APP DOKUMENTASJON

I hovedsakelig oppgaven er sikkerhetsperspektivet skalerbarhet, og tar utgangspunkt i appen for hendelsesregistrering som vi har designet og diskutert tidligere. Det er utviklet en RESTful web-service for denne appen. Det er en enkel versjon av selv utviklet appen, slik at vi har en minimal klient som konsumerer web-servicen.

I appen var utviklet flere klasser som samarbeider med hverandre. Klassen BadEvent for å jobbe med data BadEvents. Klasse DataAdapter får å jobbe med utdataene fra listen alle hendelser. Klasse NavigationEvents utviklet for navigasjonssystem i appen. Klasse register fungerer for å registrere ny bruker, jobber med forespørsel her og der brukes standart bibliotek JSON og OkHttp. Klasse RegisterEvents er utviklet for å registrere ny hendelse brukes standart metoder og standart bibliotek. Klasse ShowEvents utviklet for å jobbe med liste for å vise hendelser. Opprettet en adapter, deretter sett den for listen. I tillegg er det noen metoder

```
public static String makeRequest(final String uri, final Map<String, Object> obj) {
```
makeRequest metoden brukes for POST forespørselen til tjenesten. Bruker flere parameter og gir oss Boolean verdi.

```
public static String makeRequest(final String uri) {
```
For forbereder Message Pool for barnet tråden. Metoden brukes for GET forespørsel, og har en parameter.

```
private static OkHttpClient getUnsafeOkHttpClient() {
```
Det var utviklet metoden som installerer den tillitsfulle tilliten, opprett en ssl-stikkontaktfabrikk med vår all-trusting manager. Metoden opprette en tillitsforvalter som ikke validerer sertifikatkjeder, til api kontakt.

Ved hjelp av metoden `Toast.makeText` opprettet et pop up-varsel
Forespørsler fra ulike kunder ser det samme, om klienten er en nettleser, mobilenhet, eller noe annet. Klienten og serveren handler uavhengig, og samspillet mellom dem er bare i form av forespørsler og svar.

REST er en skalerbar og strukturert strategi for deling av ressurser. En ressurs er et informasjonsobjekt som skal deles. REST vil bidra til å opprettholde tilgjengeligheten av disse ressursene. Serveren kan deles inn i mikrotjenester. De vil behandle dataene og returnere resultatene. Dette reduserer belastningen på serveren. Siden serveren husker ingenting om brukeren som bruker API, så all nødvendig informasjon for å behandle forespørselen må leveres av klienten på hver forespørsel. Dette handler ikke om lagring av serversiden.

Fordelene med datalagring på klientsiden - bedre skalering, det brukes ikke noe minne på web-serveren, med et stort antall brukere kan det være problemer med minne på serveren. Støtte for flere servere - mulig distribusjon av innkommende forespørsler mellom et stort antall samtidige web-servere. Og det er enkelt å legge til nye servere. Det viktigste er at koden kan parallellisere forespørsler.

HATEOAS lar meg endre API uten å måtte endre klient. Det følger samme prinsipp som nettsteder. Brukeren trenger bare å kjenne hjemmesiden / domenet og kan finne ut hvordan man navigerer derfra ved hjelp av hyperkoblinger. Prinsippet er at klienten samhandler med nettverksapplikasjonen gjennom hypermedia dynamisk levert av applikasjonsserveren. For eksempel når våre brukere sender en HTTP GET-forespørsel om tilgang til en ressurs, må svaret inneholde URI-er som gjør at klientapplikasjonen raskt kan

finne alle direkte relaterte ressurser. I tillegg bør HATEOAS-koblinger inneholde beskrivelser av andre operasjoner (POST, PUT, DELETE, etc.) som hver koblet ressurs støtter, samt en tilsvarende URI for å utføre hver forespørsel.

Caching kan kontrolleres av HTTP-hoder sendt av serveren. Brønnbuffering (caching) vil bidra til å redusere serverforespørsler. Klienten er uavhengig av hvor mange lag som eksisterer mellom klienten og den faktiske serveren som svarer på forespørselen. Dette er et sentralt HTTP-prinsipp for caching av servere, omvendte proxyer og tilgang til sikkerhetsnivåer - alt gjennomsiktig for den forespørsel som klienten. Cacheable - Serverresponsen skal inneholde informasjon om hvorvidt dataene kan caches, noe som gjør det mulig for klienten og / eller mellommennene å cache data utenfor API-serveren.