Rekruttering - Ferdighetstest - Developer

Skill-test: TODO-CRUD (valgbar tech stack)

Formål: Gi deg mulighet til å vise frem din reelle kompetanse i et avgrenset, praksisnært oppsett – uavhengig av om du er sterkest på frontend, backend, fullstack eller native.

Kort om Unimicro: Vi bygger for skyen på **Microsoft Azure**, med **.NET/C#** og **Azure SQL** i kjernen. I frontend bruker vi ulike rammeverk (React/ReactJS, Angular, Vue, SvelteKit). Vi foretrekker strukturerte **REST-APIer** og utforsker **MCP** (Model Context Protocol) i AI-prosjekter.

> **Viktig:** Velg den **tech stacken du mestrer best**. **Du må nødvendigvis ikke velge en av våre teknologier.** Målet er å se hvordan du jobber og prioriterer – ikke å teste hvor raskt du kan lære noe helt nytt.

Oversikt

- **Tema:** TODO-liste (CRUD) med søk/filtrering
- **Leveranse:** Kjørbar løsning + kort README

Oppgave

Bygg en liten TODO-applikasjon som lar brukeren **liste**, **søke/filtrere**, **opprette**, **oppdatere** og **slette** oppgaver.

Funksjonelle minimumskrav (Core)

1. **Listevisning:** Vis alle tasks med feltene minst `title` og `completed`.
2. **Opprett:** Legg til ny task med validering (f.eks. ikke tom tittel, lengdebegrensning).
3. **Oppdater:** Endre tittel og/eller huk av/avhuk `completed`.
4. **Slett:** Fjern en task.
5. **Søk/Filtrering:** Søk i tittel (og gjerne tags) og filtrer på `completed` / `active`.
6. **Persistens:** Velg én av:
   - Backend-lagring (egendefinert API, database eller fil)
   - Lokal lagring (localStorage/SQLite/room/realm etc. for native)
   - Mock-API (json-server eller tilsvarende) – beskriv begrensninger i README

Ikke-funksjonelle krav (Core)

- **Lesbar kode & struktur:** Ryddige moduler/filer, navngiving og enkle abstraheringer.
- **Dokumentasjon:** Kort README (setup, kjøring, antakelser/avgrensninger, hva du ville gjort videre).

Velg spor (ta det som passer profilen din best)

Spor A – Frontend-fokus

- Rammeverk: React, Angular, Vue, SvelteKit **eller** native (Windows/iOS/Android/Flutter/MAUI)

- **Krav i tillegg til Core:**
  - State-håndtering (lokal state er OK) med tydelig dataflyt
  - Brukbarhet: tastaturnavigasjon og ARIA/tilgjengelighets-hensyn på interaktive elementer

Spor B – Backend-fokus

- Teknologi: .NET (Minimal API/ASP.NET), Node/TypeScript (Express/Fastify/Nest), Go, etc.
- **Krav i tillegg til Core:**
  - Rent REST-API for `/tasks` (se kontrakt nedenfor)
  - Input-validering og feilkoder (400/404/422/500)
  - Enkle enhetstester på service-/domene-lag

Spor C – Fullstack Light

- Sett opp en enkel FE som snakker med ditt BE-API (eller motsatt: koble FE til mock-API nå, beskriv videre plan)
- Vis at du kan løse CORS/miljøkonfig og enkel bygningsstruktur mellom lagene

---

REST-kontrakt (referanse – tilpass ved behov)

Ressurs: `Task`

```
Task:
  type: object
  required: [id, title, completed, createdAt]
  properties:
    id: { type: string, description: "UUID eller unik streng" }
    title: { type: string, minLength: 1, maxLength: 140 }
    completed: { type: boolean, default: false }
    dueDate: { type: string, format: date-time, nullable: true }
    tags:
      type: array
      items: { type: string }
    createdAt: { type: string, format: date-time }
    updatedAt: { type: string, format: date-time, nullable: true }

```

Endepunkter:

```
GET    /tasks?query=&completed=(true|false)
POST   /tasks
PUT    /tasks/{id}
DELETE /tasks/{id}

```

- **GET /tasks** returnerer liste, støtter `query` (matcher minst `title`, gjerne `tags`) og `completed`.
- **POST /tasks** validerer `title` og setter default-felt på server (id, createdAt, completed=false).
- **PUT /tasks/{id}`** tillater partiell oppdatering (title/completed/dueDate/tags) – 404 hvis ukjent id.
- **DELETE /tasks/{id}`** returnerer 204 ved suksess – 404 hvis ukjent id.

> Hvis du lager **kun frontend/native**: bruk en **mock-server** (f.eks. json-server) eller lokal lagring. Beskriv valget i README.

Eksempeldata (frivillig)

```
[
```

```
  {"id":"1","title":"Skriv README","completed":false,"tags":["docs"],"cr
  {"id":"2","title":"Implementer POST /tasks","completed":true,"tags":["
  {"id":"3","title":"Legg til søk i tittel/tags","completed":false,"tags
]

```

Stretch-mål (velg maks 2 hvis tid)

- **Sortering/paginering** (server eller klient)
- **Ytelse/observability:** enkel logging/metrics og kort notat om hva du ville overvåket i Azure, eller ligende.
- **Feilhåndtering:** Synlig, brukervennlig håndtering av minst 2 feiltyper (f.eks. valideringsfeil og nettverksfeil).
- **MCP-touch:** Beskriv (eller skissér i kode) hvordan en minimal MCP-server/handler kunne eksponert `tasks` for en AI-agent (ingen full implementasjon kreves)

Vurderingskriterier

Kategori	Vi ser etter
Problemløsning & prioritering	Forstår krav/edge cases, leverer kjerne først
Kodekvalitet & struktur	Lesbarhet, modularisering, navngiving, små commits/PR-vaner
Robusthet	Validering, feilhåndtering, forutsigbare HTTP-/UI-feil
Testing	1–2 meningsfulle tester og begrunnelse
Web/Cloud-fag	API-tenkning, dataflyt, kontrakter (evt. lagdeling i BE)
Kommunikasjon	Tenk-høyt/dokumentasjon, refleksjon i retro/demosamtale

---

Leveransekrav

- **Kode** i repo (GitHub) med **README** som beskriver:
  a. Teknologistack og hvordan starte lokalt
  b. Antakelser/avgrensninger
  c. Hva du ville gjort neste steg for prod (sikkerhet, drift, test, CI/CD)
- **Kjøring:** `npm run dev`, `dotnet run`, eller tilsvarende – lav terskel
- **Valgfrie dev-fasiliteter:** devcontainer/Codespaces, docker-compose, Azurite, json-server

Tillatte hjelpemidler

- All åpen dokumentasjon, AI verktøy og søk er OK – men **forklar** i README hva som er generert/lånt og hva som er ditt.

Fairness

- Vi forventer ikke en ferdig løsning. Velg de områdene der du mener du får vist ditt skill set. Løsningen kan være uferdig på stretch – poeng for tydelige prioriteringer og begrunnede valg.

Lykke til – vi gleder oss til å se hvordan du angriper problemet!