NTNU | Norwegian University of Science and Technology

IDATT 2106 – Teknisk krav - Prosjekt

Nytt prosjekt, nye muligheter, med bruk av allerede kjente konsepter og teknologier

Kjente teknologier og konsepter

- Dere har vært gjennom en del forskjellig innen nå
- Mange forskjellige teknologier
- Mange forskjellige teoretiske konsepter
- Mange forskjellige fag
- Samles i en praktisk setting

Spesielt relevante fag

- Ikke alle fag dere har hatt er like relevante for dette prosjektet
- Innholdet av andre fag, er dog høyst relevant:
  - Programmering 1 & 2 (byggesystemer, versjonskontroll, design patterns, unntakshåndtering, funksjonell programmering, persistens)
  - Systemutvikling (MMI/UU, dokumentasjon, prosjektstyring, utvikling av litt større system)
  - Algoritmer og datastrukturer (alt av datastrukturer man kan benytte seg av)
  - Nettverksprogrammering (HTTP, CI/CD, virtualisering, funksjonelle/async-kall, trådprogrammering)
  - Full-stack (frontend/backend, REST, design patterns, testing, arkitektur, profiler, byggesystemer, funksjonelle/async-kall, persistens)
  - Databaser (persistens og gjennfinning av data)

Source Code Management (git)

- Essensiell del av software-utvikling nå til dags
- Litt flere som skal jobbe mot samme repo enn dere vant med
- Noen utfordringer med dette
- Er greit å ha en strategi for hvordan man skal jobbe med SCMet
  - Gitflow vs. noe annet?
  - Rebasing vs. merging
  - Merge requests?
  - Code reviews?
- Yep, dere er nødt til å bruke skolens Gitlab / Github.
- Opprett gruppe med navn på formatet «idatt2106-v24-gruppenr»
  - Ha med navnet på alle gruppemedlemmene på forsiden/landing page på wikien
  - Gi meg, Grethe og Surya «reporter»-tilgang
- Ha ett prosjekt for frontend, og ett prosjekt for backend!

Byggesystemer

- Maven eller gradle for backend
- Husk at tester skal kjøres når en bygger
- Viktig at man til slutt produserer kjørbar JAR-fil
- På frontend er det større valgfrihet
  - Hovedpoenget er å kunne kjøre opp en klient med enkel kommando
  - `> npm run serve`
- Gjør det enkelt for oss å få kjørt opp både klient- og server-side!

Continuous Integration (gitlab/github)

- Automatisk bygging/og testing i Gitlab
- `gitlab-ci.yml`
  - Instruksjoner for hvordan bygge og kjøre tester
  - Har vært borti for klientsiden i nettverksprog
  - Ikke brukt dette for Maven-prosjekter(?)
- Bygger på hver push
  - Bygging er tungt!
  - IKKE push før du er sikker på at testene fungerer lokalt
- Continuous Deployment
  - Ikke en del av dette prosjektet

Eksempel prosjekt (CI / CD)

- `https://gitlab.stud.idi.ntnu.no/alexholt/spring-boot-example`
- Oppsett for vise JUnit-rapporter for hver pipeline (`gitlab-ci.yml`)
- Generering av Jacoco code coverage-rapport (`pom.xml og gitlab-ci.yml`)
- Publisering av Jacoco-rapport til Gitlab Pages (`gitlab-ci.yml`)
- Nedlastbar artefakt (JAR-fil) for Spring Boot-applikasjonen (`gitlab-ci.yml`)
- Greit utgangspunkt for eget oppsett av CI

Testing

- Programmatisk testing er et krav
- Dekningsgrad
  - 50% på backend
  - 30% på frontend
- Opp til dere selv hvordan testingen skal implementeres på backend
  - Unit/integrasjon
  - Mocking/persistente data/opprett base for hvert gjennomløp E2E?
- Må vurderes ut fra som er fordelsmessig for deres prosjekt
- Fortrinnsvis enhetstester på frontend
- Bruk det aktivt under utvikling!

Kode

- Ordnet og enkelt forståelig kode er pri. 1
  - Dokumentasjon til fornuftig grad (Javadoc/REST-endepunkter/kommentarer)
- Konsistens
- Fornuftig bruk av design patterns
  - Ikke overforbruk, men hvor det er hensiktsmessig
- Patterns dere har vært gjennom
  - Lagdelt arkitektur (standard controller/service/repo)
  - DAO/repo
  - Factory/repo
  - Decorator/wrapper
  - Observer/events
  - Singleton

REST

- RESTful backend
- Oversiktlig design (RESTful)
  - Hierarkisk oppbygning av URIer for ressursene
- Korrekt bruk av HTTP-verb
- Statuskoder (og god behandling av disse på klientsiden)
- Dokumentasjon

Sikkerhet

- Ta høyde for XSS og SQL injection (løses stort sett ved bruk av rammeverk)
- Hash passord i DB. Kan bruke:
  - Innebygde mekanismer i database
  - Implementere hashingen selv (bruk av BCrypt eller liknende)

Loggesystemer og Unntakshåndtering

- Logg feil som oppstår
- Skill mellom reelle feil som oppstår og logging for debug-formål
  - Server-siden kan være mye mer finkornet
  - Rudimentær logging om feil/debug-logging bør likevel være på klientsiden
- Husk også å behandle feil som kan oppstå på klientsiden!
  - Kort sagt: `.catch()`

Teknologier oppsummert

- Det forventes at dere benytter allerede kjente hjelpemidler/teknologier
- Noen vil fremtre som fordelsmessig, andre er påkrevd
- Påkrevde:
  - VCS (Git)
    - To distinkte prosjekter for frontend og backend, respektivt.
  - Bruk av byggesystemer som kjører tester og gjør det enkelt å kjøre systemene
  - Kode skal programmatisk testes (50% coverage for backend, 30% for front)
  - CI (men ikke CD)
    - Med testrapporter i Gitlab (se eksempelprosjekt)
  - Klient (web) og server (REST/java)-arkitektur
  - REST
  - Persistens i vanlig SQL-base

Generelle aspekter av prosjektet

- Har produkteiere/kunder
  - Setter kravene for systemet
  - Kan ombestemme seg underveis
  - Ikke anta! Spør kunden!
- Krever at dere finner tekniske løsninger selv
  - Regn med å tilegne dere ny kunnskap i løpet av prosjektet

Samarbeid – Veldig viktig

- Ikke alle trenger å gå like godt overens
- Jobber likevel mot et felles mål
- Noen kan ha ekstremt sterke meninger om hvordan ting skal gjøres
  - Forskjellige personer vektlegger forskjellige ting Ikke gitt at «min måte» er det beste for gruppa som helhet
  - Å kunne lempe på å ikke alltid få det som man vil, er en dyd
  - Man når et punkt hvor konsistens er viktigere enn en optimal løsning
- Grei å ta hensyn til hverandre
- Prat sammen
- Før man begynner med en oppgave, spør gjerne om noen har input/innsikt
  - Det er den kollektive kunnskapen til gruppa man vil utnytte

Ting å tenke på

- Faktisk vær enig om teknologier før en setter i gang
- Enkelte oppgaver er avhengige av at andre, for at de skal kunne ferdigstilles
  - Ikke begynn i feil ende!
- Noen oppgaver kan fint utføres i parallell
  - Tidsbesparende
- Alle har forskjellige kunnskapsområder og -nivåer
  - Ref. prat sammen fra forrige foil
  - Utnytt dette, men ikke overdriv det ved å legge for mye på én person
- Unngå bus factor!
  - Ekstremt vanlig problem rundt omkring i bedrifter

FAQ: kan vi bruk X i stedet for Y?

- Arkitektur er spikret (klient/server/REST)
- Backend har mindre spillerom enn frontend
- Krav for å bruke noe annet enn det vi har lært:
  - *Alle* på gruppa skal være enig om at valg av annen teknologi er OK
- Ved valg av andre teknologier får en ikke drahjelp

Oppstartxxxxxx

- Begynner på mandag, 08:15 (**UKE 16**)
- Grethe har intro-forelesning med mer inngående gjennomføring
- Produkteiere vil introdusere prosjektet/oppgaven
- Fordeles deretter til rom og går i gang med å definere oppgaver/sette opp scrum-board
- Opprett gruppe i (skolens) Gitlab med navn på formatet «idatt2106-v24-gruppenr»
  - Eks: idatt2106-v24-03 eller idatt2106-v24-07 eller idatt2106-v24-10 (kun små bokstaver)
  - Ha med navnet på alle gruppemedlemmene på forsiden/landing page på wikien
  - Gi meg, Grethe og Surya «reporter»-tilgang
- Scrum-master leder standups, men faglærere kan komme innom for å observere.
  - Scrum-master skal velges av gruppa selv, og gruppa må ha dette klart innen prosjektet begynner tirs. 19. april.
  - Standup hver morgen, samme tidspunkt hver dag (kan endres fra én sprint til annen)