

ADTS 2310-Testing av programvare

Våren 2024

Gruppe 9:

S333329 - Therese Trollbu

S375057 - Rayan Kanabi

S375060 - Jonas Karlsen

S375049 - Andreas Loar Jensen

Innholdsfortegnelse

ADTS 2310-Testing av programvare.....	1
Innholdsfortegnelse.....	2
1.0 Innledning.....	3
1.1 Oppgaven.....	3
1.2 Funksjonalitet.....	3
2.0 Testene.....	4
2.1 Enhetstest.....	4
2.2 Integrasjonstest.....	6
2.3 Systemtest.....	6
3.0 Fremgangsmåte.....	7
3.2 Samarbeid.....	7
3.2 Brukerhistorier.....	7
3.2 Verktøy.....	9
3.3 Testplan.....	10
4.0 Resultater og avvik.....	11
4.1 Enhetstesting.....	11
4.2 Integrasjonstest.....	12
Oversikt over hvilke tester som passerte og hvilken som ikke gjorde det.....	12
4.3 Systemtesting.....	13
5.0 Vurdering av produktet og arbeidet vårt.....	17
5.1 Produktet.....	17
5.2 Arbeidet vårt.....	18
6.0 Konklusjon.....	19
7.0 Vedlegg.....	20
7.1 Vedlegg A - Enhetstester.....	20
EnhetstestAdminKontoController:.....	20
EnhetstestAdminKundeController:.....	24
EnhetstestBankController:.....	29
EnhetstestSikkerhetController:.....	40
Hjelpemetoder til testene:.....	45
7.2 Vedlegg B - Integrasjonstestresultater.....	48
bankController:.....	48
kontoController:.....	49
kundeController:.....	50
Sikkerhet:.....	51
7.3 Vedlegg C - Google Sheets-ark.....	52
7.4 Vedlegg D - Microsoft Test Management (Azure) dokumentasjon.....	54

1.0 Innledning

1.1 Oppgaven

Prosjektet går ut på å teste ut en ferdig utviklet applikasjon ved å gjøre enhetstester, integrasjonstester og systemtester. Applikasjonen er en forenklet versjon av en nettbank og er skrevet i Spring Boot (Java). Under testingsperioden, var vi innstilt på å avdekke alle funksjonalitetene og eventuelle feil. Vårt arbeid med dette er dokumentert i denne rapporten.

1.2 Funksjonalitet

Applikasjonen sin funksjonalitet er beregnet for to brukere, administrator og kunde. Nedenfor er listen over funksjonalitetene hver av disse har. Disse ble utviklet til brukerhistorier senere i prosessen.

Administrator

- Logge inn og ut
- Registrere, endre og slette konto
- Registrere, endre og slette kunde

Kunde

- Logge inn og ut
- Se oversikt over egne konti og egen saldo
- Registrere og utføre betalinger
- Se alle transaksjoner
- Se kundeinfo om seg selv
- Endre kundeinformasjon

2.0 Testene

2.1 Enhetstest

Kode til enhetstestene kan bli funnet i Vedlegg A.

Enhetstester er betegnet som “white box testing”, som enkelt sagt betyr at tester utføres i kontekst av at kode er synlig til systemet. I denne typen testing, så gjør vi direkte kall på metoder og lager virtuelle klasser for å simulere hvordan dataene fungerer “bak kulissene”.

Målet med enhetstestene våre er å teste enhver metode i forskjellige scenarioer.

Eksempler på slike scenarioer er når:

- Bruker ikke er logget inn.
- Bruker er logget inn.
- “Feil” data sendes inn.
- Overflødig mengde data sendes inn.

Når vi white box tester, er det viktig å ha bred kunnskap om helheten av koden og vite hva vi forventer å få returnert fra våre metoder og klasser. Vi deler enhetstestene i stegene “arrange”, “act” og “assert”.

Arrange

Her bruker vi frameworket Mockito, som får oss til å “simulere” en database ved å kunne oppgi egendefinerte returnerte verdier fra repository-metodene. I mange tilfeller i våre tester, innebærer det at det returneres “OK”, “Feil” eller null. I tilfeller der repository-metodene forventer å returnere en spesifikk instans av en klasse, initialiserer vi også slike instanser.

Act

Dette steget går ut på å se hvordan controller oppfører seg basert på Mockitos kall fra repository.

Assert

Deretter så sammenligner man resultatet fra “act” med det vi forventer å få fra systemet.

I eksempel under kan vi se disse stegene:

```

1 Jonaskarlsen *
@Test
public void hentSaldi_loggetInn() {
    // ARRANGE:
    List<Konto> kontoer = new ArrayList<>();
    kontoer.add(new Konto(
        personnummer: "1234567890",
        kontonummer: "1234567890",
        saldo: 1000,
        type: "Lånskonto",
        valuta: "NOK",
        Hjelp.transaksjonsGenerator(
            ania: 1)));

    when(sjekk.loggetInn()).thenReturn("81010110523");

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

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

    // ASSERT:
    assertNotNull(resultat);
    assertEquals(kontoer, resultat);
}

```

I dette eksemplet, så har vi to assertions. Vi forventer at siden bruker er logget inn, så skal konto-listen returneres, derfor er “assertNotNull” på plass. Sammen med “assertEquals” under, dekker dette også tilfellet der både Mockito og Controller skulle på en eller annen måte feile og returnere null.

Element ^	Class, %	Method, %	Line, %
all	70% (7/10)	28% (26/91)	33% (114/338)
oslomet	70% (7/10)	28% (26/91)	33% (114/338)
testing	70% (7/10)	28% (26/91)	33% (114/338)
API	100% (3/3)	78% (15/19)	90% (67/74)
AdminKontoController	100% (1/1)	80% (4/5)	94% (18/19)
AdminKundeController	100% (1/1)	60% (3/5)	72% (13/18)
BankController	100% (1/1)	88% (8/9)	97% (36/37)
DAL	0% (0/2)	0% (0/19)	0% (0/171)
AdminRepository	0% (0/1)	0% (0/9)	0% (0/89)
BankRepository	0% (0/1)	0% (0/10)	0% (0/82)
Models	100% (3/3)	16% (8/48)	42% (29/69)
Konto	100% (1/1)	14% (2/14)	40% (8/20)
Kunde	100% (1/1)	16% (3/18)	42% (11/26)
Transaksjon	100% (1/1)	18% (3/16)	43% (10/23)
Sikkerhet	100% (1/1)	75% (3/4)	78% (18/23)
Sikkerhet	100% (1/1)	75% (3/4)	78% (18/23)
Main_Nettbank	0% (0/1)	0% (0/1)	0% (0/1)

Ovenfor er statistikk på Code Coverage. Det er verdt å nevne at initDB som blir brukt i alle kontrollerne, ikke ble testet.

For å simulere en stor mengde variert data, har vi laget hjelpemetoder som genererer forskjellige instanser av Kunder, Kontoer og Transaksjoner. Disse metodene lager tilfeldige strenger og numre basert på hva som forventes med Regex. Disse blir brukt til å teste om systemet klarer å håndtere større mengder data i “terskel”-testene.

2.2 Integrasjonstest

Resultatene fra integrasjonstestene kan bli funnet i Vedlegg B.

Med integrasjonstester kan vi teste samholdet mellom systemet og databasen ved å bruke POST- og GET-kall fra URL-endepunktene (F. eks:” /hentKonti”). Når vi henter informasjonen, kan vi bruke “asserts” til å bekrefte om vi fikk det forventede resultatet. Under denne prosessen bruker vi SoapUI. For at databasen skal fungere med testene våre, måtte vi lage en metode som heter “initDB”, som kort sagt initialiserte databasen. Dette gjorde det lettere for oss å starte på ny etter at en test hadde kjørt.

2.3 Systemtest

Azure-dokumentasjonen kan bli funnet i Vedlegg D.

Systemtester handler om å teste hvordan programmet fungerer “under production”, altså under normal bruk av en aktør av systemet. Systemtestene kjøres i nettleseren, slik at vi kan nøyaktig simulere hvordan systemet oppfører seg.

Brukerhistorier spiller en sentral rolle i alle testprosessene, men i systemtesting spiller dette en større rolle. For her kan vi faktisk spille av scenarioen “Bruker vil betale sine regninger”.

Imens integrasjonstest handlet om å sjekke om dataflyten fungerte, handler systemtesting om å sjekke om dataflyten skjer i samsvar med front-end. Med andre ord, spiller effektiviteten til samarbeidet mellom HTML, Javascript og Spring Boot en stor rolle.

Systemtestene ble gjort i Katalon Recorder og dokumentert i Microsoft Azure Test Management.

3.0 Fremgangsmåte

3.2 Samarbeid

Vi var innstilt på at alle skulle få prøve seg på de tre ulike testene på applikasjonen, slik at alle fikk utvikle en bredere forståelse for testprosessene og hele applikasjonen. Vi fordelte de ulike kontrollene slik at hver og en av oss hadde ansvar for sin egen kontroller under enhetstesting, integrasjonstesting og systemtesting. Dette førte til at kvaliteten på testene økte, ettersom det var forskjellige personer som testet ulike deler av applikasjonen - og dermed fikk vi gjennomført testene fra ulike perspektiv. Under integrasjonstestene og systemtestene laget vi digitalt en oversikt over hvilke kontroller som hadde blitt utført (pass or fail) slik at alle hadde en ryddig oversikt over hva som hadde blitt gjort ferdig etterhvert.

3.2 Brukerhistorier

I henhold til systemutviklingsprosessen “test-drevet utvikling”, lager vi tester basert på brukerhistorier. Disse historiene representerer nettbankens funksjonaliteter og hva slags forventninger enhver aktør har til systemet. Senere har vi delt disse opp i aktørene “Admin” og “Kunde”, ingen av disse aktørene deler funksjonalitet, men det er fellestrekk mellom dem. For eksempel bruker både admin og bruker metodene “hentKonti” og “hentSaldi”, men i forskjellige bruksområder.

Admin

Skal kunne:

1. Logge inn
2. Endre verdi i hvert enkelt felt under konto
3. Endre verdi i hvert enkelt felt under kunde
4. Slette en og en kunde
5. Slette en og en konto
6. Registrere en ny konto
7. Registrere en ny kunde

Kunde

Skal kunne:

1. Logge inn.
2. Registrere betaling.
3. Utføre betalinger.
4. Se sine betalinger.
5. Se sine transaksjoner.
6. Se sin(e) konto(er).
7. Se saldo på enhver konto.
8. Se sin kunde-info.
9. Endre hvert enkelt felt under kundeinfo.

3.2 Verktøy

Intellij Ultimate

Programmet vi bruker til å kjøre prosjektfilen og enhetstestene.

SoapUI

Programmet vi bruker til å gjøre integrasjonstesting hvor vi jobber med hver vår fil.

Google Sheets

For å dokumentere fortløpende resultater av integrasjonstestene.

Katalon Recorder

Brukes for å gjøre systemtesting.

Azure

Microsoft Test Management (MTM) som brukes til dokumentasjon av systemtestingen.

3.3 Testplan

Enhetstesting

Målet vårt var at enhetstestene skulle være så omfattende som mulig med høyest mulig prosent på Code Coverage. En av prinsippene i testing er at det ikke er mulig å teste absolutt *alt*. Dermed var det viktig å ha en realistisk “scope” og teste det mest relevante først. Dette ble naturligvis til at enhver metode ble testet minst én gang, og at det ble lagt til ytterligere tester etter hvert. Med brukerhistorier som tenkte scenarioer, kunne vi lage enhetstestene definert av dette.

Integrasjonstesting

I Google-Sheets dokumentet har vi et regneark kalt *Overview*, der man kan se en total oversikt over hvor mange tester som gjenstår og en endringslogg som dokumenterer når dokumentet sist ble endret. I *Test Overview* har vi en oversikt over alle klassene og navn på endpoints som skulle testes, hva slags resultat vi forventet, status, dato og kommentarer. Gruppen delte opp forskjellige ansvarsområder og navnet på ansvarlige står også i dokumentet. Dokumentet oppdaterer seg automatisk når man endrer status på hver test, slik at man alltid kan få en totaloversikt øverst i dokumentet og på *overview*-regnearket.

Systemtesting

Vi brukte Microsoft Azure Test Management (MTM) til dokumentasjon av systemtestingen. Der ble det laget en test suite for funksjonene til hver controller. Vi brukte det for å planlegge og fordele arbeid under *Boards*, og under *Test Plans* skrev inn steg for steg hvordan vi utførte system testingen av hver og en controller. Dokumentet oppdaterte seg automatisk når endringene ble lagret, slik at alle fikk “real-time” oppdateringer.

4.0 Resultater og avvik

4.1 Enhetstesting

✓ ! <default package> 24 sec 788 ms	✓ ! EnhetstestBankController 21 sec 842 ms
✓ ! EnhetstestAdminKontoController 311 ms	✓ hentKonti_loggetInn 15 ms
✓ slettKontoTest_ikkeLoggetInn 301 ms	✓ endre_ikkeLoggetInn
✓ registrerKontoTest_ikkeLoggetInn 1 ms	✓ hentSaldi_loggetInn 1 ms
✓ hentAlleKonti_ikkeLoggetInn 1 ms	✓ hentKonti_terskel 14 sec 21 ms
✓ endreKontoTest_ikkeLoggetInn 2 ms	✓ hentBetaling_ikkeLoggetInn 1 ms
✓ slettKontoTest 1 ms	✓ hentKundeInfo_loggetInn 1 ms
✓ endreKontoTest	✓ hentTransaksjoner_loggetInn 1 ms
✓ hentAlleKonti_loggetInn 3 ms	✓ hentTransaksjoner_ikkeLoggetInn 2 ms
✓ registrerKontoTest 1 ms	✓ registrerBetaling_loggetInn
! unnecessary Mockito stubbings 1 ms	✓ registrerBetaling_ikkeLoggetInn
✓ EnhetstestAdminKundeController 2 sec 592 ms	✓ hentBetaling_terskel 6 sec 309 ms
✓ hentAlle_loggetInn 1 ms	✓ hentTransaksjoner_terskel 13 ms
✓ endre_ikkeLoggetInn 1 ms	✓ endre_loggetInn 1 ms
✓ slett_ikkeLoggetInn	✓ utforBetaling_ikkeLoggetInn 1 ms
✓ lagreKunde_loggetInn	✓ utforBetaling_loggetInn
✓ hentAlle_terskel 2 sec 588 ms	✓ hentSaldi_terskel 1 sec 466 ms
✓ slett_loggetInn 1 ms	✓ hentBetaling_loggetInn 1 ms
✓ endre_loggetInn 1 ms	✓ hentSaldi_ikkeLoggetInn 1 ms
✓ lagreKunde_ikkeLoggetInn	✓ utforBetaling_terskel 6 ms
✓ hentAlle_ikkeLoggetInn	✓ hentKonti_ikkeLoggetInn 1 ms
! unnecessary Mockito stubbings	✓ hentKundeInfo_ikkeLoggetInn
	! unnecessary Mockito stubbings 1 ms
	✓ EnhetstestSikkerhetController 43 ms
	✓ testSjekkLoggInn_altFeil 29 ms
	✓ testSjekkLoggInn 5 ms
	✓ testInnlogget_altFeil 2 ms
	✓ testSjekkLoggInn_feilPersonnummer 2 ms
	✓ testInnlogget 1 ms
	✓ testLoggInnAdmin 2 ms
	✓ testSjekkLoggInn_feilPassord 1 ms
	✓ testLoggUt 1 ms
	! unnecessary Mockito stubbings

Som man kan se i skjermbildene over, så er det feilmeldinger fra Mockito. Etter hjelp fra andre med samme problem, har vi kommet til konklusjonen at denne feilmeldingen er uunngåelig med repository-strukturen som er på plass i applikasjonen. Det ser ut til at Mockito syns linjer som “*when(repository.hentAlleKunder()).thenReturn(kunder);*” er redundante, men i kontekst av testene våre, er de nødvendige.

Terskel-testene går ut på å sjekke om systemet klarer store “loads” av data. Vi legger ikke mye vekt på dette i vår vurdering av systemet, i og med at relevansen er diskuterbar. En konto kan potensielt ha flere tusen transaksjoner, derfor kan det være nyttig å sjekke om systemet klarer dette i det hele tatt. Resultatet av disse kan variere i test omgivelsene til IntelliJ, på grunn av varierende hardware. Et eksempel på dette, er at en av gruppe-medlemmene har mer minne tilgjengelig som da kunne kjøre testene helt feilfritt. Mens andre fikk “java.lang.OutOfMemoryError: Java heap space” feilmelding. Disse testene tar også mye lengre tid å kjøre, noe som gjør at vi velger å si at denne typen testing virker mindre realistisk å gjøre i et større prosjekt.

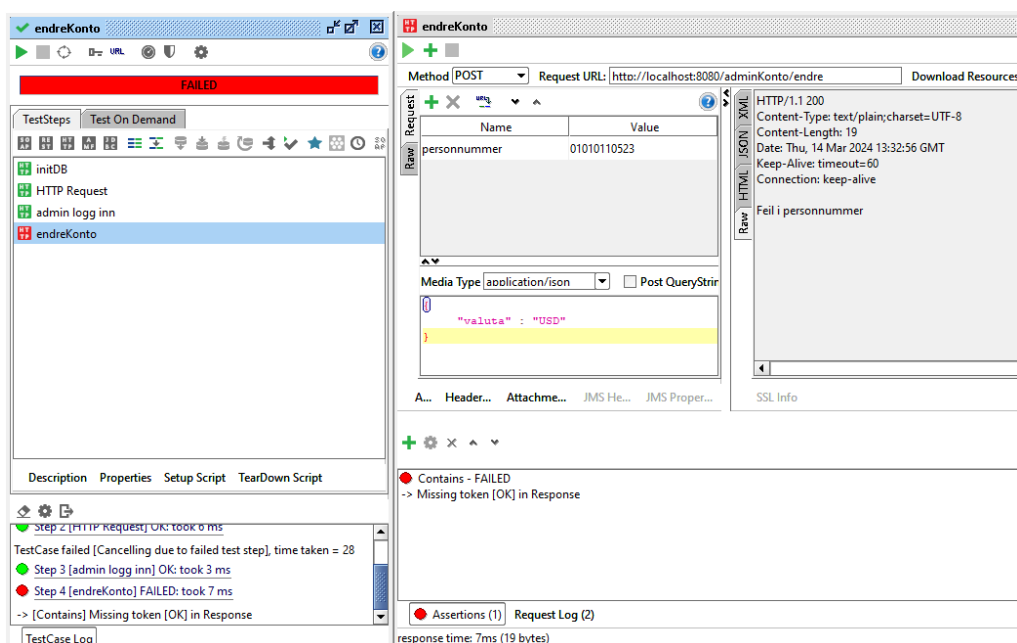
4.2 Integrasjonstest

Oversikt over hvilke tester som passerte og hvilken som ikke gjorde det.

Class	Endpoint/method	Forventet resultat	Status	Dato endret sist	Kommentar(er)	Tester
Sikkerhet	/logInnAdmin	Logget inn	Pass	01.03.24 12:24		Andreas
	/logInn	OK	Pass	01.03.24 13:48		Andreas
	/loggUt	NULL	Pass	01.03.24 13:53		Andreas
KontoController	/register	OK	Pass	07.03.24 11:40		Therese
	/hentAlle	Kontoer	Pass	07.03.24 11:40		Therese
	/endre	OK	Fail	07.03.24 11:40	Feil i personnummer	Therese
	/slett	Slettet	Pass	07.03.24 11:40		Therese
	/slett	Slettet	Pass	07.03.24 11:40		Therese
KundeController	/hentAlle	Kunder	Pass	07.03.24 12:34		Rayan
	/lagre	OK	Pass	07.03.24 13:27		Rayan
	/endre	OK	Pass	07.03.24 12:22		Rayan
	/slett	OK	Pass	07.03.24 12:20		Rayan
BankController	/hentTransaksjoner	Antall: 3, txID: 7	Pass	07.03.24 13:04		Andreas
	/hentKonti	Antall: 3, Kontonummer nr1: 105020123456	Pass	07.03.24 13:25		Andreas
	/hentSaldi	JSON-antall: 3, JSON-match	Pass	11.03.24 14:22		Jonas
	/registrerBetalings	JSON-antall: 4, JSON-match	Pass	12.03.24 18:23		Jonas
	/hentBetalinger	JSON-antall: 3, JSON-match	Pass	13.03.24 03:02		Jonas
	/utforBetalings	JSON-antall: 2, JSON-match	Pass	14.03.24 06:48		Jonas
	/hentKundeInfo	Fornavn: Lene, Etternavn: Jensen	Pass	04.03.24 13:31		Andreas
	/endreKundeInfo	Fornavn: Ryan, Etternavn: Gosling	Pass	04.03.24 12:28		Jonas

Det er ikke mulighet til å endre konto i integrasjonstesten. Feilmeldingen som dukker opp er “Feil i personnummer”, men personnummeret som er skrevet skal være riktig og er en av de som allerede eksisterer i databasen ved oppstart av programmet. Det ser ut til at endre-metoden i kontoController ikke fungerer fordi den tar inn personnummer fra loggetInn-metoden. Denne metoden fungerer kun på bruker-siden av nettbanken, i og med at den tar inn et personnummer. Personnummer blir spesifisert når bruker logger inn og dette blir lagret i session. Når man er admin, er det ingen personnummer

registrert i session, og dermed vil ikke personnummeret “matche” hva enn man skriver i integrasjonstesten.



Feilmelding i SoapUI.

4.3 Systemtesting

Oversikt over systemtestene som ble utført og hvorvidt testene passerte eller ikke.

Controller	Tester	Pass/Fail
AdminKontoController	Endre verdi i hvert enkelt felt under konto som admin	Fail
	Slette en og en konto	Fail
	Registrere en ny konto	Pass
AdminKundeController	Registrere en ny kunde	Fail
	Slette en og en kunde	Fail
	Endre kundeinformasjon som admin	Fail
BankController	Endre verdi i hvert enkelt felt under kunde	Pass
	Registrere betaling	Fail

	Utføre betalinger	Pass
	Se sine betalinger	Pass
	Se sine kontoer	Pass
	Se saldo på enhver konto	Pass
	Se sine transaksjoner	Pass
	Se Sin kunde-info	Pass
Sikkerhet	Som bruker vil jeg kunne logge inn	Pass
	Som admin vil jeg kunne logge inn	Pass
	Feilmelding ved feil innlogging som admin	Pass
	Feilmelding ved feil innlogging som bruker	Pass
	Mulighet til å logge ut	Pass

Under systemtesting var det totalt 6 tester som ikke passerte. Noe som ble oppdaget under testing, var at man ikke kunne endre kontonummer og personnummer, men vi antar at disse to er låst med vilje “by design”. Derfor er ikke de oppgitt under som avvik. De andre som ikke passerte er følgende:

Registrere betaling

Avviket til ‘Registrere betaling’ ligger i at den ikke har funksjonalitet på front-end siden. Ved å teste gjennom SoapUI så får vi bekreftet at den fungerer back-end og testene kjøres og funksjonen fungerer som ønsket. Det antas at feilen ligger i javascriptet til knappen “Registrer betaling” og at POST- og GET-kallene ikke får kjørt som de skal.

Registrere ny kunde

Her ligger det også problemer i at det ikke er noen form for funksjonalitet på front-end siden. Back-end testene får like resultater som i ‘Registrere betaling’. Dette i at man får kjørt med ønsket funksjonalitet om man gjør det gjennom SoapUI. Her er det samme antagelse om at det er problemer i POST- og GET-kallene.

Endre konto/kunde som Admin

Når man kjører test i Katalon Recorder og endrer verdien i feltene så får man tilbakemelding om at *Test Passed*, men i det man velger å laste inn siden på nytt eller logge ut og inn igjen, så er ikke den nye verdien lagret. Under er feilmeldingen som dukker opp etter ny innlogging.

Command	Target	Value
click	id=bruker	
type	id=bruker	Admin
click	id=passord	
type	id=passord	Admin
click	id=loggInn	
open	http://localhost:8080/adminKunde.html	
click	link=Endre konto	
open	http://localhost:8080/adminEndreKonto.html	
AssertValue	xpath=//div[@id='endreKonto']/table/tbody/tr/td[4]	USD

+ Add new row (Cmd/Ctrl + I)

Log Screenshots Variables Reference Self-healing

[info] Executing: | assertValue | xpath=//div[@id='endreKonto']/table/tbody/tr/td[4] | USD |


[error] Cannot read properties of undefined (reading 'trim')

[info] Pausing

[info] Time: Fri Mar 15 2024 12:56:18 GMT+0100 (Central European Standard Time) Timestamp: 1710503778219

[info] Test case failed

[info] Stop executing



Slette konto og kunde

Når slett knappen er trykket så kan man se at kontoen blir fjernet fra nettsiden. Derimot får man en feilmelding når Katalon gjør testen. Vi har en mistanke om at det skjer en feil når siden gir en pop-up-advarsel hvor man bekrefter at man ønsker å slette konto. Det er mulig at Katalon Recorder ikke har innebygde verktøy for å registrere denne handlingen.

localhost:8080 says
Slett kunde med kontonummer 105010123456

OKCancel

Type

Valuta

00LønnskontoNOK

Command	Target	Value
open	/admin/EndreKonto/API/initDB.html	
pause	1000	
chooseOkOnNextConfirmation		
open	http://localhost:8080/admin/EndreKonto.html	
click	link=Slett	
assertConfirmation	Slett kunde med kontonummer 105010123456	
assertValue	id=kontonummer1	null

LogScreenshotsVariablesReferenceSelf-healing

[info] Executing: | assertConfirmation | Slett kunde med kontonummer 105010123456 | |
[info] Executing: | assertConfirmation | Slett kunde med kontonummer 105010123456 | |
[error] Confirmation message doesn't match actual message
[info] Time: Fri Mar 15 2024 11:28:13 GMT+0100 (Central European Standard Time) Timestamp: 1710498493727
[info] Test case failed
[info] Stop executing

Netbank - Admin - Register kunde - Endre kunde - Register konto - Endre konto - Logg ut

Endre konto

Kontonummer

Balans

Type

Valuta

Personnummer

Balans

Total

Kontonummer

Balans

Type

Valuta

Personnummer

Balans

Total

Slett konto.

Command	Target	Value
chooseOkOnNextConfirmation		
open	/adminKunde/API/initDB.html	
pause	1000	
open	http://localhost:8080/adminKunde.html	
click	link=Slett	
chooseOkOnNextConfirmation		
assertConfirmation	Slett kunde med personnummer 01010110523	

Slett kunde

5.0 Vurdering av produktet og arbeidet vårt

5.1 Produktet

Gjennom testing, får man ikke bare et bilde av hvordan systemet fungerer rent teknisk, men også i perspektivet av brukervennlighet. Nettstedets oppførsel under denne testingen, kan beskrives som en minimalistisk prototype som skal bli et større og mer realisert produkt. Med disse forventningene i bildet, kan vi vurdere hva som kan bli gjort bedre.

Uten å logge inn, kan en gå inn på hvilken som helst av fanene i navigasjonsmenyen. Det går an å skrive inn verdier, men disse blir ikke sendt i og med at session ikke er satt til "Innlogget". Dette testes i "ikkeLoggetInn"-testene under enhetstestene. Et finalisert produkt hadde kanskje da bare vist alternativer til interaksjon om session var "Innlogget".

Det er flere funksjoner som ikke fungerer i front-end. Det ser ut til at javascript til knappen "Registrer betaling" ikke fungerer. POST- og GET-kallene kjøres ikke. Disse fungerer henholdsvis i backend, noe som vi ser på testresultatene i SoapUI. Dette problemet oppstår i front-end når man trykker på "Endre"-knappen i "Endre kunde"-siden mens man er admin. Samme problemstilling oppstår i "Registrer kunde" og "Registrer konto".

Problemene i henhold til standard praksis innenfor objektorientert programmering er også til stede. Attributten "avventer" i klassen "Transaksjon" kunne i stedet for å være en streng, vært av typen boolean. På grunn av at den eneste bruken til denne attributten er at den er "0" eller "1".

Under systemtesting når det skulle slettes en konto, så ser det ut til at konto blir slettet på nettsiden fordi den forsvinner, men i Katalon dukker det opp en feilmelding. Hvis man kjører testen i Katalon en gang til, så blir enda en konto slettet, men denne gangen har

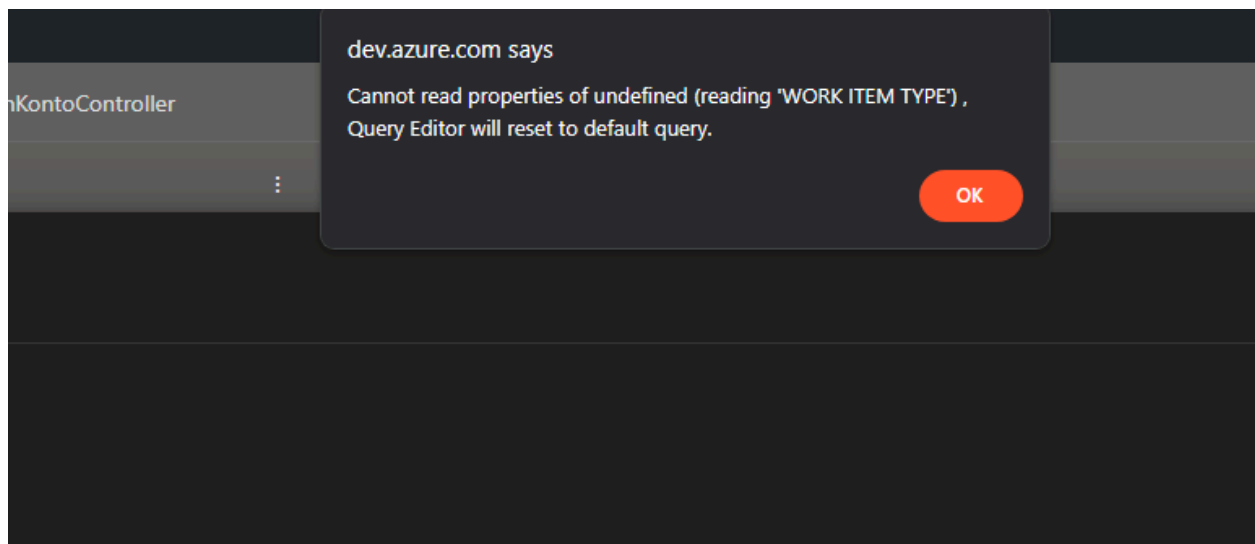
den et annet kontonummer enn det vi spesifiserte. Dette kan være en feil med testen i Katalon, men hvis ikke så er dette potensielt noe som bør endres i et eventuelt sluttprodukt.

5.2 Arbeidet vårt

Når vi reflekterer over arbeidet vårt, så har vi fått en felles forståelse for hva testing innebærer. Alle programmene, utenom IntelliJ var helt nye for oss, så det var læringspotensiale fra starten av. Gjennom jevnt arbeid på tvers av gruppen på alle programmene og testtypene, har vi fått mye ut av dette prosjektet.

Vi slet mye med Azure, fordi vi følte at det var en applikasjon med for mange funksjoner som vi ikke skulle bruke. Dette gjorde det vanskelig å navigere og effektivt dokumentere testene. Det har nok mye med at Azure er et program designet for større prosjekter og teams. Vi brukte mest sannsynlig bare en brøkdel av Azures mange funksjoner.

Azure var også veldig buggy for oss når vi sammen skulle lage user stories. Mye arbeid måtte repeteres i og med at mye ble borte. Alt annet enn brukerhistorien “endre hvert felt under konto som admin” i AdminKontoController ble gjenopprettet. Derfor er ikke denne user storien inkludert i Azure-utskriftet.



6.0 Konklusjon

Avvikene som ble oppdaget var totalt syv stykker, en under integrasjonstesting og seks stykker under systemtestingen.

Under enhetstesten dukker det opp Mockito-feilmelding som er uunngåelig med repository strukturen som er på plass i applikasjonen. Det er også en feilmelding når det kommer til tester som “hentSaldi_terskel”, men denne kommer kun opp på PC-er med mindre minne installert.

I integrasjonstesten så er det ikke mulig å endre kontoinformasjon ved innlogging som admin, dette kan være fordi det ikke er noe personnummer registrert i session ved admin innlogging, som skaper feilmeldingen “feil i personnummer”.

I systemtestingen fungerer det ikke å registrere betaling eller å registrere ny kunde på front-end siden, men det fungerer i backend. Her antar vi at feilen ligger i POST og GET-kallene. Når man skal endre enten konto eller kundeinformasjon som admin så passere testene i Katalon først, men hvis man gjør en test hvor man logger ut, så har ikke de nye verdiene blitt lagret. Når man skal slette enten konto eller kunde, så får man opp samme feilmelding når assertConfirmation skal utføres. Dette kan være på grunn av pop-up vinduet som dukker opp og at Katalon ikke har mulighet til å registrere dette.

Under dette prosjektet, har vi påtatt oss rollen som uavhengig testere; testere som ikke har vært med i selve utviklingsprosessen av koden. Med disse omstendighetene i bildet, har vi lært oss å stille oss kritisk til all kode og lese mellom linjene gjennom testing. Våre øyne eller IntelliJs statiske testing kan ikke avdekke alt, derfor er det viktig med verktøy som kan hjelpe oss.

7.0 Vedlegg

7.1 Vedlegg A - Enhetstester

EnhetstestAdminKontoKontroller:

```
package oslomet.testing;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.Mockito;
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.Arrays;
import java.util.List;

import static org.junit.Assert.assertNotEquals;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.anyString;

@RunWith(MockitoJUnitRunner.class)
public class EnhetstestAdminKontoController {

    @InjectMocks
    private AdminKontoController kontoController;

    @Mock
    private AdminRepository repository;

    @Mock
```

```

private Sikkerhet sjekk;

// hent alle // ---TEST GODKJENT-----
@Test
public void hentAlleKonti_loggetInn() {
    // ARRANGE:
    // lage en list med kontoer
    List<Konto> kontoer = Hjelp.kontoGenerator(3);

    // setter opp mock når sjekk.LoggetInn() blir kalt
    Mockito.when(sjekk.loggetInn()).thenReturn("Innlogget");

    //Setter opp et mock når aRepository.hentAlleKonti() blir kalt
    Mockito.when(repository.hentAlleKonti()).thenReturn(kontoer);

    // ACT:
    //henter faktiske kontoer fra kontocontroller
    List<Konto> resultat = kontoController.hentAlleKonti();

    // ASSERT:
    // sammenligner resultat og ser om vi får tilbake det vi har sendt
    assertEquals(kontoer, resultat);
}

@Test
public void hentAlleKonti_ikkeLoggetInn() {
    // ARRANGE:
    // lage en list med kontoer
    List<Konto> kontoer = Arrays.asList(new Konto(), new Konto(), new
Konto());

    // setter opp mock når sjekk.LoggetInn() blir kalt
    Mockito.when(sjekk.loggetInn()).thenReturn(null);

    //Setter opp et mock når aRepository.hentAlleKonti() blir kalt
    Mockito.when(repository.hentAlleKonti()).thenReturn(kontoer);

    // ACT:

```

```

//henter faktiske kontoer fra kontocontroller
List<Konto> resultat = kontoController.hentAlleKonti();

// ASSERT:
// sammenligner resultat og ser om vi får tilbake det vi har sendt
assertNull(resultat);
}

//registrer    // ----TEST GODKJENT----
@Test
public void registrerKontoTest() {
    // ARRANGE:
    // Lager ny konto
    Konto konto = new Konto();

    Mockito.when(sjekk.loggetInn()).thenReturn("Innlogget");
    Mockito.when(repository.registrerKonto(konto)).thenReturn("OK");

    // ACT:
    String resultat = kontoController.registrerKonto(konto);

    // ARRANGE:
    assertEquals("OK", resultat);
}

@Test
public void registrerKontoTest_ikkeLoggetInn() {
    // ARRANGE:
    // lager ny konto
    Konto konto = new Konto();
    Mockito.when(sjekk.loggetInn()).thenReturn(null);
    Mockito.when(repository.registrerKonto(konto)).thenReturn("OK");

    // ACT:
    String resultat = kontoController.registrerKonto(konto);

    // ASSERT:

```

```

        assertEquals("Ikke innlogget", resultat);
    }

//endre // ----TEST GODKJENT----
@Test
public void endreKontoTest() {
    // ARRANGE:
    Konto konto = new Konto();

    Mockito.when(sjekk.loggetInn()).thenReturn("Innlogget");
    Mockito.when(repository.endreKonto(konto)).thenReturn("OK");

    // ACT:
    String resultat = kontoController.endreKonto(konto);

    // ASSERT:
    assertEquals("OK", resultat);
}

@Test
public void endreKontoTest_ikkeLoggetInn() {
    // ARRANGE:
    Konto konto = new Konto();

    Mockito.when(sjekk.loggetInn()).thenReturn(null);
    Mockito.when(repository.endreKonto(konto)).thenReturn("OK");

    // ACT:
    String resultat = kontoController.endreKonto(konto);

    // ASSERT:
    assertEquals("Ikke innlogget", resultat);
}

//slett // ---TEST GODKJENT-----
@Test
public void slettKontoTest() {

```

```

// ARRANGE:
String kontonummer = "12345678901";

Mockito.when(sjekk.loggetInn()).thenReturn("Innlogget");
Mockito.when(repository.slettKonto(kontonummer)).thenReturn("Slettet");

// ACT:
String resultat = kontoController.slettKonto(kontonummer);

// ARRANGE:
assertEquals("Slettet", resultat);
}

@Test
public void slettKontoTest_ikkeLoggetInn(){
    // ARRANGE:
    String kontonummer = "12345678901";

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

    // ACT:
    String resultat = kontoController.slettKonto(kontonummer);

    // ASSERT:
    assertEquals("Ikke innlogget", resultat);
}
}

```

EnhetstestAdminKundeController:

```

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.*;
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 lagreKunde_loggetInn() {
        // ARRANGE:
        List<Kunde> kunder = new ArrayList<>();

        when(sjekk.loggetInn()).thenReturn("Innlogget");
        when(repository.hentAlleKunder()).thenReturn(kunder);

        // ACT:
        List<Kunde> ekteKunder = adminKundeController.hentAlle();
    }
}

```

```

        // ASSERT:
        assertEquals(kunder, ekteKunder);
    }

    @Test
    public void lagreKunde_ikkeLoggetInn() {
        // ARRANGE:
        List<Kunde> kunder = new ArrayList<>();

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

        when(repository.hentAlleKunder()).thenReturn(kunder);

        // ACT:
        List<Kunde> resultat = adminKundeController.hentAlle();

        // ASSERT:
        assertNull(resultat);
    }

    @Test
    public void hentAlle_loggetInn(){
        // ARRANGE:
        List<Kunde> kunder = new ArrayList<>();

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

        when(repository.hentAlleKunder()).thenReturn(kunder);

        // ACT:
        List<Kunde> resultat = adminKundeController.hentAlle();

        // ASSERT:
        assertEquals(kunder, resultat);
    }

    @Test
    public void hentAlle_ikkeLoggetInn(){

```

```

// ARRANGE:
List<Kunde> forventedeKunder = new ArrayList<>();

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

when(repository.hentAlleKunder()).thenReturn(forventedeKunder);

// ACT:
List<Kunde> resultat = adminKundeController.hentAlle();

// ASSERT:
assertNull(resultat);
}

// Terskeltest med 2 000 000 kunder:
@Test
public void hentAlle_terskel() {
    // ARRANGE:
    List<Kunde> forventedeKunder = Hjelp.kundeGenerator(2_000_000);

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

    when(repository.hentAlleKunder()).thenReturn(forventedeKunder);

    // ACT:
    List<Kunde> ekteKunder = adminKundeController.hentAlle();

    // ASSERT:
    assertEquals(forventedeKunder, ekteKunder);
}

@Test
public void endre_loggetInn() {
    // ARRANGE:
    Kunde kunde = new Kunde();

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

```

```

        when(repository.endreKundeInfo(kunde)).thenReturn("OK");

        // ACT:
        String resultat = adminKundeController.endre(kunde);

        // ASSERT:
        assertEquals("OK", resultat);
    }

    @Test
    public void endre_ikkeLoggetInn() {
        // ARRANGE:
        Kunde kunde = new Kunde();

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

        when(repository.endreKundeInfo(kunde)).thenReturn("OK");

        // ACT:
        String resultat = adminKundeController.endre(kunde);

        // ASSERT:
        assertEquals("Ikke logget inn", resultat);
    }

    @Test
    public void slett_loggetInn() {
        // ARRANGE:
        String personnummer = "123456789";

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

        when(repository.slettKunde(personnummer)).thenReturn("OK");

        // ACT:
        String resultat = adminKundeController.slett(personnummer);

        // ASSERT:

```

```

        assertEquals("OK", resultat);
    }

    @Test
    public void slett_ikkeLoggetInn() {
        // ARRANGE:
        String personnummer = "123456789";

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

        when(repository.slettKunde(personnummer)).thenReturn("OK");

        // ACT:
        String resultat = adminKundeController.slett(personnummer);

        // ASSERT:
        assertEquals("Ikke logget inn", resultat);
    }
}

```

EnhetstestBankController:

```

package oslomet.testing;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.InjectMocks;
import org.mockito.*;
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.*;

```

```

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.*;
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 hentTransaksjoner_loggetInn() {
        // ARRANGE:
        List<Transaksjon> transaksjoner = Hjelp.transaksjonsGenerator(10);
        Konto konto = new Konto("1234567890", "1234567890", 1234, "Lønnskonto",
"NOK", transaksjoner);

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

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

        // ACT:
        Konto resultat = bankController.hentTransaksjoner("1234567890", "", "");

        // ASSERT:
        assertNotNull(resultat);
        assertEquals(konto, resultat);
    }
}

```

```

}

@Test
public void hentTransaksjoner_ikkeLoggetInn() {
    // ARRANGE:
    List<Transaksjon> transaksjoner = Hjelp.transaksjonsGenerator(10);
    Konto konto = new Konto("1234567890", "1234567890", 1234, "Lønnskonto",
"NOK", transaksjoner);

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

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

    // ACT:
    Konto resultat = bankController.hentTransaksjoner("1234567890", "", "");

    // ASSERT:
    assertNull(resultat);
}

// Terskeltest på 10 000 transaksjoner på en konto
@Test
public void hentTransaksjoner_terskel() {
    // ARRANGE:
    List<Transaksjon> transaksjoner = Hjelp.transaksjonsGenerator(10_000);
    Konto konto = new Konto("1234567890", "1234567890", 1234, "Lønnskonto",
"NOK", transaksjoner);

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

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

    // ACT:
    Konto resultat = bankController.hentTransaksjoner("123456789", "", "");

    // ASSERT:

```

```

        assertNotNull(resultat);
        assertEquals(konto, resultat);
    }

    @Test
    public void hentKonti_loggetInn() {
        // ARRANGE:
        List<Konto> konti = Hjelp.kontoGenerator(10);

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

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

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

        // ASSERT:
        assertNotNull(resultat);
        assertEquals(konti, resultat);
    }

    @Test
    public void hentKonti_ikkeLoggetInn() {
        // ARRANGE:
        when(sjekk.loggetInn()).thenReturn(null);

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

        // ASSERT:
        assertNull(resultat);
    }

    // Terskeltest på 900 000 kontoer
    @Test
    public void hentKonti_terskel() {
        // ARRANGE:
        List<Konto> konti = Hjelp.kontoGenerator(900_000);
    }

```



```

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

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

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

        // ASSERT:
        assertNotNull(resultat);
        assertEquals(konti, resultat);
    }

    @Test
    public void hentSaldi_loggetInn() {
        // ARRANGE:
        List<Konto> kontoer = new ArrayList<>();
        kontoer.add(new Konto("1234567890", "1234567890",
                               1000, "Lønnskonto", "NOK", Hjelp.transaksjonsGenerator(1)));

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

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

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

        // ASSERT:
        assertNotNull(resultat);
        assertEquals(kontoer, resultat);
    }

    @Test
    public void hentSaldi_ikkeLoggetInn() {
        // ARRANGE:
        List<Konto> kontoer = new ArrayList<>();
        kontoer.add(new Konto("1234567890", "1234567890", 1000, "Lønnskonto",
                               "NOK", Hjelp.transaksjonsGenerator(1)));

```

```

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

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

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

        // ASSERT:
        assertNull(resultat);
    }

    // Terskeltest med 100 000 kontoer med samme personnummer;
    @Test
    public void hentSaldi_terskel() {
        // ARRANGE:
        List<Konto> kontoer = Hjelp.kontoGenerator(100_000, "1234567890");

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

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

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

        // ASSERT:
        assertNotNull(resultat);
        assertEquals(kontoer, resultat);
    }

    @Test
    public void registrerBetaling_loggetInn() {
        // ARRANGE:
        Transaksjon transaksjon = new Transaksjon();

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

```

```

when(repository.registrerBetaling(any(Transaksjon.class))).thenReturn("OK");

// ACT:
String resultat = bankController.registrerBetaling(transaksjon);

// ASSERT:
assertNotNull(resultat);
assertEquals("OK", resultat);
}

@Test
public void registrerBetaling_ikkeLoggetInn() {
    // ARRANGE:
    Transaksjon transaksjon = new Transaksjon();

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

when(repository.registrerBetaling(any(Transaksjon.class))).thenReturn("OK");

// ACT:
String resultat = bankController.registrerBetaling(transaksjon);

// ARRANGE:
assertNull(resultat);
}

@Test
public void hentBetaling_loggetInn() {
    // ARRANGE:
    List<Transaksjon> transaksjoner = Hjelp.transaksjonsGenerator(10);

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

    when(repository.hentBetaling(anyString())).thenReturn(transaksjoner);

// ACT:

```

```

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

        // ASSERT:
        assertNotNull(resultat);
        assertEquals(transaksjoner, resultat);
    }

    @Test
    public void hentBetalinger_ikkeLoggetInn() {
        // ARRANGE:
        List<Transaksjon> transaksjoner = Hjelp.transaksjonsGenerator(10);

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

        when(repository.hentBetalinger(anyString())).thenReturn(transaksjoner);

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

        // ASSERT:
        assertNull(resultat);
    }

    // Terskeltest med 4 000 000 transaksjoner:
    @Test
    public void hentBetalinger_terskel() {
        // ARRANGE:
        List<Transaksjon> transaksjoner =
Hjelp.transaksjonsGenerator(4_000_000);

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

        when(repository.hentBetalinger(anyString())).thenReturn(transaksjoner);

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

        // ASSERT:

```

```

        assertNotNull(resultat);
        assertEquals(transaksjoner, resultat);
    }

    @Test
    public void utforBetaling_loggetInn() {
        // ARRANGE:
        List<Transaksjon> transaksjoner = Hjelp.transaksjonsGenerator(10);
        transaksjoner.get(0).setAvventer("0");

        when(sjekk.loggetInn()).thenReturn("1234567890");
        when(repository.utforBetaling(10)).thenReturn("OK");
        when(repository.hentBetaling(anyString())).thenReturn(transaksjoner);

        // ACT:
        List<Transaksjon> resultat = bankController.utforBetaling(10);

        // ASSRT:
        assertNotNull(resultat);
        assertEquals(transaksjoner, resultat);
    }

    @Test
    public void utforBetaling_ikkeLoggetInn() {
        // ARRANGE:
        List<Transaksjon> transaksjoner = Hjelp.transaksjonsGenerator(10);
        transaksjoner.get(0).setAvventer("0");

        when(sjekk.loggetInn()).thenReturn(null);
        when(repository.utforBetaling(10)).thenReturn("OK");
        when(repository.hentBetaling(anyString())).thenReturn(transaksjoner);

        // ACT:
        List<Transaksjon> resultat = bankController.utforBetaling(10);

        // ASSERT:
        assertNull(resultat);
    }
}

```

```

// Terskeltest på 4000 transaksjoner:
@Test
public void utforBetaling_terskel() {
    // ARRANGE:
    List<Transaksjon> transaksjoner = Hjelp.transaksjonsGenerator(4000);
    transaksjoner.get(0).setAvventer("0");

    when(sjekk.loggetInn()).thenReturn("1234567890");
    when(repository.utforBetaling(10)).thenReturn("OK");
    when(repository.hentBetaling(anyString())).thenReturn(transaksjoner);

    // ACT:
    List<Transaksjon> resultat = bankController.utforBetaling(10);

    // ASSERT:
    assertNotNull(resultat);
    assertEquals(transaksjoner, resultat);
}

@Test
public void hentKundeInfo_loggetInn() {
    // ARRANGE:
    Kunde enKunde = new Kunde("01010110523",
        "Lene", "Jensen", "Askerveien 22", "3270",
        "Asker", "22224444", "HeiHei");

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

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

    // ACT:
    Kunde resultat = bankController.hentKundeInfo();

    // ASSERT:
    assertNotNull(resultat);
    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 endre_loggetInn() {
    // ARRANGE:
    Kunde kunde = new Kunde("0987654321",
        "Lene", "Jensen", "Askerveien 22", null,
        "Asker", "22224444", "HeiHei");

    when(sjekk.loggetInn()).thenReturn("1234567890");

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

    // ACT:
    String resultat = bankController.endre(kunde);

    // ASSERT:
    assertNotNull(resultat);
    assertEquals("OK", resultat);
}

@Test
public void endre_ikkeLoggetInn() {
    // ARRANGE:
    Kunde kunde = new Kunde("0987654321",
        "Lene", "Jensen", "Askerveien 22", null,

```

```

        "Asker", "22224444", "HeiHei");

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

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

    // ACT:
    String resultat = bankController.endre(kunde);

    // ASSERT:
    assertNull(resultat);
}
}

```

EnhetstestSikkerhetController:

```

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.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNull;

```



```

import static org.mockito.ArgumentMatchers.anyString;
import static org.mockito.Mockito.*;

@RunWith(MockitoJUnitRunner.class)
public class EnhetstestSikkerhetController {

    @InjectMocks
    private Sikkerhet sikkerhetsController;

    @Mock
    private BankRepository repository;

    @Mock
    MockHttpSession session;

    @Before
    // Nødvendig for å sette en session-attributt før kallet til controlleren.
    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 testSjekkLoggInn() {
        // ARRANGE:
        when(repository.sjekkLoggInn(anyString(),
anyString())).thenReturn("OK");

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

        // ASSERT:
        assertEquals("OK", resultat);
        verify(session).setAttribute("Innlogget", "12345678901");
    }

    @Test
    public void testSjekkLoggInn_altFeil() {
        // ARRANGE:
        when(repository.sjekkLoggInn(anyString(),
anyString())).thenReturn("Feil");

        // ACT:
        // Feil passord:
        String resultat = sikkerhetsController.sjekkLoggInn("12345678901",
"NeiNei");

        // ASSERT:
        assertEquals("Feil i personnummer eller passord", resultat);
    }

    // Feil passord i følge regex:
    @Test
    public void testSjekkLoggInn_feilPassord() {
        // ARRANGE:

```

```

        when(repository.sjekkLoggInn(anyString(),
anyString())) .thenReturn("Feil");

        // ACT:
        // Feil passord, er under 6 tegn:
        String resultat = sikkerhetsController.sjekkLoggInn("12345678901",
"Hei");

        // ASSERT:
        assertEquals("Feil i passord", resultat);
    }

    @Test
    public void testSjekkLoggInn_feilPersonnummer() {
        // ARRANGE:
        when(repository.sjekkLoggInn(anyString(),
anyString())) .thenReturn("Feil");

        // ACT:
        // Feil personnummer i følge regex, er under 6 tegn:
        String resultat = sikkerhetsController.sjekkLoggInn("123", "HeiHei");

        // ASSERT:
        assertEquals("Feil i personnummer", resultat);
    }

    @Test
    public void testInnlogget() {
        // ARRANGE:
        session.setAttribute("Innlogget", "12345678901");

        // ACT:
        String resultat = sikkerhetsController.loggetInn();

        // ASSERT:
        assertEquals("12345678901", resultat);
        verify(session).setAttribute("Innlogget", "12345678901");
    }

```

```

    }

    @Test
    public void testInnlogget_altFeil() {
        // ARRANGE:
        when(repository.sjekkLoggInn(anyString(),
anyString())) .thenReturn("Feil");

        // ACT:
        String resultat = sikkerhetsController.loggetInn();

        // ASSERT:
        assertNull(resultat);
    }

    @Test
    public void testLoggInnAdmin() {
        // ARRANGE:
        session.setAttribute("Innlogget", "Admin");

        // ACT:
        String resultat = sikkerhetsController.loggetInn();

        // ASSERT:
        assertEquals("Admin", resultat);
        verify(session).setAttribute("Innlogget", "Admin");
    }

    @Test
    public void testLoggUt() {
        sikkerhetsController.loggUt();

        // Sjekker om session er blitt endret:
        verify(session).setAttribute("Innlogget", null);
    }
}

```

Hjelpemetoder til testene:

```
package oslomet.testing;

import oslomet.testing.Models.Konto;
import oslomet.testing.Models.Kunde;
import oslomet.testing.Models.Transaksjon;

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

public class Hjelp {

    // HJELPE-METODER TIL Å FULLFØRE TESTER
    public static List<Konto> kontoGenerator(long antall) {
        List<Konto> kontoer = new ArrayList<>();

        for (int i = 0; i < antall; i++) {
            kontoer.add(new Konto(tilfeldigString(11), tilfeldigString(11),
tilfeldigDouble(5), "Lønnskonto", "NOK", transaksjonsGenerator(10)));
        }

        return kontoer;
    }

    //Lager n antall kontoer til samme person
    public static List<Konto> kontoGenerator(long antall, String personnummer)
{
        List<Konto> kontoer = new ArrayList<>();

        for (int i = 0; i < antall; i++) {
            kontoer.add(new Konto(personnummer, tilfeldigString(11),
tilfeldigDouble(5), "Lønnskonto", "NOK", transaksjonsGenerator(10)));
        }
    }
}
```

```

        return kontoer;
    }

    public static List<Kunde> kundeGenerator(long antall)    {
        List<Kunde> kunder = new ArrayList<>();

        for (int i = 0; i < antall; i++)    {
            kunder.add(new Kunde(tilfeldigString(2), tilfeldigString(7),
tilfeldigString(7), tilfeldigString(12), tilfeldigString(4),
tilfeldigString(6), tilfeldigString(8), tilfeldigString(20)));
        }

        return kunder;
    }

    // HJELPE-METODER TIL Å FULLFØRE TESTER
    public static List<Transaksjon> transaksjonsGenerator(int antall)    {
        List<Transaksjon> transaksjoner = new ArrayList<>();

        // dato er "" fordi repository-metoden endrer det for oss
        for (int i = 0; i < antall; i++)    {
            transaksjoner.add(new Transaksjon(tilfeldigInt(5),
tilfeldigString(11), 100, "", tilfeldigString(50), tilfeldigString(5),
tilfeldigString(11)));
        }

        return transaksjoner;
    }

    public static List<Transaksjon> transaksjonsGenerator(int antall, String
kontonr)    {
        List<Transaksjon> transaksjoner = new ArrayList<>();

        // dato er "" fordi repository-metoden endrer det for oss
        for (int i = 0; i < antall; i++)    {
            transaksjoner.add(new Transaksjon(10, tilfeldigString(11),
tilfeldigDouble(3), "", tilfeldigString(10), "1", kontonr));
        }
    }

```

```

    }

    return transaksjoner;
}

private static String tilfeldigString(int n) {
    String tegn = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" + "0123456789"
    + "abcdefghijklmnopqrstuvwxyz";

    StringBuilder s = new StringBuilder(n);
    for (int y = 0; y < n; y++) {
        int index = (int)(tegn.length() * Math.random());
        s.append(tegn.charAt(index));
    }
    return s.toString();
}

private static int tilfeldigInt(int n) {
    String tegn = "0123456789";

    StringBuilder s = new StringBuilder(n);
    for (int y = 0; y < n; y++) {
        int index = (int)(tegn.length() * Math.random());
        s.append(tegn.charAt(index));
    }
    return Integer.parseInt(s.toString());
}

private static double tilfeldigDouble(int n) {
    String tegn = "0123456789";

    StringBuilder s = new StringBuilder(n);
    for (int y = 0; y < n; y++) {
        int index = (int)(tegn.length() * Math.random());
        s.append(tegn.charAt(index));
    }
    return Double.parseDouble(s.toString());
}

```

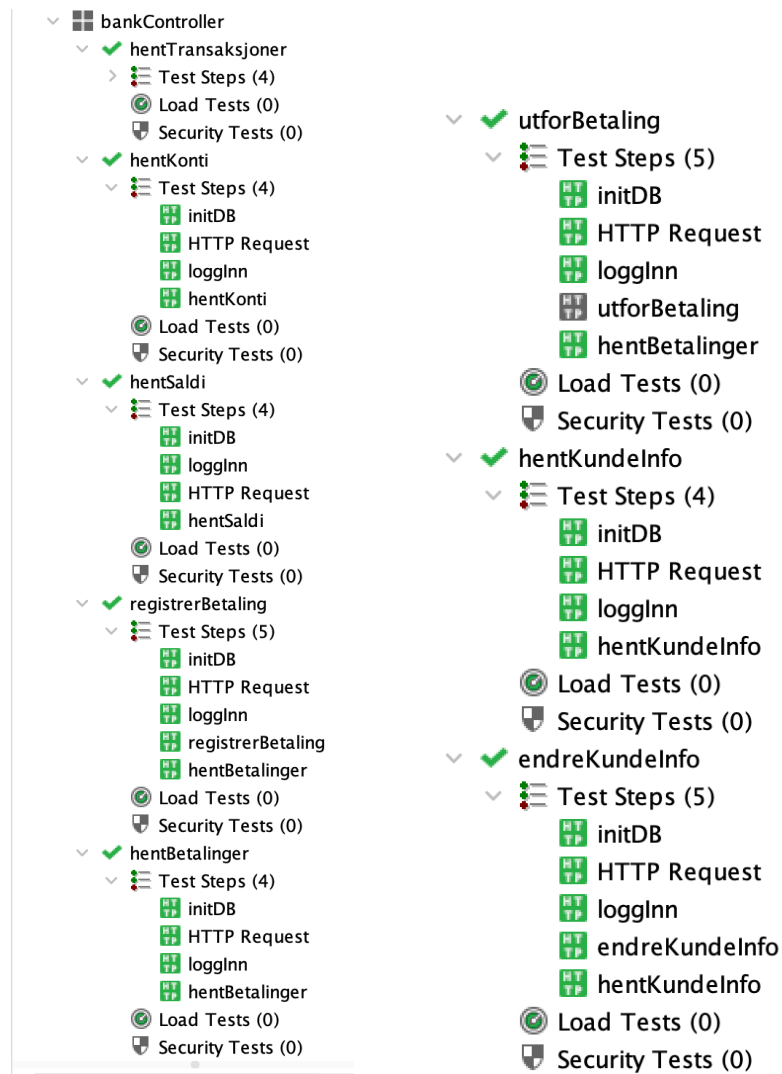
```

}
}


































```

7.2 Vedlegg B - Integrasjonstestresultater







































bankController:



kontoController:
















- ▼  kontoController
 - ▼  registrerKonto
 - ▼  Test Steps (4)
 -  initDB
 -  HTTP Request
 -  AdminLogin
 -  RegistrerKonto
 -  Load Tests (0)
 -  Security Tests (0)
 - ▼  hentKonti
 - ▼  Test Steps (4)
 -  initDB
 -  HTTP Request
 -  Admin logg inn
 -  hentAlleKonto
 -  Load Tests (0)
 -  Security Tests (0)
 - ▼  endreKonto
 - ▼  Test Steps (4)
 -  initDB
 -  HTTP Request
 -  admin logg inn
 -  endreKonto
 -  Load Tests (0)
 -  Security Tests (0)
 - ▼  slettkonto
 - ▼  Test Steps (4)
 -  initDB
 -  HTTP Request
 -  logg inn admin
 -  slettKonto
 -  Load Tests (0)
 -  Security Tests (0)

kundeController:

- ▼  kundeController
 - ▼  slettKunde
 - ▼  Test Steps (4)
 -  initDB
 -  HTTP Request
 -  adminLogginn
 -  slettKunde
 -  Load Tests (0)
 -  Security Tests (0)
 - ▼  endreKunde
 - ▼  Test Steps (4)
 -  initDb
 -  adminLogginn
 -  HTTP Request
 -  endreKunde
 -  Load Tests (0)
 -  Security Tests (0)
 - ▼  hentAlle
 - ▼  Test Steps (4)
 -  initDB
 -  adminLogginn
 -  HTTP Request
 -  hentAlle
 -  Load Tests (0)
 -  Security Tests (0)
 - ▼  lagreKunde
 - ▼  Test Steps (4)
 -  initDB
 -  adminLogginn
 -  HTTP Request
 -  lagreKunde
 -  Load Tests (0)
 -  Security Tests (0)
 - ▼  hentKundeInfo
 - ▼  Test Steps (1)
 -  hentKundeInfo
 -  Load Tests (0)
 -  Security Tests (0)

--

Sikkerhet:

- ▼  Sikkerhet
 - ▼  loggInn
 - ▼  Test Steps (4)
 -  initDB
 -  HTTP Request
 -  loggInn
 -  loggUt
 -  Load Tests (0)
 -  Security Tests (0)
 - ▼  loggInnAdmin
 - ▼  Test Steps (2)
 -  HTTP Request
 -  loggInnAdmin
 -  Load Tests (0)
 -  Security Tests (0)

7.3 Vedlegg C - Google Sheets-ark

ADTS2310 - Testing av Programvare		
Test Results Summary (Auto Calculated)	Number	Percent
Total Test Cases Passed	18	95%
Total Test Cases Failed	1	5%
Total Test Cases Remaining	0	0%
Change Log	Person	Date
Opprettet dokument	Gruppe	01.03.24
Lagt til metoder på integrasjonstester	Therese	01.03.24
Oppdatert "Sikkerhet" på integrasjonstester, 3 nye tester	Andreas	01.03.24
Oppdatert "Sikkerhet" på integrasjonstester, 2 nye tester	Jonas	04.03.24
Oppdatert "KontoController" på integrasjonstester, 4 nye tester	Therese	07.03.24
Oppdatert "KundeController" på integrasjonstester, 4 nye tester	Rayan	07.03.24
Oppdatert "BankController" på integrasjonstester, 2 nye tester	Andreas	07.03.24
Oppdatert "BankController" på integrasjonstester, 1 ny test	Jonas	11.03.24
Oppdatert "BankController" på integrasjonstester, 1 ny test	Jonas	12.03.24
Oppdatert "BankController" på integrasjonstester, 1 ny test	Jonas	13.03.24
Oppdatert "BankController" på integrasjonstester, 1 ny test	Jonas	14.03.24

Test Summary		Instructions				
Total Passed	18	Status needs either "Pass", "Fail" or "Not started".				
Total Failed	1	By writing this under status, the numbers on the left will update automaticly.				
Total Remaining	0					
Class	Endpoint/met hod	Forventet resultat	Status	Dato endret sist	Kommentar(er)	Tester
Sikkerhet	/loggInnAdmin	Logget inn	Pass	01.03.24 12:24		Andreas
	/loggInn	OK	Pass	01.03.24 13:48		Andreas
	/loggUt	NULL	Pass	01.03.24 13:53		Andreas
KontoContr oller	/register	OK	Pass	07.03.24 11:40		Therese
	/hentAlle	Kontoer	Pass	07.03.24 11:40		Therese
	/endre	OK	Fail	07.03.24 11:40	Feil i personnumm er	Therese
	/slett	Slettet	Pass	07.03.24 11:40		Therese
KundeContr oller	/hentAlle	Kunder	Pass	07.03.24 12:34		Rayan
	/lagre	OK	Pass	07.03.24 13:27		Rayan
	/endre	OK	Pass	07.03.24 12:22		Rayan
	/slett	OK	Pass	07.03.24 12:20		Rayan
BankContr oller	/hentTransaksj oner	Antall: 3, txID: 7	Pass	07.03.24 13:04		Andreas
	/hentKonti	Antall: 3, Kontonummer nr1: 105020123456	Pass	07.03.24 13:25		Andreas
	/hentSaldi	JSON-antall: 3, JSON-match	Pass	11.03.24 14:22		Jonas
	/registrerBetali ng	JSON-antall: 4, JSON-match	Pass	12.03.24 18:23		Jonas
	/hentBetaling er	JSON-antall: 3, JSON-match	Pass	13.03.24 03:02		Jonas

	/utforBetaling	JSON-antall: 2, JSON-match	Pass	14.03.24 06:48		Jonas
	/hentKundeInfo	Fornavn: Lene, Etternavn: Jensen	Pass	04.03.24 13:31		Andreas
	/endreKundeinfo	Fornavn: Ryan, Etternavn: Gosling	Pass	04.03.24 12:28		Jonas

7.4 Vedlegg D - Microsoft Test Management (Azure) dokumentasjon