ITP Refleksjon

Tema: 01 Verktøy, sette opp miljøer
Studentnummer: 136664
Gruppenr: 03
Ordtelling: 1464

I denne refleksjonen skal jeg hovedsakelig fokusere på valg av utviklingssted, altså hvor det er man skriver kode. Men jeg skal også litt innom bruken vår av hjelpescripts under utvikling.

Valg av utviklingssted

Vi valgte å ikke utvikle i eclipse che, men istedenfor utvikle lokalt på vår egen maskin. Dette førte til at vi fikk en miks av de vanligste operativsystemene, altså Mac, Windows og Linux (debian). Vi testet appen vår i eclipse che før innleveringer, og litt underveis, for å sikre oss at den kjørte fint der. Dermed endte vi opp med å utvikle appen for fire forskjellige plattformer. Valget å utvikle lokalt hadde både fordeler og ulemper.

Problemer med utviklingssted

Den mest åpenbare ulempen er at vi støtte på en del OS spesifikke problemer som var krevende. Dette er et av hovedproblemene eclipse che prøver å fikse: “Remove inconsistency between developer environments” (Eclipse che, u.å.). For eksempel så får ikke de på Windows pakket programmet med jpackage og jlink grunnet et manglende bibliotek kalt wix, som vi ikke klarte å løse og til slutt ga opp på. Vi brukte en del tid på dette som vi hadde sluppet hvis alle hadde brukt eclipse che siden det fungerte fint der. Det var ikke bare windows det var problemer på.

Vi hadde også en (Thobias) som kjørte Linux (debian), og han fikk en del problemer med å kjøre ui tester. Det var ekstremt frustrerende at ui testene skulle ta over musen når de skulle kjøre, spesielt når de brukte opp mot et minutt. Derfor valgte jeg å kjøre ui testene headlessly, og sånn jeg forstår det så kjører de på en slags VM i bakgrunnen da, men jeg er litt usikker på detaljene i hvordan det fungerte. Dette fungerte ekstremt bra for alle utenom Thobias, som måtte bruke lang tid for å finne ut at han måtte kjøre denne kommandoen `xvfb-run -a <cmd>` hver gang han skulle kjøre ui testene.

Det var også noen små problemer med at alle ikke brukte samme editor. I likhet med de fleste i gruppen bruker eclipse che vscode, som er godt egnet for de fleste bruksområder. Jeg er det eneste unntaket på gruppen siden jeg prøver å lære meg vim. Dette fungerte greit og det oppsto ikke noen problemer helt til vi skulle legge til autoformatering. Vi valgte å bruke `spotless` for å autoformatere koden slik at vi ikke trengte å fikse alle checkstyle warnings manuelt, så jeg la til i pom-en at den skulle

autoformatere på validate steget. Jeg brukte bare vim når jeg la til denne endringen og testet lokalt med terminalen, og alt virket som det skulle. Men det viste seg at vscode, eller en extension, kjører `mvn clean install/verify` hele tiden for å validere koden, som gjør at koden autoformateres mens du endrer på den (var et mareritt). Vi fant til slutt ut av årsaken og fjernet autoformateringen, se denne committen `9609fef` (Knudsen, 2025), men det var en god del sløsing med tid. Samtidig lærte jeg mye av å kunne bruke vim og angrer ikke på dette valget.

Ulemper med eclipse che

Men samtidig så hadde det oppstått noen problemer hvis vi istedenfor hadde brukt eclipse che, for vi har allerede hatt en del tekniske problemer med eclipse che. Det første var adgangsnøkler til github og lignende, og så tok det lang tid før vi fant ut av hvordan vi kunne åpne selve desktopen. Vi fant aldri ut av hvordan man kunne installere extensions som er kritisk for å kunne utvikle med et språk som Java, men det skal sies at vi aldri prøvde så hardt å få dette til. Det hadde vært tidkrevende og litt demotiverende, men alle disse problemene hadde vært løselig hvis vi hadde gått inn for å løse dem.

Det som hadde vært et større problem er de potensielle problemene som kunne oppstå. Et problem kunne ha vært å ønske å installere et lokalt cli-verktøy som `pre-commit` , som jeg testet ut. Sånn jeg forstår det så blir hele containeren som du utvikler i slettet (utenom koden) etter en stund med inaktivitet. Dette kunne ha skapt stor frustrasjon og tidstap siden alt må installeres på nytt hver gang du skal begynne å utvikle, og du må eventuelt manuelt installere cli-verktøy som du ønsker å benytte deg av.

Et annet problem er at man ikke lærer å utvikle på lokal maskin. De fleste i gruppen har ikke tenkt til å bruke eclipse che videre i utvikling, og kommer nok til å utvikle lokalt. Det er derfor bedre for fremtiden å lære seg hvordan maskinen man har fungerer. Noen av oss har også brukt lang tid på å sette opp maskinen vår slik at den er tilpasset vår bruk og store deler av dette er nok ikke mulig i eclipse che

Fremtidig løsning

En fremtidig løsning kan være å utvikle lokalt på maskin, men i en devcontainer. Det eksisterer allerede litt konfigurasjon for dette i repoet `devcontainer.json` (Knudsen, 2025), som stammer fra template repoet. Da

beholder man noen av fordelene med å utvikle på lokal maskin, samtidig som man sikrer en mer uniform utviklingsplattform på tvers av teamet (VSCode, 2025). Se denne siden for mer informasjon (VSCode, 2025).

Hjelpescripts

Jeg lagde python scripts for å automatisere mange maven kommandoer. Grunnen til å velge python er at det fungerer på alle plattformer, og jeg har en del forkunnskap i det. Vi begynte med å bruke bash for scriptsene siden det er lettere med enkle kommandoer, men vi lærte raskt at det ikke fungerte på Windows. En av hovedårsakene til å bruke scripts var å slippe å huske på hvilke flagg man måtte sette for å kjøre spesifikke ting, som for eksempel `-DskipPackage=false` for å aktivere pakking. Det lot oss også kjøre kompilering/testing av modulene i parallell med flagget `-T 1C` i scriptsene, hvor dette flagget allokerer antall CPU-kjerner i tråder (Marwell, 2024). Det kommer noen warnings fra å kjøre byggingen parallelt for noen moduler, men det virker som alt kjører som det skal, og siden det bare er utviklingsscript så tenkte vi det var greit.

Kompleksitet

Det at vi gjemte en del kompleksitet inne i scriptsene førte nok til at flere på gruppen slet litt med å forstå hvordan alt hang sammen. Dette er en vanskelig avveining, for man vil på den ene siden slippe å huske alle flaggene man må sette, men på den andre siden så er det uheldig hvis det hemmer den overordnede forståelsen av strukturen til prosjektet.

Tekniske problemer

Det var noen problemer som kan ha vært forårsaket av scriptsene. Et problem som ofte oppsto var at serveren av og til ikke ble avsluttet helt og fortsatte å lytte på porten, selv om den ikke kjørte i noe synlig vindu. Dette var ekstremt frustrerende for man kunne ikke kjøre en ny instans av serveren siden den eksisterende okkuperte porten. På mac/linux måtte vi da finne id-en til serverprosessen og deretter manuelt drepe det i terminalen. På Windows hadde vi ikke noen god måte å løse det på og restartet hele pc-en som oftest hvis problemet oppsto.

Det kan ha vært scriptsene som har skapt dette problemet siden de kjørte modulene i parallell, noe som ikke var støttet av alle modulene, men det kan

også ha vært noe annet, som en dårlig konfigurering av serveren. Vi fikk ikke tid til å løse dette problemet, men en mulig måte å løse dette på kunne ha vært å teste om å fjerne parallelliteten ville fungert. En annen løsning, og mest sannsynlig den vi hadde brukt, kunne vært å ha lagt til litt cleanup kode i scriptsene for å sikre at serveren ikke kjører lenger, og dette er faktisk implementert i `eclipse_run.sh`.

Fremtidig løsning
De tekniske problemene er overkommelige, selv om de nok vil ta en del tid å fikse. Så det er hovedsakelig kompleksiteten som man må avveie mot praktikaliteten i å slippe å memorisere kommandoer. I vårt prosjekt gikk vi kanskje litt for mye i retning av scripts, og vi burde nok bare hatt dem for ofte brukte sekvenser av kommandoer. Dette vil da muligens bare tilsvare et kjørescript, formateringsscript og verifiseringsscript, så en halvering i antall scripts.

Referanser

Eclipse Che (u. å.) *Introduction to Eclipse Che*. Eclipse che documentation. Hentet 30. november 2025 fra: https://eclipse.dev/che/docs/stable/overview/introduction-to-eclipse-che/#what-is-che

Knudsen, Thobias. Kristiansen, Jonas. Müller-Grud, Jonathan. Nordvik, Henrik. Samset, Erlend. Vårum, Sivert (2025) *GR2503*. Github. Hentet 30. november 2025 fra: https://git.ntnu.no/IT1901-2025-groups/gr2503

Marwell, Benjamin. Rosenvold, Kristian (2024, 6. desember) *Parallel builds in Maven 3*. Apache Confluence. Hentet 30. november 2025 fra: https://cwiki.apache.org/confluence/display/MAVEN/Parallel+builds+in+Maven+3

VSCode (2025, 12. november) *Developing inside a Container*. Visual studio code documentation. Hentet 30. november 2025 fra: https://code.visualstudio.com/docs/devcontainers/containers