Utviklingsplan med Iterasjoner og Prioritert Funksjonalitet

Iterasjon 1 – Grunnleggende Infrastruktur & Domenemodell

- **Mål**: Sett opp grunnleggende infrastruktur (Kotlin, Spring Boot, PostgreSQL) og utvikle domenemodellen for systemet.

- **Aktiviteter**:

    1. Sett opp Spring Boot-prosjekt med nødvendige avhengigheter (Liquibase, PostgreSQL, Spring Data JPA).
    2. Sett opp integrasjon med Flowcase. La en HTTP-klient som håndterer kommunikasjonen med API´et
    3. Lag domeneobjektene som **Consultant**, **Customer**, **ProjectRequest**, **Skill**, etc.
    4. Definer repositoriene (f.eks. `ConsultantRepository`, `ProjectRequestRepository`).
    5. Lag de første tjenestene som håndterer lagring og oppdatering av konsulenter og kundeforespørsler.
    6. Implementer grunnleggende validering og forretningslogikk i service-laget (matchmaking-logikk kan komme i senere iterasjoner).

Iterasjon 2 – AI Integrasjon og Matchmaking

- **Mål**: Integrer AI for å generere forslag til matcher mellom konsulenter og kundeforespørsler.

- **Aktiviteter**:

    1. Implementer AI-basert matching som bruker en enkel scoring-mekanisme for å vurdere hvor godt en konsulent passer til en kundeforespørsel basert på ferdigheter og erfaring.
    2. Bruk en AI-modell som GPT-4 (eller en annen ML-modell) for semantisk matching, og integrer dette i systemet via et REST API.
    3. Bygg en testable tjeneste som kan bruke denne logikken til å generere forslag.
    4. Lag grensesnitt og enkle endepunkter for å hente forslagene for match (for selger/leder).

Iterasjon 3 – Automatisering og Varsler

- **Mål**: Implementer automatiske varsler og fristpåminnelser for kundebehov.

- **Aktiviteter**:

    1. Implementer automatiserte varsler (via e-post eller Slack) som informerer om frister og status på kundeforespørsler.
    2. Bruk Spring Scheduler eller Quartz for å håndtere tidsstyring og varsling.
    3. Legg til støtte for å sende påminnelser for å følge opp prosjekter i god tid før frister.
    4. Test og verifiser at systemet kan håndtere tidsfrister og oppgaver på en effektiv måte.

Iterasjon 4 – Brukergrensesnitt og Admin-Portal

- **Mål**: Bygg et admin-grensesnitt for å administrere konsulenter, kunder, prosjektforespørsler og match.

- **Aktiviteter**:

    1. Bygg en enkel REST API som kan brukes av et frontend-grensesnitt.
    2. Bygg en React eller Vue.js-applikasjon som lar administratorer opprette, oppdatere og administrere alle enheter (konsulenter, kunder, prosjektforespørsler).
    3. Legg til funksjonalitet for å visualisere tilgjengeligheten i konsulentpoolen og få oversikt over matchresultater.
    4. Test og verifiser grensesnittet med brukere.

Iterasjon 5 – Dokumentasjon og Eksempelbruk

- **Mål**: Dokumenter systemet og forbered det for presentasjon.

- **Aktiviteter**:

    1. Skriv grundig dokumentasjon for systemet.
    2. Forbered konkrete brukerhistorier og eksempler på hvordan AI kan hjelpe utviklere med generering av kode og funksjonelle tester (TDD).
    3. Skriv eksempler på hvordan systemet kan utvides med flere AI-funksjoner i fremtiden.
    4. Gjør en siste test av hele systemet for å sikre at det fungerer som forventet.

Forberedelser og øvelse for workshop

Brukerhistoriene kan brukes for å få AI til å generere funksjonell kode med tilhørende tester.

Steg-for-steg øvelse:

1. Pakk ut kodebasen og åpne den i IntelliJ.
2. Opprett en ny `feature/xyz` branch fra `main`.
3. Bruk Git-kommando: `git checkout <hash>` for å hente utgangspunktet før featuren ble sjekket inn.
4. Skriv inn brukerhistorien og akseptansekriteriene i en prompt til AI.
5. La AI generere kode + tester (TDD).
6. Verifiser at det kompilerer og at testen kjører grønt.
7. Refaktorer etter Clean Code-prinsipper og valider at testen fortsatt passerer.
8. Skriv en commit med en klar, meningsfull melding.

Forberedelser:

1. Test dette på forhånd og ha eksempelkode ferdig i én commit. Vær nøye med commit-meldinger
2. For workshopen:
    a. Deltakere sjekker ut koden fra main i en egen branch
    b. Ha en `git checkout <hashcode>` klar, slik at deltakere kan sjekke ut koden slik den så ut før feature i referanse-prosjektet var sjekket inn
    c. Deltakere gjennomfører øvelsen med å få AI til å generere kode som beskrevet i *"Steg-for-steg øvelse"*

Brukerhistorier (eksempler)

🔹 Brukerhistorie 1 – Opprette prosjektforespørsel

- **Som:** en selger
- **Vil jeg:** kunne opprette en prosjektforespørsel fra kunde raskt
- **Slik at:** jeg kan få AI-foreslåtte konsulenter før deadline og levere et kvalifisert team

Akseptansekriterier:

- Det skal være mulig å registrere forespørsel med ønskede ferdigheter, startdato og sluttdato
- Systemet skal validere at frist for svar er før oppstart
- Etter registrering, skal systemet automatisk foreslå minst én relevant konsulent basert på tilgjengelighet og kompetanse
- AI-forslag skal kunne vises med score og begrunnelse

🔹 Brukerhistorie 2 – Oppdatere tilgjengelighet og ferdigheter

- **Som:** en konsulent
- **Vil jeg:** oppdatere min tilgjengelighet og ferdigheter
- **Slik at:** jeg blir matchet med relevante oppdrag

Akseptansekriterier:

- Jeg kan legge inn eller oppdatere informasjon om hvilke perioder jeg er tilgjengelig
- Jeg kan legge til eller redigere mine ferdigheter (f.eks. språk, teknologier)
- Endringer skal påvirke fremtidige matchingsforslag umiddelbart
- Endringer loggføres med tidspunkt

🔹 Brukerhistorie 3 – Oversikt over kapasitet og behov

- **Som:** en leder
- **Vil jeg:** ha oversikt over kommende forespørsler og kapasitet i konsulentpoolen
- **Slik at:** vi kan proaktivt planlegge bemanning og unngå tapte muligheter

Akseptansekriterier:

- Dashboard viser alle aktive og kommende forespørsler med status og frister
- Kapasitetsoversikt viser tilgjengelige konsulenter og deres profil
- Systemet markerer hvilke forespørsler som mangler aktuelle kandidater
- Visningen kan filtreres på tid, kompetanse og kunde

🔹 Brukerhistorie 4 – Automatisk varsling

- **Som:** system
- **Vil jeg:** sende e-postvarsler 48 timer før siste frist for en åpen forespørsel
- **Slik at:** salgsapparatet får varsling i tide og rekker å respondere

Akseptansekriterier:

- Systemet sjekker alle åpne forespørsler med frist innen 48 timer
- Varsel sendes til ansvarlig selger og eventuelt teamleder
- Innhold i e-posten inkluderer forespørselsdetaljer og gjeldende matchstatus
- Varsling skal være planlagt som en periodisk jobb (cron/scheduler)

🔹 Brukerhistorie 5 – Daglig synkronisering med Flowcase

- **Som:** system
- **Vil jeg:** hente ferdigheter og CV-data fra Flowcase daglig
- **Slik at:** matchingsgrunnlaget alltid er oppdatert

Akseptansekriterier:

- Systemet henter oppdatert konsulentinformasjon (skills, erfaring) fra Flowcase-API
- Synkroniseringen skjer som en automatisk daglig jobb
- Endringer oppdateres i lokal database
- Eventuelle feil i integrasjon loggføres og varsles

Plan for Presentasjon/Fagkveld og Workshop

Fagkveld – Struktur og Temaer

1. **Introduksjon til DDD, SOLID og Clean Code**
    - Hva er DDD? Hvordan kan vi bruke DDD til å utvikle et bedre system?
    - SOLID-prinsippene i praksis: Hvordan strukturere koden for bedre vedlikehold og utvidbarhet?
    - Clean Code: Prinsipper og beste praksis for ren og lesbar kode.
2. **Hvordan AI kan hjelpe utviklere**
    - Bruk AI til å analysere brukerhistorier og generere kodeforslag.
    - Generere funksjonelle tester i TDD-stil ved hjelp av AI.
    - Hvordan AI kan veilede utviklere i DDD, SOLID og Clean Code.
3. **Demonstrasjon av systemet**
    - Vis frem det utviklede systemet, fra konsulent- og kundemodell til AI-basert matchmaking.
    - Forklar hvordan systemet følger DDD, SOLID og Clean Code-prinsippene.
4. **Spørsmål og diskusjon**
    - Åpen diskusjon om AI, DDD og hvordan de kan brukes sammen for å forbedre utviklingsprosesser.
    - Diskuter utfordringer og muligheter med å implementere AI i systemutvikling.

Workshop – Struktur og Temaer

1. **Introduksjon og mål for workshop**
    - Presentasjon av de viktigste målene for workshoppen: å utvikle en funksjonell prototype og forstå hvordan AI kan hjelpe utvikleren.
2. **Utvikling i grupper**
    - Del deltakerne inn i grupper og gi dem oppgaver basert på brukshistoriene.
    - Grupper skal bruke AI til å generere kode og funksjonelle tester for et gitt case.
3. **AI-verktøy**
    - Vis hvordan AI kan brukes til å generere forslag, lage kode og gi tilbakemelding på kvaliteten av koden i realtid.
4. **Refleksjon og oppsummering**
    - Hver gruppe presenterer hva de har utviklet.
    - Diskuter hvordan AI har hjulpet, og hvordan man kan implementere AI i sine egne prosjekter.
    - Oppsummering av viktig lærdom fra dagen.

Teknologivalg og Ekstra Verktøy

- **Spring Boot**, **Kotlin**, **PostgreSQL**, **Liquibase** (som nevnt tidligere).
- **AI-teknologi**: OpenAI GPT-4 for naturlig språkbehandling og semantisk matching.
- **Frontend**: React eller Vue.js for admin-portal.
- **Automatisering**: Spring Scheduler eller Quartz for tidsstyring og varsling.
- **Testverktøy**: JUnit, Mockito, Spring Test for TDD og funksjonelle tester.

Markdown (.md) template som AI prompt

Klipp og lim markdown-koden nedenfor og rediger den med din brukerhistorie for å generere funksjonell kode med tilhørende test.

```
# Kontekst
Vi utvikler et internt produkt hos Cloudberries for matching av
IT-konsulenter til kundeforespørsler. Vi bruker Kotlin, Spring Boot
og følger prinsippene i Domain-Driven Design, SOLID og Clean Code.
Data lagres i PostgreSQL og styres med Liquibase.

Kodebasen er lastet opp og gir full oversikt over domenemodeller,
services og repositories. Du skal lage funksjonell kode og en
tilhørende test i TDD-stil basert på følgende brukerhistorie.

# Brukerhistorie
Som: [rolle]
Jeg vil: [mål]
Slik at: [verdi]

## Akseptansekriterier:
- [punkt 1]
- [punkt 2]
- ...

# Oppgave
Generer:
1. Kotlin-kode som implementerer funksjonaliteten i en ny service
eller ved å utvide en eksisterende service.
2. Enhetstest i TDD-stil for funksjonaliteten.
3. Følg prinsippene for DDD (riktig plassering i
domene/service/applikasjonslag), SOLID (f.eks. SRP og Dependency
Inversion), og Clean Code (navngiving, modulerbarhet).
4. Kommentér kort hva som skjer i koden der det trengs.

Svar kun med kodeblokkene og kort forklaring. Ikke skriv hele
domenemodellen på nytt – bruk eksisterende strukturer.

# Teknologi og rammeverk
- Kotlin
- Spring Boot
- Spring Data JPA
```

- Liquibase (database håndteres i separat changelog)
- JUnit 5 og MockK for testing