Hybridutveckling med Flutter

11:15:59

Del 2: Utveckling Native-app med Flutter

Dagens agenda

1. Förtydliganden: Uppgift 3 : Dart
2. Repetition: Kursupplägg
3. Repetition: Förväntade studieresultat
4. Repetition: Dart vs. Flutter
5. Introduktion uppgift 4-6
6. Uppstart Flutter
    - Installation
    - Projektstruktur
    - Emulator
    - Widgets
    - Material Design
    - BuildContext

Förtydliganden: Uppgift 3 : Dart

1. Arkitekt Perra har gjort en del bra och dåliga val
    - Krävs djup förståelse av Perras intentioner för att upptäcka problemen
    - Det är okej att göra ändringar till Perras kod
        - Det är till och med uppmuntrat. Du har tagit över projektet.
    - CTO Petter bryr sig om funktionaliteten och att kunden blir nöjd.
    - Det går absolut att lösa uppgiften utan att ändra i Perras kod

Uppgift 3 : Uppenbara fel

1. Datamodellerna är anpassade för att representera en relationsdatabas
    - Men datamodellerna ska lagras i en NOSQL-databas (dokument)
2. Datamodellen för en clique innehåller data som flera användare förväntas skriva till samtidigt
    - Introducerar race-condition, data kan bli förlorat då användare skriver över varandras data
3. Det bloc state som laddat all data till ett clique innehåller allt data som ska visas
    - Men är inte strukturerat som ett GUI vill ha det för att smidigt rita ut.
    - När GUI får detta state måste det först sortera och även matcha scores till users
        - Operation som gör blockerar GUI utritningen och gör applikationen långsammare

Uppgift 3 : Lösningar på uppenbara fel

1. Fel 1: Datamodellerna är anpassade för att representera en relationsdatabas
    - Lösning: undvik relationer som tvingar klienten själv utföra joins genom att hämta från flera kollektioner
    - Lösning: duplicera data där nödvändigt
2. Fel 2: Datamodellen för en clique innehåller data som flera användare förväntas skriva till samtidigt
    - Lösning: separera datat så att användare skriver till egna dokument istället för samma
    - Lösning: t.ex. genom att introducera sub-kollektioner istället för fält (exempel i firebase console)
3. Fel 3: Clique bloc state ej strukturerat som ett GUI vill ha det
    - Lösning: låt sortering / mappning ske i bloc istället för GUI.
    - Lösning: undvik behovet av att mappa user - score genom att spara ytterligare information om en user i ett score

Lösningar: kodexempel

```
class Clique {
  final String id;
  final String name;
  // final Map<UserId, ParticipantScore> participantScoreMap; ← removed

  // represent clique scores as separate documents stored in a clique sub-collection
  // removes need of mapping User → UserId and removes race condition on Clique writes
}

// newly introduced, each user updates their own score.
class Score {
  final String user_id;
  final String user_name; // duplication of data, totally fine, removes need of user collection query
  final int score;
}

class User {
  final String name;
  final String email;
  final String id;
}
```

Lösningar: kodexempel - fortsättning

```
final class CliqueLoadingSuccess extends CliqueState {
  final Clique clique; // Clique
  // final List<User> participants; ← removed

  // the GUI previously would have had to map the participant ids + scores in the Clique
  // to the User list which had to be fetched based on what User ids exist in clique.participants map.
  // ...
  // Nice job Arkitekt Perra ?

  final List<Score> participantScoresSorted;

  // already sorted by bloc when returned.
  // remember Score includes user name so
  // GUI has everything it needs
}
```

Uppgift 3 : Arkitektur föreslagna förändringar

Öppna bild/draw.io

Uppgift 3 : Lösningar på uppenbara fel

1. Fel 4: Firedart saknar en del operationer / funktionalitet som finns i officiella Flutter klienten
    - Anmärkning 1: Samma begränsningar kvarstår om vi vill använda Firebase för desktop (windows/mac/linux)
    - Anmärkning 2: Flutter SDK:n går ej att använda heller i bloc_test, utan då behövs Firebase mockas/stubbas
    - Lösning 1: Anpassa din problemlösningsförmåga och kom förbi begränsningarna
    - Lösning 2: Migrera din kod till ett Flutter projekt redan nu och byt firebase klient
2. Fel 5: Uppgiften är för svår för att lösa på en vecka
    - Lösning: Inlämningsdatum har ändrats till **30/10/2023**

Repetition: Kursupplägg

1. Programmering i Dart
   - v.38 - v.41
   - Examineras med **tre** praktiska uppgifter
   - Modelkod(library) för tre olika applikationer (MVC)
     - Kod lämnas in (via GitHub?)
2. Utveckling Native-app med Flutter
   - v.42 - v.51
   - Examineras med **tre** praktiska uppgifter
   - Slutförande av de påbörjade applikationerna (M**VC**)
     - Kod + kort video demo

Repetition: Förväntade studieresultat

1. Dart
   - ~~Grundläggande Dart syntax~~
   - ~~Strukturer, variabler, funktioner, iterering~~
   - Utveckla mobilappar i Dart
2. Flutter
   - ~~State Management (BLoCs), Arkitektur~~
   - Widgets, layouter, animationer
   - ~~Firebase plattform~~
   - Designa och skapa användarvänliga UI
   - Testa och distribuera mobilappar på olika plattformar

Repetition: Dart vs. Flutter

1. Dart
   - Programmeringsspråk
   - Optimerad för snabba appar
   - Stöder webb, mobil och skrivbord
2. Flutter
   - UI-verktygslåda (framework) drivet av Dart
     - UI-komponenter som är Dart klasser
   - Bygg vackra appar för flera plattformar
   - Open source
3. Förhållande
   - Dart ger språket och exekveringsmiljön för Flutter
   - Flutter utökar Darts förmåga för UI-utveckling

Introduktion uppgift 4-6

1. Publiceras på sti.learning direkt efter dagens föreläsning :-)
    - Ni vet redan vilken funktionalitet de ska ha
    - Ni har själva kodat klart den kod ni ska använda för funktionaliteten
    - Det som saknas är de färdiga applikationerna
    - Få konkreta krav på uppgifterna
    - Stor kreativ frihet i hur ni ska stödja funktionaliteten i era GUI:n
    - Finns en del förslag för vad som går att lägga till / förbättra / fokusera på för betyget VG

Uppstart Flutter

1. Installation
2. Projektstruktur
3. Emulator
4. Widgets
5. Material Design
6. BuildContext

Installation: ~ 30-60 minuter

`https://docs.flutter.dev/get-started/install`

Flutter Projektstruktur

1. `lib/`
   - `main.dart` - Appens startpunkt
   - Kan innehålla underkataloger som `widgets/`, `models/`, `views/` etc.
2. `assets/`
   - För bilder, ikoner, och andra statiska resurser
3. `pubspec.yaml`
   - Konfigurationsfil för projektet - Definierar beroenden och resurser
4. `test/`
   - Innehåller tester för appen
5. `android/` & `ios/` & `linux` & `macos` & `web`
   - Kataloger för platform-specifik kod - rörs sällan

Emulator

1. Följ installationshänvisningar
    - Går även att använda fysisk enhet med usb-kabel
    - Går även över wifi med lite setup
    - **För kursen:** Android Studio - AVD Manager
    - **För kursen:** Web - Chrome

Flutter Widgets Koncept

1. Grundsten
   - Allt i Flutter är en `Widget`
   - Byggstenar för användargränssnittet
   - Använd existerande eller bygg egna
   - En Widget är en Dart `class` som kan `extend`as
2. Det finns många existerande Widgets
   - T.ex. `Text`, `Row`, `Column`, `Scaffold`
3. Egenskaper
   - Widgetar har egenskaper (properties)
   - Bestämmer widgetens utseende och beteende
   - Kan finnas en del `defaults` som du kan välja att själv konfigurera

Flutter Widgets Koncept - fortsättning

1. Kombinera Widgetar
    - Widgetar kan inneslutas i andra widgetar. Ofta säger man att en widget kan ha ett `child`
    - Nästlade widgetar skapar en `widget tree structure`, går att se över i DevTools
2. Återanvändning
    - Möjlighet att återanvända och anpassa widgetar
    - Skapar konsekvent och återanvändbart UI
    - Notera: Detta är inte en kurs i att skapa designsystem.
        - Vi kommer använda Material Design som är Googles egna designsystem.
        - Men vi kan absolut göra ändringar till de Widgets vi får från Material Design.
        - Eller skapa egna från grundkomponenterna i Flutter.

Material Design i Flutter

1. Designsystem skapat av Google
   - Riktlinjer för skapande av visuellt sammanhängande appar
2. Material Widgets
   - Flutter levererar inbyggda Material Design-widgets
   - T.ex. `AppBar`, `FlatButton`, `Card`
3. Interaktion & Rörelse
   - Animeringar och övergångar för bättre användarupplevelse
   - Ripple-effekter, skuggor, med mera.
4. Teman
   - Möjlighet att skapa och tillämpa egna teman
   - Konsistent färgschema och typografi

BuildContext i Flutter

1. Definition
    - En referens till widgetens plats i widget-trädet
    - Innehåller data som tillhör widgetens specifika plats i widget-trädet
2. Varför det är viktigt
    - Används för injection av state, styling och routing till Widgets.
    - Används ofta i byggmetoder (`build`)
        - När en Widget byggs kan den välja att läsa information från widgetens **UNIKA** BuildContext

BuildContext i Flutter - fortsättning

1. Hierarki & Ägande
   - Varje widget har sin egen BuildContext
   - Widgets skapar och ger sina barn en ny specifik BuildContext
2. Användningsfall
   - Navigering mellan skärmar med `Navigator`
   - Använda temadata med `Theme.of(context)`
   - Läsa värden från `BLoC` eller andra state management-lösningar
   - Få information från föräldern t.ex. gällande hur stor en Widget kan bli.
     - En förälder kan ha introducerat så kallade `constraints` som påverkar barnen.

Demo?

```
Demo time!
```

Tack för idag!

Glöm inte att ställa frågor! Finns på Team!

17 oktober 2023

Stockholms Tekniska Institut