# Test- och slutrapport av Secureme CRM-system

Här sammanfattas resultaten för testningen av programmet Secureme, ett CRM-system med primär funktionalitet mot kundärendehantering. För tydlighetens skull har jag delat upp varje avsnitt i tre underkategorier baserade på vilken typ av testning som är gjord. För samtliga detaljer se kalkylarket Testplanering\_Testfall.xlsx.

## Ej genomförda tester

Funktionalitet som inte testades var Ny användare, Visa alla ärenden, Autogenererat mejl vid nytt ärende samt visa alla användare

## Genomförda tester

### API-tester

Inom API-testningen genomfördes 16 tester, vilket motsvarar programmets samtliga endpoints. Testresultatet av dessa fördelade sig enligt nedan:

* Utan anmärkning: 4
* Med behov av refaktorering/bugg: 4
* Innehållande säkerhetsbrister: 8

Däremot fungerade den grundläggande funktionaliteten i alla API som förväntat. För samtliga API saknas DTO (Data Transfer Object) och i många fall skickas en hel klass med data till frontend. I ett fåtal fall finns ett anpassat objekt i returen. Detaljer om refaktorering och säkerhetsbrister diskuteras under Resultat.

### Enhetstester

Enhetstester genomförs inte i testet, eftersom logiken i backend uteslutande finns i API-endpoints, vilka har till uppgift att serva frontend med data. Logiken är också byggd utifrån att säkerställa att kommunikationen mellan frontend och backend fungerar och genererar olika typer av felmeddelanden i backend samt frontend när grundlogiken inte uppfylls. För att förtydliga detta kan API-endpointen /api/cases lyftas. Här finns det både try-catch för att testa så att metoden kan köras och att databaskopplingen fungerar men också if-satser som kontrollerar så att datan som kommer från frontend inte är tom och att insättningen i databasen fungerar genom att läsa av om det skickas tillbaka ett nytt id. Hade det varit så att programmet hade haft mer ”business logic” som skulle var essentiell för programmet hade det varit mer passande att testa enheter i backend. Ett sådant exempel skulle kunna vara när funktionaliteten Statistik hade implementerats. Däremot hade det varit bra att testa mer i frontend, eftersom där finns mer logik att testa. Men eftersom detta inte har genomförts i samband med kursen har jag inte funnit tid till att sätta mig in i ytterligare ett sätt att genomföra enhetstester.

### Gränssnittstestning

I GUI-testningen genomfördes 8 test, vilket motsvarar funktionaliteterna Nytt ärende, Kundchatt och Kundmedarbetarechatt. Dessa fördelade sig enligt följande:

* Utan anmärkning: 3
* Med behov av refaktorering: 3
* Med bug: 1
* Ej fullständigt testad: 1

På grund av tidsbrist testades inte funktionaliteten Ny användare, vilket ursprungligen låg med i testplanen. Ett generellt problem med gränssnittstestningen var avsaknaden av klassnamn och id, vilket diskuteras vidare under Resultat.

## Resultat

## Identifierade buggar och säkerhetsbrister

Vid testningen saknades restriktioner på API-endpoints, vilket gör att samtliga endpoints går att komma åt utan att vara inloggad. Vid GUI-testning framkom att frontend till stor del saknar classnamn och id. Detta tillsammans med användningen av MUI, som har egna förinställda classnamn och har specialbyggda element gjorde att GUI var svårtestad.

## Rekommendationer och reflektioner

Förväntad förbättring av funktionalitet i programmet efter testningen utifrån kravspecifikationen:

I hela programmet:

* API som sätter ärendehandläggare på ett ärende tillsammans med en interaktion i frontend. Detta måste göras manuellt i databasen för att slå igenom i programmet
* Säkerhetsperspektiv: Sätta fler checkar både i backend och frontend, så att t.ex. en inaktiv medarbetare inte kan logga in. Programmets databas har näst intill inga restriktioner, så det går t.ex. att skapa flera användare med samma uppgifter.
* Ta bort ej använd/fungerande funktionaliet: Funktionaliteten att ta bort en användare finns i både frontend och backend men fungerar inte på grund av databasrestriktioner mellan cases och users där det finns en främmandenyckel. Sedan bör en användare inte kunna tas bort i databasen, förutom vid specifika fall, utan här skulle en soft delete behöva göras.

I databasen:

* skalbarhet till flera företag, eftersom databasen och därmed backend har byggt kategorier baserat på enum, vilka är svårare att anpassa till andra typer av företag.
* Saknad funktionalitet i frontend: Inloggningshantering.

I backend:

* Enhetlighet i namnkonventioner, exempelvis namngivning av klassvariabler i backend.
* Saknad funktionalitet i backend: DTO för att hantera vilken data som skickas till frontend.
* Fel data skickas till frontend när ett ärende skapas: I endpointen där ett nytt ärende returneras id för det insatta meddelandet och inte det insatta ärendet. Detta kan skapa problem, bland annat i samband med att det unika token som kunden ska ha tillgång till för att komma åt sin kundchatt. I samband med detta är just denna endpoint uppbyggd så att den unika token inte kan skickas med i det autogenererade mejlmeddelandet, för de variabler som används läser data från frontend inte från databasen, där ärendets token genereras. Detta behöver alltså refaktoreras.

I frontend:

* Hantera meddelandeavsändare genom session istället för komponenter i frontend. Hur programmet är uppbyggt just nu ger två problem: dels skalbarhet i frontend, eftersom komponenter blir beroende av varandra och ändrar du i en komponent påverkar det den underliggande komponenten i det fallet att en ny komponent ska implementeras. Dels skapar det en säkerhetsrisk i samband med att nå en chatt och vem som skickar ett meddelande.
* Identifierare i frontend: Eftersom frontend till stor del är uppbyggd av Material UI-element som kommer med egna klassnamn i överflöd och till stor del avsaknad av egenkomponerade klassnamn och id.

Lärdomar som jag drar av att testa ett system är flera men primärt att i en funktionalitet tänka ur fler perspektiv, inte bara att något ska fungera utan att det måste finnas ett säkerhetsperspektiv på flera nivåer: Vilken data är tillgänglig och för vem? Hur mycket data behövs för att frontend ska fungera? Vilken data returneras när en API-endpoint är lyckad? Samtidigt behöver programmet planeras mer noggrant när det kommer till att namngivning i både backend och frontend, både ur ett namnkonventionsperspektiv men också för att skapa en enhetlighet genom programmet, oavsett vem som programmerar en funktionalitet. Vidare behöver ett program i större utsträckning testas samtidigt som funktionalitet läggs till för att se till tidigare tillagda funktioner inte påverkas, och i så fall lägga upp en plan för hur detta ska lösas. Det är ganska otroligt vad man kan upptäcka i ett program vid testning som man inte tänkte på i samband med programmeringen av programmet. Det blir stora likheter med hur man ska skriva en bra text: gör texten så bra du kan utifrån de uppsatta premisserna. Sedan låter man texten ligga, för att sedan utifrån premisserna granska texten igen.