

Apputvikling



Kristin, Alliana og Julia

Nå skal vi prate om alt og ingenting :)

**SLUTT Å BRUKE KODE FRA
CHAT-GPT SOM DERE IKKE
FORSTÅR.**

Design og krav

Før dere begynner:

- Hvilke funksjonelle og ikke-funksjonelle krav skal appen ha?
- De funksjonelle kravene definerer hva applikasjonen skal kunne gjøre og hvilken funksjonalitet som skal vektlegges.
- Ikke-funksjonelle krav er produktkrav, organisatoriske krav og eksterne krav. Formålet med disse kravene er å sikre utviklingen av en stabil og pålitelig applikasjon som er rask og enkel å bruke.
- Need to have / nice to have - PRIORITER kravene

Eksempel på krav:

Funksjonelle krav:

1. Systemet skal beregne hvordan strømforbruket og -prisen vil endre seg ettersom været endrer seg de neste dagene.
2. Systemet skal vise strømprisen for stedet som brukeren velger.

Ikke-funksjonelle: Produktkrav

- Systemet skal være stabilt og pålitelig, og ikke krasje under normal bruk.
- Strømprisene skal vises i løpet av maksimalt to sekunder etter at brukeren har gjort en relevant forespørsel.

Organisatoriske:

- Det skal være en grundig og tydelig dokumentasjon av utviklingsprosessen.
- Det skal ikke koste noe å utvikle appen.
- Prosjektet og applikasjonen skal være ferdig i løpet av 2-3 måneder.

Design i startfasen

- Hva vil dere at appen skal gjøre? Diskuter og tegn.
- Hva vil brukerne? Spørreundersøkelser, intervjuer osv.
- Tips: Crazy 8 !!
- Ikke meld dere ut - alle må bidra - ingen kan BARE kode
 - Alt kan komme på eksamen
- Ha i bakhodet: Hvordan kan dette testes?
- Ha i bakhodet: WCAG



Backlog - lurt å ha

Backlog

Implementere og fullføre funksjonalitet som viser riktig vær-ikon på “Hjem”-skjermen, basert på data som blir hentet gjennom Location Forecast APIet.

Implementere funksjoner som henter data fra andre strømregioner og tilrettelegge for at brukere kan endre region i drop-down menyen i “Systeminnstillinger”-skjermen.

Bearbeide grafen i “Hjem”-skjermen og lage en linje som følger utviklingen av strømpriser gjennom dagen og “krysser” grafen for å vise nåværende pris. Dette for å spesifisere ytterligere hva strømprisen er akkurat nå.

Arkitektur

I startfasen, så hjelper det å visualisere og tenke over

Hvordan appen skal fungere:

- Hva skal appen gjøre?
- Hvilke type data har vi / trenger vi?
- Hvordan skal vi representere dataen?
- Må vi gjøre noe med dataen?

Under utviklingsfasen:

Lettere å vedlikeholde, skalere og teste appen

Endringer i arkitektur i store prosjekter er tidkrevende => teknisk gjeld

- Interfaces som bearbejder customobjekter
- Dummy data
- ViewModels og Views vet hva slags formater

Lar oss jobbe parallelt i alle lag!

```
21 interface ForecastRepository {
22     suspend fun makeSunEvents(
23         forecastGroupedByDate: SortedMap<LocalDate, List<WeatherPerHour>>, lat: String, long: String
24     ): List<SunEvent>
25
26     suspend fun findClosestWeather(
27         sunsetTime: LocalDateTime, weatherData: List<WeatherPerHour>
28     ): WeatherPerHour
29
30     suspend fun getWeatherConditions(weatherData: WeatherPerHour): WeatherConditions
31     suspend fun interpretCloudCondition(cloudAreaFraction: Double): CloudConditions
32     suspend fun interpretHumidity(humidity: Double): AirConditions
33     suspend fun getRating(
34         cloudConditionLow: CloudConditions,
35         cloudConditionMedium: CloudConditions,
36         cloudConditionHigh: CloudConditions,
37         airCondition: AirConditions
38     ): WeatherConditionsRating
39 }
40
41 /**
42  * An implementation of [ForecastRepository].
43  * @property sunriseDataSource instance of [SunriseDataSource]
44  * @property locationForecastDataSource instance of [LocationForecastDataSource]
45  * @property goldenHourBlueHourDataSource instance of [GoldenHourBlueHourDataSource]
46  */
47 class ForecastRepositoryImpl(
48     private val sunriseDataSource: SunriseDataSource = SunriseDataSource(),
49     private val locationForecastDataSource: LocationForecastDataSource = LocationForecastDataSource(),
50     private val goldenHourBlueHourDataSource: GoldenHourBlueHourDataSource = GoldenHourBlueHourDataSource()
51 ) : ForecastRepository {
52     /**
53      * Given a map containing lists of [WeatherPerHour] objects (where the key is a [LocalDate] and
54      * the values are the lists), this function creates a [SunEvent] object for each date, containing
55      * the time of sunset and a statement on the conditions for photography.
56      */
57 }
```

**Viktigste er å komme
i gang ASAP!**

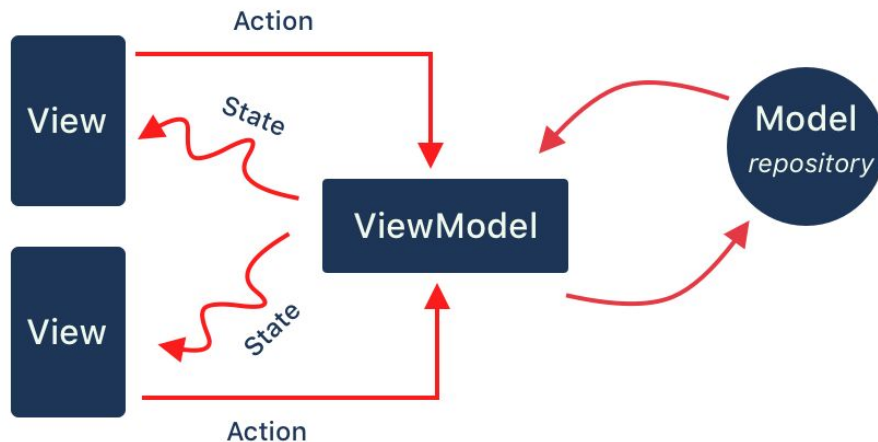
MVVM

MODEL-VIEW-VIEWMODEL

- Standard arkitektur i Android

Hvorfor?

- Klart skille mellom User Interface (UI) og appens data



UI-laget

Ansvaret til View:

Presentere state til bruker (ved å observere state i ViewModel)

Muliggjøre brukerinteraksjon

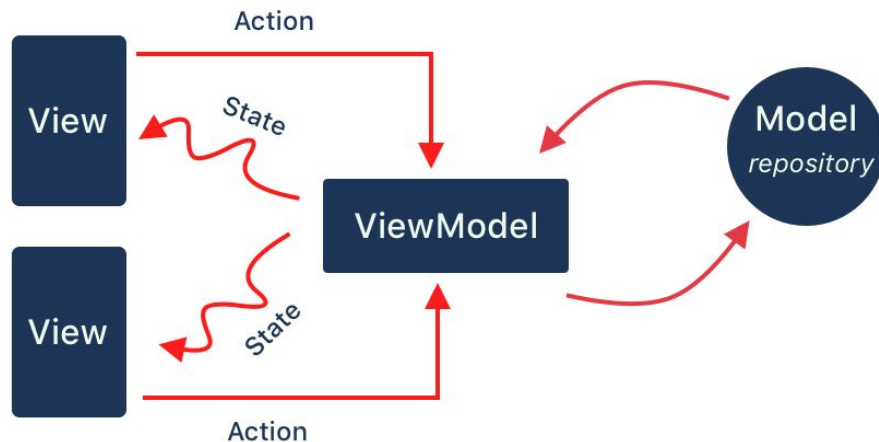
Ansvaret til ViewModel:

Ekspone state til View-et

Oppdatere state

Henting av data fra repo

Reagere på brukerinput



Data-laget

Ansvaret til Model:

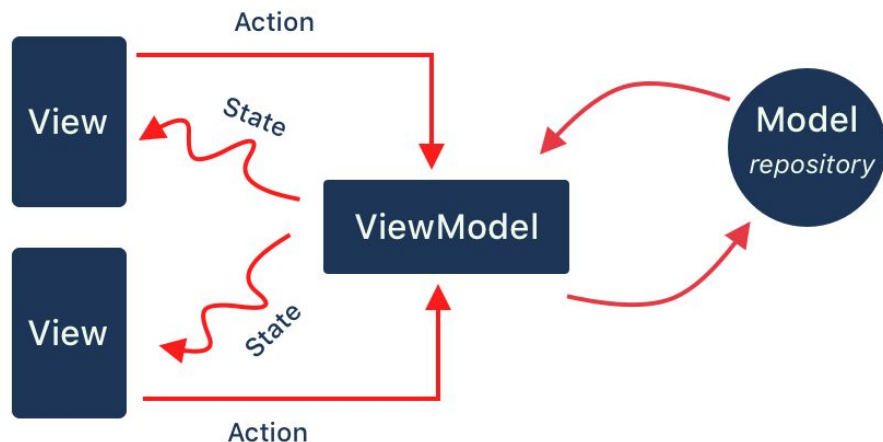
Hente data fra kilde/endepunkt

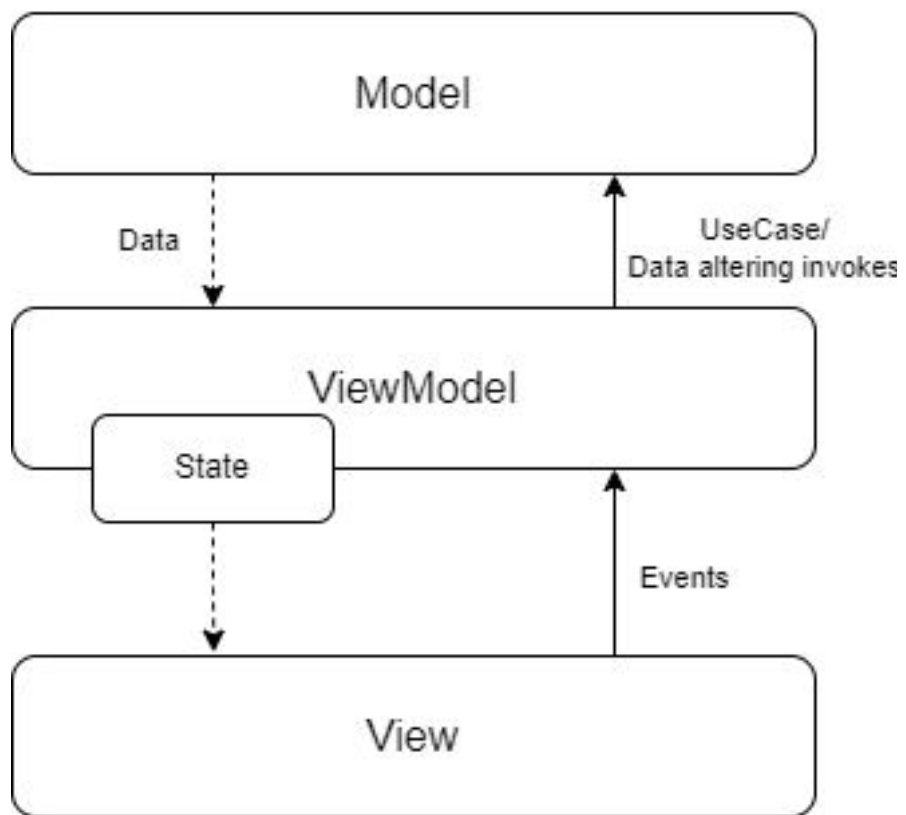
Gjøre et API-kall

Behandle dataen

Presentere ferdig behandlet data til ViewModel

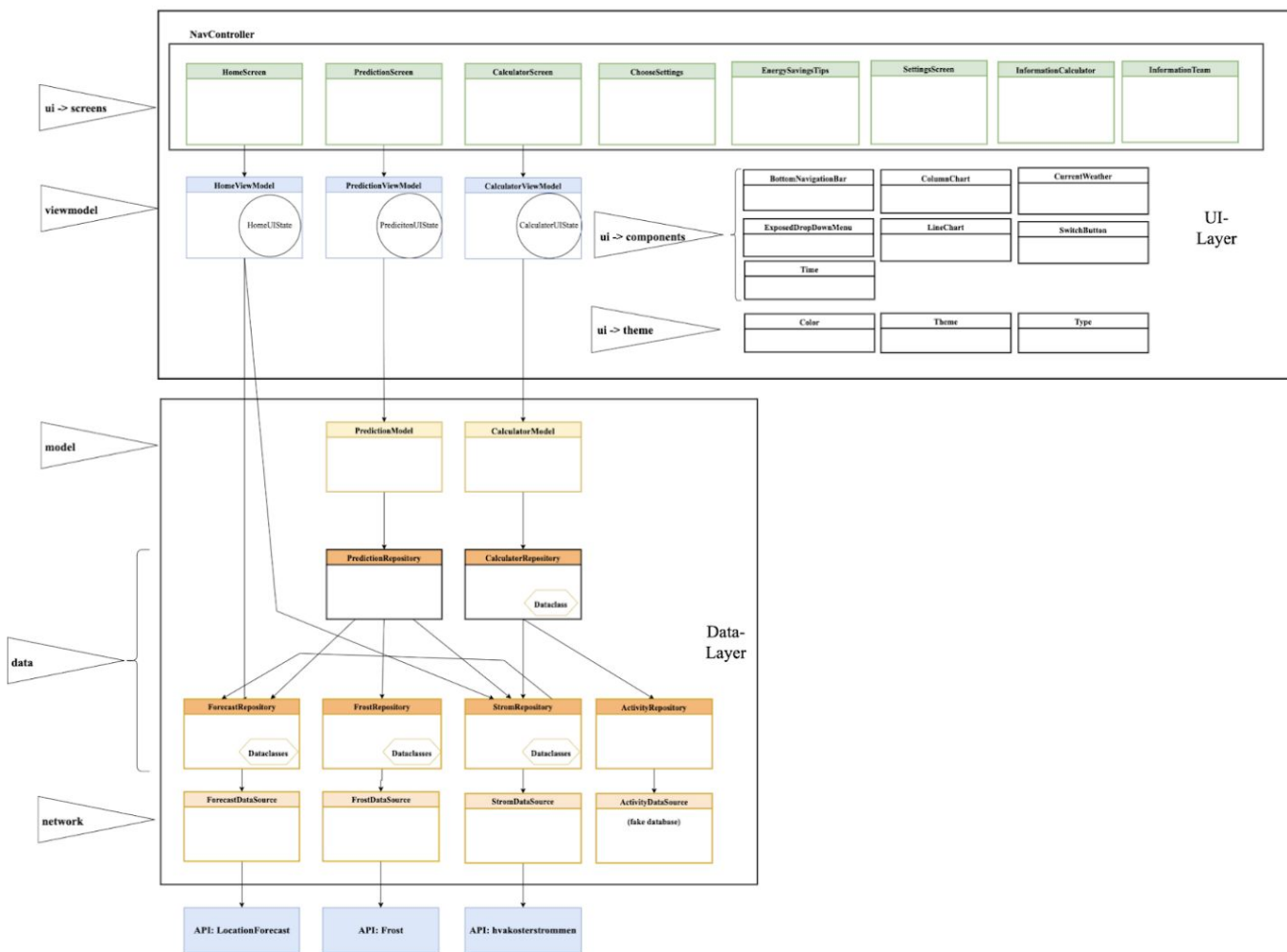
Behandling av hentet data skal kun skje i datalaget





Plasering (mappe)

Oversikt over filer i arkitekturen for WattIsThePrice



Høy kohesjon vs Lav kobling

et objekt med moderat ansvar og utfører et begrenset antall oppgaver

ulike ansvarsoppgaver i forskjellige klasser/moduler

avhengig av få andre objekter/moduler har lav kobling

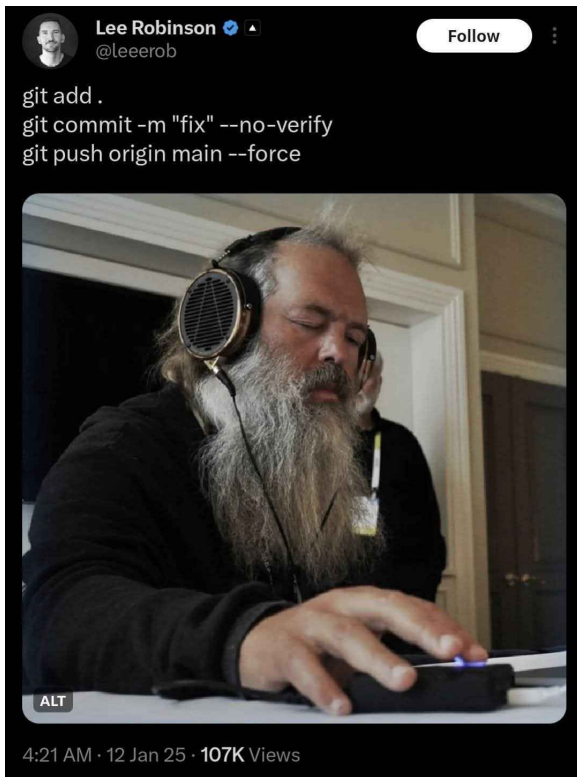
Vi lager få (oftest én) og enkle grensesnitt mellom klasser og moduler

Separation of concerns: Modulær kode

- Gjenbruk kode om dere kan
- Moduler - feks én modul for datatilgang, en for forretningslogikk, en for brukergrensesnitt
- Funksjoner - innenfor en klasse
- Klasser
- Testbarhet: Alle modulerbør kunne testes hver for seg

Kode med git

- Bli tidlig enig om branchingstrategi
 - Trunk-based, git flow, github flow
 - Hva du kaller det er egentlig ikke så viktig
- Én feature, én branch, én person
 - To folk jobber ikke på samme ting
- Ofte veldig kavete å branche fra en branch
- Ofte veldig kult å synce med main før man gjør pull request
- Commit hver gang du endrer noe
 - Med kommentar om hva du endret
- Push etter hver økt



LAB