Testing av programvare

ADTS2310

Prosjektoppgave - Testing av Nettbank

PROSJEKTGRUPPE 1

Elise Hjartholm Tryti s374988

Fartun M Said s374964

Kevin Olsen s354651

Iuliana Tarnovschi s374978

TABLE OF CONTENTS

1. IDENTIFIKASJON

Laget: 15.01.2024

Sist oppdatert: 15.03.2024

Forfattere:

Elise Hjartholm Tryti s374988

Fartun M Said s374964

Kevin Olsen s354651

Iuliana Tarnovschi s374978

Hva testes her: Vi har gjennomført tester på en enkel Nettbank nettside ved å benytte enhetstesting, integrasjonstesting og systemtesting.

2. SAMMENDRAG

I denne rapporten dokumenterer vi i detaljer prosessen av testingen utført for systemet Nettbank. Vi har gjennomført tre hovedtyper av testing: enhetstesting, integrasjonstesting og systemtesting. Enhver programvareutviklingsprosess bør inkludere disse testtypene da det vil øke kvaliteten på systemet. Det er her vi oppdager feil og mangler og kan da rette opp før lansering. Rapporten presenterer en individuell detaljert testplan for alle testtypene med resultater og evaluering av disse.

3. TESTPLAN

3.1 ENHETSTESTING-TESTPLAN

3.1.1 IDENTIFIKASJON

Hva testes her: Tester hver Kontroller metode individuelt.

3.1.2 INNLEDNING

Enhetstesting er praksisen med å teste hver del av koden separat, inkludert individuelle kontrollermetoder. Dens betydning ligger i å sikre full testdekning, slik at eventuelle feil i applikasjonen identifiseres tidlig. Dette gir både kunden og utvikleren innsikt i hvor feilen oppstår, noe som er avgjørende for å rette opp og forbedre applikasjonens pålitelighet og kvalitet.

3.1.3 TESTOBJEKTER

Består av alle kontrollerne:

AdminKundeController: Kontroller med CRUD-funksjonalitet for kundeobjekter i systemet, som representerer kontoer til de ulike brukerne i systemet med tilhørende personinformasjon.

AdminKontoController: Kontroller som er knyttet bankkontoene til kundene i systemet som inneholder funksjonalitet kun tilgjengelig for administratoren.

BankController: Denne kontrolleren inneholder nødvendig funksjonalitet som brukere av nettbanks systemet må ha tilgjengelig. Dette innebærer å kunne utføre betalinger, å se kontooversikt ha oversikt over tidligere transaksjoner osv.

Sikkerhet: Kontroller som håndterer innlogging for både brukere og administratorer.

Testing: Alle metoder som gir en returverdi testes.

3.1.4 FREMGANGSMÅTE

Framgangsmåte for enhetstesting av kontrollerne i nettbankapplikasjonen inkluderer flere trinn for å sikre en grundig og pålitelig testprosess. Først og fremst er det viktig å organisere testene i separate Java-filer, hver dedikert til kontrolleren den tester. Dette gir en ryddig struktur som gjør det enklere å identifisere og håndtere feil. Deretter utføres enhetstestene uavhengig av en ekstern database ved å simulere metodekall på de fire aktuelle kontrollerne. Under selve testingen fokuseres det på å teste alle metoder som gir en returverdi, og det inkluderer også testing av Regex og grensetilfeller for å håndtere variert input. Etter gjennomføringen av testene overvåkes kode dekning for å sikre at testene dekker en tilstrekkelig mengde av koden. Denne framgangsmåten sikrer en omfattende og grundig enhetstestprosess som bidrar til å opprettholde kvaliteten og påliteligheten til nettbankapplikasjonen.

3.1.4.1 OVERORDNET TESTSTRATEGI

For å sikre kvaliteten og påliteligheten til nettbankapplikasjonen, er det avgjørende å implementere en grundig enhetsteststrategi. Dette innebærer å utføre enhetstester på kontrollerne i applikasjonen. Testene vil omfatte alle metoder som gir en returverdi og vil inkludere testing av alle mulige utfall, inkludert Regex og grensetilfeller. Målet er å oppnå en linjedekning på minst 95% for alle testklasser.

3.1.4.2 PROSEDYRE FOR PLANLEGGING OG GJENNOMFØRING

1. STARTET TIDLIG MED TESTING FOR Å IDENTIFISERE OG RETTE EVENTUELLE FEIL TIDLIG I UTVIKLINGSPROSESSEN.

2. ORGANISER TESTENE I SEPARATE JAVA-FILER SOM SAMSVARER MED KONTROLLERNE.
3. UTFØR ENHETSTESTENE UAVHENGIG AV EKSTERN DATABASE VED Å SIMULERE METODEKALL PÅ FIRE KONTROLLERE VED HJELP AV MOCKITO.
4. TEST ALLE METODER MED RETURVERDI OG INKLUDER OGSÅ TESTING AV REGEX OG GRENSETILFELLER FOR Å HÅNDTERE VARIERTE INPUT.
5. OVERVÅK KODE DEKNING UNDER TESTING FOR Å SIKRE AT ALLE METODER ER TESTET GRUNDIG.

VED Å FØLGE DENNE PROSEDYREN, VIL APPLIKASJONEN DRA NYTTE AV EN GRUNDIG OG PÅLITELIG ENHETSTESTSTRATEGI SOM BIDRAR TIL Å SIKRE DENS KVALITET .

3.1.4.3 KRITERIER FOR GRUNDIGHET I TESTEN – HVA MÅ TIL FOR AT TESTEN SKAL ANSES SOM GRUNDIG

På grunn av den minimale størrelsen på applikasjonen, velger vi å teste alle metodene i kontrollere som er tilgjengelige. I enhetstestene tester vi samtlige utfall for alle metodene i kontrollerne. Vi ønsker at hver test skal ha en line coverage på minst 95 prosent for å godkjenne enhetstesten.

3.2 INTEGRASJONSTESTING-TESTPLAN

3.1.1 IDENTIFIKASJON

Hva testes her: Samspillet mellom Controller, Repository og database.

3.1.2 INNLEDNING

Integrasjonstesting er viktig for å teste metodene enkeltvis opp mot databasen der kunde og kontoinformasjon ligger. Dette ble testet fra både et kunde standpunkt og fra et administrator standpunkt. Integrasjonstestene benytter enhetstestene som sjekker blant annet innlogging, og regex-feil. For å oppnå automatisering av testene ble SoapUI benyttet.

3.1.3 TESTOBJEKTER

AdminKontoController

En kontroller som inneholder funksjoner for endring, sletting, registrering og henting av alle konti som ligger i databasen. Alle konti er knyttet til kundens personnummer og inneholder blant annet informasjon om kontotype, saldo og transaksjoner.

Alle metodene ble testet

AdminKundeController

En kontroller som inneholder funksjoner for endring, sletting, registrering og henting av alle kundeobjektene som ligger i databasen. Hvert kundeobjekt inneholder blant annet informasjon om navn, adresse, postnummer og passordet som brukes til innlogging.

Alle metodene ble testet

BankController

En kontroller som inneholder bankfunksjoner for kunden. Kunden kan blant annet hente saldo, utføre betalinger og se konti. Kunden bruker personnummer og passord for å logge inn her.

Alle metodene ble testet

Sikkerhet

Sikkerhet inneholder metoder for innlogging for både admin og kunde. Kontrolleren inneholder også metoder for utlogging.

Alle metodene ble testet

AdminRepository

Et repository med metoder knyttet til både AdminKontoController og AdminKundeController. Repositoryet knytter kontrollerne til databasen.

Alle metoder knyttet til Konto og Kunde kontrollerne ble testet

BankRepository

Et repository med metoder knyttet til BankController. Repositoryet knytter kontrolleren til databasen.

3.1.4 FREMGANGSMÅTE

Formålet med integrasjonstestene er å sjekke samhandling mellom Controller/Repository mot databasen. Dette oppnås ved å lage ulike test cases som for eksempel å hente alle kunder som en admin eller å gjennomføre en betaling som kunde. Her utføres http-kall. At output er korrekt, blir verifisert med assertions på output. For å automatisere testene har vi benyttet SoapUI som et verktøy. SoapUI tar inn ulike test cases og kan teste samhandlingen mellom kontroller og database og koble dem sammen. Resultater fra test casene og deres output skal dokumenteres i et Excel-ark.

3.1.4.1 OVERORDNET TESTSTRATEGI

Som en helhet skal samspillet mellom kontroller og database testes. Å benytte en integrasjonstest er nyttig for å teste systemet fra back-end siden uten behov for et brukergrensesnitt. Dette kan tidlig gjøre det enkelt å se hva som må utbedres. Nettbanks-systemet må kjøres for å kunne utføre http-kallene. Det er også viktig at for hver suite ble først databasen initialisert og deretter admin/kunde ble logget inn. Testene gir en ide om hvilke komponenter som fungerer bra sammen og hvilke som ikke oppfyller forventningene. Ved å skrive resultatene inn i et Excel-ark får vi en god oversikt over hvor feilene ligger.

3.1.4.2 PROSEDYRE FOR PLANLEGGING OG GJENNOMFØRING

Før selve integrasjonstestene blir gjennomført blir det laget flere test-cases i Excel-arket som skal brukes til å utføre testene med. Test-casene skal være brukerhistorier som f.eks. å hente alle kunder, endre på en kunde eller utføre betalinger. Alle casene som skal være testet skal være basert på metodene i kontrolleren som henting, endring, registrering og sletting. I Excel-arket ble det skrevet inn en brukerhistorie og hvis input var nødvendig så ble input skrevet inn under “Test Data”.

Når brukerhistoriene var ferdigstilte brukte vi SoapUI for å opprette automatiserte tester. Først ble et nytt prosjekt og en ny Test Suite opprettet. Det ble lagd tre test-suites, en for hver kontroller. Dette vil si AdminKontoController, AdminKundeController, og BankController. Hver Test Suite har ett Test Case med tester for initialisering av database, innlogging, henting, registrering, sletting og endring. På hvert Test Step ble det brukt assertions for å sjekke for korrekt databehandling. Hvis assertions feilet, var ikke testen korrekt. For å forsikre oss om at output var som forventet så ble det i tillegg til en assertion der det sto “OK” i HTML også kjørt metoder som henter dataene etter hver endring, sletting eller registrering.

Når Test Stepsene var fullførte så ble resultatet skrevet inn i Excel-arket for god oversiktlighet.

3.1.4.3 KRITERIER FOR GRUNDIGHET I TESTEN

For å forsikre en grundig testing av kontrollerne så må alle brukerhistorier testes, både som admin og kunde. Alle metodene må testes med assertions i tillegg, for å forsikre at output faktisk er korrekt og at dataen er behandlet riktig. I tillegg bør det testes for tilfeller der kunder/admin er logget inn, ikke logget inn eller feil i f.eks. regex og hvordan det blir behandlet.

3.3 SYSTEMTESTING-TESTPLAN

3.3.1 IDENTIFIKASJON

Hva testes her: Vi tester at de ulike komponentene i systemet samarbeider sømløst som en helhet, og har de ønskede funksjonene tilgjengelige slik det er forventet fra brukernes perspektiv.

3.3.2 INNLEDNING

Får å kunne utføre systemtestingen må enhetstestene og integrasjonstestene være utført. Systemtesting er en type blackbox testing som ikke har noe med selve koden å gjøre, men med systemet som helhet. Testene definert gjenspeiler funksjonene som er forventet av systemet. Det vil innebære testing av de funksjonelle og ikke-funksjonelle kravene til systemet. Vi fokuserer oss på de funksjonelle kravene og definerer dem som brukerhistorier. Videre utføres de i et bestemt testmiljø som er konfigurert for å automatisere testprosessen og etterligne det virkelige driftsmiljøet.

Systemtesting er en viktig del av programvareutviklingsprosessen, og den utføres for å sikre at systemet oppfyller brukernes krav og fungerer som forventet i ulike scenarioer.

3.3.3 TESTOBJEKTER

Systemtesting omfatter hele systemet og undersøker samspillet mellom dets ulike komponenter, snarere enn å vurdere komponentene individuelt. Under systemtesting evalueres brukergrensesnittet på klientens side, som er beskrevet gjennom HTML, JavaScript og CSS-koden. Feilhåndtering er også et sentralt aspekt som blir testet. Videre er databasen et annet viktig testobjekt, hvor lagring og henting av data er essensielle elementer. Disse operasjonene er beskrevet i bl.a. repositoryene. Repositoryene, sammen med kontrollermetodene, styrer hvordan systemet skal reagere fra serverens side.

3.3.4 FREMGANGSMÅTE

Målet med systemtesting er å verifisere at brukernes funksjonelle krav er oppfylt, og at systemet utfører de nødvendige prosessene og interaksjonene for å imøtekomme disse kravene. Dette oppnås ved å blant annet skrive brukerhistorier som beskriver typiske scenarioer med spesifikke brukerroller, funksjoner og ønskede verdier. I vårt nettbanks-system inkluderer brukerrollene administrator og kunder av systemet.

Vi bruker Azure DevOps-plattformen, for å organisere brukerhistoriene som er lagret som separate testcases. I testcasene dokumenterer vi stegene brukeren må utføre i brukergrensesnittet og det forventede resultatet for hvert steg. Dette gir tydelighet rundt hva forventede resultatet skal være etter hver handling fra brukerens perspektiv.

Etter å ha definert brukerhistoriene, tar vi opptak av programmet ved hjelp av Katalon. Vi gjennomfører testene ved å gjenskape brukernes trinn som ble definert i testcasene basert på brukerhistoriene. Katalon brukes som automatiseringsverktøy for å sikre konsistent utførelse av testene. Resultatene vi får fra testene utført i Katalon registreres og dokumenteres i Azure DevOps, sammen med årsakene til mislykkede tester.

3.3.4.1 OVERORDNET TESTSTRATEGI

Systemtestens strategi starter med å definere brukshistorier for å klargjøre hvilke funksjoner systemet skal tilby. Deretter fokuserer vi på at kravene satt av brukerne er oppfylt gjennom testingen. Det overordnede målet er å oppdage og løse feil i systemet og vurdere funksjoner som ble godkjent før systemet lanseres. Det er i systemtesting at vi får sett om systemet er klar til bruk.

3.3.4.2 PROSEDYRE FOR PLANLEGGING OG GJENNOMFØRING

1. Definere og registrere brukshistoriene for å forstå hvilken funksjonalitet skal testes. Dette gjør vi i Azure DevOps.

2. Ta opptak av handlingene gjort av brukeren ut ifra brukshistorien for å kunne teste at systemet gjør det som er forventet. Dette gjør vi ved å lage ny testcases og trykke record mens vi utfører handlingene manuelt første gangen. Deretter blir testen automatisert.

3. Initialisere databasen for å tilbakestille databasen slik at testene kan kjøres uavhengig av hverandre. Dette gjør vi ved å legge til en ny open kommando helt øverst med url-en og initDB som target. I praksis betyr det at vi kan teste en testcase av gangen i stedet for hele testsuiten.

4. Skrive assert kommandoer i Command feltet, ønsket mål i Target-feltet og om nødvendig forventet verdi i Value-feltet for å sette en sjekk på at utført handlingen gir resultatet som ble definert i koden.

5. Om assert kommandoene ikke gir feil vil testen være godkjent, ellers har testen mislykket og vi registrer resultatet i DevOps sammen med feilen under “Bugs”.

6. Systemtestingen er gjennomført dersom alle brukshistoriene har blitt testet uavhengig av resultat.

3.3.4.3 KRITERIER FOR GRUNDIGHET I TESTEN

For at testingen skal kunne anses som grundig, må alle definerte brukshistorier testes. I et system vil det være ulike roller med ulike målsettinger, og derfor må brukerhistorier utarbeides for å dekke behovene til disse rollene. I vårt nettbanksystem fokuserer vi på to hovedroller og det er administrator og vanlig kunde hos nettbanken. Det er ikke mulig å teste absolutt alt, men det er viktig å prioritere og teste det viktigste for å sikre at systemet oppfyller målene til de involverte personene/systemene.

I systemtestingen er det viktig å teste både de funksjonelle og ikke-funksjonelle kravene. Dette sikrer at testingen dekker både brukernes forventede funksjonalitet ved å simulere en brukers handling og systemets ytelse generelt, som vil blant annet inkludere responstid og evnen til å håndtere høy trafikk (flere samtidige brukere). Alle disse vil kunne avgjøre systemets effektivitet og brukeropplevelse. I tillegg er det viktig at data i systemet blir håndtert på en sikker måte- systemtesting skal sjekke at den gjør det.

4. TESTMILJØ

Enhetstest:

Brukte verktøy: Intelji

For å utføre testene brukte vi Intelji som vi har kjennskaper fra tidligere. I tillegg brukte vi Mockito til å opprette mock-objekter.

Integrasjonstest:

Brukte verktøy: SoapUI, Excel

Verktøyet SoapUI ble brukt for å automatisere integrasjonstestene slik at testene var konsekvente. Excel ble brukt til dokumentasjon.

Systemtest:

Brukte verktøy: Chrome sin utvidelse- Katalon, Azure DevOps

Katalon ble brukt som testautomatiseringsverktøy for å ta opptak av stegene utført av en bruker ut ifra brukerhistoriene for å teste funksjonen.

Azure DevOps verktøyet ble brukt til dokumentering av resultater og andre detaljer slik som steg og forventet resultat for hvert steg og årsak til feil ved mislykkede tester.

5. TESTRESULTATER

Enhetstesting:

Enhetstest for metodene i **BankController:**

```
package oslomet.testing;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.MockitoJUnitRunner;
import oslomet.testing.API.BankController;
import oslomet.testing.DAL.BankRepository;
import oslomet.testing.Models.Konto;
import oslomet.testing.Models.Kunde;
import oslomet.testing.Models.Transaksjon;
import oslomet.testing.Sikkerhet.Sikkerhet;

import java.util.ArrayList;
import java.util.List;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.anyString;
import static org.mockito.Mockito.when;

@RunWith(MockitoJUnitRunner.class)
public class EnhetstestBankController {

    @InjectMocks
    // denne skal testes
    private BankController bankController;

    @Mock
    // denne skal Mock'es
    private BankRepository repository;

    @Mock
    // denne skal Mock'es
    private Sikkerhet sjekk;

    @Test
    public void hentKundeInfo_loggetInn() {

        // arrange
        Kunde enKunde = new Kunde("01010110523", "01010110523",
                "Lene", "Jensen", "Askerveien 22", "3270",
                "Asker", "22224444");
```

```
        when(sjekk.loggetInn()).thenReturn("01010110523");

        when(repository.hentKundeInfo(anyString())).thenReturn(enKunde);

        // act
        Kunde resultat = bankController.hentKundeInfo();

        // assert
        assertEquals(enKunde, resultat);

    }

    @Test
    public void hentKundeInfo_IkkeloggetInn() {

        // arrange
        when(sjekk.loggetInn()).thenReturn(null);

        //act
        Kunde resultat = bankController.hentKundeInfo();

        // assert
        assertNull(resultat);
    }

    @Test
    public void hentKonti_LoggetInn()  {
        // arrange
        List<Konto> konti = new ArrayList<>();
        Konto konto1 = new Konto("105010123456", "01010110523",
                720, "Lønnskonto", "NOK", null);
        Konto konto2 = new Konto("105010123456", "12345678901",
                1000, "Lønnskonto", "NOK", null);
        konti.add(konto1);
        konti.add(konto2);

        when(sjekk.loggetInn()).thenReturn("01010110523");

        when(repository.hentKonti(anyString())).thenReturn(konti);

        // act
        List<Konto> resultat = bankController.hentKonti();

        // assert
        assertEquals(konti, resultat);
    }
```

```
    @Test
    public void hentKonti_IkkeLoggetInn()  {
        // arrange

        when(sjekk.loggetInn()).thenReturn(null);

        // act
        List<Konto> resultat = bankController.hentKonti();

        // assert
        assertNull(resultat);
    }

    @Test
    public void utforBetaling_OK() {
        // arrange
        List<Transaksjon> betalinger = new ArrayList<>();
        Transaksjon enTransaksjon = new Transaksjon(2, "20102012345", 400.4,
"2015-03-20", "Skagen", "1", "105010123456");

        betalinger.add(enTransaksjon);

        Konto konto = new Konto("05068924604", "41925811793",
                13495.41, "Brukskonto", "NOK", betalinger);

        konto.setTransaksjoner(betalinger);

        when(sjekk.loggetInn()).thenReturn(konto.getPersonnummer());

        when(repository.utforBetaling(enTransaksjon.getTxID())).thenReturn("OK");
        when(repository.hentBetalinger(anyString())).thenReturn(betalinger);

        // act

        List<Transaksjon> resultat =
bankController.utforBetaling(enTransaksjon.getTxID());

        // assert
        assertEquals(betalinger, resultat);
    }
    @Test
    public void utforBetaling_ikkeOK() {

        // arrange
        List<Transaksjon> betaling = new ArrayList<>();
        Transaksjon enBetaling = new Transaksjon(2, "20102012345", 400.4, "2015-
03-20", "Skagen",
                "1", "105010123456");

        Konto konto = new Konto("05068924604", "41925811793",
                13495.41, "Brukskonto", "NOK", betaling);
```

```
        when(sjekk.loggetInn()).thenReturn(null);

        // act
        List<Transaksjon> resultat =
bankController.utforBetaling(enBetaling.getTxID());

        // assert
        assertNull((resultat));
    }
    @Test
    public void  endre_OK() {
        // arrange
        Kunde kunde = new Kunde("01010110523", "Lene", "Jensen",
                "Askerveien 22", "3270", "Oslo", "22224444", "HeiHei");

        when(sjekk.loggetInn()).thenReturn(kunde.getPersonnummer());


when(repository.endreKundeInfo(any(Kunde.class))).thenReturn(kunde.getPersonnummer
());

        // act
        String resultat = bankController.endre(kunde);

        // assert
        assertEquals(kunde.getPersonnummer(), resultat);
    }
    @Test
    public void  endre_ikkeOK() {
        // arrange
        Kunde kunde = new Kunde("01010110523", "Lene", "Jensen",
                "Askerveien 22", "3270", "Oslo", "22224444", "HeiHei");

        when(sjekk.loggetInn()).thenReturn(null);

        // act
        String resultat = bankController.endre(kunde);

        // assert
        assertNull(resultat);
    }
    @Test
    public void hentBetalinger_OK() {

        // arrange
        List<Transaksjon> transaksjoner = new ArrayList<>();
        Transaksjon transaksjon = new Transaksjon(2, "123456789101", 23.5,
                "2012-03-11", "send", "1", "23456789101");
        transaksjoner.add(transaksjon);
```

```
        Konto konto1 = new Konto("115111133557", "02020211533",
                800, "Lønnskonto", "NOK", transaksjoner);

        when(sjekk.loggetInn()).thenReturn("115111133557");

when(repository.hentBetalinger(konto1.getPersonnummer())).thenReturn(transaksjoner
);

        // act
        List<Transaksjon> resultat = bankController.hentBetalinger();

        // assert
        assertEquals(transaksjoner, resultat);
    }
    @Test
    public void hentBetalinger_ikkeOK() {

        // arrange
        when(sjekk.loggetInn()).thenReturn(null);

        // act
        List<Transaksjon> resultat = bankController.hentBetalinger();

        // assert
        assertNull(resultat);
    }
    @Test
    public void registrerbetaling_OK() {
        // arrange
        List<Transaksjon> konto1transaksjoner = new ArrayList<>();
        Transaksjon enTransaksjon = new Transaksjon(2, "20102012345", 400.4,
"2015-03-20", "Skagen", "1",
                "105010123456");

        konto1transaksjoner.add(enTransaksjon);

        Konto konto1 = new Konto("05068924604", "41925811793",
                13495.41, "Brukskonto", "NOK", konto1transaksjoner);

        when(sjekk.loggetInn()).thenReturn(konto1.getPersonnummer());
        when(repository.registrerBetaling(enTransaksjon)).thenReturn("OK");

        // act
        String resultat = bankController.registrerBetaling(enTransaksjon);

        // assert
        assertEquals("OK", resultat);
    }
```

```
    @Test
    public void registrerbetaling_ikkeOK() {

        // arrange
        Transaksjon enTransaksjon = new Transaksjon(2, "20102012345", 400.4,
"2015-03-20", "Skagen", "1",
                "105010123456");

        when(sjekk.loggetInn()).thenReturn(null);

        // act
        String resultat = bankController.registrerBetaling(enTransaksjon);

        // assert
        assertNull(resultat);
    }
    @Test
    public void hentSaldi_loggetInn() {

        // arrange
        List<Konto> saldi = new ArrayList<>();
        Konto konto3 = new Konto("115111133557", "02020211533",
                800, "Lønnskonto", "NOK", null);
        saldi.add(konto3);

        when(sjekk.loggetInn()).thenReturn("115111133557");

        when(repository.hentSaldi(anyString())).thenReturn(saldi);

        // act
        List<Konto> resultat = bankController.hentSaldi();

        // assert
        assertEquals(saldi, resultat);
    }
    @Test
    public void hentSaldi_IkkeLoggetInn() {
        // arrange

        when(sjekk.loggetInn()).thenReturn(null);

        // act
        List<Konto> resultat = bankController.hentSaldi();

        // assert
        assertNull(resultat);
    }
```

```
    @Test
    public void hentTransaksjoner_OK() {

        // arrange
        List<Transaksjon> transaksjons = new ArrayList<>();

        Transaksjon tr1 = new Transaksjon(2, "123456789101", 23.5,
                "2012-03-11", "send", "1", "23456789101");
        Transaksjon tr2 = new Transaksjon(3, "123456789101", 23.5,
                "2021-04-11", "send", "1", "23456789101");

        transaksjons.add(tr1);
        transaksjons.add(tr2);


        List <Konto> konti = new ArrayList<>();
        Konto konto1 = new Konto("115111133557", "02020211533",
                800, "Lønnskonto", "NOK", transaksjons);

        konti.add(konto1);

        konto1.setTransaksjoner(transaksjons);

        when(sjekk.loggetInn()).thenReturn("115111133557");

        when(repository.hentTransaksjoner(anyString(), anyString(),
anyString())).thenReturn(konto1);

        // act
        Konto resultat = bankController.hentTransaksjoner("115111133557","2011-01-
01", "2013-01-01");

        // assert
        assertEquals(konto1, resultat);
    }
    @Test
    public void hentTransaksjoner_ikkeOK() {

        Konto konto1 = new Konto("115111133557", "02020211533",
                800, "Lønnskonto", "NOK", null);

        when(sjekk.loggetInn()).thenReturn(null);

        // act
        Konto resultat = bankController.hentTransaksjoner(null,null, null);

        assertNull(resultat);
    }
```

```
    @Test
    public void test_initDB(){

        when(repository.initDB(any())).thenReturn("OK");

        String resultat = bankController.initDB();

        assertEquals("OK", resultat);
    }
}
```

Enhetstesting for metodene i **AdminKundeController:**

```
package oslomet.testing;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.MockitoJUnitRunner;
import oslomet.testing.API.AdminKundeController;
import oslomet.testing.DAL.AdminRepository;
import oslomet.testing.Models.Kunde;
import oslomet.testing.Sikkerhet.Sikkerhet;

import java.util.ArrayList;
import java.util.List;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;
import static org.mockito.ArgumentMatchers.anyString;
import static org.mockito.Mockito.when;
@RunWith(MockitoJUnitRunner.class)
public class EnhetstestAdminKundeController {
    @InjectMocks
    // denne skal testes
    private AdminKundeController AdminKundeController;

    @Mock
    // denne skal Mock'es
    private AdminRepository repository;

    @Mock
    // denne skal Mock'es
    private Sikkerhet sjekk;

    @Test
    public void test_HentAlleOk(){
        // arrange
        List<Kunde> kundeList = new ArrayList<>();
```

```
        Kunde kunde1= new Kunde("01010110523",
"01010110523","Lene","Jensen","Askerveien 22","3270","22224444","HeiHei");
        Kunde kunde2=new Kunde("01010110523",
"12345678901","Per","Hansen","Osloveien 82","1234","12345678","HeiHei");

        kundeList.add(kunde1);
        kundeList.add(kunde2);

        when(repository.hentAlleKunder()).thenReturn(kundeList);
        when(sjekk.loggetInn()).thenReturn("01010110523");

        // act
        List<Kunde> resultat = AdminKundeController.hentAlle();
        //assert
        assertEquals(kundeList, resultat);
    }
    @Test
    public void test_HentAlleIkkeOK(){
        // arrange
        when(sjekk.loggetInn()).thenReturn(null);
        //act
        List<Kunde> resultat = AdminKundeController.hentAlle();
        //assert
        assertNull(resultat);
    }
    @Test
    public void test_LagreOK(){
        // arrange
        Kunde kunde1= new Kunde("01010110523",
"01010110523","Lene","Jensen","Askerveien 22","3270","22224444","HeiHei");
        when(sjekk.loggetInn()).thenReturn("01010110523");
        when(repository.registrerKunde(kunde1)).thenReturn("OK");
        //act
        String resultat = AdminKundeController.lagreKunde(kunde1);
        //assert
        assertEquals("OK", resultat);
    }
    @Test
    public void test_LagreIkkeOK(){
        // arrange
        Kunde kunde1= new Kunde("01010110523",
"01010110523","Lene","Jensen","Askerveien 22","3270","22224444","HeiHei");
        when(sjekk.loggetInn()).thenReturn(null);
        //act
        String resultat = AdminKundeController.lagreKunde(kunde1);
        //assert
        assertEquals("Ikke logget inn",resultat);
    }
```

```
    @Test
    public void test_EndreOK(){
        // arrange
        Kunde enkunde= new Kunde("01010110523",
"01010110523","Lene","Jensen","Askerveien 22","3270","22224444","HeiHei");
        when(sjekk.loggetInn()).thenReturn("01010110523");
        when(repository.endreKundeInfo(enkunde)).thenReturn("OK");
        //act
        String resultat = AdminKundeController.endre(enkunde);
        //assert
        assertEquals("OK", resultat);
    }
    @Test
    public void test_EndreIkkeOK(){
        // arrange
        Kunde enkunde= new Kunde("01010110523",
"01010110523","Lene","Jensen","Askerveien 22","3270","22224444","HeiHei");
        when(sjekk.loggetInn()).thenReturn(null);
        //act
        String resultat = AdminKundeController.endre(enkunde);
        //assert
        assertEquals("Ikke logget inn",resultat);
    }
    @Test
    public void test_SletteOK(){
        // arrange
        when(sjekk.loggetInn()).thenReturn("01010110523");
        when(repository.slettKunde(anyString())).thenReturn("OK");
        //act
        String resultat = AdminKundeController.slett("01010110523");
        //assert
        assertEquals("OK", resultat);
    }
    @Test
    public void test_SletteIkkeOK(){
        // arrange
        when(sjekk.loggetInn()).thenReturn(null);
        //act
        String resultat = AdminKundeController.slett("01010110523");
        //assert
        assertEquals("Ikke logget inn",resultat);
    }

}
```

Enhetstesting for metodene i **AdminKontoController:**

```
package oslomet.testing;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.MockitoJUnitRunner;
import oslomet.testing.API.AdminKontoController;
import oslomet.testing.DAL.AdminRepository;
import oslomet.testing.Models.Konto;
import oslomet.testing.Sikkerhet.Sikkerhet;
import java.util.ArrayList;
import java.util.List;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.when;
@RunWith(MockitoJUnitRunner.class)
public class EnhetstestAdminKontoController {
    @InjectMocks
    // denne skal testes
    private AdminKontoController AdminKontoController;

    @Mock
    // denne skal Mock'es
    private AdminRepository repository;

    @Mock
    // denne skal Mock'es
    private Sikkerhet sjekk;
        @Test
        public void test_HentAlleKontiOk() {
            // arrange

            List<Konto> kontoList = new ArrayList<>();

            Konto konto1 = new Konto("01010110523", "105010123456",
                    720, "Lønnskonto", "NOK", null);

            kontoList.add(konto1);

            when(sjekk.loggetInn()).thenReturn("01010110523");
            when(repository.hentAlleKonti()).thenReturn(kontoList);
            //act
            List<Konto> resultat = AdminKontoController.hentAlleKonti();
            //assert
            assertEquals(kontoList, resultat);
        }
```

```
@Test
        public void test_HentAlleIkkeOK() {
            // arrange
            when(sjekk.loggetInn()).thenReturn(null);
            //act
            List<Konto> resultat = AdminKontoController.hentAlleKonti();
            //assert
            assertNull(resultat);
        }

        @Test
        public void test_RegOK() {
            //arrange
            Konto konto1 = new Konto("01010110523", "105010123456",
                    720, "Lønnskonto", "NOK", null);

            when(sjekk.loggetInn()).thenReturn("105010123456");

            when(repository.registrerKonto(konto1)).thenReturn("OK");

            // act
            String resultat = AdminKontoController.registrerKonto(konto1);

            // assert
            assertEquals("OK", resultat);

        }

        @Test
        public void test_RegIkkeOK() {
            //arrange
            Konto konto1 = new Konto("01010110523", "105010123456",
                    720, "Lønnskonto", "NOK", null);

            when(sjekk.loggetInn()).thenReturn(null);

            // act
            String resultat = AdminKontoController.registrerKonto(konto1);

            // assert
            assertEquals("Ikke innlogget", resultat);

        }
        @Test
        public void test_endreOK() {
            //arrange
            Konto konto1 = new Konto("01010110523", "105010123456",
                    720, "Lønnskonto", "NOK", null);

            when(sjekk.loggetInn()).thenReturn("01010110523");
            when(repository.endreKonto(any(Konto.class))).thenReturn("OK");
```

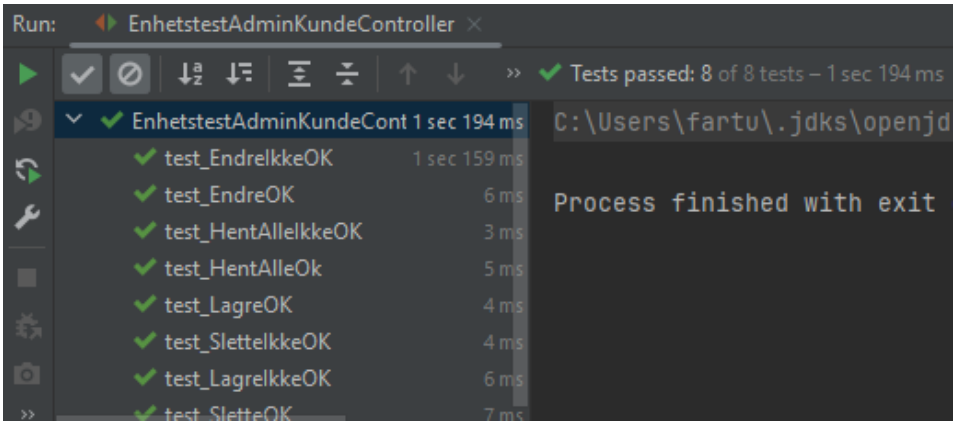
```
            //act
            String resultat = AdminKontoController.endreKonto(konto1);
            //assert
            assertEquals("OK", resultat);
        }
        @Test
        public void test_endreIkkeOK() {
            //arrange
            Konto konto1 = new Konto("01010110523", "105010123456",
                    720, "Lønnskonto", "NOK", null);

            when(sjekk.loggetInn()).thenReturn(null);
            //act
            String resultat = AdminKontoController.endreKonto(konto1);

            //assert
            assertEquals("Ikke innlogget", resultat);
        }
        @Test
        public void test_SletteOK(){
            // arrange
            Konto konto1 = new Konto("01010110523", "105010123456",
                    720, "Lønnskonto", "NOK", null);

            when(sjekk.loggetInn()).thenReturn(konto1.getPersonnummer());
            when(repository.slettKonto(konto1.getKontonummer())).thenReturn("OK");
            //act
            String resultat =
AdminKontoController.slettKonto(konto1.getKontonummer());
            //assert
            assertEquals("OK", resultat);
        }
        @Test
        public void test_SletteIkkeOK(){
            // arrange
            Konto konto1 = new Konto("01010110523", "105010123456",
                    720, "Lønnskonto", "NOK", null);
            when(sjekk.loggetInn()).thenReturn(null);
            //act
            String resultat =
AdminKontoController.slettKonto(konto1.getKontonummer());
            //assert
            assertEquals("Ikke innlogget",resultat);
    }
    }
```

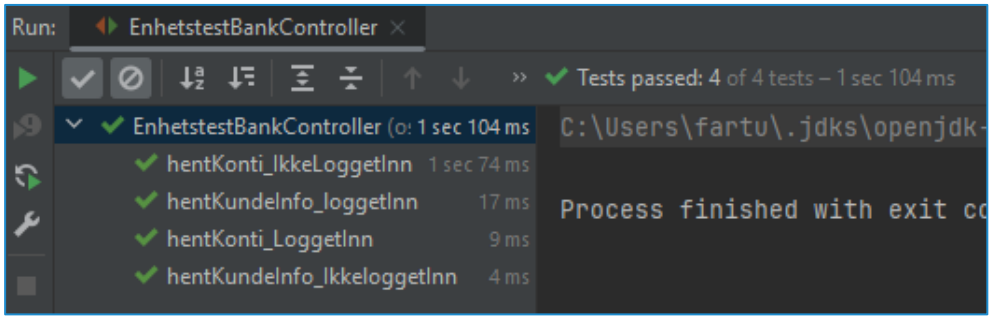
Enhetstesting for metodene i klassen **EnhetsSikkerhet:**

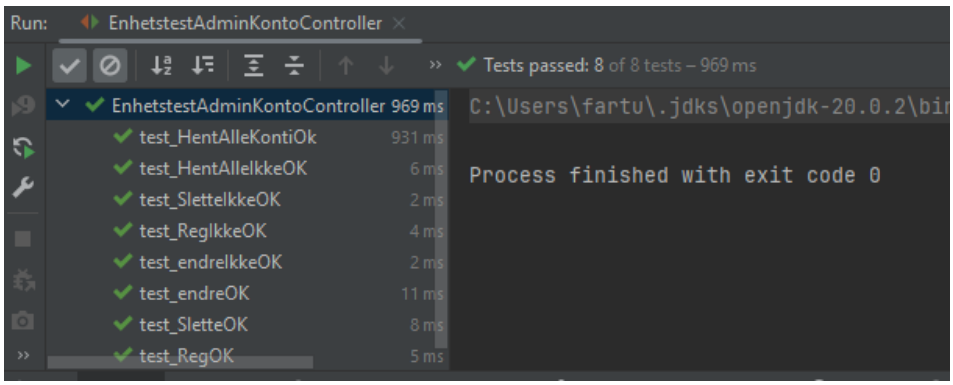
```
package oslomet.testing;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.invocation.InvocationOnMock;
import org.mockito.junit.MockitoJUnitRunner;
import org.mockito.stubbing.Answer;
import org.springframework.mock.web.MockHttpSession;
import oslomet.testing.DAL.BankRepository;
import oslomet.testing.Sikkerhet.Sikkerhet;
import java.util.HashMap;
import java.util.Map;
import static org.junit.jupiter.api.Assertions.assertNull;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.ArgumentMatchers.anyString;
import static org.mockito.Mockito.*;

@RunWith(MockitoJUnitRunner.class)
public class EnhetstestSikkerhet {
    @InjectMocks
    // denne skal testes
    private Sikkerhet sikkerhetsController;

    @Mock
    // denne skal Mock'es
    private BankRepository repository;

    @Mock
    // denne skal Mock'es
    private MockHttpSession session;

    @Before
    public void initSession(){
        Map<String,Object> attributes = new HashMap<String,Object>();

        doAnswer(new Answer<Object>(){
            @Override
            public Object answer(InvocationOnMock invocation) throws Throwable {
                String key = (String) invocation.getArguments()[0];
                return attributes.get(key);
            }
        }).when(session).getAttribute(anyString());

        doAnswer(new Answer<Object>(){
            @Override
            public Object answer(InvocationOnMock invocation) throws Throwable {
                String key = (String) invocation.getArguments()[0];
```

```
                Object value = invocation.getArguments()[1];
                attributes.put(key, value);
                return null;
            }
        }).when(session).setAttribute(anyString(), any());

    }
    @Test
    public void test_SjekkLoggetInnOK() {
        // arrange
        when(repository.sjekkLoggInn(anyString(),anyString())).thenReturn("OK");

        // act
        String resultat = sikkerhetsController.sjekkLoggInn("12345678901",
                "HeiHei");

        // assert
        assertEquals("OK", resultat);
    }
    @Test
    public void test_sjekkLoggInnIkkeOk(){

        // act
        String resultat = sikkerhetsController.sjekkLoggInn("01010110523688",
"HeiHei12345");

        // assert
        assertEquals("Feil i personnummer", resultat);
    }
    @Test
    public void test_LoggInnOk(){
        //arrange
        session.setAttribute("Innlogget", "12345678901");
        //act
        String resultat= sikkerhetsController.loggetInn();
        //assert
        assertEquals("12345678901",resultat);
    }
    @Test
    public void test_LoggInnIkkeOk(){
        // arrange
        session.setAttribute("Innlogget", null);
        //act
        String resultat= sikkerhetsController.loggetInn();
        //assert
        assertNull(resultat);
    }
    @Test
    public void test_LoggUtOk() {
        // arrange
        session.setAttribute("Innlogget", "12345678901");
```

```
        // act
        sikkerhetsController.loggUt();
        String resultat = (String) session.getAttribute("Innlogget");
        // assert
        assertNull(resultat);
    }

    @Test
    public void test_LoggInnAdminOk(){
        // arrange
        session.setAttribute("Innlogget", "Admin");
        //act
        String resultat= sikkerhetsController.loggInnAdmin("Admin","Admin");
        //assert
        assertEquals("Logget inn",resultat);
    }
    @Test
    public void test_LoggInnAdminIkkeOk(){
        // arrange
        session.setAttribute("Innlogget", null);
        //act
        String resultat=
sikkerhetsController.loggInnAdmin(anyString(),anyString());
        //assert
        assertEquals("Ikke logget inn",resultat);
    }

}
```





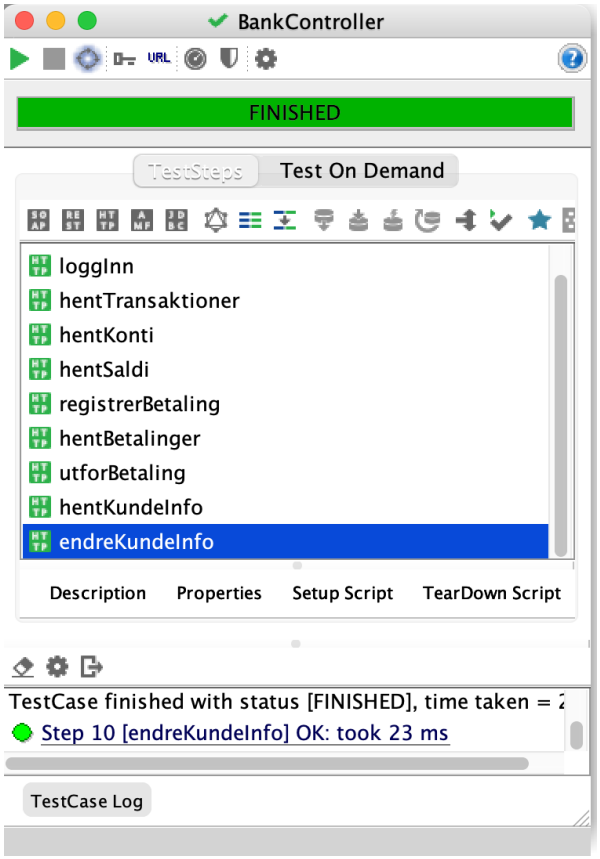


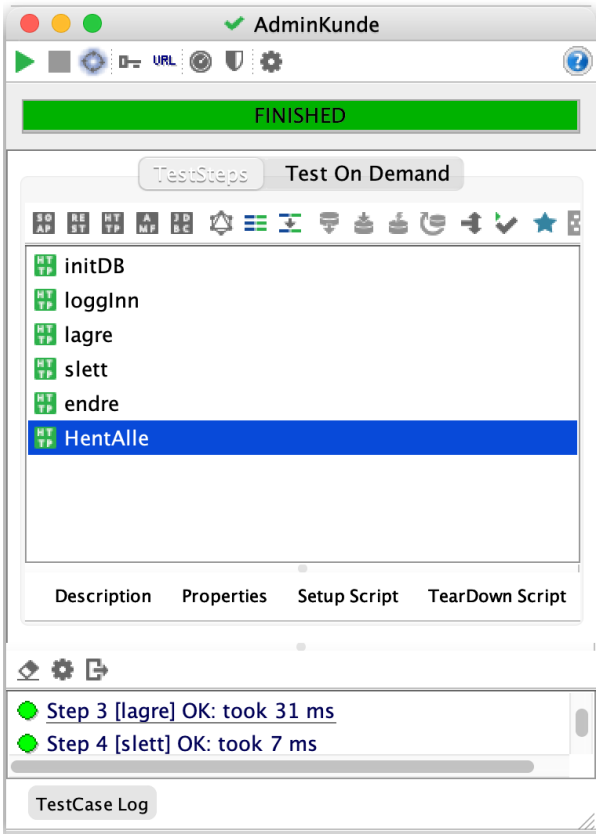
Vi kan observere at det er 100% Code Coverage for alle fire klassene. Under kjøringen av kodekjerne ble det vist grønne linjer på siden, som indikerer at alle metodene ble testet. Dersom vi hadde fått røde linjer på siden, ville det betydd at det var deler av koden som ikke ble testet. I slike tilfeller måtte vi ha utviklet tester for å oppnå full kodedekning.

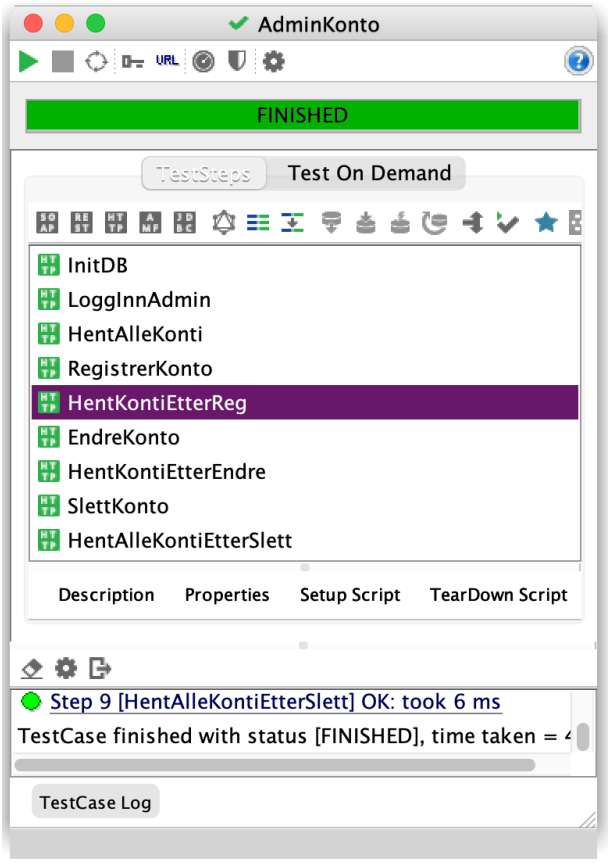
Element	Class, %	Method, %	Line, %
oslomet.testing	60% (6/10)	29% (26/89)	27% (92/333)
API	100% (3/3)	100% (17/17)	100% (69/69)
AdminKontoController	100% (1/1)	100% (4/4)	100% (17/17)
AdminKundeController	100% (1/1)	100% (4/4)	100% (16/16)
BankController	100% (1/1)	100% (9/9)	100% (36/36)

Integrasjonstesting:

Resultater fra SoapUI samt skjermbilder fra en Excel-fil som viser testresultater:




AdminKunde
FINISHED
TestSteps
Test On Demand
initDB
loggInn
lagre
slett
endre
HentAlle
Description
Properties
Setup Script
TearDown Script
Step 3 [lagre] OK: took 31 ms
Step 4 [slett] OK: took 7 ms
TestCase Log


AdminKonto
FINISHED
TestSteps
Test On Demand
InitDB
LoggInnAdmin
HentAlleKonti
RegistrerKonto
HentKontiEtterReg
EndreKonto
HentKontiEtterEndre
SlettKonto
HentAlleKontiEtterSlett
Description
Properties
Setup Script
TearDown Script
Step 9 [HentAlleKontiEtterSlett] OK: took 6 ms
TestCase finished with status [FINISHED], time taken =
TestCase Log

BankController:

Test Case ID	Test Case Description	Expected Result	Test Data	Actual Result	Status	Tester	Date/Time	Comments
BAC-0001	Initialisere databasen med InitDB	OK		OK	Pass	Emelie	02.03.24	Database initialisert
BAC-0002	Logge inn som kunde	OK	personnummer=01010110523 passord=HeiHei	OK	Pass	Emelie	02.03.24	Logget inn som kunde
BAC-0003	Hent transaktioner	Transaktioner Array	Kontonummer, FraDato, TilD	Transaktioner Array	Pass	Emelie	02.03.24	Transaktioner hentet
BAC-0004	Hent Konti	Konto Array	personnummer=01010110523	Konto Array	Pass	Emelie	02.03.24	Konton hentet
BAC-0005	Hent Saldi	Konto Array	personnummer=01010110523	Konto Array	Pass	Emelie	02.03.24	Saldi Hentet
BAC-0006	Registrer betalning	OK	Transaktion transaktion	OK	Pass	Emelie	02.03.24	Betalning registrert
BAC-0007	Hent betalinger	Transaktion Array	personnummer=01010110523	Transaktion Array	Pass	Emelie	02.03.24	Betalinger hentet
BAC-0008	Utför betaling	Transaktion Array	txID=1	Transaktion Array	Pass	Emelie	02.03.24	Betaling utfört
BAC-0009	Hente kunde info	Kunde kunde	personnummer=01010110523	Kunde kunde	Pass	Emelie	02.03.24	Kunde info hentet
BAC-0010	Endre kunde info	OK	Kunde kunde	OK	Pass	Emelie	02.03.24	Kunde info endret

AdminKunde:

Test Case ID	Test Case Description	Expected Result	Test Data	Actual Result	Status	Tester	Date/Time	Comments
AKU-0001	Initialisere databasen med InitDB	OK		OK	Pass	Emelie	02.03.24	Database initialisert
AKU-0002	Logge inn som admin	OK	bruker=Admin, passord=Admin	OK	Pass	Emelie	02.03.24	Logget inn som admin
AKU-0003	Lagre ny Kunde	OK	Kunde kunde	OK	Pass	Emelie	02.03.24	Kunde lagret
AKU-0004	Slett Kunde	OK	personummer = 01010110523	OK	Pass	Emelie	02.03.24	Kunde slettet
AKU-0005	Endre Kunde	OK	Kunde kunde	OK	Pass	Emelie	02.03.24	Kunde Endret
AKU-0006	Hent alle kunder	Kunde Array		Kunde Array	Pass	Emelie	02.03.24	Alla kunder hentet

AdminKonto:

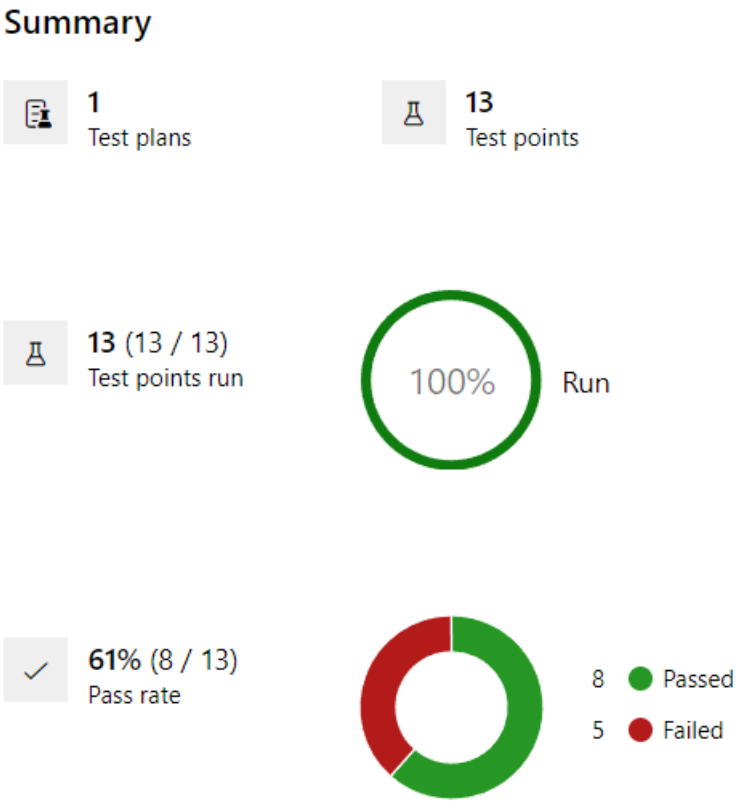
Test Case ID	Test Case Description	Expected Result	Test Data	Actual Result	Status	Tester	Date/Time	Comments
AKO-0001	Initialisere databasen med InitDB	Database initialisert		Database initialisert	Pass	Elise	04-03-24 10:00pm	Tilbakemelding "OK"
AKO-0002	Logge inn som admin	Logget inn som admin	bruker=Admin, passord=Admin	Logget inn som Admin	Pass	Elise	04-03-24 10:00pm	Tilbakemelding "Logget inn"
AKO-0003	Hente alle kontoer i DB	Alle kontoene i DB er hentet		Alle kontoene i DB er hentet	Pass	Elise	04-03-24 10:00pm	Fikk alle tre kontoene som lå i DB tilbake, assertions for Path count og kontonumre var riktige
AKO-0004	Registrere en ny konto i DB	Konto registrert	personnummer=12345678901 kontonummer=444444444444 saldo=414 type=Brukskonto valuta=NOK	Kontoen er registrert i DB	Pass	Elise	04-03-24 10:00pm	Tilbakemelding "OK"
AKO-0005	Hente alle kontoer i DB etter den nye kontoen er registrert	Alle kontoene i DB er hentet med oppdaterte verdier		Alle kontoene i DB er hentet	Pass	Elise	04-03-24 10:00pm	Fikk alle fire kontoene som lå i DB tilbake, assertions var riktige
AKO-0006	Endre en konto i DB	Konto er endret	personnummer=12345678901 kontonummer=444444444444 saldo=9000000 type=Brukskonto valuta=EUR	Kontoen er endret i DB	Pass	Elise	04-03-24 10:00pm	Tilbakemelding "OK"
AKO-0007	Hente alle kontoer i DB etter den nye kontoen etter verdiendringer i den nye kontoen	Alle kontoene i DB er hentet med oppdaterte verdier		Alle kontoene i DB er hentet	Pass	Elise	04-03-24 10:00pm	Fikk alle fire kontoene som lå i DB tilbake, assertions var riktige og endringer var oppdaterte
AKO-0008	Slette en konto i DB	Konto er slettet	kontonummer=444444444444	Kontoen er slettet i DB	Pass	Elise	04-03-24 10:00pm	Tilbakemelding "OK"
AKO-0009	Hente alle kontoer i DB etter den nye kontoen er slettet	Alle kontoene i DB er hentet med oppdaterte verdier		Alle kontoene i DB er hentet	Pass	Elise	04-03-24 10:00pm	Fikk alle tre kontoene som lå i DB tilbake, assertions var riktige

Systemtesting:

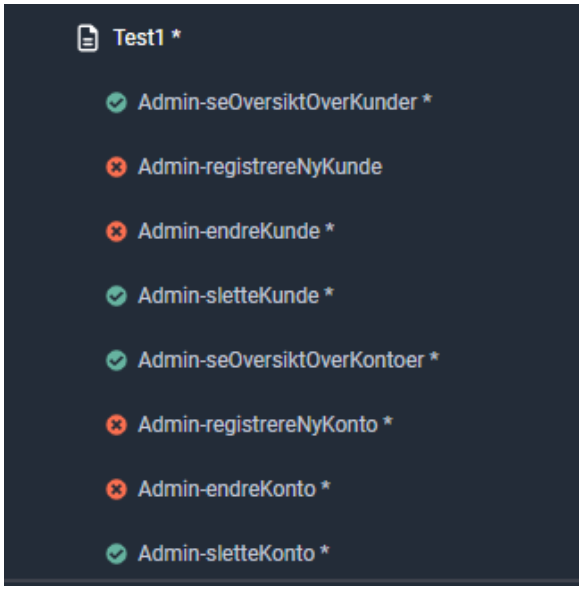
Brukerhistorier (Systemtest):

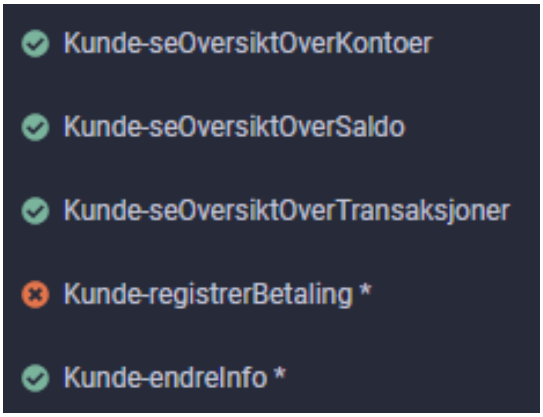
Type bruker	Brukerhistorie
Kunde/bruker	Som kunde ønsker jeg å kunne se oversikt over alle transaksjonene knyttet til mine kontoer for å beholde og se historikken.
Kunde/bruker	Som bruker ønsker jeg å kunne se oversikt over alle kontoene (mine) slik at jeg kan administrere dem slik jeg ønsker.
Kunde/bruker	Som bruker ønsker jeg å kunne se oversikt over saldoene til mine kontoer slik at jeg kan vite saldoen.
Kunde/bruker	Som bruker ønsker jeg å kunne registrere/utføre en betaling slik at jeg kan betale regninger.
Kunde/bruker	Som kunde ønsker jeg kunne endre info (om meg selv) slik at riktig informasjon blir lagret.
Admin	Som administrator ønsker jeg å se oversikt over alle kontoene i systemet for å kunne administrere dem.
Admin	Som administrator ønsker jeg å kunne registrere en ny konto slik at de kan lagres i systemet/databasen.
Admin	Som administrator ønsker jeg å kunne endre en eksisterende konto slik at riktig informasjon kan lagres.
Admin	Som administrator ønsker jeg kunne slette en konto slik at uønskede kontoer ikke ligger i systemet.
Admin	Som administrator ønsker jeg å kunne se oversikt over alle kunder for å kunne administrere dem.
Admin	Som administrator ønsker jeg å kunne registrere nye kunder slik at de legges til i systemet/databasen.
Admin	Som administrator vil jeg kunne endre kundeinformasjon for å kunne oppdatere opplysningene til kunden.
Admin	Som administrator vil jeg kunne slette kunder for å slette dem fra systemet.

Sammendrag av testresultatene hentet fra DevOps:



Testresultatene fra Katalon:





Testplanen og resultatene hentet fra Azure DevOps:

Test suite 32: 15 : Som administrator ønsker jeg å kunne se oversikt over alle kunder for å kunne administrere dem.

Test cases (1)

Test case 45: Se oversikt over kunder

PROPERTIES

Test Case Id:	45
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp "Nettbank" siden	
2	Logg inn som administrator	Får loggt inn og får opp liste over alle kunder i systemet	

LINKS

ID	WorkItemType	Link type	Title
15	User Story	Tests	Som administrator ønsker jeg å kunne se oversikt over alle kunder for å kunne administrere dem.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Passed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 31: 14 : Som administrator ønsker jeg å kunne registrere nye kunder slik at de legges til i systemet/databasen.

Test cases (1)

Test case 44: Registrering av kunde

PROPERTIES

Test Case Id:	44
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp Nettbank siden	
2	Logg inn som administrator	Får opp listen med kunder i systemet	
3	Gå til "Registrer kunde" vinduet	Får opp vinduet med innfyllingsskjema	
4	Fyll ut skjema med kundeinfo og trykk "Registrer kunde"	Får registrert kunde	

LINKS

ID	WorkItemType	Link type	Title
55	Bug	Tests	Registrering av kunde Failed Knappen fungerer ikke
14	User Story	Tests	Som administrator ønsker jeg å kunne registrere nye kunder slik at de legges til i systemet/databasen.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Failed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 30: 13 : Som administrator vil jeg kunne endre kundeinformasjon for å kunne oppdatere opplysningene til kunden.

Test cases (1)

Test case 43: Endring av kunde

PROPERTIES

Test Case Id:	43
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp vinduet for å logge inn	
2	Logg inn som administrator	Får opp listen av kunder	
3	Gå til vindu "Endre kunde"	Får opp "Endre kunde" vinduet	
4	Skriv endringen i ønsket felt deretter trykk "Endre"	Får endret kunden	

LINKS

ID	WorkItemType	Link type	Title
59	Bug	Tests	Endring av kunde Failed Knappen fungerer ikke
13	User Story	Tests	Som administrator vil jeg kunne endre kundeinformasjon for å kunne oppdatere opplysningene til kunden.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Failed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 27: 12 : Som administrator vil jeg kunne slette kunder for å slette dem fra systemet.

Test cases (1)

Test case 42: Sletting av kunde

PROPERTIES

Test Case Id:	42
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp Nettbank siden	
2	Logg inn som administrator	Får opp listen til alle kundene i systemet	
3	Trykk på "Slett" knappen	Får opp pop up boks til godkjenning av handling	
4	Trykk på "OK"	Får slettet kunden	

LINKS

ID	WorkItemType	Link type	Title
12	User Story	Tests	Som administrator vil jeg kunne slette kunder for å slette dem fra systemet.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Passed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 36: 19 : Som administrator ønsker jeg å se oversikt over alle kontoene i systemet for å kunne administrere dem.

Test cases (1)

Test case 49: Se oversikt over kontoer

PROPERTIES

Test Case Id:	49
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp siden "Nettbank"	
2	Logg inn som administrator	Får opp liste av alle kunder i systemet	
3	Gå til vinduet "Endre konto"	Får oversikt over alle kontoene	

LINKS

ID	WorkItemType	Link type	Title
19	User Story	Tests	Som administrator ønsker jeg å se oversikt over alle kontoene i systemet for å kunne administrere dem.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Passed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 35: 18 : Som administrator ønsker jeg å kunne registrere en ny konto slik at de kan lagres i systemet/databasen.

Test cases (1)

Test case 48: Registrering av ny konto

PROPERTIES

Test Case Id:	48
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp "Nettbank siden"	
2	Logg inn som administrator	Får opp liste over alle kunder	
3	Gå til "Registrer konto" vinduet	Får opp det vinduet med skjema med inputfelt	
4	Fyll ut med info deretter trykk på knappen "Registrer konto"	Får registrert kontoen	

LINKS

ID	WorkItemType	Link type	Title
56	Bug	Tests	Registrering av ny konto Failed Knappen fungerer ikke
18	User Story	Tests	Som administrator ønsker jeg å kunne registrere en ny konto slik at de kan lagres i systemet/databasen.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Failed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 34: 17 : Som administrator ønsker jeg å kunne endre en eksisterende konto slik at riktig informasjon kan lagres.

Test cases (1)

Test case 47: Endring av konto

PROPERTIES

Test Case Id:	47
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp "Nettbank" siden	
2	Logg inn som administrator	Får opp liste over alle kunder i systemet	
3	Trykk på "Endre konto" vinduet	Får opp liste over alle kontoer	
4	Skriv endringen i ønsket felt deretter trykk "Endre"	Får endret konto	

LINKS

ID	WorkItemType	Link type	Title
57	Bug	Tests	Endring av konto Failed Knappen fungerer ikke
17	User Story	Tests	Som administrator ønsker jeg å kunne endre en eksisterende konto slik at riktig informasjon kan lagres.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Failed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 33: 16 : Som administrator ønsker jeg kunne slette en konto slik at uønskede kontoer ikke ligger i systemet.

Test cases (1)

Test case 46: Sletting av konto

PROPERTIES

Test Case Id:	46
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp logg inn feltet	
2	Logg inn som administrator	Får opp liste over alle kunder	
3	Gå til vinduet "Endre konto"	Får opp liste over alle kontoer	
4	Trykk "Slett" på uønsket konto	Får slettet kontoen	

LINKS

ID	WorkItemType	Link type	Title
16	User Story	Tests	Som administrator ønsker jeg kunne slette en konto slik at uønskede kontoer ikke ligger i systemet.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Passed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 40: 23 : Som bruker ønsker jeg å kunne se oversikt over alle kontoene (mine) slik at jeg kan administrere dem slik jeg ønsker.

Test cases (1)

Test case 53: Se oversikt over alle kontoer som kunde

PROPERTIES

Test Case Id:	53
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Kommer til Nettbank siden	
2	Trykk på "Logg inn"	Får opp liste over alle kontoer	

LINKS

ID	WorkItemType	Link type	Title
23	User Story	Tests	Som bruker ønsker jeg å kunne se oversikt over alle kontoene (mine) slik at jeg kan administrere dem slik jeg ønsker.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Passed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 39: 22 : Som bruker ønsker jeg å kunne se oversikt over saldoene til mine kontoer slik at jeg kan vite saldoen.

Test cases (1)

Test case 52: Se overikt over saldo

PROPERTIES

Test Case Id:	52
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp Nettbank siden	
2	Trykk på "Logg inn"	Får opp liste over alle kontoer	
3	Gå til "Saldo" vinduet	Får opp saldoen til de ulike kontoene	

LINKS

ID	WorkItemType	Link type	Title
22	User Story	Tests	Som bruker ønsker jeg å kunne se oversikt over saldoene til mine kontoer slik at jeg kan vite saldoen.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Passed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 41: 24 : Som kunde ønsker jeg å kunne se oversikt over alle transaksjonene knyttet til mine kontoer for å beholde og se historikken.

Test cases (1)

Test case 54: Se oversikt over alle transaksjoner som kunde

PROPERTIES

Test Case Id:	54
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp Nettbank siden	
2	Trykk på "Logg inn"	Får opp liste med alle kontoer	
3	Gå til "Transaksjoner" vinduet	Får opp felt for kontonr og intervallet den skal søke for transaksjoner	
4	Velg kontonr, fra dato, til dato deretter trykk på "Søk"	Får opp liste over alle transaksjoner i det gitte intervallet	

LINKS

ID	WorkItemType	Link type	Title
24	User Story	Tests	Som kunde ønsker jeg å kunne se oversikt over alle transaksjonene knyttet til mine kontoer for å beholde og se historikken.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Passed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 38: 21 : Som bruker ønsker jeg å kunne registrere/utføre en betaling slik at jeg kan betale regninger.

Test cases (1)

Test case 51: Registrering av betaling

PROPERTIES

Test Case Id:	51
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp Nettbank siden	
2	Trykk på "Logg inn"	Får opp liste over alle kontoene	
3	Gå til vinduet "Registrer betaling"	Får opp skjema med inputfeltene for å registrere en betaling	
4	Fyll ut feltene deretter trykk på "Registrer betaling"	Får registrert betaling	

LINKS

ID	WorkItemType	Link type	Title
58	Bug	Tests	Registrering av betaling Failed Knappen fungerer ikke
21	User Story	Tests	Som bruker ønsker jeg å kunne registrere/utføre en betaling slik at jeg kan betale regninger.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Failed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

Test suite 37: 20 : Som kunde ønsker jeg kunne endre info (om meg selv) slik at riktig informasjon blir lagret.

Test cases (1)

Test case 50: Endring av info som kunde

PROPERTIES

Test Case Id:	50
Assigned To:	Kevin Olsen
State:	Design

STEPS

#	Action	Expected value	Attachments
1	Gå til localhost:8080	Får opp "Nettbank" siden	
2	Trykk på "Logg inn"	Får opp kontoene til innloggene kunden	
3	Gå til vinduet "Kunde-info"	Får opp feltene med kundeinfo	
4	Utfør de nødvendige endringene deretter trykk på "Endre info"	Får endret infoen	

LINKS

ID	WorkItemType	Link type	Title
20	User Story	Tests	Som kunde ønsker jeg kunne endre info (om meg selv) slik at riktig informasjon blir lagret.

LATEST TEST OUTCOME

Outcome	Tester	Configuration	Run by	Date completed	Duration	Build number
Passed	Kevin Olsen	Windows 10	Kevin Olsen	mandag 11. mars 2024	0:00:00.000	0

6. AVVIK I FORHOLD TIL TESTPLANEN

I starten delte vi ansvaret for de ulike testene, slik at hver person kunne fokusere på en spesifikk testtype og bli dyktigere innen den. Dette tillot oss å fordype oss i hver testtype og deretter gjennomgå resultatene sammen for å få en helhetlig forståelse av testdekket. En utfordring i begynnelsen var å forstå de forskjellige testverktøyene og hvordan vi skulle anvende dem effektivt for vårt formål. Ettersom applikasjonen vi testet var ikke omfattende eller komplisert, har denne prosessen gått relativt raskt og smidig.

Integrasjonstestene ble utført senere enn det timeplanen for faget tilsa, men fremdeles innenfor akseptabel tid. Når de først ble gjennomført var det ikke mye avvik. En av utfordringene vi møtte var å finne riktig format for parameterverdiene, da key+value metoden ikke alltid fungerte. Vi fant ut at bruk av JSON-formatet var mer pålitelig. Vi støtte også på problemer med spesialtegn som «ø» og «å», spesielt i forbindelse med «Lønnskonto», da SoapUI ikke godtok disse tegnene. Andre viktige momenter som vi mista på starten, var behovet for å inkludere «Drop table if exixts» i SQL-filen for å kunne kjøre integrasjonstestene vellykket.

Under systemtestingen opplevde vi problemer med å registrere tidligere kjørte tester i Katalon. Ettersom det ikke var mulig å dele testsuitene, sto vi overfor utfordringen med å enten lage alle testene på en datamaskin eller dele oppgaven (men vi kunne ikke samarbeide om testene). Vi hadde også problemer med å finne riktig assert-kommando som testet det vi faktisk trengte å teste. På den positive siden har dokumenteringen av testene i Azure DevOps gått greit og bidratt til å ha oversikt over prosessen.

7. GODKJENNINGSKRITERIER

Godkjenningskriteriene til systemet: 95% av testene definert skal være utført og ingen skal være av type A og B kategori.

Kategori A: "Feil i en eller flere funksjoner som forårsaker stopp. For eks. Maskin/system/funksjon må startes på nytt."

Kategori B: "Feil har alvorlige konsekvenser i systemet eller tilstøtende systemer, for eksempel feil i databaser, alvorlige feil i rapporter slik at disse tolkes feil etc."

Kategori C: "Feil med mindre alvorlige konsekvenser, for eksempel mindre grad av ustabilitet, mindre funksjons-/egenskapsmangler etc."

Kategori D: "Feil uten alvorlige konsekvenser, for eksempel layout-/trykkfeil i skjermbilder, rapporter eller dokumentasjon som ikke har konsekvenser for forståelsen av disse."

8. VURDERING AV TESTENS GRUNDIGHET

Vurdering av enhetstestene: Testingen av Nettbank-applikasjonen var en vellykket prosess. Dette skyldes hovedsakelig to faktorer: først oppnådde vi 100% Code Coverage, og vi testet hele kontroller-siden grundig. For det andre var applikasjonen selv ikke særlig kompleks, noe som gjorde det enkelt å gjennomføre testingen. Denne kombinasjonen av fullstendig kodedekning og enkelhet i applikasjonen resulterte i at det ikke var noen avvik eller feil som ble oppdaget under testingen.

Vurdering av integrasjonstestene: Integrasjonstestene kunne vært gjennomført grundigere. Når systemet ble testet gjorde vi bare tester der brukeren var innlogget og det ble ikke utført tester der brukeren f.eks. ikke var innlogget eller hadde regex feil. Vi mangler også testing for å sikre at kun autoriserte brukere har tilgang til spesifikke metoder slik som det administratoren har.

Vurdering av vår systemtesting: Vi har ikke utført en grundig systemtesting, men ut ifra de testene vi har utført vil ingen av de mislykkede testene føre til stopp i maskinen eller systemet dermed er de ikke av kategori A. De er av kategori B da for eksempel mangel på evnen til å registrere nye kunder eller endre info om enten konto eller kunde vil føre til alvorlige konsekvenser for systemet. Dette vil føre til at systemet ikke fungerer som den skal da vanlige funksjoner ikke godkjennes. Databasen blir ikke oppdatert. Testfeilene kan også klassifiseres som kategori C ettersom feil ved registrering av konto vil kunne føre til frustrasjon hos kunden.

9. GJENSTÅENDE AKTIVITETER

Enhetstestene er basert på kontrollermetodene som er gitt. Integrasjonstestene våre mangler blant annet viktige sjekker for ikke-innloggede tilfeller, for eksempel om en innlogget kunde får tilgang til de samme rettighetene som en administrator, noe som ikke skal skje. Et annet eksempel kan være at metodene i systemet skal ikke være tilgjengelige

hvis man ikke er logget inn generelt. Selv om de utvalgte integrasjonstestene har blitt godkjent, indikerer dette ikke at integrasjonstestingdelen er fullstendig eller godkjent.

Brukerhistoriene som er skrevet i systemtestingdelen, representerer kun en del av de mulige brukshistorier, og det kan være flere som bør legges til og testes. Det er avgjørende å rette opp feil som blir oppdaget under systemtestingen.

Som tidligere nevnt er de gjennomførte testene en av de viktigste, men det er også andre testnivåer som burde utføres, for eksempel akseptansetesting. Dette nivået kommer etter systemtesting ettersom det er viktig at systemet aksepteres av kunden. Noe som er viktig at systemet oppfyller da systemet skal gjøre det kunden ønsket seg.

10. AVVIK I PRODUKTET

Alle enhetstestene ble godkjent og hadde full codecoverage, dette betyr at godkjenningskravene til testene ble oppfylt. Og selv om integrasjonstestene også ble godkjent har vi ikke testet nok tilfeller for å kunne vurdere produktet ut ifra kun disse testene. I nettbank vil sikkerhet være en av de viktigste egenskapene som systemet bør ha. Kunden skal kunne stole på at kontonummeret og personnummeret deres er kun tilgjengelig for personell med spesiell rettighet til det. Det bør dermed også utarbeides tester som tester dette.

Koden til Nettbanks-systemet vårt er ikke fullstendig til å kunne ha det ute i produksjon. Dette kan vi se ut ifra de feilende testene fra systemtestingen. Der oppdaget vi at systemet utfører ikke de mest grunnleggende funksjonalitetene beskrevet i brukerhistoriene. Systemet er dermed ikke klar til bruk. Alle følgende testcases ble ikke godkjent «Admin-registrerNyKunde», «Admin-registrerNyKonto», «Admin-endreKonto», «Admin-endreKunde» og «Kunde-registrerBetaling». For alle disse testene ville ikke knappen fungere slik det står i koden. For eksempel skal «Registrer betaling» knappen sende kunden til siden som heter «Utført betaling» noe som ikke skjedde som gjør at testen feilet. I praksis skal kunden kunne registrere en betaling og den skal være tilgjengelig for han.

11. VURDERING AV PRODUKTET SOM ER TESTET

Produktet som har blitt testet er ikke klart for brukere ettersom feilene som har blitt avdekket i systemtestene har vært såpass omfattende. Når brukere ikke får utført det de ønsker på applikasjonen og heller blir møtt med feil, kommer ikke systemet slik det er nå til å være et ønsket produkt på markedet. For å forbedre programvaren må feilene som er funnet i systemtesten utbedres slik at brukere får utført det de ønsker på systemet. Produktet bør også testes ytterligere etter feilene fra systemtesten er rettet.

Vi observerte også at fra kundeperspektiv fremstår det ikke så estetisk tiltalene som det ideelt sett burde være. Vi kom frem til at det er nødvendig med en forbedret designprosess for å oppnå et mer tiltalene utseende som bedre samsvarer med kundens forventninger til

et nettbanks system. I tillegg bør det også inkludere flere funksjoner som er typiske for en moderne nettbank. Dagens utforming fremstår som for enkel. Den bør utarbeides til å utvide funksjonalitet og gjøre det mer komplett og brukervennlig.

12. SAMMENDRAG AV AKTIVITETER OG LÆRDOM

I dette prosjektet har vi fått verdifull erfaring og innsikt i testing av programvare, fra planlegging til gjennomføring av ulike testmetoder. Vi har forstått viktigheten av grundig testing for å sikre høy kvalitet på den utviklede programvaren. Selv om applikasjonen var av begrenset omfang, ble det avdekket flere programfeil som måtte rettes opp. Dette understreker betydningen av jevn og kontinuerlig testing gjennom hele utviklingsprosessen. Den største utfordringen vi møtte på var å bli kjent med de nye verktøyene brukt for de ulike testene. På denne måten har vi fått bedre forståelse for de verktøyene og deres relevans i programvaretesting.