**Prosjekt Cicero/Enonic**

(Testing prosess)

<https://cicero.oslo.no/no/om-oss/grafisk-profil>

**Manuelle Tester**

1. Påvirkningsanalyse:

- Identifiser hvilke deler av applikasjonen som er påvirket av endringene i API-ene. Dette inkluderer eksisterende funksjonalitet som var avhengig av de gamle API-ene, samt eventuell ny funksjonalitet som er introdusert.

\* Det finnes prosjekt og dokumentasjon i JIRA/Confluence, trenger tilgang til prosjekt (Sigvard/Tom Arild ?).

\* Teknologi/Verktøy i gamle prosjekt (Enonic Guillotine 5- Using the GraphQL API ):

<https://developer.enonic.com/docs/guillotine/5.x>

Teknologi/Verktøy i den nye prosjekt (Enonic Guillotine 7- GraphQL API for Enonic XP ):

<https://developer.enonic.com/docs/guillotine/stable>

2. Utforskende Tester:

- Utfør utforskende tester i de områdene av applikasjonen som er påvirket av endringene i API-ene. Dette vil tillate deg å oppdage mulige problemer som ikke var dokumentert eller forventet.

\*Eksempel med Guillotine:

a) 5 ***Scaled Image URL***

Et bilde som inneholder tekst, Font, line, skjermbilde

Automatisk generert beskrivelse

b) 7(latest) ***Scaled Image URL***

Et bilde som inneholder tekst, skjermbilde, programvare, Font

Automatisk generert beskrivelse

3. Datavalidering:

- Verifiser at dataene som hentes fra den nye kilden (den nye applikasjonen som håndterer databasene) er korrekte og håndteres riktig i applikasjonen.

4. Dokumentasjon av Testtilfeller:

- Dokumenter detaljerte testtilfeller som inkluderer inndata og forventede utdata for hver av de modifiserte API-ene. Dette vil være nyttig både for manuelle tester og for å opprette automatiserte tester.

**Automatiserte Tester**

1. Automatisering av Regresjonstester:

- Utvikle automatiserte tester for testtilfeller som kjøres regelmessig. Siden API-ene har endret seg, er det avgjørende å ha automatiserte regresjonstester for å oppdage problemer som kan oppstå på grunn av disse endringene.

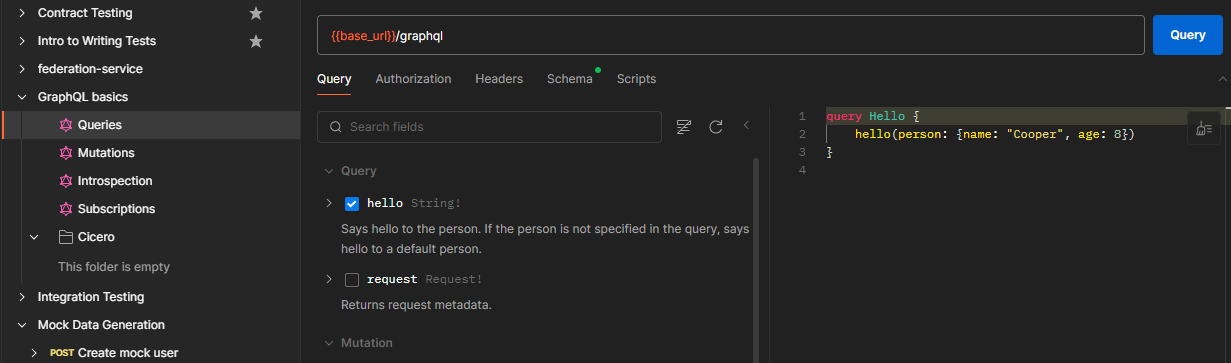
2. Kontinuerlig Integrasjon (CI):

- Gjennomfør en CI-pipeline som automatisk kjører enhetstester og integrasjonstester hver gang det gjøres endringer i koden. Dette sikrer at eventuelle feil som introduseres av endringene i API-ene oppdages raskt.

3. Bruk av API-testverktøy:

- Bruk verktøy som Postman for å automatisere testene av de nye API-ene. Disse verktøyene vil tillate deg å lage tester som validerer at API-ene responderer korrekt og håndterer dataene på en passende måte.

Query:



Test resultat:

Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare

Automatisk generert beskrivelse

4. Ytelsestester:

- Vurder å gjennomføre ytelsestester ved hjelp av verktøy som JMeter eller Gatling for å sikre at de nye API-ene er effektive og ikke introduserer flaskehalser i applikasjonen.

Et bilde som inneholder tekst, skjermbilde, programvare, Multimedieprogramvare

Automatisk generert beskrivelse

**Anbefalte Verktøy**

1. Postman: For automatiserte API-tester og dokumentasjon av tester.

2. Jenkins eller GitLab CI/CD\*\*: For å innpasse de automatiserte testene i en kontinuerlig integrasjonspipeline.

3. Selenium: Hvis applikasjonen har et grafisk grensesnitt, for å automatisere UI-tester som validerer interaksjonen med de nye API-ene.

4. JMeter eller Gatling: For å utføre ytelsestester og sikre at de nye API-ene håndterer arbeidsbelastningen effektivt.

**Andre anbefalinger**

1. Kommunikasjon med Utviklingsteamet: Jobb tett med utviklerne for å forstå nøyaktig hvordan API-ene har endret seg og hva som forventes av dem. Dette vil hjelpe deg med å designe bedre tester.

2. Tester i Preproduksjonsmiljø: Sørg for at alle tester kjøres i et miljø som er så likt produksjonsmiljøet som mulig før endringene settes i produksjon.

Denne kombinasjonen av teknikker vil hjelpe med å dekke både funksjonell og ikke-funksjonell validering av endringene som er gjort i applikasjonen, og sikre at de nye endringene i API-ene ikke påvirker applikasjonens stabilitet og ytelse negativt.

**Noen tester for pilotprosjektet**

A- Test med ChatGPT agent ‘Test Master’ og Enonic-Guillotine:

1)GraphQL Guillotine 5 til 7

***Example: Scaled Image URL****- Retrieve the image contents and generate absolute URLs to these images cropped to 800x200px*

{

guillotine {

query(contentTypes:"media:image") {

displayName

... on media\_Image {

imageUrl(scale:"block(800,200)",type:absolute)

}

}

}

}

2)Fra ‘Test Master’

For å konvertere den nevnte GraphQL-spørringen fra Enonic Guillotine 5 til Guillotine 7, må vi justere strukturen for å være kompatibel med de endrede API-funksjonene i Guillotine 7. Her er hvordan spørringen ser ut i den nye versjonen:

Graphql (Kopier kode)

{

guillotine {

queryDsl(query: {

bool: {

must: [

{

term: {

field: "\_nodeType",

value: "media:image"

} } ] }

}) {

total

hits {

displayName

... on media\_Image {

imageUrl(params: {

scale: "block(800,200)"

type: "absolute"

}) } } } } }

**Forklaring på endringene:**

1. **Bytte fra query(contentTypes: "media:image") til queryDsl:**
   * I Guillotine 7 bruker vi queryDsl for å utføre komplekse søk. I stedet for å bruke contentTypes, bruker vi nå "\_nodeType"-feltet for å filtrere på typen media:image.
2. **Bruke bool og must:**
   * Dette er en fleksibel måte å spesifisere søkekriterier på. I denne spørringen bruker vi must-arrayen for å angi at vi må matche "\_nodeType": "media:image".
3. **Resultatstruktur (hits):**
   * Resultatene fra spørringen returneres nå innenfor en hits-array. Dette gjør det enklere å håndtere flere resultater, og det gir mer informasjon om søket, som det totale antallet treff (total).
4. **Oppdatering av imageUrl-feltet:**
   * I stedet for å sende skalering og type direkte i imageUrl-funksjonen, bruker vi nå params for å spesifisere disse innstillingene. Dette gir en mer strukturert og utvidbar måte å håndtere parametere på.

\*Dokumentasjon ved Enonic.com var annerledes:

***Example: Scaled Image URL****- Retrieve the image contents and generate absolute URLs to these images cropped to 800x200px*

{

guillotine {

queryDsl(query: {

term: {

field: "type",

value: {

string: "media:image"

}

}

}){

displayName

... on media\_Image {

imageUrl(scale:"block(800,200)",type:absolute)

} } } }

Ignacio Tejera

Test Manager