

Datahantering – backend

TDP028 Entreprenöriell programmering

Översikt

- Navbar-navigering
- Null safety
- Datahantering
 - Provider
- Datahämtning
 - Async
 - Firestore
- Modulär kod

Hot reload

- *Hot reload* injicerar kod som förändrats, ritar om appen
 - Bevarar app state
 - *Web app stödjer inte hot reload just nu (2024)*
- *Hot restart* startar om appen
 - Förlorar app state (t.ex. inloggning, kundvagn)

Navbar

Navbar

- Lätt att navigera mellan skärmarna
- Bra om användarna vill navigera ofta mellan en grupp skärmar
 - Behöver inte stega framåt och backa flera steg till önskad skärm
 - Skärmarnas tillstånd (innehåll) bevaras

Kombinera GoRoute med Navbar

```
final __parentKey = GlobalKey<NavigatorState>();
```

```
final __shellKey = GlobalKey<NavigatorState>();
```

GoRoute

```
parentNavigatorKey = __parentKey
```

ShellRoute

GoRoute // Bottom Navigation

```
parentNavigatorKey = __shellKey
```

GoRoute // Bottom Navigation

```
parentNavigatorKey = __shellKey
```

GoRoute // Doesn't need Bottom Navigation

```
parentNavigatorKey = __parentKey
```

Navbar

```
return Scaffold(  
  body: widget.child,  
  bottomNavigationBar: BottomNavigationBar(  
    onTap: () => changeTab(currentIndex),  
    currentIndex: currentIndex,  
    items: const [  
      BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),  
      BottomNavigationBarItem(icon: Icon(Icons.chat), label: 'Chat'),
```

Byt tabb med go_router

```
void changeTab(int index) {  
  switch(value){  
    case 0:  
      context.go('/');  
      break;  
    case 1:  
      context.go('/chat');  
      break;  
  }  
  setState() {  
    currentIndex = value;  
  }  
}
```

ShellRoute

- För att kombinera navbar med go_router navigering
- Läs mer:
 - <https://medium.com/@ahm4d.bilal/using-gorouters-shellroute-in-flutter-for-nested-navigation-777a9a20642f>

Null safety

Null safety

default

String name // Non-nullable type. Cannot be 'null'

String? name // Nullable type. Can be 'null' or string

I första fallet kommer kompilatorn kunna hjälpa till att upptäcka kod där name riskerar att bli null

Läs mer:

- <https://dart.dev/null-safety#using-variables-and-expressions>
- <https://dart.dev/null-safety/understanding-null-safety>

? = null-aware operator

```
String? notAString = null;  
print(notAString?.length);
```

- Var beredd att var kan vara null
 - Hoppa över 'length' och returnera null (hade annars fått NullPointerException och krasch)

?? (dubbelt frågetecken)

Om null ...

Ex1:

String a = b ?? 'hello';

Ex2:

Text(items.getData ?? 'No data')

! (utropstecken)

- Avänd **!** för att konvertera från nullable to non-nullable datatyp

```
String? iKnowItsNotNull = "";  
print(iKnowItsNotNull!.length);
```

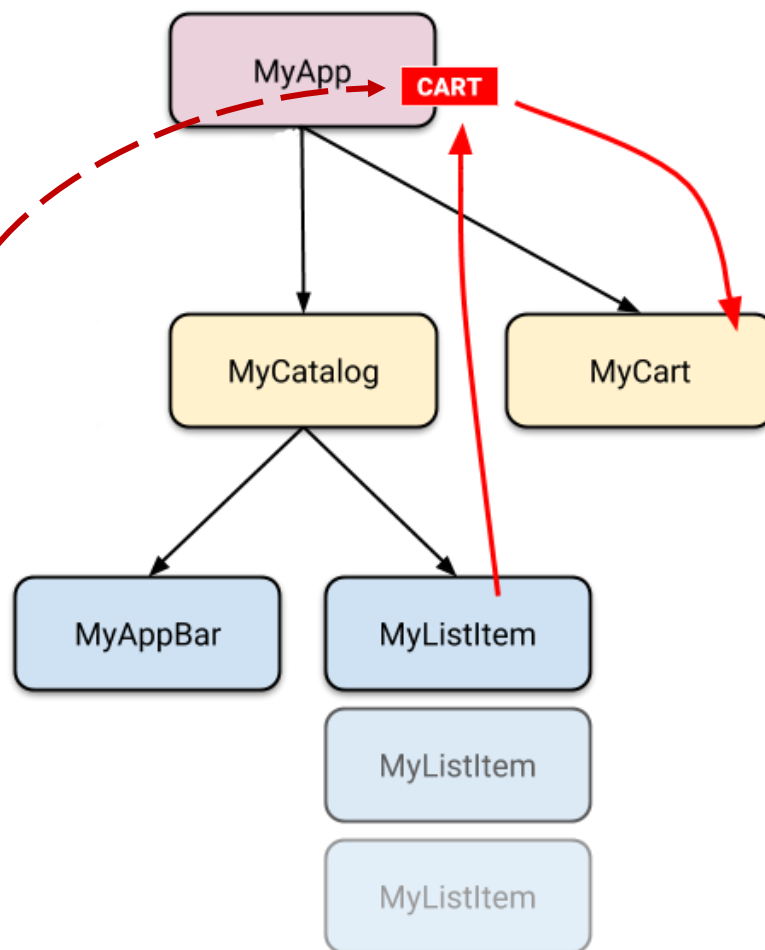
- Läs mer:
 - <https://dart.dev/null-safety/understanding-null-safety#non-null-assertion-operator>

State management

Tillståndshantering

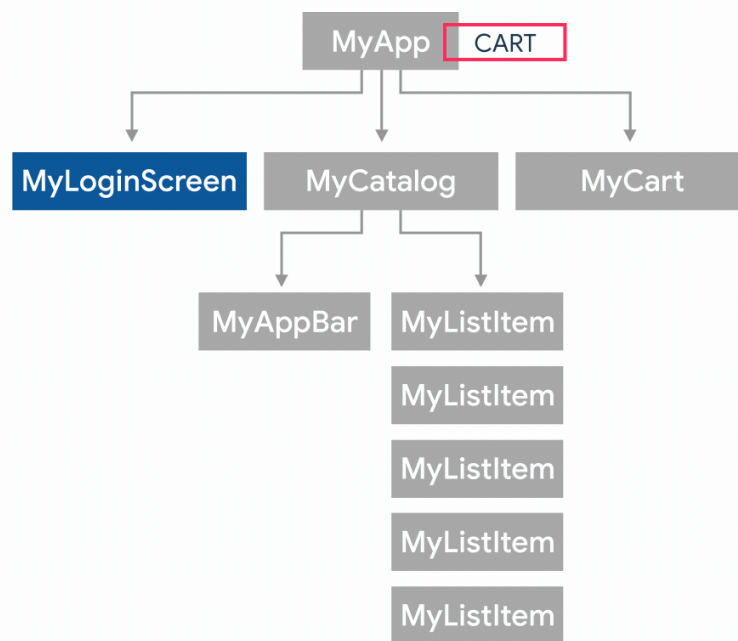
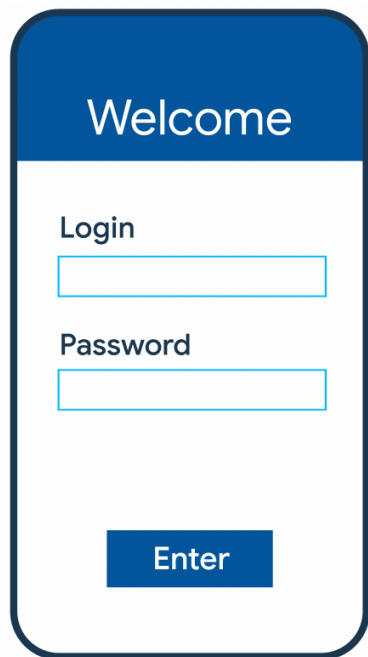
Fall när tillstånd bör lagras på app-nivå

- När en widget behöver påverka annan del i trädet
- Ex.
 - Anv. klickar på ListItem
 - Cart** uppdateras
 - Triggar omritning av MyCart



Ett fel och ett rätt sätt att hantera state

- Behöver meddela topp-nivå (app-tillstånd) och sedan rita om de delar av skärmen som berörs



Fel sätt: Skicka ner callback i widget-trädet

```
class MyApp {  
  void myTapCallback(Item item) {  
    print('User tapped on $item');  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    return SomeWidget(  
      MyClickabelWidget(myTapCallback),  
    );  
  }  
}
```

Fel sätt: Callback

- Lagra allt i AppState och registrera callbacks från Widgets i appen
 - Callback = ”Kalla på min funktion vid uppdatering, tack!”
- Stor nackdel
 - appState-instansen måste skickas neråt för att widgets ska kunna regga sina callbacks

Fel: MyClickableWidget anropar callback

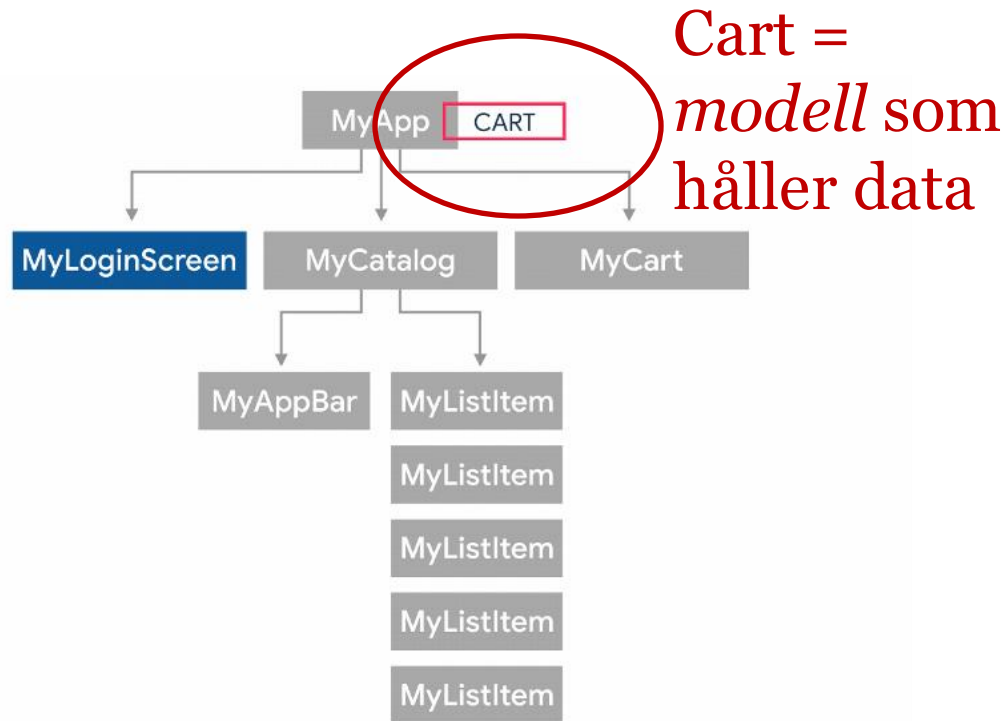
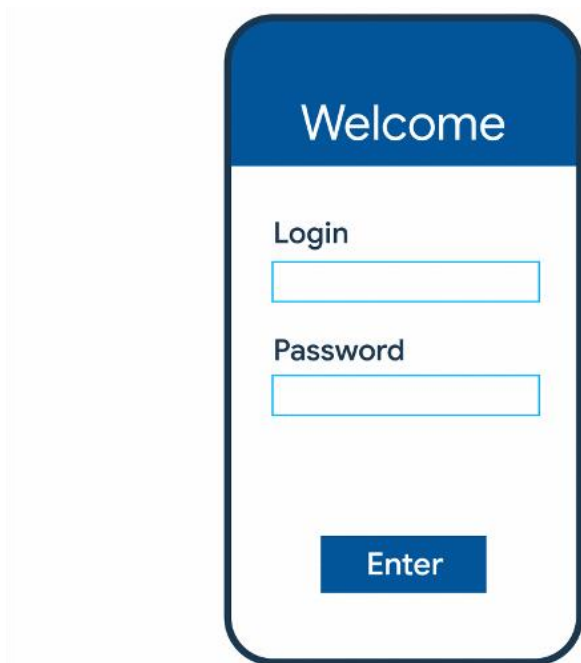
```
class MyClickableWidget extends StatelessWidget {  
  
  final int index;  
  final void Function(int) callBack;  
  
  const MyClickableWidget({super.key, required this.callBack});  
  
  @override  
  Widget build(BuildContext context) {  
    return SomeWidget(  
      title: Text(...),  
      onClick: () {callBack(index);}  
    );  
  }  
}
```

Fel: Rörigt att skicka in appState i widget-trädet

- Måste ofta skickas vidare i många led
- Rörigt att samla all information på ett ställe
 - Många tillståndsvariabler att hålla reda på
 - Många olika uppdateringar
 - Från många olika håll (vem har förstört ...?)

Rätt sätt: Använd Provider

- Provider hjälper hantera *modellen* som håller data



Provider biblioteket

- Skriv i terminal:

```
> flutter pub add provider
```

Alt. lägg till ny dependency i pubspec.yaml:

dependencies:

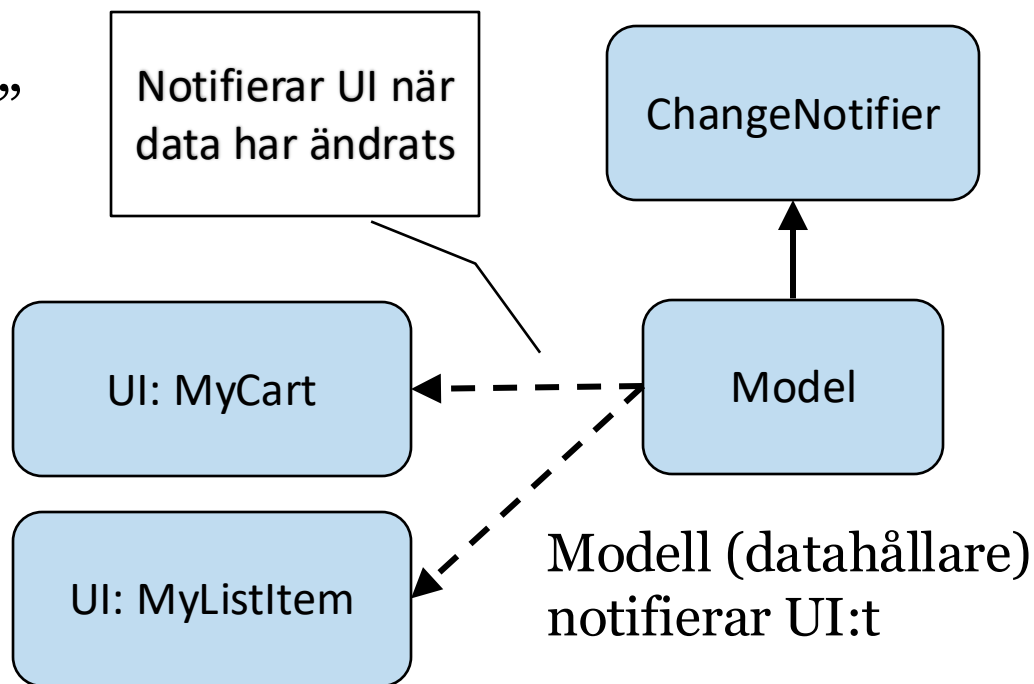
```
provider: ^6.1.2 # ta senaste versionen
```

- Importera

```
import 'package:provider/provider.dart';
```

Modellen kan ärva ChangeNotifier

- Kommer notifiera de som har "anmält" intresse
- Provider hjälper till med kontaktförmedlingen
 - Fördel: Slipper explicita beroenden i koden



notifyListeners()

```
class CartModel extends ChangeNotifier {  
    final List<String> _boughtItems = [];
```

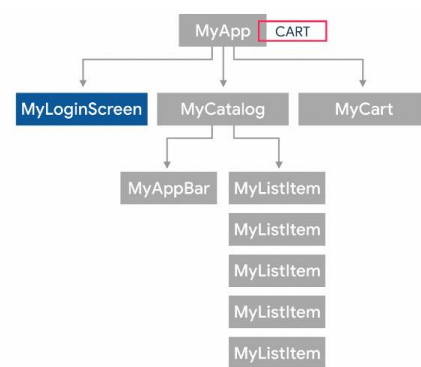
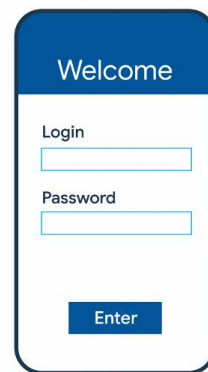
```
    List<String> get items => _boughtItems;
```

```
    void buy(String merchName) {  
        _boughtItems.add(merchName);  
        notifyListeners();  
    }
```

Wrap:a MyApp i en ChangeNotifierProvider

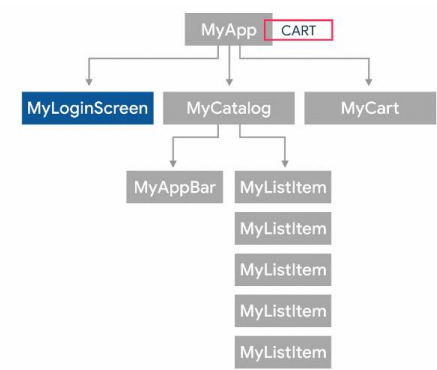
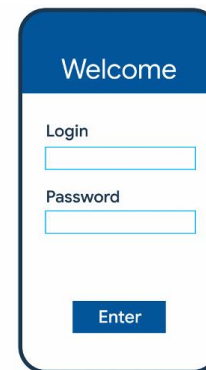
// ska ligga ovanför routing i widget-trädet

```
void main() {  
  runApp(  
    ChangeNotifierProvider(  
      // hanterar instansen av vår modell  
      create: (context) => CartModel(),  
      child: const MyApp(),  
    ),  
  );  
}
```



a. Vill ha uppdateringar: Consumer

```
return Consumer<CartModel>(  
  builder: (context, cart, child) {  
    return Text('Total price: ${cart.totalPrice}');  
  },  
);
```



b. Vill ha uppdateringar: watch

@override

```
Widget build(BuildContext context) {  
  // want to be notified if Cart has changed  
  CartModel cart = context.watch<CartModel>();  
  
  return Scaffold(  
    body: ListView.builder(  
      itemCount: cart.items.length,  
      itemBuilder: (context, index) =>  
        ListTile(title: Text(cart.items[index])),  
    ),  
  );  
}
```

c. Vill ha passiv tillgång

// just want access, no notifications needed

CartModel cart =

Provider.of<CartModel>(context, **listen: false**);

return Card(

child: ListTile(

title: Text(itemName),

trailing: const Text('buy'),

onTap: () => **cart.add(itemName)**,

Använd hellre Consumer:

- kan läggas längre ner i trädet
- mindre risk för att klanta med context

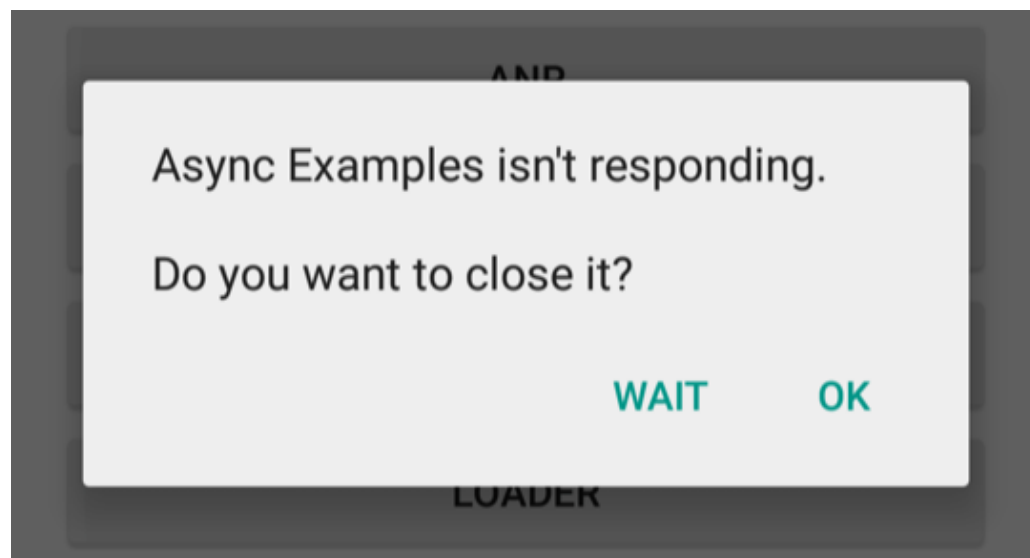
Provider

- Intelligent hantering av tillstånd hos data
 - Ger 'heads up' när data har förändrats
- Läs mer:
 - <https://docs.flutter.dev/data-and-backend/state-mgmt/simple>
- Kodexempel:
 - https://github.com/flutter/samples/blob/main/provider_shopper/lib/main.dart

Async

Android: Application not responding (ANR)

- Appen ska inte vänta på långa nätverksanrop



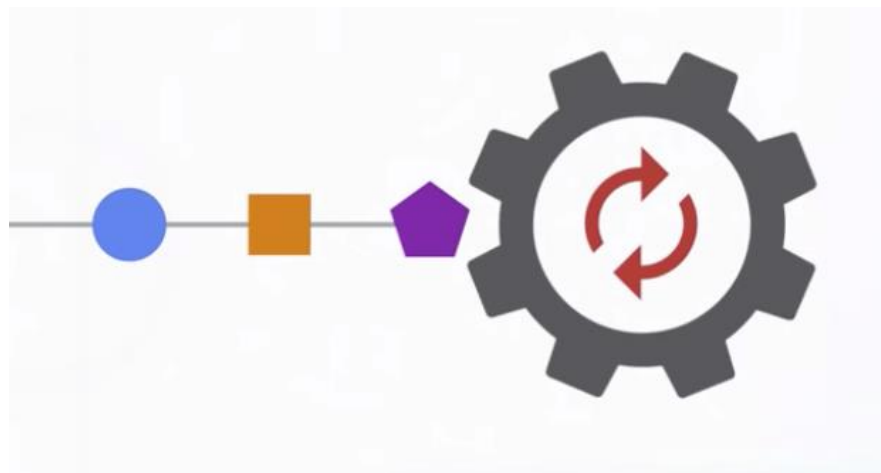
Exempel från Android

Trådar och virtuella processer (isolates)

- Trådar (threads)
 - Varje tråd fungerar som en egen parallell process
 - Flera trådar kan köras på en CPU-kärna
- Flutter har 'isolates' med eget minne
 - Isolates kommunicerar via meddelanden

Event loop

- UI-tråden, eller UI-isolate, kör main event-loop
 - Får inte blockeras av tung beräkning



- Läs mer:
 - https://www.youtube.com/watch?v=vl_AaCgudcY

Olika sätt att undvika blockering

1. Använd asynkrona widgets
2. Använd asynkrona bibliotek
3. Starta ny isolate manuellt

1. FutureBuilder

```
child: FutureBuilder<String>(
  future: _calculation, // a previously-obtained Future
  builder: (BuildContext context, AsyncSnapshot<String> snapshot) {
    List<Widget> children;
    if (snapshot.hasData) {
      children = <Widget>[
        const Icon(Icons.check_circle_outline),
        Text('Result: ${snapshot.data}'),
```

Pågående tung
process

Builder som blir
anropad med
delresultat

1. FutureBuilder

- Tre fall att ta hand om, dvs. visa olika saker på skärmen:
 1. `snapshot.hasData`
 2. `snapshot.hasError`
 3. annars: vänteläge (inte klar ännu)
- Läs mer:
 - <https://api.flutter.dev/flutter/widgets/FutureBuilder-class.html>

Future

- Påbörjat resultat från en process som fortfarande kör
- Ska typas
 - `Future<String>`
 - Kommer att bli en sträng när processen är färdig
- Future gör att event-loopen kan fortsätta
 - Man har ju redan ett slags svar i handen...
 - På de ställen i koden där man verkligen behöver svaret, måste man vänta in processen

Vänta på en långsam process

- `async` – `await`

```
Future<String> createOrderMessage() async {  
    var order = await fetchUserOrder();  
    return 'Your order is: $order';  
}
```

Async – await

- Main UI-isolate blockeras inte, eftersom funktionen returnerar en future under tiden
 - return-satsen kommer dock att dröja
- Läs mer
 - <https://dart.dev/libraries/async/async-await>

2. Asynkrona bibliotek

```
var client = HttpClient();  
try {  
    HttpClientRequest request = await client.get('localhost', 80, '/file.txt');  
    // Optionally set up headers...  
    // Optionally write to the request object...  
    HttpClientResponse response = await request.close();  
    // Process the response  
    final stringData = await response.transform(utf8.decoder).join();  
    print(stringData);  
} finally {  
    client.close();  
}
```

Felhantering

```
throw FormatException('Expected at least 1 section');
```

```
try {  
    breedMoreLlamas();  
} catch (e) {  
    print('Error: $e'); // Handle the exception first  
} finally {  
    cleanLlamaStalls(); // Then clean up  
}
```

Felhantering

- Läs mer:
 - <https://dart.dev/language/error-handling>

3. Manuella sättet

- Starta ny asynkron isolate
 - Kör det tunga jobbet
 - Den kommer att anropar din callback när klart
- Läs mer på:
 - <https://docs.flutter.dev/perf/isolates>

Backend – Firestore

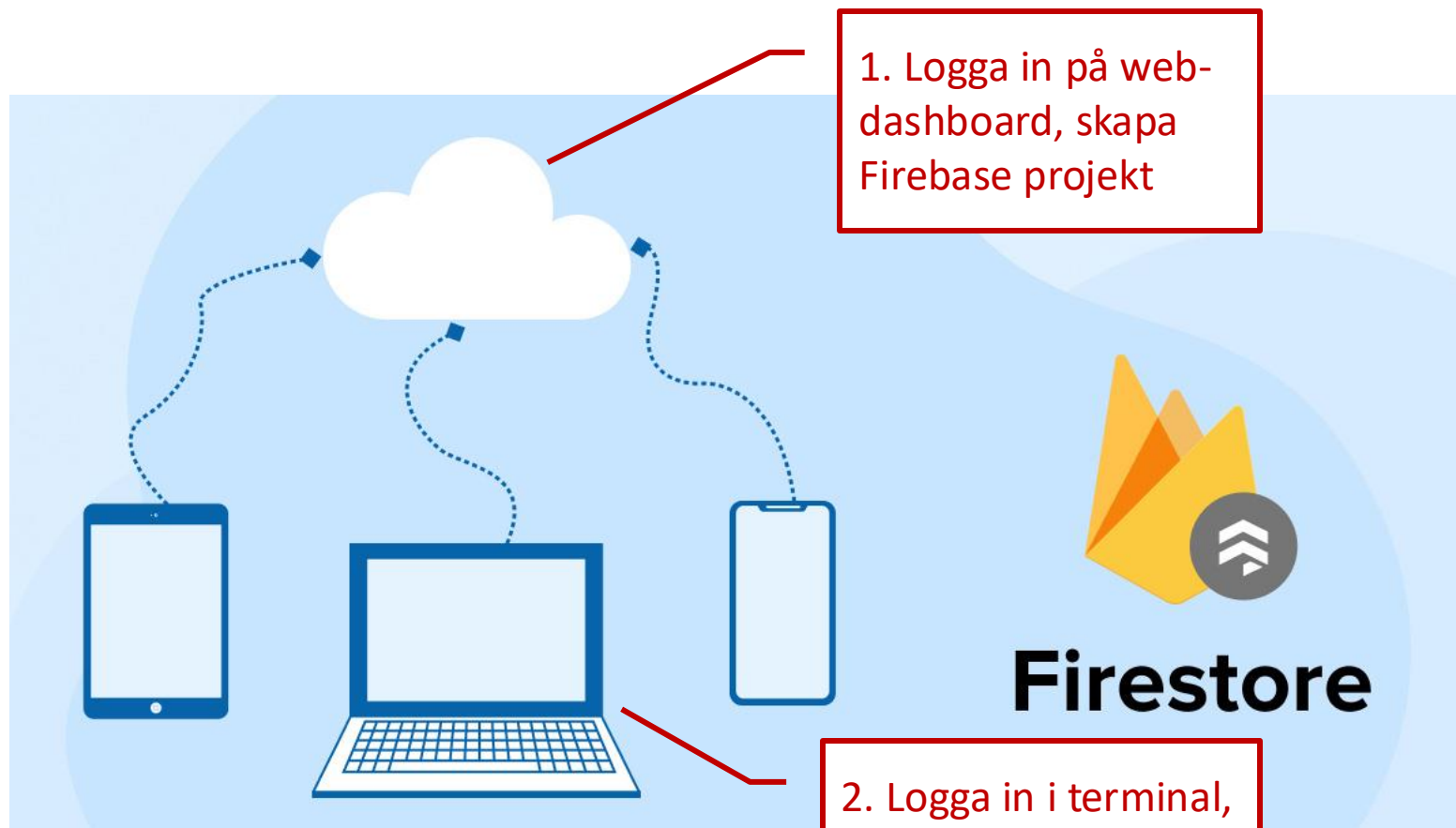
Bra startpunkt

- Sätt upp Firebase:
 - <https://firebase.google.com/docs/flutter/setup?platform=ios>
- Quickstart för Firestore:
 - <https://firebase.google.com/docs/firestore/quickstart>

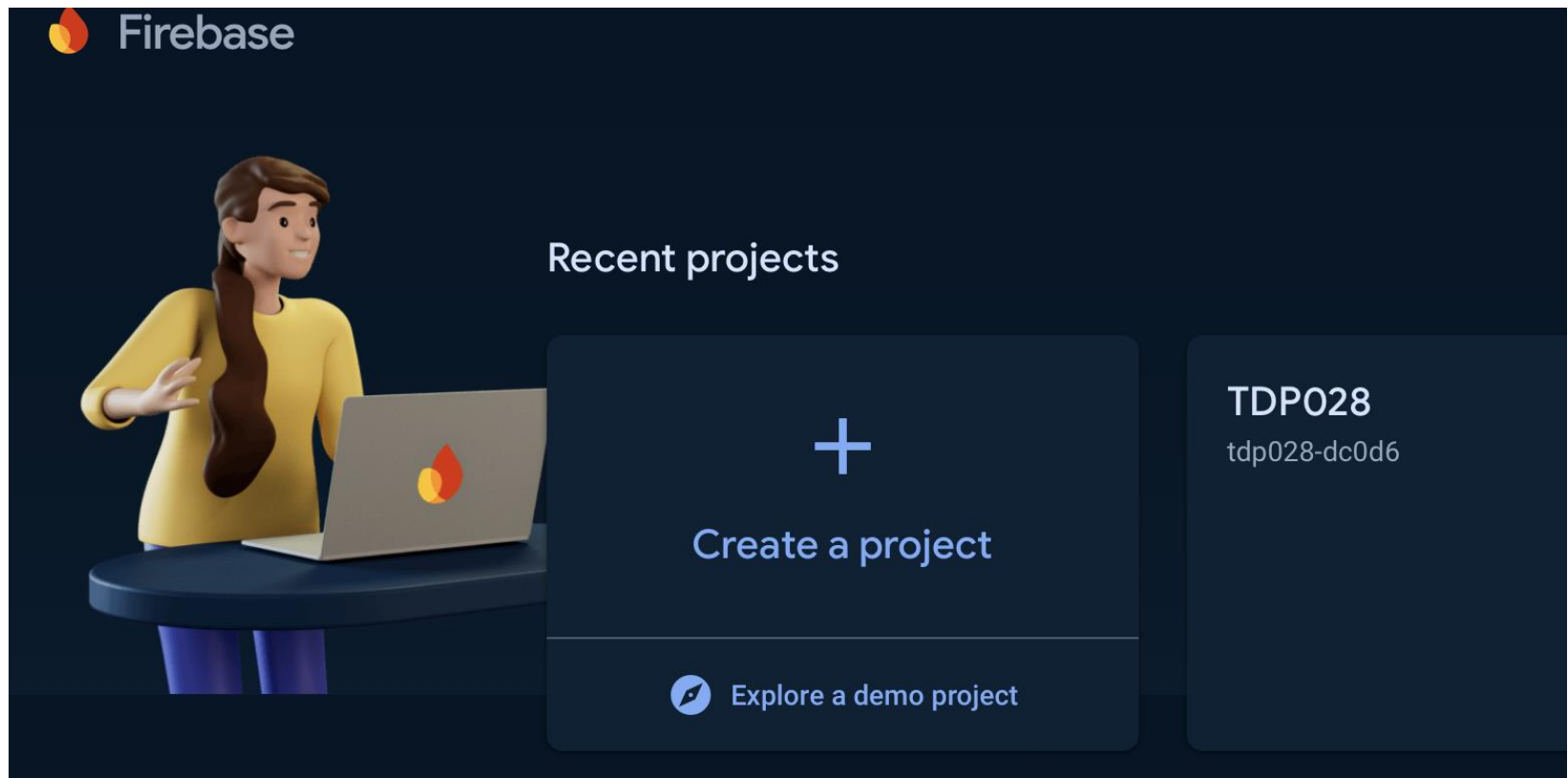
Cloud Firestore vs Realtime database

- Firestore
 - Dokument, ger mer struktur
- Realtime database
 - json, lite enklare koncept
- Läs mer:
 - <https://blog.flutter.wtf/firestore-gui-tools/>

Behöver sättas upp från två håll



1. Skapa Firestore projekt (på webben)



Sätta upp Firebase

× Add Firebase to your Flutter app

1 Prepare your workspace

The easiest way to get you started is to use the FlutterFire CLI.

Before you continue, make sure to:

- Install the [Firebase CLI](#) and log in (run `firebase login`)
- Install the [Flutter SDK](#)
- Create a Flutter project (run `flutter create`)

Next

Install and run the FlutterFire CLI

Initialize Firebase and add plugins

2. Från utvecklarkåll

- Öppna terminal
- Installera Firebase CLI (Command Line Interface)
 - > `curl -sL https://firebase.tools | bash`
 - > **firebase login**

Woohoo!

Firebase CLI Login Successful

You are logged in to the Firebase Command-Line interface. You can immediately close this window and continue using the CLI.

3. Från utvecklarkhåll (forts)

- Nu är moln och utvecklar miljö ihopkopplade
- Installera FlutterFire CLI
 - > `dart pub global activate flutterfire_cli`
 - > `export PATH="$PATH":"$HOME/.pub-cache/bin"`
 - > `flutterfire configure --project=tdpo28-dcod6`



Projektet som
skapades i Firestore

https://console.firebase.google.com/...

The screenshot displays the Firebase console for project TDP028. The left sidebar contains navigation links for Project Overview, Generative AI, Build with Gemini, Project shortcuts (Firestore Database, App Check), Product categories (Build, Run, Analytics), All products, and Related development tools (IDX, Checks). The main content area shows the 'Project Overview' tab with a list of apps. A red circle highlights the 'shopping_firestore_provider (android)' app, and a red arrow points to its settings icon. A 'Gemini in Firebase' banner is visible below the app list. The 'Build' section at the bottom shows Firestore usage graphs for Reads and Writes.

Apps List:

App Name	Platform	App ID
10 apps		
shopping (android)	Android	liu.shopping_firestore_provider.app
shopping_firestore_provider (android)	Android	liu.shopping_firestore_provider.app

Gemini in Firebase: Get answers to questions about Firebase products, use cases, and features. Generate code for development and shorten your troubleshooting process with new insights—with the assistance of a natural language chat interface directly in the Firebase console.

Build Section:

- Firestore:**
- Reads (current):** 9
- Writes (current):** 0

Usage Graphs:

The graphs show usage over time (Sep 13 to Sep 19). The 'Reads' graph shows a peak around Sep 16. The 'Writes' graph shows a peak around Sep 16.

https://console.firebase.google.com/...

The screenshot displays the Firebase console interface. On the left, a sidebar contains navigation links: Project Overview (with a settings gear icon), Generative AI (with a 'Build with Gemini' button), Project shortcuts (Firestore Database, App Check), Product categories (Build, Run, Analytics), All products, and Related development tools (IDX, Checks). The main content area is titled 'Project settings' for project 'TDP028'. It features an 'Environment' section with an 'Environment type' of 'Unspecified'. Below this is the 'Your apps' section, which lists several apps. A red circle highlights the 'shopping_firestore_provider (android)' app. To the right of this app, the 'SDK setup and configuration' section provides instructions and links to 'See SDK instructions' and 'google-services.json'. Further down, the 'App ID' is listed as '1:1067431162115:android:9f2172c11bbd090e87b936', and the 'App nickname' is 'shopping_firestore_provider (android)'. The 'Package name' is 'liu.shopping_firestore_provider.app'. At the bottom, there is a section for 'SHA certificate fingerprints' and an 'Add fingerprint' button.

Environment

This setting customizes your project for different stages of the app lifecycle

Environment type: Unspecified

Your apps

Android apps

- shopping (android) liu.rit
- shopping_firestore_provider (android) liu.shopping_firestore_provider.app**
- shopping_fs_nav_prov (android) liu.sho
- shopping_provider (android) liu.tdp0

Apple apps

- shopping (ios) com.example.shopping
- shopping_fs_nav_prov (macos) com.example.shoppingFsNavProv
- shopping_provider (ios) com.example.shoppingProvider

SDK setup and configuration

Need to reconfigure the Firebase SDKs for your app? Revisit the SDK setup instructions or just download the configuration file containing keys and identifiers for your app.

[See SDK instructions](#) [google-services.json](#)

App ID: 1:1067431162115:android:9f2172c11bbd090e87b936

App nickname: shopping_firestore_provider (android)

Package name: liu.shopping_firestore_provider.app

SHA certificate fingerprints: [redacted] Type

[Add fingerprint](#)

> flutterfire configure (forts)

- Which platforms should your configuration support (use arrow keys & space to select)?
 - Alla alternativ är valda från början
 - Mellanslag för att 'toggle:a'
 - Tryck bara Enter för att välja alla plattformar

Från utvecklarkhåll (forts)

Installera Firebase-core

```
> flutter pub add firebase_core
```

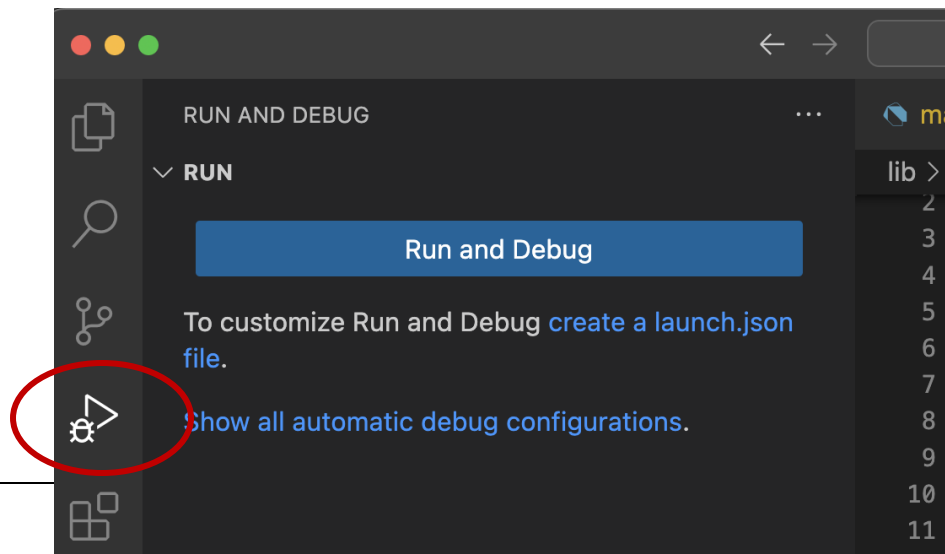
Firestore

- Sätt upp Firestore plugin till Flutter

> `flutter pub add cloud_firestore`

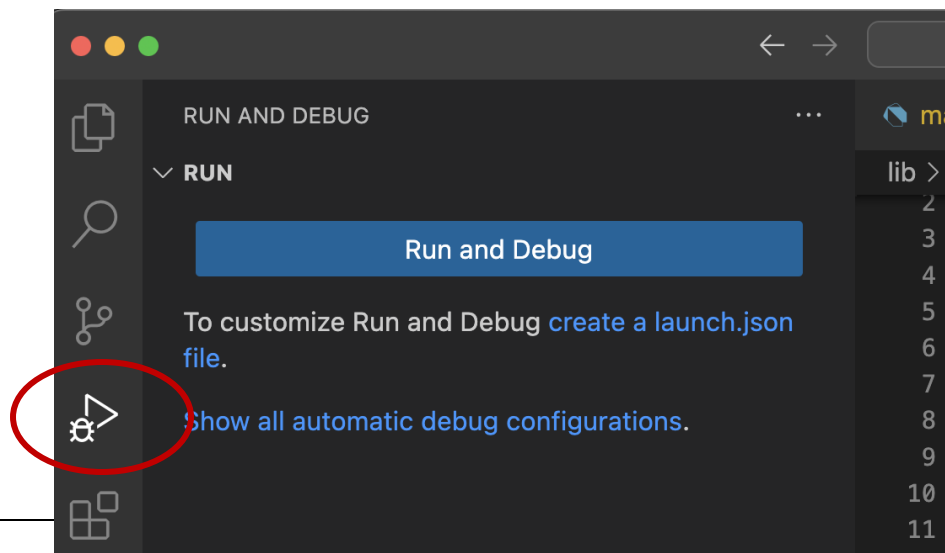
- Bygg om (kompilera om) appen:

> Kör appen



Kör appen

- OBS! Kompilering till iOS och MacOS tar en evighet!
För att snabba upp lite:
 - https://firebase.google.com/docs/firestore/quickstart#set_up_your_development_environment

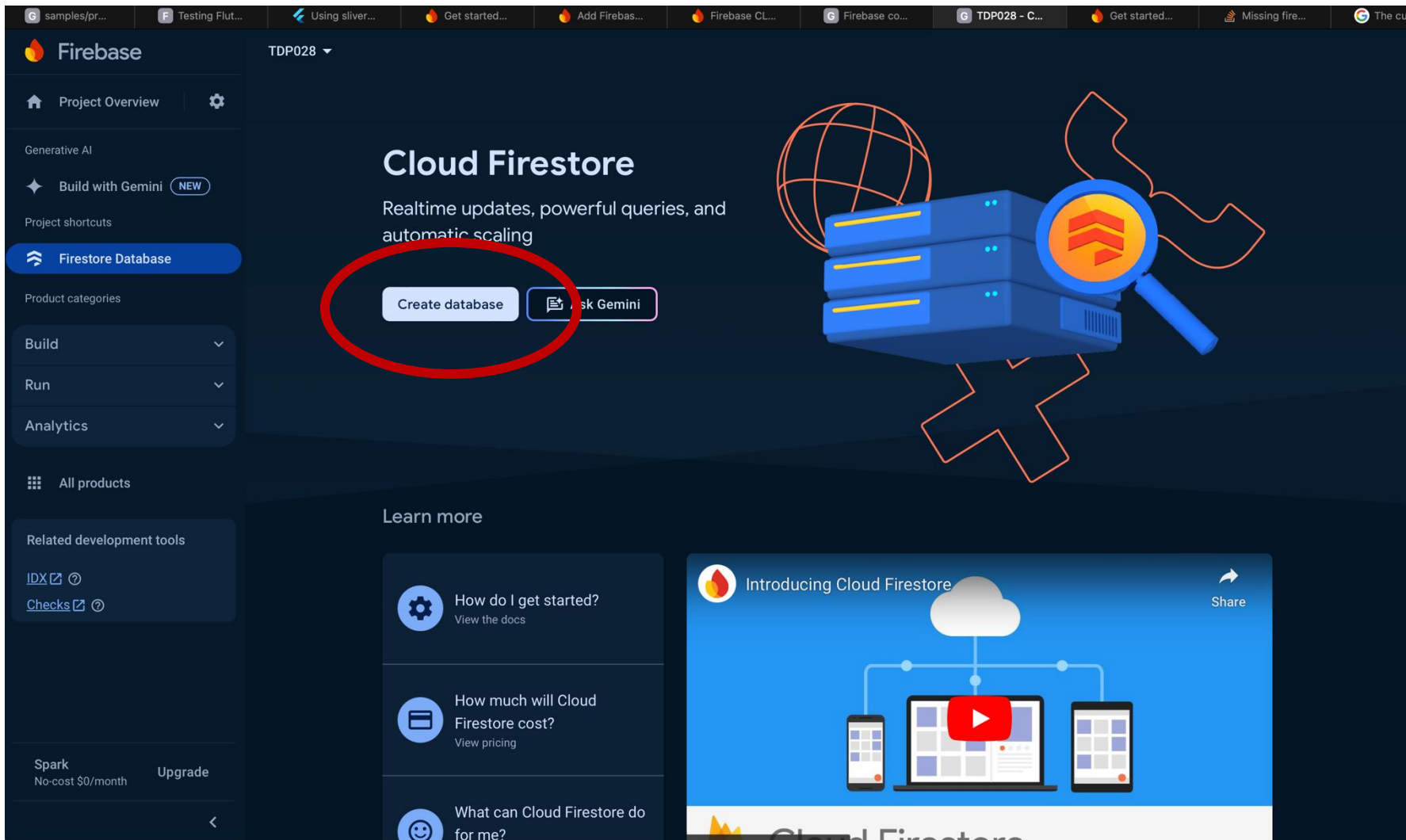


Importera och initialisera *Firebase*

```
import 'package:firebase_core/firebase_core.dart';  
import 'package:[your_package]/firebase_options.dart';
```

```
Future<void> initializeFirebase() async {  
  await Firebase.initializeApp(  
    options: DefaultFirebaseOptions.currentPlatform,  
  );  
}
```

Dags att skapa databasen



The screenshot shows the Firebase console interface for a project named 'TDP028'. The left sidebar contains navigation options: 'Project Overview', 'Generative AI' (with a 'Build with Gemini' button), 'Project shortcuts', 'Firestore Database' (highlighted with a blue bar), 'Product categories', 'Build', 'Run', 'Analytics', 'All products', and 'Related development tools' (with links to 'IDX' and 'Checks'). The main content area features a large illustration of server racks and a magnifying glass over the Firebase logo. The text 'Cloud Firestore' is prominently displayed, followed by the description 'Realtime updates, powerful queries, and automatic scaling'. Below this, the 'Create database' button is circled in red, and a 'Ask Gemini' button is visible next to it. At the bottom, there is a 'Learn more' section with three links: 'How do I get started?', 'How much will Cloud Firestore cost?', and 'What can Cloud Firestore do for me?'. A video player titled 'Introducing Cloud Firestore' is also visible in the bottom right corner.

Importera och initialisera *Firestore*

- Firestore är själva databasen (Firebase är en samling molntjänster)

```
import 'package:cloud_firestore/cloud_firestore.dart';
```

```
final db = FirebaseFirestore.instance;
```

Referens till en samling eller dokument

```
CollectionReference cartCollRef = db.collection('cart');
```

```
DocumentReference cartDocRef =  
    db.collection('cart').document(...);
```

- Mer om hur Firestore är organiserad
 - https://www.youtube.com/watch?v=v_hR4K4auoQ

Läs vidare

- <https://firebase.google.com/docs/firestore/query-data/listen>

Möjliga småproblem

Vid problem

- Kör flutter doctor
 - Kolla att alla installationer är OK
- Googla sedan på det felmeddelande du har fått
 - Oftast något namn eller någon version i en inställningsfil som är fel

Inkoppling Firestore till Android platform

- Vid frågan om vilket app-namn
 - Testa att bara göra return
 - Kommer då att få rätt app-namn (enligt de inställningsfiler som finns i projektet)
 - Om man skriver själv, trunkeras namnet, och blir ”fel”

Ex. Problem att köra på Android

- Problem med Android – Firebase kopplingen
 - “No matching client found for package name”
- Dubbelkolla appens namn i följande filer:
 - androidManifest.xml
 - build.gradle
 - google-services.json
- Ska matcha namnet i Firestore-projektet

Ex. Problem med Chrome

- Fastnar i “Waiting for debug connection..”
- Testa att köra
 - // rensa allt kompilerat från projektet
 - flutter clean
- Och sedan
 - // bygg om och kör i verbose mode
 - flutter run --verbose -d chrome
- Vid behov
 - flutter pub cache repair

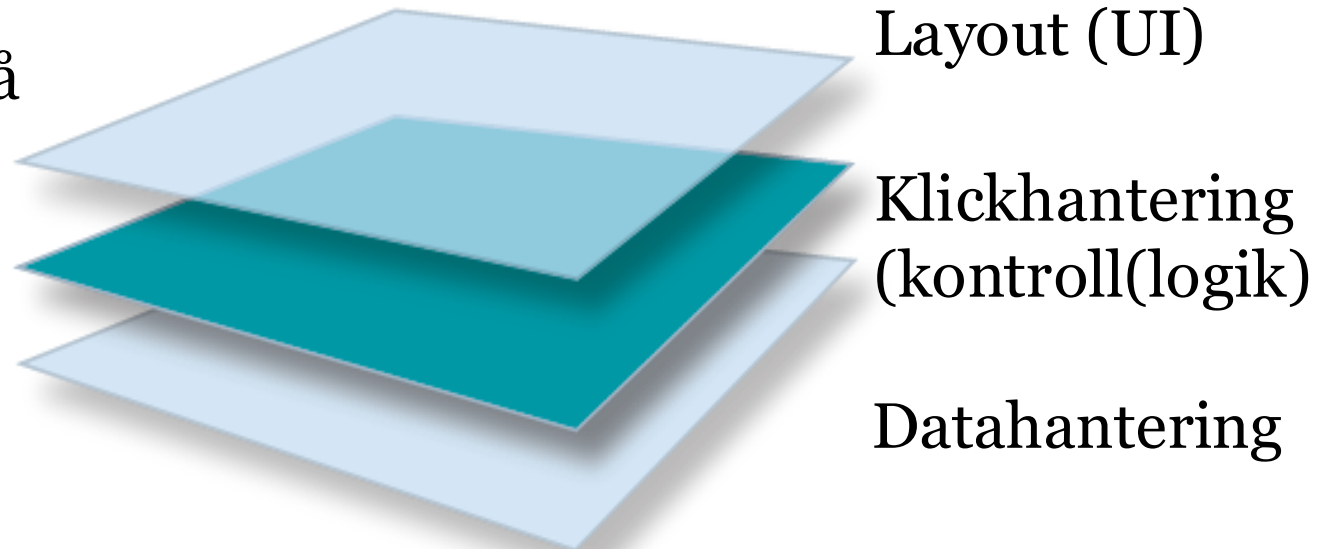
Modulär kod

Modulär kod

- Ofta naturligt att dela upp koden i skärmar
 - main_screen mapp
 - Kod för widget:en
 - Hjälp-klasser
 - Datastrukturer för att hålla data
 - Kod för att hämta data
 - second_screen mapp
 - ...

Återanvändbara skärmbitar: Widgets

- Återanvända
- Flytta
- Lätt ändra på



Flutter-app arkitektur

- Tre lager:
 - Presentation layer
 - Business logic layer
 - Data layer
- Referenser (beroenden) ska aldrig gå åt sidan
 - Viktigt att UI kan bytas ut (ritas om) utan att arkitektur-lagren nedanför får hängande pekare

Beroenden

- Varje widget ska vara självtilräcklig
 - Utbytbar, flyttbar, lätt att ändra
- Beroenden bör gå rakt uppåt i widget-trädet
 - Widgets ska *inte* kommunicera med varandra direkt
 - Ingen ”sidledes” kommunikation i widget-trädet
- Idealet
 - Widgets ska bara veta om sin context

Dependency injection

Dependency injection

- Vill inte ha direkta beroenden
 - T.ex. inte veta hur en instans ska skapas
- Objekt skapas istället externt
 - Skickas med färdigt till ”ägarens” konstruktor
 - Eller via setters
 - Eller till build via context (t.ex. m.h.a. provider-biblioteket)

Exempel på injection

```
class Car {  
    Engine? engine;  
    List<Wheel> wheels;  
  
    Car(this.wheels, {this.engine});  
  
    void setEngine(Engine newEngine) {  
        engine = newEngine;  
    }  
}  
  
Car car = Car(wheels).setEngine(engine);
```

Dependency injection

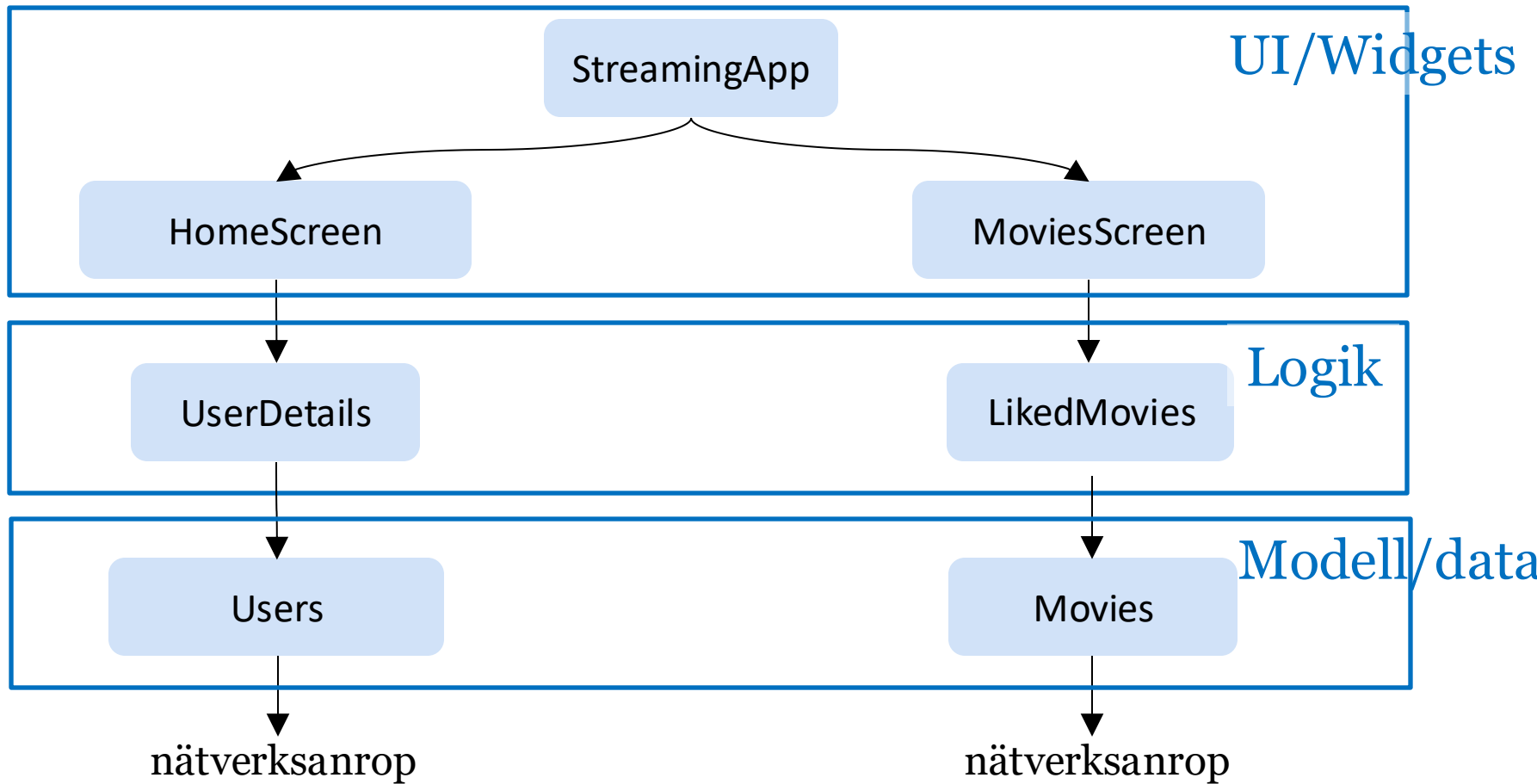
- Fördelar
 - Tydligare vilka andra klasser som denna klass behöver
 - Lättare att underhålla
 - Lättare att byta ut komponenten
 - Lättare att testa (skicka in mocks)

Model View ViewModel (MVVM) och Repository designmönstren

MVVM

- Vill ha så lite fasta beroenden som möjligt mellan UI och data-lagret
- ViewModel i Android
 - Injiceras till UI:t
 - Notifierar UI:t när data ändrats
- Provider i Flutter
 - Kan skapa providers var som helst i widget-trädet
 - Kan notifiera när data ändrats

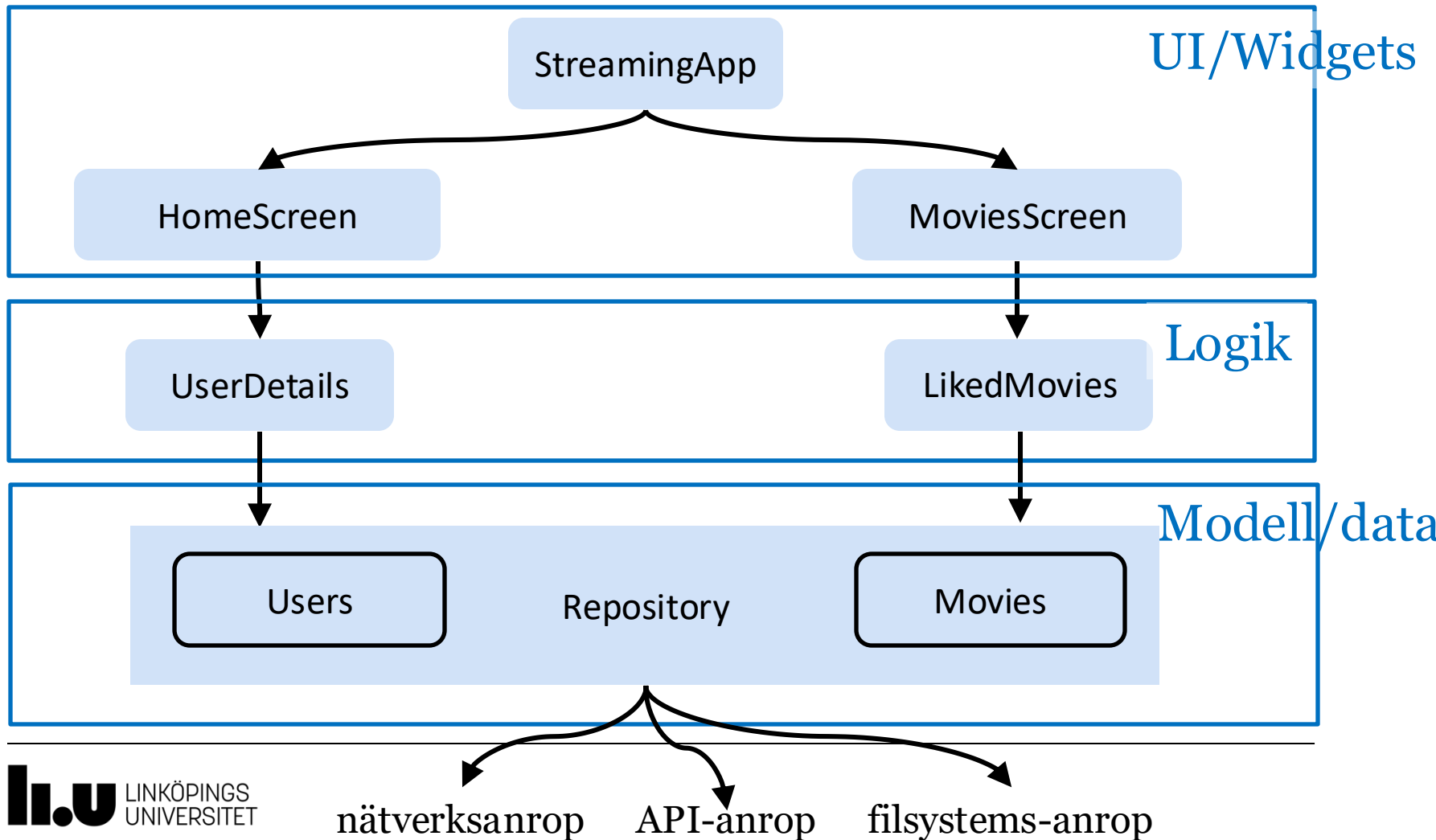
App-struktur



Inte helt optimalt

- **Infrastruktur (nätverk, filhantering, etc.) utströdda**
- Lätt att införa buggar
 - Glömmer att ändra på ett ställe
- Arbetsamt
 - Byta från Firestore till egen databas
 - Behöver ändra på flera ställen
- Svårt att testa, eftersom nätverksanrop tätt kopplade till (inbyggda i) varje data-klass

Bättre app-struktur



Läs mer

- Repository pattern:
 - <https://fluttersamples.com>
- Youtube:
 - <https://www.youtube.com/watch?v=x6C2ozhZHw8>

rita.kovordanyi@liu.se

www.liu.se