-tester

<u>Kontrollerar individuella metoder för användare (krav):</u>
Inloggning:

* standard-användare.

Efter inloggning:

* Se saldo på sitt bankkonto

* Aktuell lön
* Roll I företaget
* Begära ändring av roll eller aktuell lön
* Ta bort sin egen användare genom att skriva användarnamn + lösenord

Kontrollerar individuella metoder för Admin (krav):
Inloggning:
* admin-användare.

Efter inloggning:
* Se saldo på sitt bankkonto
* Aktuell lön
* Roll I företaget
* Se aktuella användare och deras lösenord
* Se om användare begärt ändrad roll eller lön och i så fall ändra dessa värden
* Avancera system en månad så att lönen betalas ut till användare
* Skall kunna skapa användare lokalt
  * Användarnamn och lösenord ska bestå av bokstäver och siffror (krav)
* Ta bort användare genom att skriva användarnamn och tillhörande lösenord

Så unit tester kommer implementera i alla klasser som skapas bortsett från metoder som använder scannern. Däremot kommer all text som tas emot av scannern kontrolleras i separata metoder. Utskrift till konsolen kontrolleras via sanity test.

Integrationstester

Integrationstester har använts genom att köra multipla unit test i alla klasser.

Stress test

Genomförs genom att skapa och radera 10 000 användare. Detta påverkar ju mer prestanda på datorn än programmet då programmet inte kraschar av det.

Smoke testning

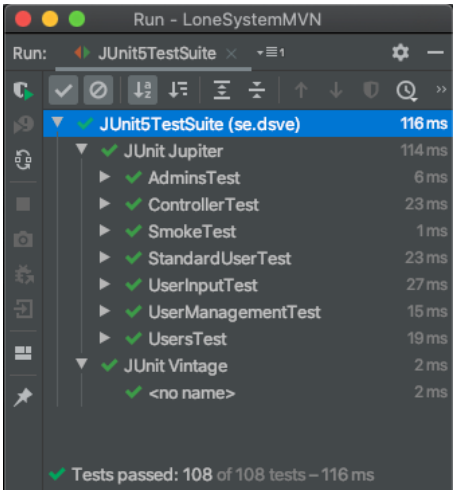
Här kontrollerar jag så att grundfunktionalitet så som inloggning och utloggning fungerar i ett första steg. När funktionaliteten för månadskörning är inlagd kommer smoke testningen uppdateras till att inkludera även det.
Rev 1:
Kontrollera inloggning så man kommer in till meny-system samt logga ut i ett och samma test.
Rev 2:
Samma som deltest 1 plus att verifiera en månadskörning.

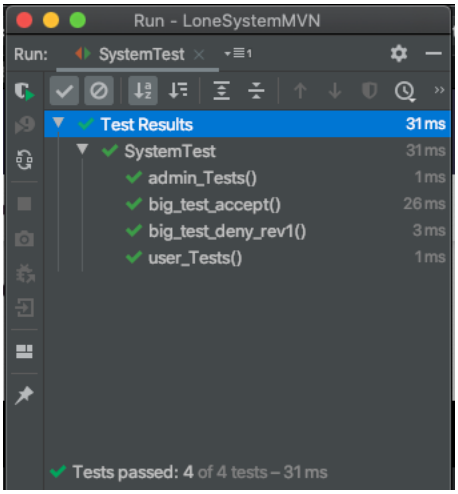
System test

Kör alla unit- och integrations-test i en körning i en JUnit5 Test Suite.

Dessutom har jag en klass som heter SystemTest som använder alla metoder som man kan använda som användare av systemet, borträknat metoder som använder “scanner” och “print/println” till konsolen.

Utöver detta så har programmet gåtts igenom manuellt med all funktionalitet för både admin och standardanvändare.





Sanity test

Här har jag kört programmet och visuellt kontrollerat att allt som skrivs ut till konsolen ser vettigt ut. Här kördes även de manuella Monkey testerna.

Hittade tex följande logikfel som låg utanför testklasserna vid sanity test:

- Om man försöker radera sitt konto med fel uppgifter så loggades man ut ändå.
- Även admin loggades ut när den raderade användare.
- Missat att kontrollera för dubbletter vid ny användare.
- Användare kunde radera varandra.

Monkey testing

Används lite enklare variant på för att testa UserInput klassen. All inläsning sker med till strängar, sedan behandlas strängen olika beroende på om det som läses in är en integer eller double. Rena textsträngar sker ingen behandling på. UserInput klassen ska fånga alla eventuella Exceptions så de inte fortplantar sig till resten av programmet.

Regression testing

Varje klass skrevs i ordning, så att säga nedifrån och upp. Jag började med User-klassen som har användarnamn och lösenord, därifrån ärver admin och lägger till en Boolean för "isAdmin". Sedan ärver "Standard-user" från Admin-klassen och adderar resterande variabler.

Då vi använder TDD så ser utvecklingsförfarandet ut enl:

1. Testerna för User skrivs, sedan skrivs metoderna för User. Enbart klassen testas.
2. Testerna för Admin skrivs, sedan skrivs metoderna för Admin. Enbart klassen testas.
3. Testerna för StandardUser skrivs, sedan skrivs metoderna för StandardUser. Enbart klassen testas.
4. Klassen som skapar objekten UserManagement skrivs enligt samma princip som ovan och när all tester går igenom så körs även testerna för punk 1-3 för att säkerställa att inget gått fel på vägen.
5. UserInput skrivs enlig TDD men testas bara separat då den ej påverkar övriga klasser direkt.
6. View-klassen innehåller bara de olika menyerna och testas ej.
7. Menu-Klassen innehåller respektive switch för menyval och testas ej.
8. Controller-klassen sköter all logic i programmet och skrivs enligt TDD, detta är den enskilt största testklassen på ca 40 test för att säkerställa funktionaliteten. Här körs regression testing med oftare intervaller eftersom denna klassen rör allt i hela programmet.

Utvärdering

Refaktorering

Har inte behövt göra så mycket refaktorering i efterhand då jag ofta gjort det direkt i anslutning till att koden fungerar då det ökar läsbarheten. Vissa metoder och viss funktionalitet refaktorerar jag inte då det skulle minska läsbarheten.

Kvalitetssäkring

Jag har för att hålla kvalitetssäkringen högt valt att dela upp enligt MVC-modellen. Det lyckades inte helt då viss funktionalitet gled över till Controller-klassen när det istället hade kunnat ligga i tex UserManagement.

Majoriteten av metoderna har skrivits för att vara enkelt testbara. Så har jag behövt använda tex scanner så har jag valt att lyfta ut all annan funktionalitet enbart för att kunna testa så mycket som möjligt. Jag la inte tid på att kontrollera scanner eller print/println då de är ju klasser som tillhandahålls av jdk och förväntas göra det de ska.

De flesta metoder är void eller boolean då jag vill kunna säkerställa att varje metod gör vad de ska utan att vara beroende av olika returvärden som man måste hålla reda på.