<\*\*\* flutter \*\*\* >

1. outline 안보일때 ctrl 누르고 widget 클릭

2. ctrl + alt + L : 자동 들여쓰기

=> 와 {} 는 같은 기능을 한다. 함수호출에 사용

3. Dynamic : 변수에 어떤값이 올지 모를 때 사용함. 할당된 변수값의 종류에 영향없이 타입이 변경된다.

4. Initstate : 위젯의 생성자 코드 실행이후 처음으로 실행되는 메서드

-만약 구동이 에러가 없는데 구동 안되는거면 애뮬레이터 용량문제일수 있다. ADB 삭제 후 다시 만들거나 edit 들어가서 internal Storage를 4096으로 늘린다.

File -> project counstruct -> project JDK -> android studio default

5. scaffold 의미 : 앱의 시작을 알리는 위젯. 모두다 여기서 시작함

6. 변수 생성시 초기화로 late 해줘야 하는 이유 -> 플러터는 null 값을 구분해 줘야함

7. required 의미 : 꼭 써줘야 할 변수에 적용 시킴

8. final과 const 를 사용하는 이유:

변경하지 못하게 하는 것도 있지만, 컴파일 시 매번 컴파일 하지 않아도 되는 내용이라고 알려줘서 더 빠르게 컴파일을 할 수 있게 만들기 위함이다.

9. Scaffold( body: SafeArea(...

-> SafeArea는 안전영역을 의미하며, 앱 바에서 표시되지 않고, 앱 바의 밑을 영역으로 사용한다.

10. stateless vs stateful

1. stateless: 위젯이 리로드될때마다 호출, 변하지 않는 데이터를 표시
   1. 정적인 위젯으로 , 상호작용을 하지 않고, 데이터가 화면에 고정되어 있는 것
   2. 화면이 로드될 때 한번만 그려지는 state가 없는 위젯, 변경이 필요하지 않은 data로 구성되어있어 이벤트 또는 사용자와 상호작용하지 않는다.
2. stateful : 위젯이 초기화할떄 한번만 호출 ( 페이지 이동할때마다 초기화 일어나는것임, 하지만 페이지의 내용이 변경되는것에 대해서는 반응하지 않는다. )
   1. 데이터가 전송되어서 화면에 영향을 줄 경우, 버튼을 누르거나 행동을 취했을 떄 사용 -> 사용자의 사용에 따라 변경하는 것
   2. 위젯이 동작하는 동안 data변경이 필요한 경우 사용, 화면을 다시 그려서 변경된 부분을 위젯에 반영하는 동적인 위젯

ex) dispose(), bulid(), initstate(실행될 때 가장 먼저 실행되는 함수 )

11. 데이터를 가져올 때 비동기식이 되면 안된다. 왜냐하면 데이터를 입력하고 가져오는 시간보다 출력시간이 빠르면 데이터가 없는것으로 간주하기 때문이다.

ex)void getData() async{ String name = await Future.delayed(Duration(seconds : 3), (){ return 'won';});

=> name 변수에 won을 가져오고 3초간 기다린다.

12. json 정보 크롤링 해오기

1. get('주소') 메소드 이용해서 Response객체로 받는다. -> 주소 크롤링 하기 위해서는 pubspec.yaml 에다가 http 추가하고, import로 (package: http/http.dart ) 를 가져온다.

2. 가져온 Response 객체는 문자열 형식으로, map 형식으로 바꿔서 사용해야 한다.

3. Map data = jsonDecode(response.body); -> jsonDecode사용하기 위해서는 ( import 'dart: convert') 해줘야함

4. print(data['title'])

13. Future<타입> : 지금은 없지만 미래에 요청한 데이터 혹은 에러가 담길 그릇 -> 해당 타입이 나온다는 것을 미리 알려주는 것으로, 비동기 함수에서 사용된다.

14. state란 앱에서 사용되는 data를 의미

1. app state : 앱 전반에 걸쳐 사용되는 data

2. widget state : 위젯 내부에서만 사용되는 data

3. statelessWidget : 변경될 data가 없다.

4. setState( () {} ) => 부모 state를 호출한다. stateful 에서만 사용됨

15. 정렬 Row와 Column 의 정렬

1. Row일때는 주축이 mainAxisAlignment 가 되서 가로를 기준으로 center에 정렬이 된다.

2. Column일 경우 주축이 세로가 되서 세로화면을 기준으로 center로 정렬된다.

1. 위젯 : 독립적으로 실행되는 작은 프로그램으로, 주로 바탕화면 등에서 날씨를 나타냄
   1. Flutter는 모든 것이 위젯으로 이루어져 있어서, 위젯을 많이 알수록 많은 기능을 사용할 수 있다.
2. 실행순서 : Myapp -> MaterialApp -> MyHomePage -> Scaffold -> Appvar or center 등등
3. 스케폴드 안에서 화면 구성이 이루어진다.
4. Children vs child
   1. Children : 여러개의 자식위젯을 가질 경우
   2. Child : 한가지의 위젯을 사용할 경우
5. Container : 하나로 묶는 역할을 함 -> row나 column같이 텍스트 배치에 유용
6. Pageview : 메인화면에서 화면 넘길 때 사용하는 것
7. sizedBox : 다음 컨텐츠 표시할 때 공백을 주기 위해서 사용
8. InkWell : onTap() 을 이용해서 눌려졌을때 어디로 이동하게 만듬
9. SingleChildScrollView : 스크롤 방향을 옆으로 줄 수 있는 것
10. Drawer : 메뉴바에 만들기
11. 폰트 스타일은 dart파일에 저장해서 import로 불러온다. -> color: Color(그냥 변수로 사용하면 됨 )
12. BottomNavigationBar : 밑에 네비게이션 바 생성
13. Required: 반드시 기록이 되어야 하는 변수를 만들 때
14. 스크롤 기능 : ListView
15. Context 의미
    1. Build 함수는 Scaffold 함수를 리턴할 때 widget tree상에서 어디에 위치하는지에 대한 정보를 context라는 변수에 넣어서 리턴해준다.

<FireBase 연동 시작 >

1. Firebase의 data type은 Map이다.
2. Firebase는 컬렉션 형태로 저장되며, 이 컬랙션은 document(문서) 형태로 저장된다.
   1. 컬랙션안에 문서(doc) 이 있고, 문서안에 필드가 있다. 필드안에 데이터가 저장된다.
3. Provider -> stream하면서 사용 => 데이터의 통로 같은 것, 데이터 전역에서 사용하고자 할 경우 이용
4. Pub.dev 에서 install 할것들
   1. firebase\_auth -> firebase auth 이용시 기본설정
   2. cloud\_firestore. -> firebase store 이용시 기본설정
   3. 연동한것들을 pubspec.yaml에 추가해줌
5. Firebase 에 프로젝트를 생성한다.
6. Android 문양을 선택한다.
7. Android 패키지이름은 파일에서 만든 패키지 이름으로 사용한다. ->
   1. android studio ->android디렉토리 -> app -> build.gradle -> applicationId 에서 확인할 수 있다.
      1. Build.gradle 열면 에러남. -> <에러 발생시 처리 방법> 으로 이동
8. 구성파일 다운로드 -> android studio 프로젝트의 app 디렉토리에 추가
9. apply plugin: 'com.google.gms.google-services' 과 같은 형태를 복사해서 android studio -> 이번엔 app 안에 있는 build.gradle이 아닌 android 디렉토리에 있는 build.gradle 선택 -> classpath 밑에 추가
10. applicationID “com.xxx~” 의 형태인 것을 firebase프로젝트 생성시 기입한다.

<에러 발생시 처리 방법 >

1. Android 패키지이름은 파일에서 만든 패키지 이름으로 사용한다. ->
   1. android studio ->android디렉토리 -> app -> build.gradle -> applicationId 에서 확인할 수 있다.
   2. Build.gradle 파일열면 “cannot resolve symbol properties” 에러 뜸
      1. 해결방법 :
         1. File -> Invalidate Chache/Restart. -> Invalidate Chache Restart. 눌러서 android studio 재시작
         2. build.gradle 파일 닫고(열려있으면 또 에러남. 꼭 닫아야함 ) -> Tools -> flutter -> open for editing in android studio 클릭 -> 실행 후 file-> open -> 실행하고자하는 프로젝트 실행
2. 만약 android -> app -> build.gradle 에서 minSdkVersion(최소지원안드로이드sdk버전)이 21보다 낮다면 에러가 날것이다. 따라서 수정해줘야 한다.
   1. minSdkVersion 21 과 targetSdkVersion 28 으로 변경해준다.
   2. 그 밑에 multiDexEnabled true 를 추가해준다.
3. Cannot run with sound null safety, because the following dependencies : don't support null safety: - package:flutter\_spinkit => 이 에러는null값 때문에 생기는 에러임 따라서 셋팅해줘야함
   1. Run -> Edit configurations -> Additional run args 에 --no-sound-null-safety 을 추가한다.
4. main에서 runApp전에 await Firebase.initializeApp() 을 해준다. Firebase 사용하기위해 먼저 초기화하는 단계

<로그인 폼>

1. 로그인을 편하게 하는 방법 -> firebase사이트 -> 빌드 -> authentication -> start -> 허용할 로그인 방법을 체크

2. AuthService클래스 생성해서 firebase\_auth 패키지 import

3. UserCredential 로 로그인 체크 -> 이때 시간이 많이 걸리니 async 이용해서 비동기식으로 가야함

4. Streams : 데이터나 이벤트가 들어오는 통로이다. (비동기 이벤트)

* 앱을 만들다보면 데이터 처리할 일이 많은데, 어느 타이밍에 데이터가 들어올지 확실히 알기가 어렵다.
* Firestore에서의 변경사항을 즉각적으로 반응한다.
* Snapshot : 순간순간 업데이트 된 data 라고 생각하면 됨
* Ex ) 네트워크에서 데이터를 받아와서 ui에 보여주는 상황일경우, 속도차이로인해 언제 데이터를 다 받을지 모른다. 이때 스트림은 데이터를 만드는 곳과 소비하는 곳을 따로둬서 문제를 해결
* 순서 : 데이터 만들기(변경) -> 연결 -> 데이터 처리
* Map형식을 사용해서 다룰 수 있다.
* StreamProvider vs SteamBuilder
  + StreamBuilder is a widget that comes with Flutter, and rebuilds itself every time the stream gets updated, that's it's only job.
  + StreamProvider is a widget that comes with the [Provider package](https://pub.dev/packages/provider), it's built using StreamBuilder, but combines this with [InheritedWidget](https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html), allowing you to pass information through the tree of widgets efficiently
  + Streambuilder() : 스트림에서 이벤트가 나올때마다 문장을 재생성하여 가장 최신의 이벤트로 작업할 수 있게 해준다.
    - Stream : 스트림 빌더에 쓰일 스트림을 정한다.
    - Builder : 스트림 데이터를 받아 UI를 그리는 일을 한다.
  + StreamProvicer :
* streamController() : 여러 스트림을 관리하는 것으로, 데이터가 추가되거나 이벤트 종료시 실행된다.

5. Provider : 복수의 화면에서 데이터를 공유할 수 있게 하는 것 -> 전역에서 쓰기 위해서는 MaterialApp 감싸면됨

* 모든 화면에서 데이터의 공유가 가능하게 하고 더 나아가 데이터의 변경이 발생하였을 시 해당 데이터를 참조하는 화면구성
* 2개로 나눠진다. 데이터를 생산하는 곳, 데이터를 소비하는곳 (provider: 데이터 소비(가져오는 곳), StreamProvider: 데이터 생성)
* 형태 : 데이터 생성 : streamprovider<type>.value( value:생성할 값, child: Container(),)
* 형태 : 데이터 소비 : var data Provider.of<type>(context)

1. TextFormField : 입력할수 있는 text를 만들어준다.
   1. Decoration : InputDecoration() : 텍스트 필드를 꾸며준다.
      1. 하나의 클래스를 만들어서 공통으로 스타일을 적용
   2. Validator : 유효성검사
   3. onChanged : 내용이 입력됐을 경우 setState로 전역변수에 값 전달
   4. decoration: InputDecoration( 꾸밀 내용).copyWith(hintText: ‘아메리카노’),
2. 유효성 검사 : 로그인시 이메일이 맞는지, 글자수 맞는지 판단가능
   1. Validate() 메소드 이용
   2. 비밀번호 \* 처리 하는 방법 : obscureText : true,
3. FirebaseAuth 에서는 User에 대한 변화를 stream을 통해 전송한다.
   1. UserCredential : firebase\_auth에 있는 객체로, firebase의 객체를 받아올 때 사용
   2. createUserWithEmailAndPassword( email: email, password: password) : 신규 사용자를 생성
   3. signInWithEmailAndPassword( email: email, password: password) : 생성된 사용자인지 확인
4. firestore 이용
   1. 데이터를 컬렉션으로 저장하는 저장소 -> db와 같다고 보면 됨
   2. 사용자가 회원가입할 때 생성되는 uid를 이용해서 firestore에 저장한다.
   3. 저장형태는 databaseService 클래스를 만들고, 그 클래스의 생성자로 uid를 넘긴다.
   4. DatabaseService 클래스는 컬랙션을 만들고, CollectionReference 와 FirebaseFirestore 객체 이용
   5. 생성자로 받아온 uid를 이용해 같이 넘어온 매개변수로 고유한 사용자 정보들을 저장한다.
   6. (생성한컬랙션명).doc(uid).set({ ‘저장명’: 매개변수1, ‘저장명’: 매개변수2})
5. Snapshots() : stream 데이터를 얻을 때 사용
6. ListTile 위젯이용해서 화면에 타일 모양으로 표시
   1. Leading : CirecleAvatar
   2. Title : 첫 줄 텍스트
   3. subtitle : 2번째줄 텍스트
7. GlobalKey : Form의 ID값으로 사용된다.
   1. \_formKey.currentState.validate() 의 형태로 사용되며 폼 내부의 모든 자식들의 validator가 실행
   2. 위젯이 상태를 잃지 않으면서 부모를 바꿀 수 있도록 한다.
   3. 특정 위젯의 정보를 완전히 다른 위젯 트리에서 접근 가능하게 해줄 수 있다. -> 비밀번호검사에 사용
8. Firestore 에서 가져오는 형태가 map이기 때문에 .toList()로 해줘야 한다.
9. DropdownButtonFormField() : 드롭다운 형태로 나오게 함
10. Slider : 슬라이드 형식으로 값을 표현 , 색을 이용해서 0이면 연하게, max면 진하게 설정가능
11. \_currentStrength ?? userData.strength : ??는 널 연산자로, 널이 아니면 \_currentStrength에다가 userDatastrength를 넣어준다.

< web - Firebase 문법 >

1. Firebase 에서 데이터 가져오기
   1. App.js에서 Const db = firebase.firestore(); => db.setting({timestampInSnapshots : true}); => db를 셋팅한다.
   2. db.collection(‘cafes’).get().then( (snapshot) => { snapshot.docs.forEach(doc => { console.log( doc.data()) }) })
      1. doc(문서)의 형태로 가져와서 data 해줘야함
      2. then()은 데이터를 가공해서 가져오기 위함이다.
      3. 이때 doc.id 이 문서를 뜻하고, doc.data().name 이 해당 문서안의 필드에 있는 정보를 뜻한다.
   3. 조건이용
      1. Db.collection(‘cafes’).where(‘name’, ‘==’, ‘wonyoung’).get().then( (snapshot) => {….});
   4. 정렬해서 가져오기 => 정렬하려면 firebase에서 index 설정을 해줘야 오류가 안난다.
      1. DB.collection(‘cafes’).where( 조건).orderBy(‘name’).get().then(…);
2. fireBase에 데이터 저장
   1. db.collection(‘cafes’).add({ name : form.name.value });
3. firebase 데이터 삭제
   1. db.collection(‘cafes’).doc(id).delete();
      1. Collection.doc()은 단일 문서를 찾는 역할을 한다. -> id를 지우면 해당 문서가 지워져서 안에있는 데이터(필드)들이 삭제된다 .
4. Firebase의 내용을 실시간으로 반영
   * 1. db.collection('cafes').orderBy('city').onSnapshot(snapshot => {

let changes = snapshot.docChanges();

changes.forEach(change => {

if(change.type == ‘added’){

첫 화면 호출}

Else if(change.type == ‘removed’){

삭제할 곳

1. firebase 업데이트
   1. db.collection(‘cafes’).doc( 해당문서ID).update({ name: ‘change\_name’});
   2. db.collection(‘cafes’).doc( 해당문서ID).set ({ name: ‘change\_name’}); => 업데이트가 아니라 새로 정의하는것, 이때 해당 문서 안의 데이터가 name 빼고 다 지워진다.