Todo 앱
- http 통신

2 Flutter에서 제공하는 통신의 종류

Flutter 통신 패키지 종류

1. **Http** (Rest API)
2. **gRPC (**Google Remote Procedure Call)
3. **GraphQL** (Graph Query Language - by Facebook)
4. **web socket**
5. **Firebase** (by Google)

1. Http
   - http1: dio, getX, chopper
   - http2: dio + dio_http2_adapter
   - dio support: retrofit

1. Http
   - http1: dio, getX, chopper
   - http2: dio + dio_http2_adapter
   - dio support: retrofit

- Dio (https://github.com/thecodexhub/flutter-dio-example)

```
class DioClient {
  DioClient()
      : _dio = Dio(
          BaseOptions(
            baseUrl: Endpoints.baseURL,
            connectTimeout: Endpoints.connectionTimeout,
            receiveTimeout: Endpoints.receiveTimeout,
            responseType: ResponseType.json,
          ),
        )..interceptors.addAll([
            AuthorizationInterceptor(),
            LoggerInterceptor(),
          ]);
```

```
Future<User?> createUser({required User user}) async {
  try {
    final response = await _dio.post(Endpoints.users, data: user.toJson());
    return User.fromJson(response.data);
  } on DioError catch (err) {
    final errorMessage = DioException.fromDioError(err).toString();
    throw errorMessage;
  } catch (e) {
    if (kDebugMode) print(e);
    throw e.toString();
  }
}

Future<void> deleteUser({required int id}) async {
  try {
    await _dio.delete('${Endpoints.users}/$id');
  } on DioError catch (err) {
    final errorMessage = DioException.fromDioError(err).toString();
    throw errorMessage;
  } catch (e) {
    if (kDebugMode) print(e);
    throw e.toString();
  }
}
```

- Dio (https://github.com/thecodexhub/flutter-dio-example)

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

// ignore: constant_identifier_names
const String API_KEY =
    'cdc9a8ca8aa17b6bed3aa3611a835105bbb4632514d7ca8cf55dbbc5966a7cae';

//* Request methods PUT, POST, PATCH, DELETE needs access token,
//* which needs to be passed with "Authorization" header as Bearer token.
class AuthorizationInterceptor extends Interceptor {
  @override
  void onRequest(RequestOptions options, RequestInterceptorHandler handler)
    if (_needAuthorizationHeader(options)) {
      options.headers['Authorization'] = 'Bearer $API_KEY';
    }
    super.onRequest(options, handler);
  }

  bool _needAuthorizationHeader(RequestOptions options) {
    if (options.method == 'GET') {
      return false;
    } else {
      return true;
    }
  }
}
```

```dart
class LoggerInterceptor extends Interceptor {
  @override
  void onError(DioError err, ErrorInterceptorHandler handler) {
    final options = err.requestOptions;
    final requestPath = '${options.baseUrl}${options.path}';
    logger.e('${options.method} request => $requestPath');
    logger.d('Error: ${err.error}, Message: ${err.message}');
    return super.onError(err, handler);
  }

  @override
  void onRequest(RequestOptions options, RequestInterceptorHandler handler) {
    final requestPath = '${options.baseUrl}${options.path}';
    logger.i('${options.method} request => $requestPath');
    return super.onRequest(options, handler);
  }

  @override
  void onResponse(Response response, ResponseInterceptorHandler handler) {
    logger.d('StatusCode: ${response.statusCode}, Data: ${response.data}');
    return super.onResponse(response, handler);
  }
}
```

- Dio + Retrofit

```dart
@RestApi(baseUrl: 'http://localhost:8080/')
abstract class TodoClient {
  factory TodoClient(Dio dio, {String? baseUrl}) {
    return _TodoClient(dio);
  }

  @GET('/todos')
  Future<List<Todo>> getTodoList();

  @POST('/todos')
  Future<void> addTodo(@Body() Todo todo);

  @PUT('/todos/{id}')
  Future<void> updateTodo(@Path("id") int id, @Body() Todo todo);

  @DELETE('/todos/{id}')
  @Headers(<String, dynamic>{
    "Content-Type": "text/plain",
    "Content-Length": 0,
  })
  Future<void> removeTodo(@Path("id") int id);
}
```

- GetX (https://github.com/flutter-devs/rest_api_getx)

```
class CityProvider extends GetConnect {
  @override
  void onInit() {
    // All request will pass to jsonEncode so CasesModel.fromJson()
    httpClient.defaultDecoder = CityModel.listFromJson;
    httpClient.addRequestModifier((request) {
      request.headers['Authorization'] = 'Bearer sdfsdfgsdfsdsdf12345678';
      return request;
    });
  }
  Future<Response<List<CityModel>>> getCity() => get<List<CityModel>>(
      'https://servicodados.ibge.gov.br/api/v1/localidades/estados'
  );

  Future<Response<CityModel>> postCity(body) =>
      post<CityModel>('http://192.168.0.101:3000/items', body,
          decoder: (obj) => CityModel.fromJson(obj));
}
```

- chopper (https://github.com/lejard-h/chopper/blob/develop/chopper/example/definition.dart)

```dart
@ChopperApi(baseUrl: '/resources')
abstract class MyService extends ChopperService {
  static MyService create(ChopperClient client) => _$MyService(client);

  @Get(path: '/{id}')
  Future<Response> getResource(
    @Path() String id,
  );

  @Get(path: '/', headers: {'foo': 'bar'})
  Future<Response<Map>> getMapResource(
    @Query() String id,
  );

  @Get(path: '/resources')
  Future<Response<List<Map>>> getListResources();

  @Post(path: '/')
  Future<Response> postResourceUrlEncoded(
    @Field('a') String toto,
    @Field() String b,
  );
```

```dart
@Post(path: '/multi')
@multipart
Future<Response> postResources(
  @Part('1') Map a,
  @Part('2') Map b,
  @Part('3') String c,
);

@Post(path: '/file')
@multipart
Future<Response> postFile(
  @Part('file') List<int> bytes,
);
```

2. gRPC

- 패키지: grpc

```
client = BroadcastClient(
  ClientChannel(
    "10.0.2.2",
    port: 8080,
    options: ChannelOptions(
      credentials: ChannelCredentials.insecure(),
    ),
  ),
);
```

- Http2, 웹소켓 기반
- 내부적으로 protocol buffer 사용
- 한번 연결되고 서로 데이터 주고받음

```
Future<Close> sendMessage(String body) async {
  return client.broadcastMessage(
    Message()
      ..id = user.id
      ..content = body
      ..timestamp = DateTime.now().toIso8601String(),
  );
}
```

```
Stream<Message> recieveMessage() async* {
  Connect connect = Connect()
    ..user = user
    ..active = true;

  await for (var msg in client.createStream(connect)) {
    yield msg;
  }
}
```

3. GraphQL

- 패키지: graphql

```
{
  hero {
    name
    height
    mass
  }
}
```

HTTP Request

```
{
  "hero": {
      "name": "Luke Skywalker",
      "height": 1.72,
      "mass": 77
  }
}
```

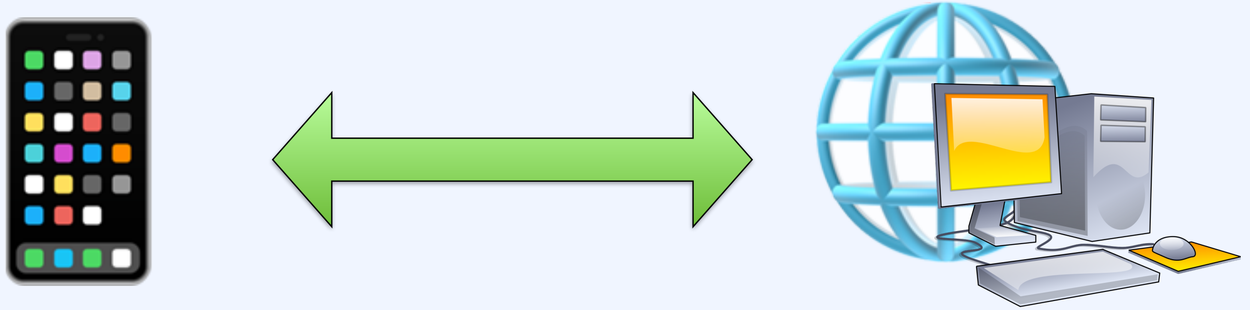
HTTP Response

4. Websocket

패키지: - web_socket_channel

- socket_io_client

- web_socket_client



```
final wsUrl = Uri.parse('ws://localhost:1234')
var channel = WebSocketChannel.connect(wsUrl);

channel.stream.listen((message) {
  channel.sink.add('received!');
  channel.sink.close(status.goingAway);
});
```

5. Firebase (Google)

- Firestore - DB저장
- Storage - 파일 저장
- Cloud Functions - 서버 함수 수행

장점:

1. 특정 요청/용량까지는 무료로 사용이 가능 (MAU 유저 천명~5천명도 커버)
2. 서버를 구동시킬 필요가 없다. 구글 서버에서 모두 알아서 동작. (미국/한국/유럽등 지정가능)