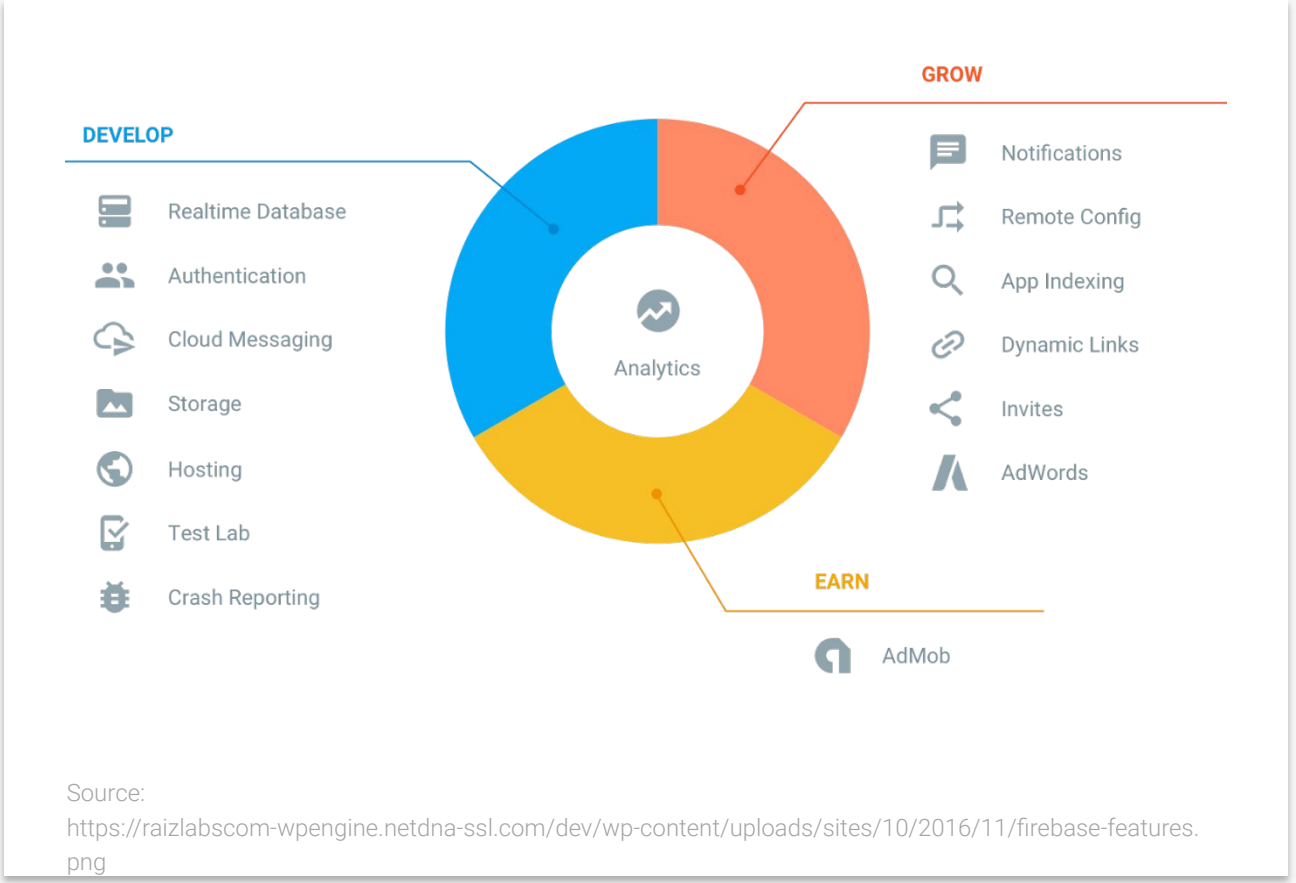
Firebase

Quickly development
of high-quality app

Agenda

1. Features
2. DB Advantages
3. Cloud Firestore
4. Authentication
5. ML Kit
6. Pricing

Features



Source: https://raizlabscom-wpengine.netdna-ssl.com/dev/wp-content/uploads/sites/10/2016/11/firebase-features.png

DB Advantages

Realtime Database

- Nothing

Cloud Firestore

- Data is easier to organize at scale
- Offline support for web clients
- Indexed queries with compound sorting and filtering
- Atomic write and transaction operations
- Scaling will be automatic
- Simpler, more powerful security for mobile, web, and server SDKs.

DB Disadvantages

Realtime Database

- All

Cloud Firestore

- Beta

Cloud Firestore

Web

1. Package install

```
npm install firebase --save
```

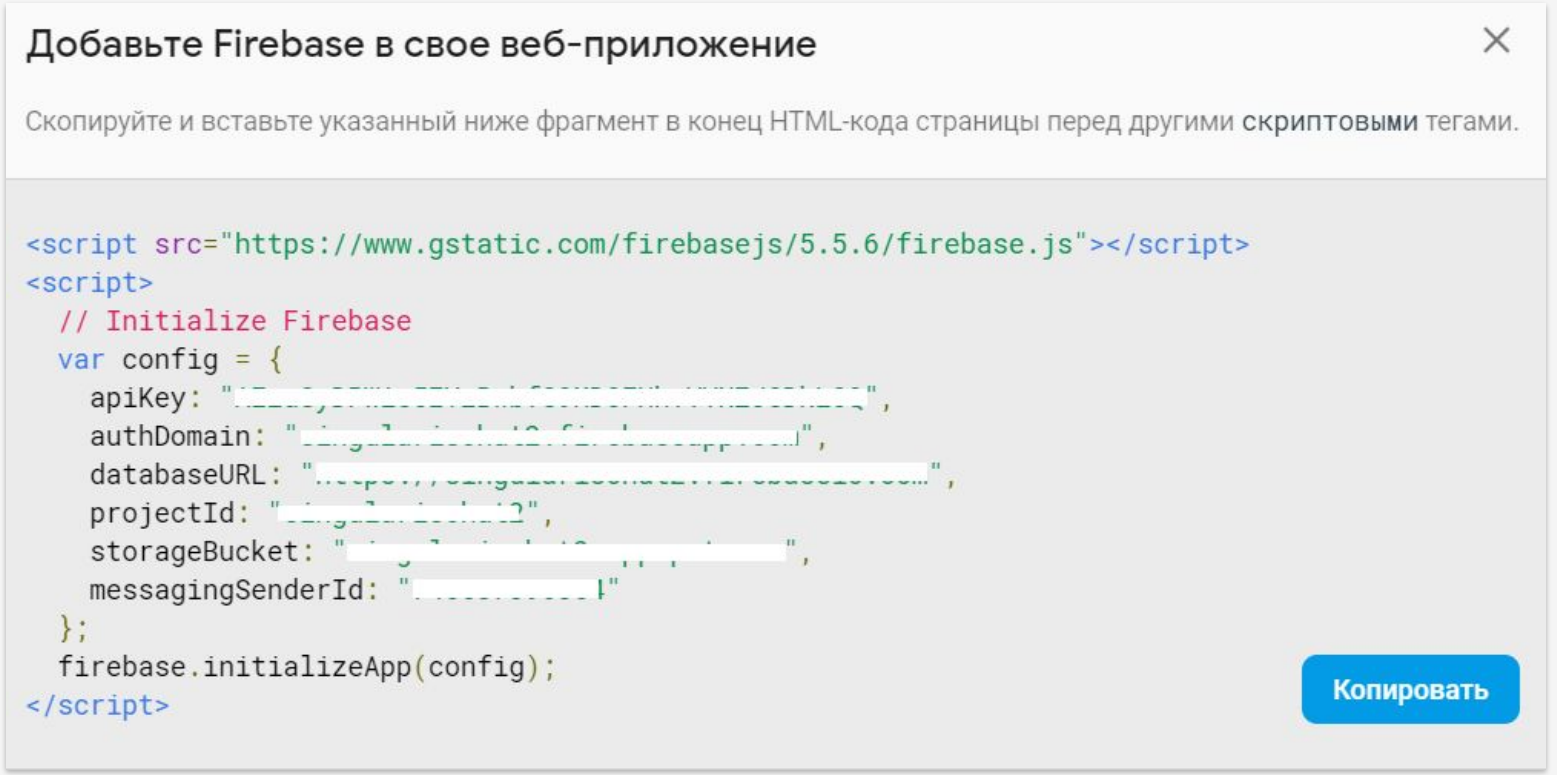
2. Manually require

```
const firebase = require("firebase");
```

3. Initialize

```
firebase.initializeApp({
  apiKey: '### FIREBASE API KEY ###',
  authDomain: '### AUTH DOMAIN ###',
  projectId: '### PROJECT ID ###'
});

// Disable deprecated features
firebase.firestore().settings({
  timestampsInSnapshots: true
});
```



Cloud Firestore

Android

1. Add lib to app/build.gradle

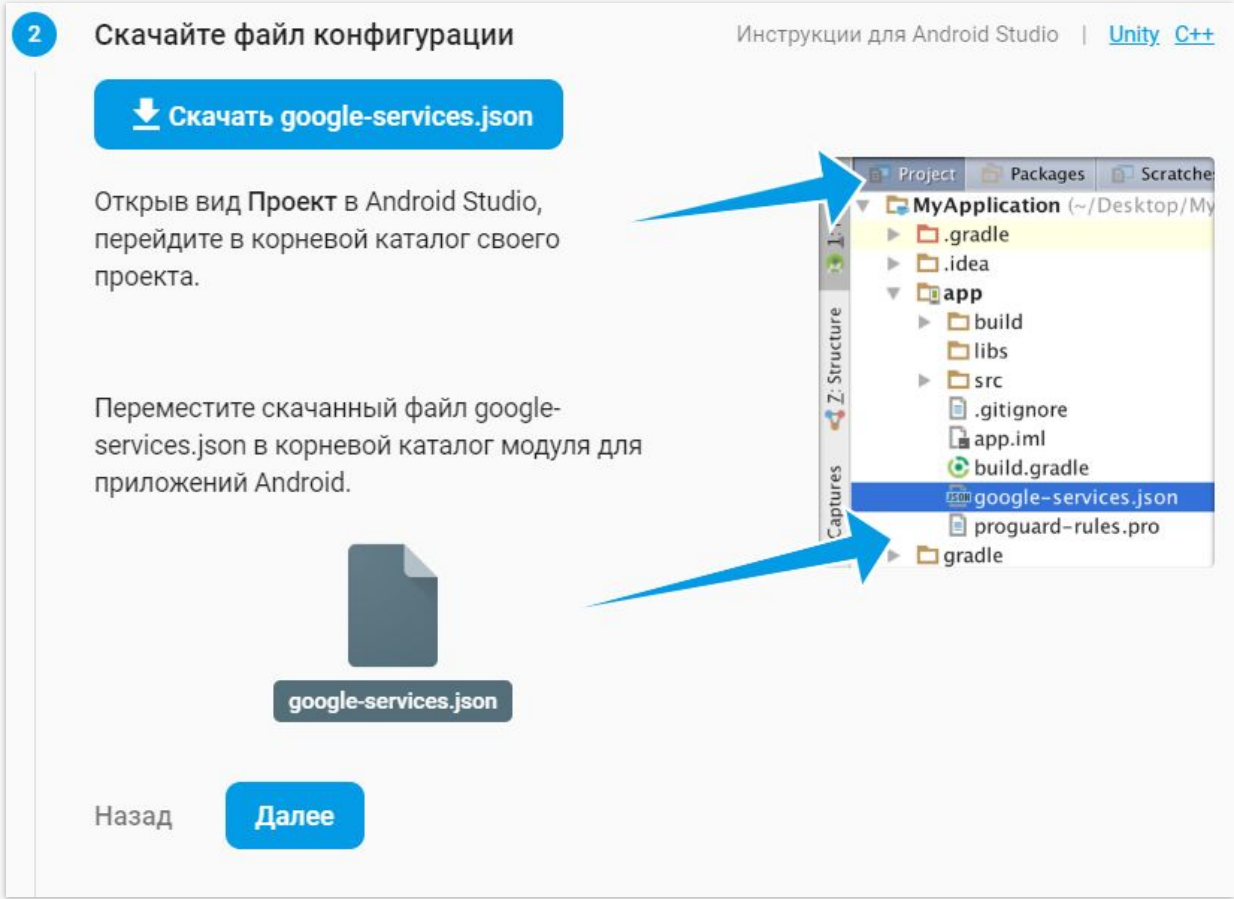
```
npm install firebase --save
```

2. Manually require

```
implementation
'com.google.firebase:firebase-firestore:17.1.2
'
```

3. Initialize

```
FirebaseFirestore db =
FirebaseFirestore.getInstance();
```



Cloud Firestore

Data model

1. NoSql
2. Each document contains a set of key-value pairs
3. All documents must be stored in collections
4. The names of documents within a collection are unique
5. Subcollections can be stored in document
6. Collections and documents are created implicitly
7. If you delete all of the documents in a collection, it no longer exists

Add data

```
// Add a new document with a generated id.
db.collection("cities").add({
    name: "Tokyo",
    country: "Japan"
})
.then(function(docRef) {
    console.log("Document written with ID:
", docRef.id);
})
.catch(function(error) {
    console.error("Error adding document: ",
error);
});
```

```
db.collection("cities").doc("LA").set({
    name: "Los Angeles",
    state: "CA",
    country: "USA"
});
```

```
// Add a new document with a generated
id.
var newCityRef =
db.collection("cities").doc();

// later...
newCityRef.set(data);
```

Update data

```
var washingtonRef =
db.collection("cities").doc("DC");

// Set the "capital" field of the city 'DC'
return washingtonRef.update({
    capital: true
})
.then(function() {
    console.log("Document successfully updated!");
})
.catch(function(error) {
    // The document probably doesn't exist.
    console.error("Error updating document: ",
error);
});
```

```
// Create an initial document to update.
var frankDocRef = db.collection("users").doc("frank");
frankDocRef.set({
    name: "Frank",
    favorites: { food: "Pizza", color: "Blue", subject:
"recess" },
    age: 12
});

// To update age and favorite color:
db.collection("users").doc("frank").update({
    "age": 13,
    "favorites.color": "Red"
})
.then(function() {
    console.log("Document successfully updated!");
});
```

Delete data

```
// Delete document by id

db.collection("cities").doc("DC").delete().then(f
unction() {
    console.log("Document successfully
deleted!");
}).catch(function(error) {
    console.error("Error removing document: ",
error);
});

// Remove the 'capital' field from the document
var removeCapital = cityRef.update({
    capital:
firebase.firestore.FieldValue.delete()
});
```

```
// Delete collections


// Deleting collections from a Web client is not
recommended.
// Deleting collections from an iOS client is not
recommended.
// Deleting collections from an Android client is not
recommended.
```

Transaction

```
var sfDocRef = db.collection("cities").doc("SF");

db.runTransaction(function(transaction) {
    return transaction.get(sfDocRef).then(function(sfDoc) {
        if (!sfDoc.exists) {
            throw "Document does not exist!";
        }
        var newPopulation = sfDoc.data().population + 1;
        if (newPopulation <= 1000000) {
            transaction.update(sfDocRef, { population: newPopulation });
            return newPopulation;
        } else {
            return Promise.reject("Sorry! Population is too big.");
        }
    });
}).then(function(newPopulation) {
    console.log("Population increased to ", newPopulation);
}).catch(function(err) {
    // This will be an "population is too big" error.
    console.error(err);
});
```

Some data

```
var citiesRef = db.collection("cities");

citiesRef.doc("SF").set({
    name: "San Francisco", state: "CA", country:
"USA",
    capital: false, population: 860000,
    regions: ["west_coast", "norcal"] });

citiesRef.doc("LA").set({
    name: "Los Angeles", state: "CA", country:
"USA",
    capital: false, population: 3900000,
    regions: ["west_coast", "socal"] });

citiesRef.doc("DC").set({
    name: "Washington, D.C.", state: null,
country: "USA",
    capital: true, population: 680000,
    regions: ["east_coast"] });
```

Get data

```
var docRef =
db.collection("cities").doc("SF");

docRef.get().then(function(doc) {
    if (doc.exists) {
        console.log("Document data:",
doc.data());
    } else {
        // doc.data() will be undefined
in this case
        console.log("No such
document!");
    }
}).catch(function(error) {
    console.log("Error getting
document:", error);
});
```

Filtering

Where(<field path>, <operator>, <value>)

```
db.collection("cities").where("capital", "==",
true)
    .get()
    .then(function(querySnapshot) {
        querySnapshot.forEach(function(doc) {
            // doc.data() is never undefined for
query doc snapshots
            console.log(doc.id, " => ",
doc.data());
        });
    })
    .catch(function(error) {
        console.log("Error getting documents: ",
error);
    });
```

Filtering

Where(<field path>, <operator>, <value>)

```
citiesRef.where("regions", "array-contains",
"west_coast")
```

Compound

```
citiesRef.where("state", ">=", "CA").where("state",
"<=", "IN")
citiesRef.where("state", "==",
"CA").where("population", ">", 1000000)
```

Filtering

OrderBy(<field path>, <direction>)

```
citiesRef.orderBy("name", "desc")
```

Compound

```
citiesRef.orderBy("state").orderBy("popul
ation", "desc")
```

Filtering

Limit(<number>)

```
citiesRef.li
mit(2)
```

Compound

```
citiesRef.limit(2)
.limit(1)
```

Filtering

startAt(<documentSnapshot>)

startAfter(<documentSnapshot>)

endAt(<documentSnapshot>)

endBefore(<documentSnapshot>)

Realtime updates

```
db.collection("cities").doc("SF")
    .onSnapshot(function(doc) {
        var source = doc.metadata.hasPendingWrites ?
"Local" : "Server";
        console.log(source, " data: ", doc.data());
    });
```

Realtime updates

```
let unsubscribe =
db.collection("cities").where("state", "==",
"CA")
    .onSnapshot(function(snapshot) {

snapshot.docChanges().forEach(function(change)
{
            if (change.type === "added") {
                console.log("New city: ",
change.doc.data());
            }
            if (change.type === "modified") {
                console.log("Modified city: ",
change.doc.data());
            }
            if (change.type === "removed") {
                console.log("Removed city: ",
change.doc.data());
            }
        });
    });
unsubscribe();
```

createDocument	POST /v1beta1/{parent=projects/*/databases/*/documents/**}/{collectionId} Creates a new document.
delete	DELETE /v1beta1/{name=projects/*/databases/*/documents/*/**} Deletes a document.
get	GET /v1beta1/{name=projects/*/databases/*/documents/*/**} Gets a single document.
list	GET /v1beta1/{parent=projects/*/databases/*/documents/*/**}/{collectionId} Lists documents.
write	POST /v1beta1/{database=projects/*/databases/*}/documents:write Streams batches of document updates and deletes, in

onCreate, onUpdate, onDelete, onWrite

```
exports.countNameChanges = functions.firestore
    .document('users/{userId}')
    .onUpdate((change, context) => {
      const data = change.after.data();
      const previousData = change.before.data();

      // We'll only update if the name has changed. // This is crucial to
prevent infinite loops.
      if (data.name == previousData.name) return null;

      // Retrieve the current count of name changes
      let count = data.name_change_count;
      if (!count) {
        count = 0;
      }

      // Then return a promise of a set operation to update the count
      return change.after.ref.set({
        name_change_count: count + 1
      }, {merge: true});
    });
```

Limitations and guarantees

Cloud Firestore is currently in beta which may result in unexpected behavior.

A few known limitations include:

- It may take up to 10 seconds for a function to be triggered after a change to Cloud Firestore data

- As with all background functions, event ordering is not guaranteed. In addition, a single event may result in multiple Cloud Functions invocations, so for the highest quality ensure your functions are written to be idempotent.

Cloud Firestore


```
service cloud.firestore {
  match
/databases/{database}/documents
{
    match /<some_path>/ {
      allow read, write: if
<some_condition>;
    }
  }
}
```

```
// Allow read/write access on all documents to any user signed
in to the application
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth.uid != null;
    }
  }
}
```

Cloud Firestore


```
service cloud.firestore {
  match /databases/{database}/documents {
    match /cities/{city} {
      // Make sure a 'users' document exists for the requesting
user before
      // allowing any writes to the 'cities' collection
      allow create: if
exists(/databases/$(database)/documents/users/$(request.auth.ui
d))

      // Allow the user to delete cities if their user document
has the
      // 'admin' field set to 'true'
      allow delete: if
get(/databases/$(database)/documents/users/$(request.auth.uid))
.data.admin == true
    }
  }
}
```

Access call limits

- 10 for single-document requests and query requests.
- 20 for multi-document reads, transactions, and batched writes. The previous limit of 10 also applies to each operation.

For example, imagine you create a batched write request with 3 write operations and that your security rules use 2 document access calls to validate each write. In this case, each write uses 2 of its 10 access calls and the batched write request uses 6 of its 20 access calls.

For each set operation, Cloud Firestore updates:

- One ascending single-field index per non-array field
- One descending single-field index per non-array field
- One array-contains single-field index for the array field

Cloud Firestore

Collection	Field indexed
cities	↑ name
cities	↑ state
cities	↑ country
cities	↑ capital
cities	↑ population
cities	↓ name
cities	↓ state
cities	↓ country
cities	↓ capital
cities	↓ population
cities	array-contains tags

If you attempt a compound query with a range clause that doesn't map to an existing index, you receive an error. The error message includes a direct link to create the missing index in the Firebase console.

```
citiesRef.where("country", "==",
"USA").orderBy("population", "asc")
citiesRef.where("country", "==", "USA").where("population",
"<", 3800000)
citiesRef.where("country", "==", "USA").where("population",
">", 690000)
```

Cloud Firestore

- Not bad NoSQL with Transactions, Indexes, Filtering
- Realtime updates
- Offline support
- Fast development
- Platform support: Web, iOs, Android, Java, Python, NODE.js, .Net

Authentication

Get started

Sign up

```
firebase.auth().createUserWithEmailAndPassword(email,
password)
.catch(function(error) {
  // Handle Errors here.
  var errorCode = error.code;
  var errorMessage = error.message;
  // ...
});
```

Authentication

Get started

Sign in

```
firebase.auth().signInWithEmailAndPassword(email, password)
.catch(function(error) {
  // Handle Errors here.
  var errorCode = error.code;
  var errorMessage = error.message;
  // ...
});
```

Authentication state observer

```
firebase.auth().onAuthStateChanged(function(use
r) {
  if (user) {
    // User is signed in.
    var displayName = user.displayName;
    var email = user.email;
    var emailVerified = user.emailVerified;
    var photoURL = user.photoURL;
    var isAnonymous = user.isAnonymous;
    var uid = user.uid;
    var providerData = user.providerData;
    // ...
  } else {
    // User is signed out.
    // ...
  }
});
```

Ml Kit

Feature	On-device	Cloud
Text recognition	+	+
Face detection	+	
Barcode scanning	+	
Image labeling	+	+
Landmark recognition		+
Custom model inference	+	

Ml Kit

Recognized Text	
Text	Wege der parlamentarischen Demokratie
Blocks	(1 block)

Ml Kit

Block 0	
Text	Wege der parlamentarischen Demokratie
Frame	(117.0, 258.0, 190.0, 83.0)
Corner Points	(117, 270), (301.64, 258.49), (306.05, 329.36), (121.41, 340.86)
Recognized Language Code	de
Lines	(3 lines)

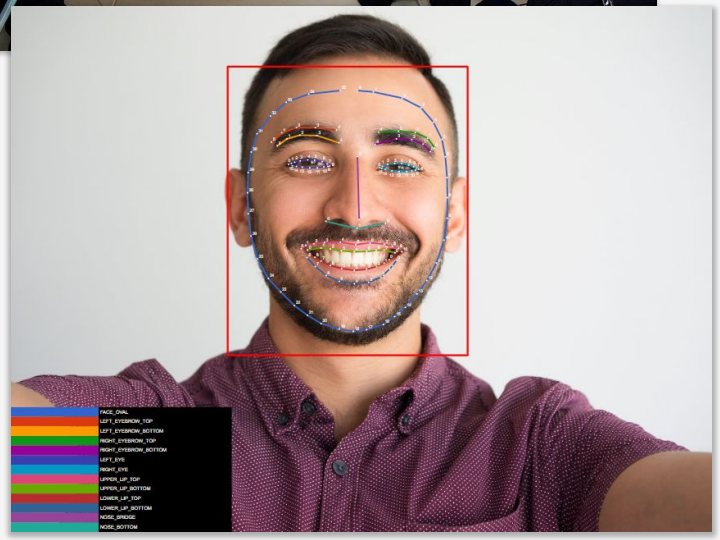
Ml Kit

Line 0	
Text	Wege der
Frame	(167.0, 261.0, 91.0, 28.0)
Corner Points	(167, 267), (255.82, 261.46), (257.19, 283.42), (168.36, 288.95)
Recognized Language Code	de
Elements	(2 elements)

Ml Kit

Element 0	
Text	Wege
Frame	(167.0, 263.0, 59.0, 26.0)
Corner Points	(167, 267), (223.88, 263.45), (225.25, 285.41), (168.36, 288.95)

Ml Kit



Face 1 of 3		
Bounding polygon	(884.880004882812, 149.546676635742), (1030.77197265625)…	
Angles of rotation	Y: -14.054030418395996, Z: -55.007488250732422	
Tracking ID	2	
Facial landmarks	Left eye	(945.869323730469, 211.867126464844)
	Right eye	(971.579467773438, 247.257247924805)
	Bottom of mouth	(907.756591796875, 259.714477539062)
Feature probabilities	Smiling	0.88979166746139526
	Left eye open	0.98635888937860727
	Right eye open	0.99258323386311531

Ml Kit

Result	
Corners	(49,125), (172,125), (172,160), (49,160)
Raw value	2404105001722

Result	
Corners	(87,87) (612,87) (612,612) (87,612)
Raw value	WIFI:S:SB1Guest;P:12345;T:WEP;;
WiFi information	
SSID	SB1Guest
Password	12345
Type	WEP

Ml Kit


C1
C2
Valais
Valais
Valais
Valais
Valais
♡F.C.P.♡
Loterie Romande
Rhône FM
Valais

Ml Kit

On device

Description	Stadium
Knowledge Graph entity ID	/m/019cfy
Confidence	0.9205354

Description	Sports
Knowledge Graph entity ID	/m/06ntj
Confidence	0.7531109

Cloud

Description	soccer specific stadium
Knowledge Graph entity ID	/m/0404y4
Confidence	0.95806926

Description	player
Knowledge Graph entity ID	/m/02vzx9
Confidence	0.9797604

Ml Kit

Result

Description	Brugge
Geographic Coordinates	51.207367, 3.226933
Knowledge Graph entity ID	/m/0drjd2
Bounding Polygon	(20, 342), (651, 342), (651, 798), (20, 798)
Confidence Score	0.77150935

Pricing

Realtime Database	Free	$25/month	Pay as you go
Simultaneous connections help	100	100k	100k/database
Stored data	1 GB	2.5 GB	$5/GB
GB downloaded	10 GB/month	20 GB/month	$1/GB per month
Multiple databases per project	-	-	+

Pricing

Cloud Firestore	Free	$25/month	Pay as you go
Stored data	1 GB	2.5 GB	$0.18/GB
Bandwidth	10 GB/month	20 GB/month	?
Document writes	20K/day	100K/day	$0.18/100K
Document reads	50K/day	250K/day	$0.06/100K
Document deletes	20K/day	100K/day	$0.02/100K

Pricing

ML Kit	Free	$25/month	Pay as you go
On-device APIs	+	+	+
Custom Model Hosting/Serving	+	+	+
Cloud Vision APIs	-	-	$1.5/1K