Flutter

Flutter create project_name
Code .

jks file

- A keystore is a container of certificates, private keys etc. There are specifications of what should be the format of this keystore and the predominant is the #PKCS12. JKS is Java's keystore implementation. There is also BKS etc. These are all keystore types.

FlutterActivity

- For android, main activity extends FlutterActivity

Dart

Data types

- Int, String, num(int), var, list(array), set, map
- _(underscore) means that variable is private
- Example
    - Var array = [12,”name”,20];
    - list<int> num = [];
- Set
    - Same as list
    - Does Not include duplicate elements
    - Example
        - set<int> num = {1,2,3}
        - Duplicate elements will be auto removed
- Map
    - Same as hash map
    - Key-value pair
    - Example
        - Map<String, String> map = {“name”: “Joyal”}
        - Var map = {“name”:”joya;”,
                “Age”:20}

      print(map[‘name’])

Functions

- Function overloading not supported (same function name, but different parameters)
- Example
    - Void sum()
    - Void sum(int a)
- Dart supports optional parameters

    - Void sum({int? A, int? B}){ }
- Dart supports required parameters
    - Void sum(required int a, int b) { }
- Optional + required
    - Void sum({required int a, int? b}) { }
- Default value
    - Void sum ({required int a, int b = 0}) { }

- Future function
    - To call a function after some times or function completion is not fixed
    - **Await** is only used with future function
    - future<int> sum(int a) async { }
    - Await future,delayed(Duration(seconds :3))

Final

- Can assign only once
- Example
    - List as final, we can add, remove values, but cannot add other list
    - Final list<int> num = List.empty();
    - num.add(1);
    - Num = [10,20] (not possible)
- Final string name;
- Name = “some name”

Constant

- Compile time constant
- Must initialize at declaring variable
- Example
    - Const string name = “some name”
    - Const string name = “”; (not possible)

Class

- First word of the class must be capital
- Example

```
Class Person{
        String name = “joyal”,
        Int age = 25, }
Final person = Person()
        //getting value
         print( person.name)
         print(person.age)

        //setting value
```

Person.name = “joyal”

- constructor

```
Class person{
        String name;
        Person(String name) {
                this.name= name;
        }
```

- Final in class

```
Class person{
        Final sting name;
Person(this.name);
}

person = Person(“joe”);
person.name = “joyal” (not possible)
```

- Getter and setter

```
Person{
        String _name;

        Void getName(){
                Return _name;
        }

        Void setName(string name){
                _name = name;
        }
}

// person,setName(“joyal”);
```

Inheritance

- Dart doesn't support multiple inheritance
- Dart support multilevel inheritance (using mixins)
- Example

```
class Animal {
  void sayAnimal() {}
}

class Human extends Animal {
  void sayHuman() {}
}
```

```
final human = Human();
human.sayAnimal()
human.sayHuman();

Override method
class Animal {
  void sayAnimal() {}
}

class Human extends Animal {
  void sayHuman() {}

  @override
  void sayAnimal(){
    super.sayAnimal();
  }
}
```

Abstract

- Cannot create objects
- Only can override or inherent
- Example
  - Abstract class Animal{ }

Interface

- Only has declaration
- Not definition
- Using **implements** keyword

```
abstract class Animal {
  void sayAnimal() {}
}

class Human implements Animal {
  void sayHuman() {}

  @override
  void sayAnimal(){

  }
}
```

Mixin

- Enables multiple inheritance

- Example

```
mixin Animal1 {
  void sayAnimal() {}
}

mixin Animal2 {
  void sayAnimal() {}
}

class Human with Animal1, Animal2 { }
```

Flutter

Project structure

- Dart tools, idea = environment files
- Pubspec.yaml = add libs
- Analysis-options = rules for code

Scaffold

- A screen, or to define a screen
- It has 2 parts (header, body)
- Header is appbar (optional)
- Safe area -> status bar

Container

- Width : double.infinate
- Height: double.infinate

Column

- Downward flow (vertical)
- Cannot apply color
- mainAxies -> top to bottom
  Item1
  Item 2

Row

- Horizontal flow
- Cannot apply color
- mainAxis -> left to right
  Item1, item2

Child

- Only one widget

Children

- Multiple widgets

- Widget declared Inside an array []

Button

- Textbutton
- Elevated button
- Iconbutton (icon:Icon(Icon.info))

Box Decoration

- Color
- Border radius (circular(5))
- Border (border.all)

Textfield

- Input decoration
- Border (outlineInputBorder)
- hintText

Padding

- padding(edgeInset.all(10))
- Cannot apply color
- edgeInset.only(left:20)

Textfield

- Final _textcontroller = TextEditingController();
- _textcontroller.text

State

- What value UI holds
- Example
    - Switch has true/false, so true/false holds UI value/state

StateLess

- Widget has no value
- Not changeable, static value

StateFull

- Dynamic
- It has a state

setState

- Used to update UI
- It is only used with statefull widget
- Example

    setState(() { })

List

- List.generate(100, (index) => Text(“hi”))

Divider

- Create a divider line

ListView

- Separated listview
  - It has a separator between items
  - It has 3 parameters
    - Itembuilder (build context, index)(items)
    - Seperatorbuild (build context, index) (separator)
    - Itemcount
- Listview builder
  - No separator
  - Same as separated
  - Effective (only load visible data, mask other data)

ListTile

- It has title, subtitle, leading widget, trailing widget

Circle avatar

- Display image in circular format
- widget-> CircleAvatar()

Image Loading

- Network image
  - NetworkImage(‘url’)
- Assets
  - Create a folder named ‘assets’ (images, fonts, gifs)

Making listview

- Widget -> **ListTile**()
- Example

```
ListView.separated(
    itemBuilder: (context, index) {
    return ListTile(
    title: Text("List Tile $index"),
    onTap: () {});
    },
    );
```

```
},
separatorBuilder: (context, index) {
return Divider();
},
itemCount: 10,
)),
```

Navigation

- Navigation can done using navigator class (**Navigator**)
- Or it can be done using **routes** defined in main dart file
- Example

```
Navigator.of(context).push(MaterialPageRoute(builder: (context) {
   return ScreenThree( ); }
or
  Navigator.of(context).push(MaterialPageRoute(builder: (context) =>
LoginScreen()));
```

Using routes (under home:)

```
routes: {
'screen_1': (context) {
 return ScreenOne();
},
'screen_2': (context) {
 return ScreenTwo();
}
},
```

Arguments to page

- Add parameter in constructor
- https://github.com/joyal670/Flutter_hive (ScreenThree)
- Example

```
final String name;
const ScreenThree({super.key, required this.name});
```

Shared preference

- Add dependency using https://pub.dev/
- Declare and Initialize shared preference
- Shared pref is a **future** function
- Example

```
late SharedPreferences preferences;
main() async {
  WidgetsFlutterBinding.ensureInitialized();
  preferences = await SharedPreferences.getInstance();
  runApp(const MyApp());
}

await preferences.setString("key", “value”);
final savedValue = preferences.getString("key");
```

Text Form

- Widget -> **TextFormField**
- Inorder to get text from textform, we need to use controller
- Example

```
final _textController = TextEditingController();
 TextFormField(
  controller: _textController,
),
print(_textController.text);
```

Images

- Add assets folder
- Add image path in pubspec.yaml
- Example

```
Image.asset('assets/images/unnamed.jpg'),
Image.asset(
 'assets/images/unnamed.jpg',
        height: 200,
       width: 200,
 ),
```

Life cycle

- createState()
  - This method is called when we create another Stateful Widget.

```
class HomeScreen extends StatefulWidget {

  HomeScreen({Key key}) : super(key: key);

  @override
```

```
  HomeScreenState<StatefulWidget> createState() =>
HomeScreen();
}
```

- initState()
  - it is called precisely once for each State object. If we characterize or add some code in the initState() method this code will execute first even before the widgets are being built.
  - This method needs to call super.initState() which essentially calls the initState of the parent widget (Stateful widget).

```
@override
void initState(){
        super.initState();
}
```

- didChangeDependencies()
  - This method is called following the initState() method whenever the widget initially is constructed.

```
@override
void didChangeDependencies() {
}
```

- build()
  - This strategy is the main method as the rendering of all the widgets.
  - It is called each time when we need to render the UI Widgets on the screen.

```
@override
Widget build(BuildContext context) {
  return Scaffold()
}
```

- didUpdateWidget()
  - This strategy is utilized when there is some adjustment of the configuration by the Parent widget.
  - It is essentially called each time we hot reload the application for survey the updates made to the widget

```
@protected
void didUpdateWidget(Home oldWidget) {
  super.didUpdateWidget(oldWidget);
}
```

- setState()

- The setState() method illuminates the framework that the internal state of this item has changed in a manner that may affect the UI which makes the structure plan a build for this State of the object.

```
void function(){
  setState(() {});
}
```

- deactivate()
  - This method is considered when the State is removed out from the tree, however, this strategy can additionally be re-embedded into the tree in another part.

```
@override
void deactivate(){
  super.deactivate();
}
```

- deactivate()
  - It is considered when the object and its State should be eliminated from the Widget Tree forever and won't ever assemble again.

```
@override
void dispose(){
  super.dispose();
}
```

Splash Screen

- Called inside async function(future class)
- Example

```
navigateToLoginPage() async {
  Future.delayed(Duration(seconds: 3));
}

Future<void> navigateToLoginPage() async {
  Future.delayed(Duration.zero, () {
    Navigator.of(context).push(MaterialPageRoute(builder: (context) {
      return LoginScreen();
    }));
  });
}
```

Snackbar

- Inorder to use snack bar outside the scaffold, we must use **ScaffoldMessenger.of(context)**
- Example

```
ScaffoldMessenger.of(context).showSnackBar(SnackBar(
  content: Text("Password not match"),
  margin: EdgeInsets.all(8),
  behavior: SnackBarBehavior.floating,
  backgroundColor: Colors.red,
  duration: Duration(seconds: 10),
));
```

Alert Dialog

- Two types of dialog are there,
  - Simple dialog (customized dialogs)
  - Alert dialog (yes or no button, or 3 buttons)
- Example

```
showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text('errot'),
        content: Text('wsfdf'),
        actions: [
          TextButton(
              onPressed: () {
                Navigator.of(context).pop();
              },
              child: Text('close'))
        ],
      );
    });
```

UI update/notify

- We can notify the UI, for data change either by using setState() or valueListenableBuilder
- setState() is used with stateFullWidget
- valueListenableBuilder is used with stateLessWidget
- Widget -> valueListenableBuilder(

```
valueListenble: ,
Builder: (BuildContext ctx, int newValue, Widget? child){
Return Text();
```

Bottom sheet

- Widget -> **showModalBottomSheet()**
- Example

```
showModalBottomSheet(
    context: context,
```

```
builder: (context) {
  return Container(
    width: double.infinity,
    height: 500,
    color: Colors.red,
    child: ListView(
      children: [
        Text("Title"),
        TextButton(
          onPressed: () {
            Navigator.of(context).pop();
          },
          child: Text('Close'))
      ],
    ),
  );
});
```

Bottom navigation

- Widget -> BottomNavigationBar
- Example

```
int _currentIndex = 0;
final _pages = [HomeScreen(), SearchScreen(), AccountScreen()];

body: _pages[_currentIndex],
bottomNavigationBar: BottomNavigationBar(
  currentIndex: _currentIndex,
  selectedItemColor: Colors.blue,
  unselectedItemColor: Colors.grey,
  onTap: (newIndex) {
    setState(() {
      _currentIndex = newIndex;
    });
  },
  items: const [
    BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),
    BottomNavigationBarItem(icon: Icon(Icons.person), label: 'Account'),
    BottomNavigationBarItem(icon: Icon(Icons.search), label: 'Search'),
  ],
),
```

Map

- Creating a list from other list

Drop down(spinner)

- Widget -> DropdownButtonFormField
- Example

```
DropdownButtonFormField(
        hint: const Text('Select fruits'),
        onChanged: (value) {},
        items: _list.map((String items) {
          return DropdownMenuItem(
            value: items,
            child: Text(items),
          );
        }).toList(),
      ),
```

Model class and adapter

- Create a model class

```
class UserModel {
  final String userName;
  final String password;
 UserModel({required this.userName, required this.password});
}
```

- Add adapter class (for CURD operations)

```
ValueNotifier<List<UserModel>> userModelListener = ValueNotifier([]);
void addUser(UserModel model) {
        userModelListener.value.add(model);
         print(model.userName);
         userModelListener.notifyListeners();
}
```

- Listing

```
SafeArea(
    child: ValueListenableBuilder(
      valueListenable: userModelListener,
      builder: (context, value, child) {
        return ListView.separated(
          itemBuilder: (context, index) {
            final data = value[index];
            return ListTile(
              title: Text(data.userName),
              subtitle: Text(data.password),
              onTap: () {},
            );
```

```
        },
        separatorBuilder: (context, index) {
          return const Divider();
        },
        itemCount: value.length);
    },
  ),
);
```

Hive

- Local db in flutter
- Add dependency (builder and generator)
- Hive type id is just like primary key for table (identify the table, with ID)
- Fields are annotate with @HiveField
- Every table is considered as box
- Hive stores its data in boxes containing key-value sets.
- We must open box, before performing any option
    - Example
        - final studentTable = await Hive.openBox<UserModel>('student_db');
- Value ValueNotifier are used for notifying changes
- hive: ^2.0.4
- hive_flutter: ^1.1.0

- hive_generator: ^2.0.0
- build_runner: ^2.1.5

- **flutter packages pub run build_runner build**
- Example
    - https://github.com/joyal670/Flutter_hive
    - https://medium.flutterdevs.com/hive-database-with-typeadapter-in-flutter-7390d0e515fa

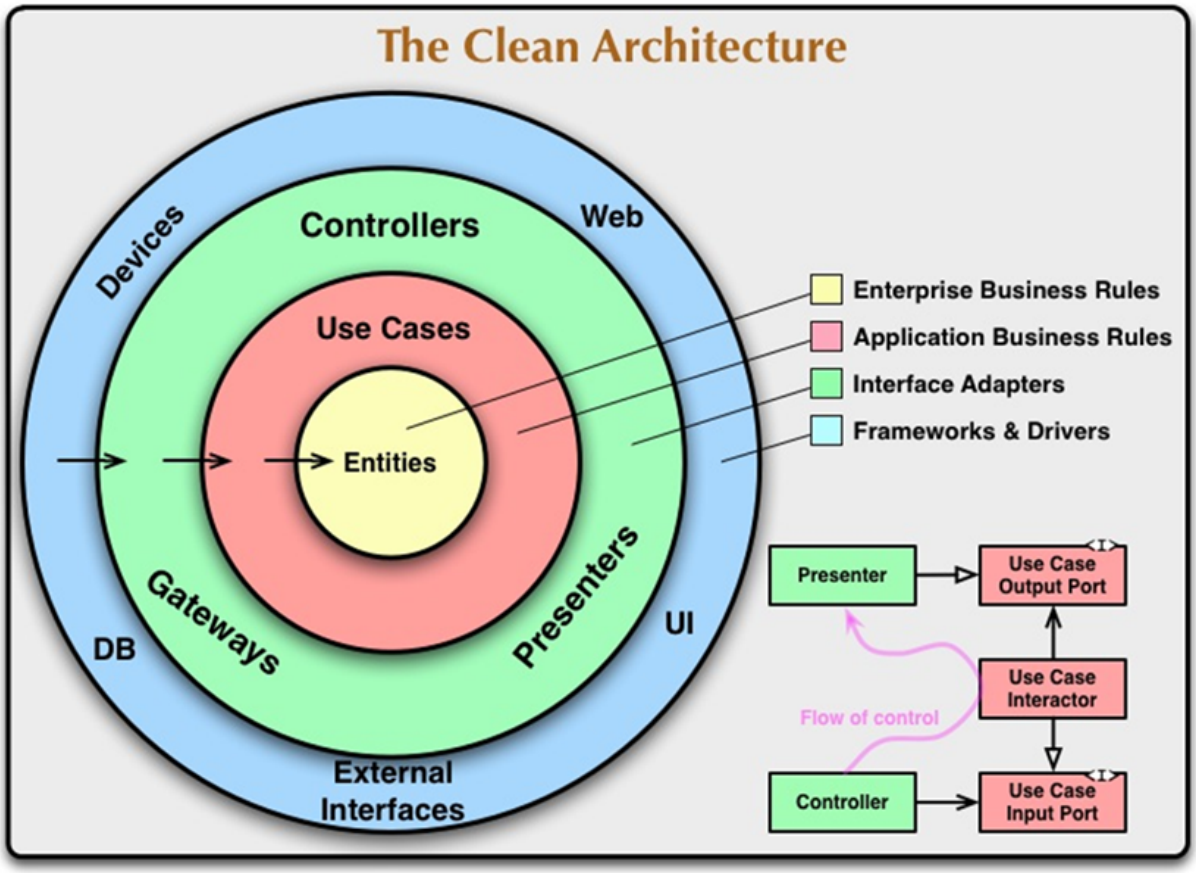
SQFlite

- It store data as map
- Hive store data as class(model class)
- Example
    - https://github.com/joyal670/Flutter_hive/tree/db

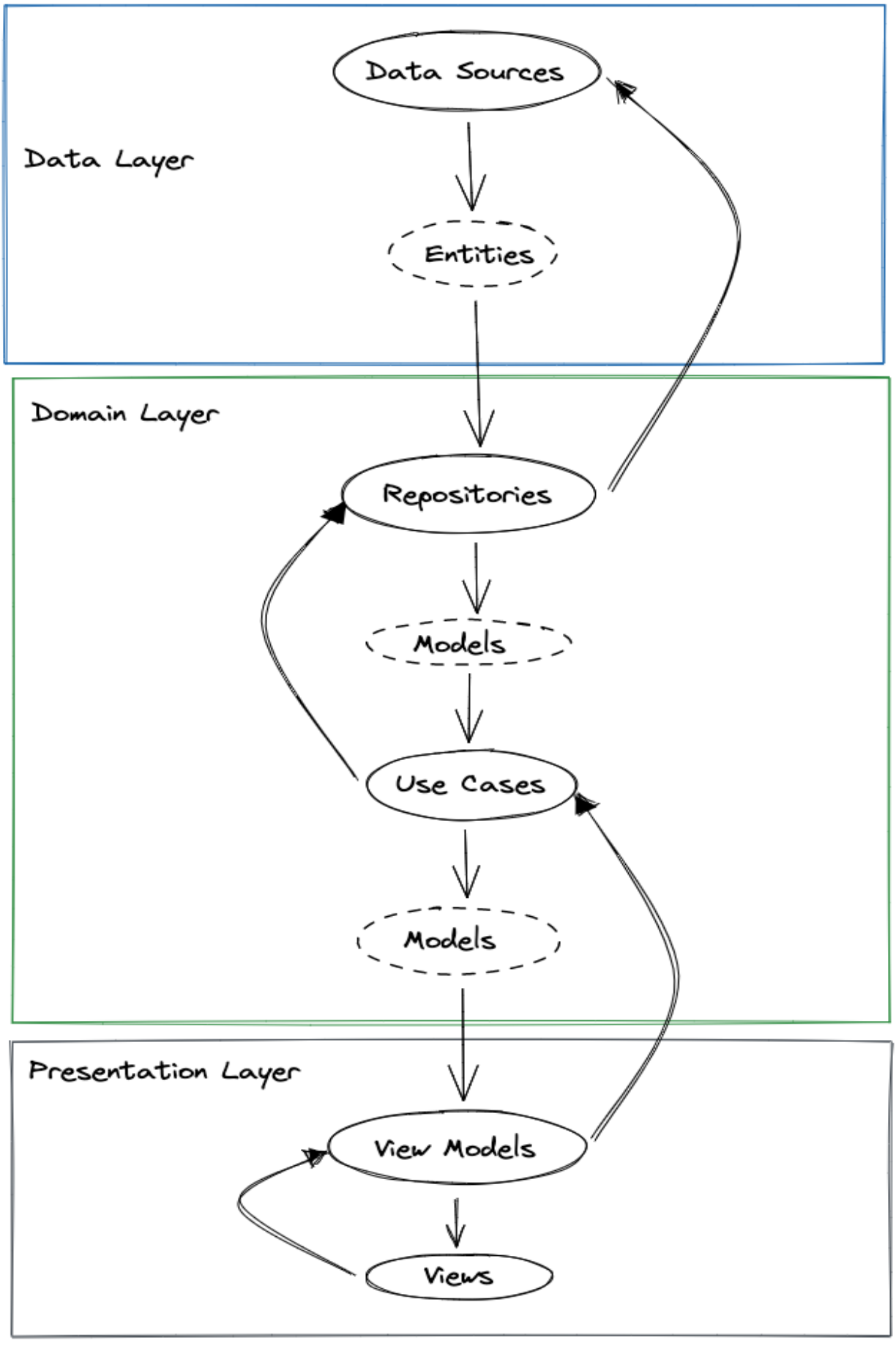
Bottom navigation/ Tab Layout

- https://github.com/joyal670/Flutter_money_management

Flutter clean architecture



- "Clean Architecture" was coined by Robert C Martin
- is a software design philosophy that organizes code in such a way that business logic is kept separate from technical implementation (databases, APIs, frameworks).
- Three layers
- Data layer, domain layer and presentation layer,
-


Data Layer
Data Sources
Entities
Domain Layer
Repositories
Models
Use Cases
Models
Presentation Layer
View Models
Views

- Models
    - Domain models represent real-world objects that are related to the problem or domain space

Firebase

- Dependency
    - firebase_database: ^10.2.2
    - firebase_core: ^2.13.1
- Add android and ios google.json file
    - For ios, it in ios/Runner package
    - For android, android/app/src
    - Add app and project level dependency
- Init firebase before runApp() function

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const MyApp());
}
```

- Initialize firebase usingFutureBuilder
- DatabaseReference has 2 methods,
    - Listen() - listen to database changes
    - get() - only listen to once
- Adding data to firebase
    - databaseReference.child(“”).set(value)
- Removing data from firebase
    - databaseReference.child(“”).remove();

State management

- State management packages like provider, riverpod, mobex, getx, bloc
- Used to manage the state of a widget, which reduces to rebuild the widget entirely
- Bloc
    - Event
        - Ui changes, actions, buttons clicks etc
    - Main bloc
    - State
        - a value that corresponding widget holds
- Add plugin called bloc in vs code
    - Right click - bloc- add new bloc
    - Calling events, 2 methods
        - context.read<CounterBloc>.add(IncrementEvent());
        - BlocProvider.of<CounterBloc>(context).add(IncrementEvent());