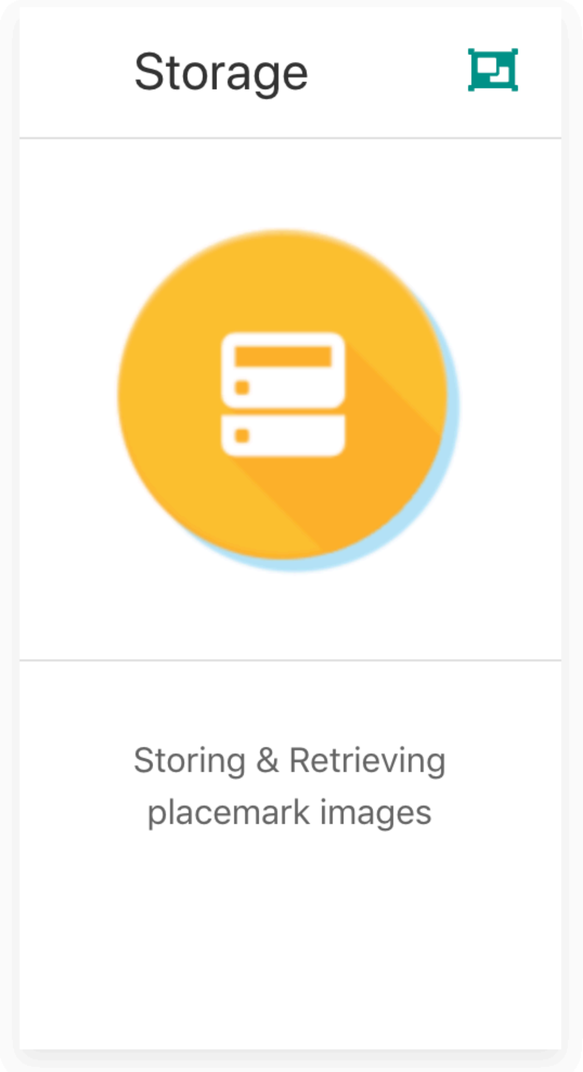
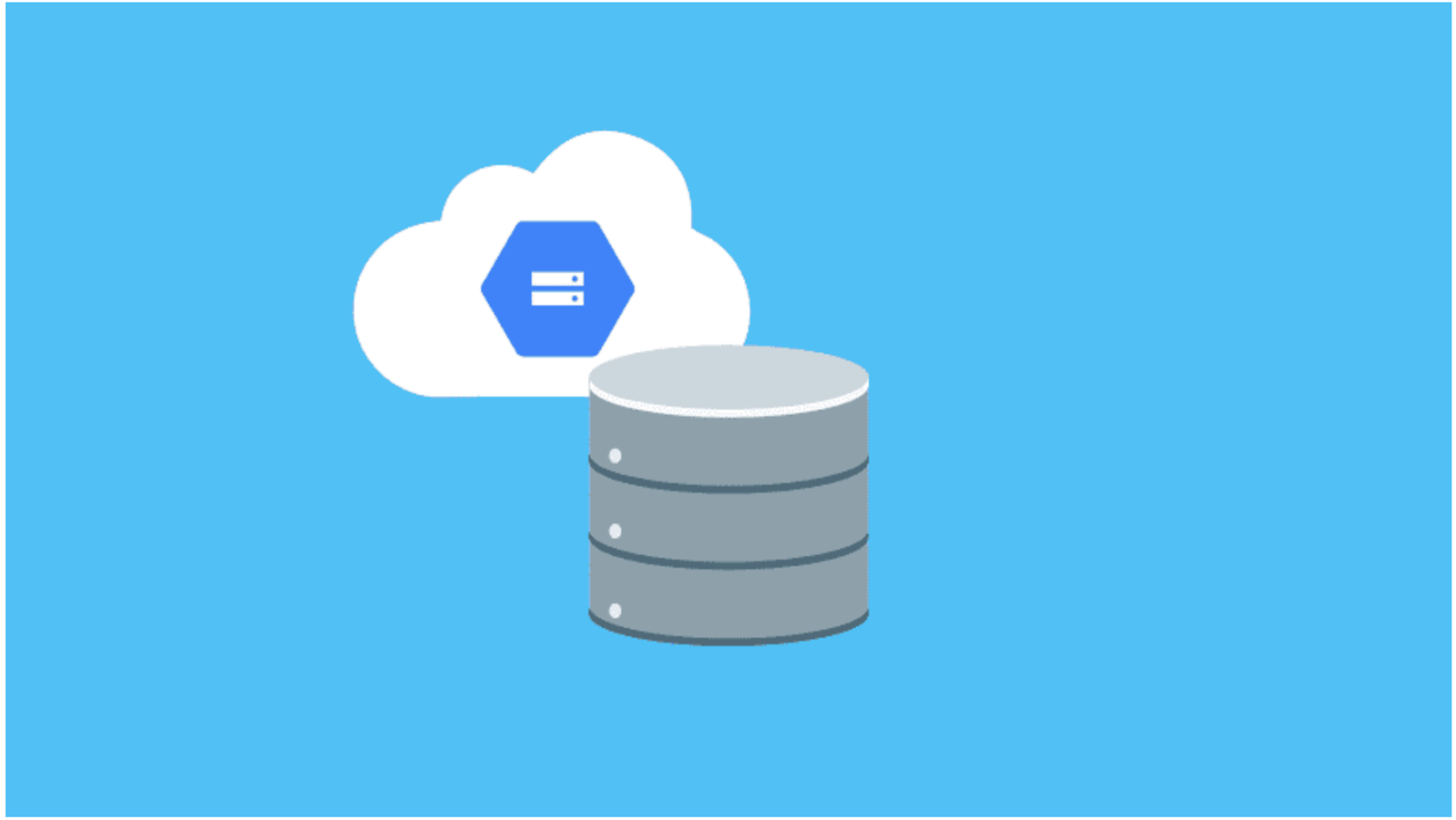
Firebase Database



Store your users' photos and videos

Cloud Storage is designed to help you quickly and easily store and serve user-generated content, such as photos and videos.

Cloud Storage



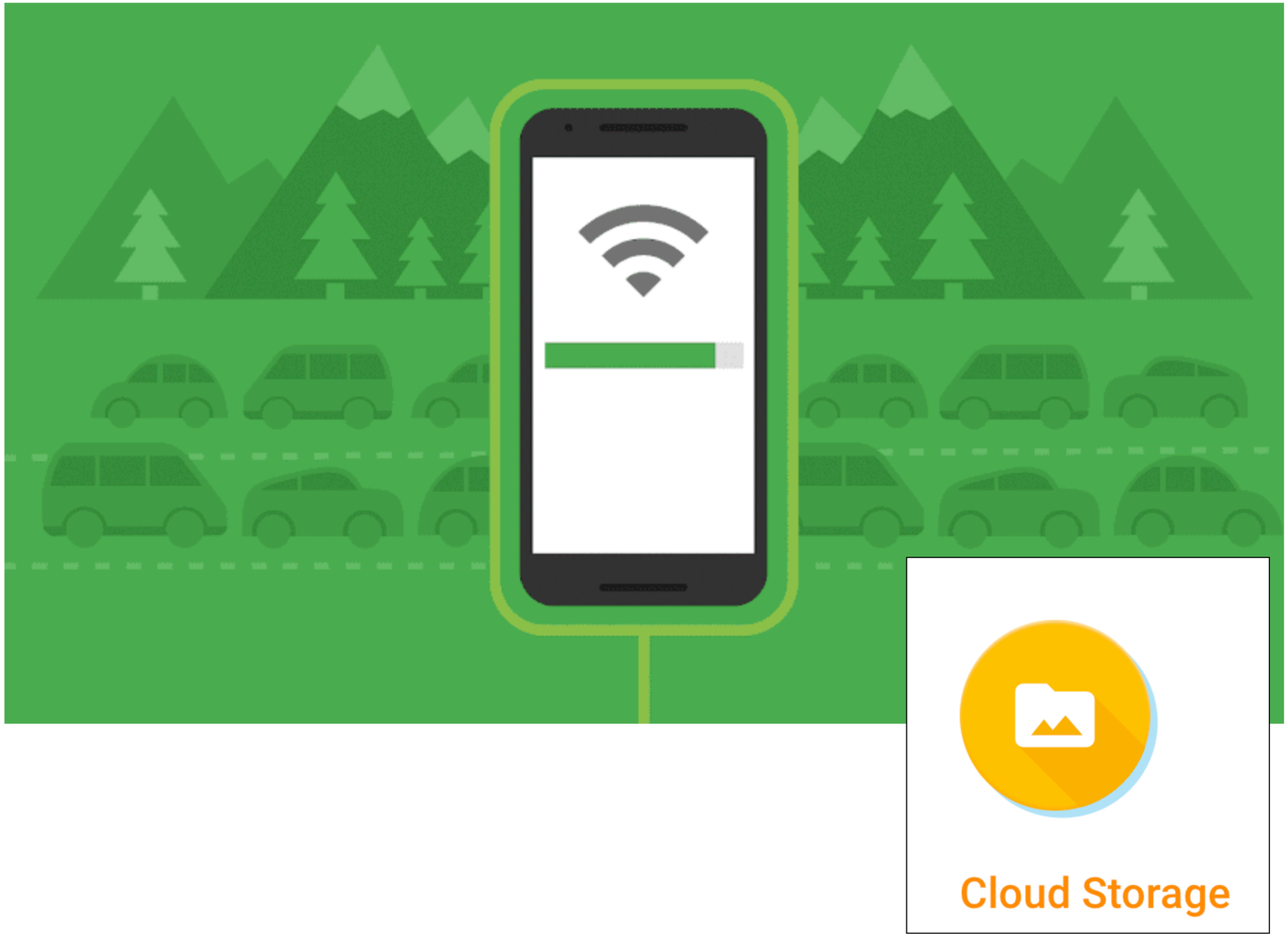
Build at Google scale

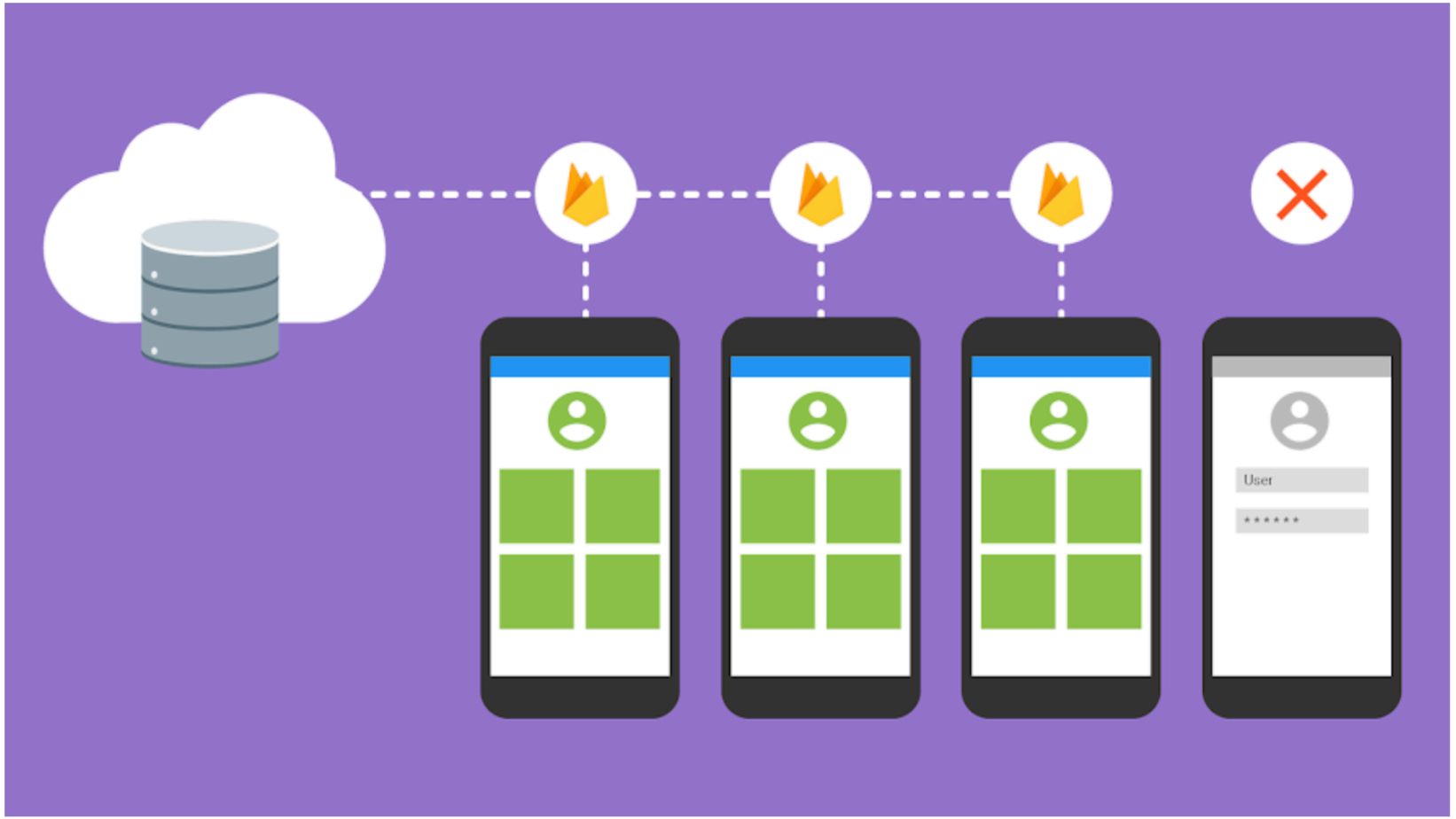
Our infrastructure is built for when your app goes viral. Effortlessly grow from prototype to production using the same technology that powers apps like Spotify and Google Photos.

Cloud Storage

Robust uploads and downloads

Your users aren't always online, so we built the Firebase SDK for Cloud Storage with mobile connectivity in mind. It will automatically pause and resume your transfers as the app loses and regains mobile connectivity, saving your users time and bandwidth.



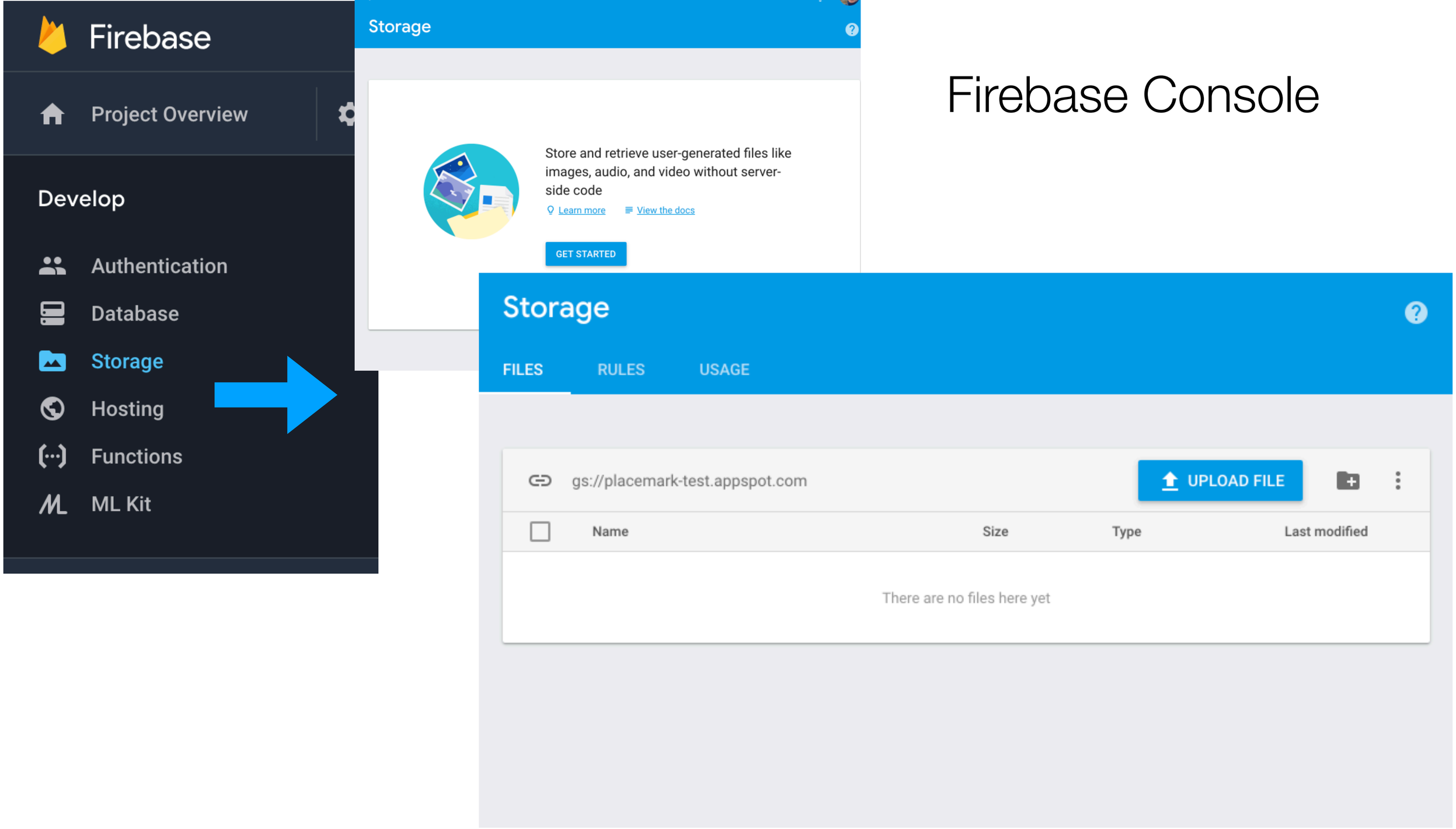


Strong user-based security

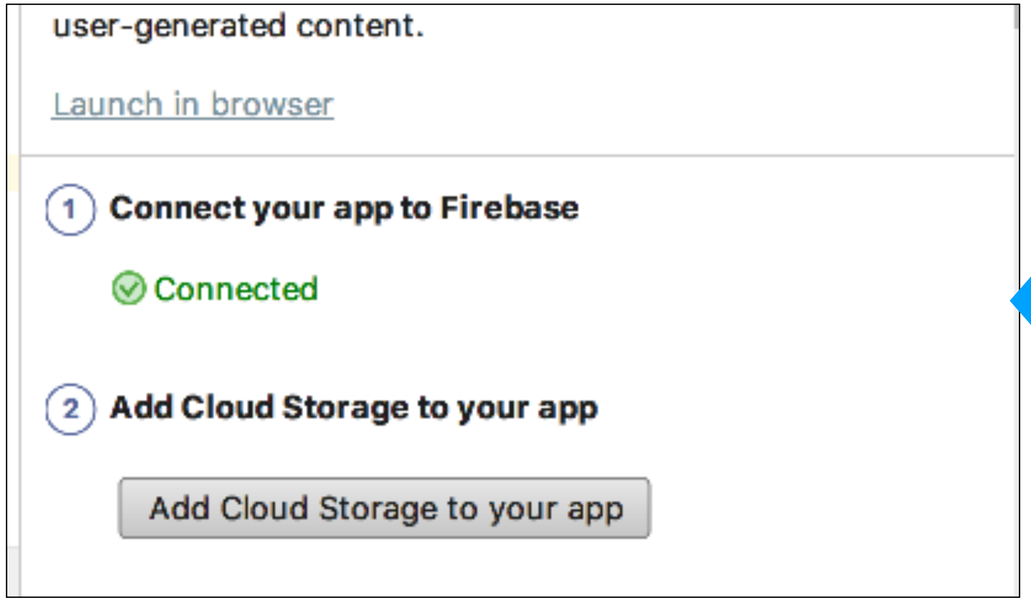
The Firebase SDK for Cloud Storage integrates with Firebase Authentication to provide simple and intuitive access control. You can use our declarative security model to allow access based on user identity or properties of a file, such as name, size, content type, and other metadata.

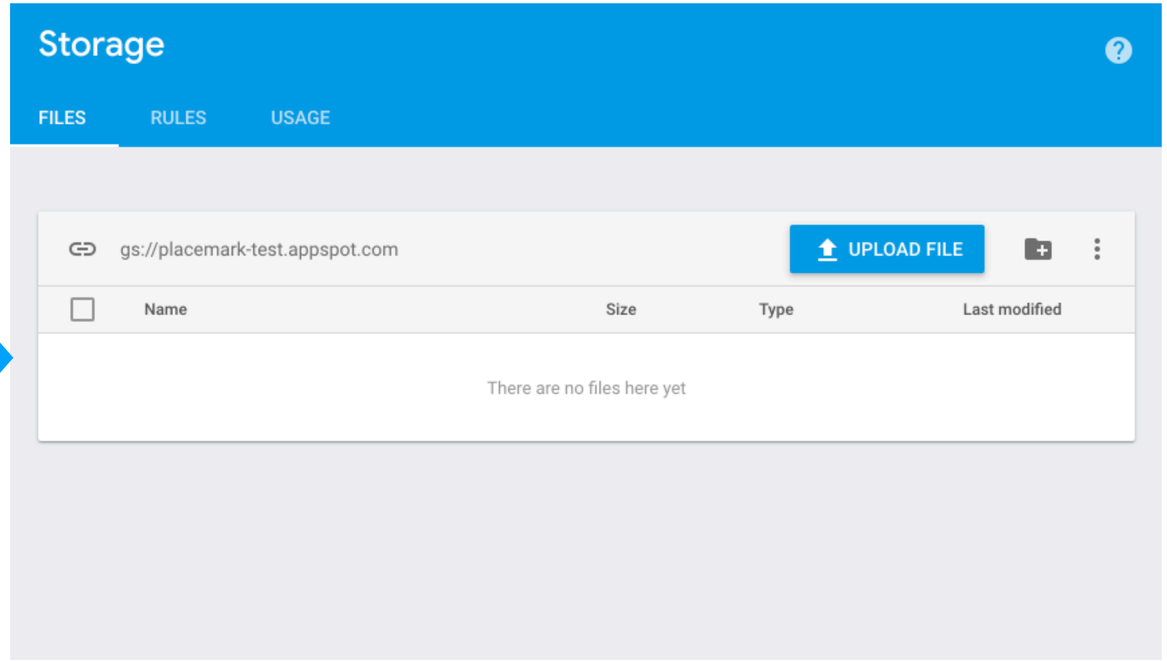
Cloud Storage

Firebase Console



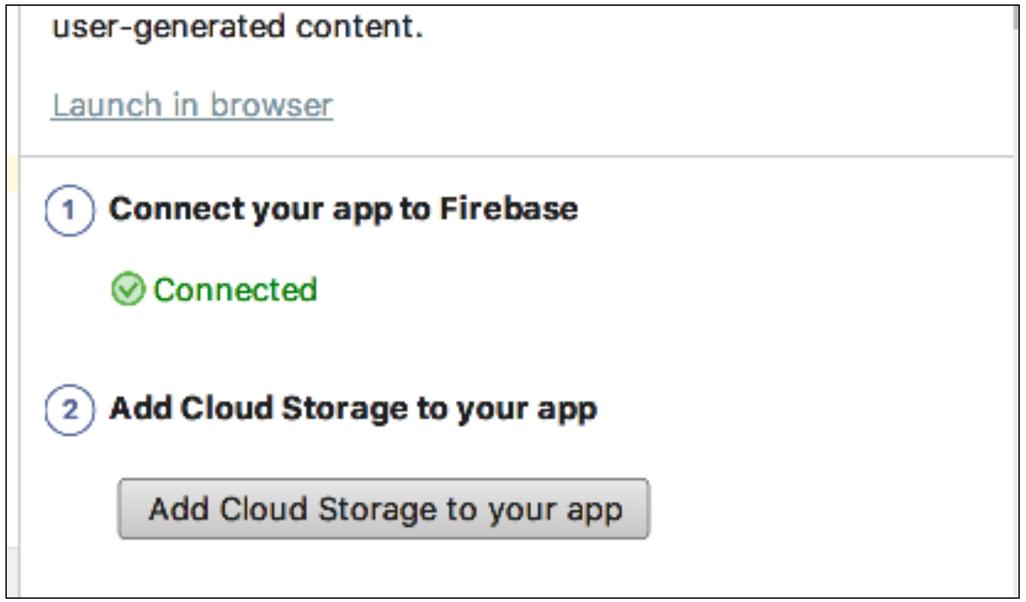
Firebase Assistant in Studio





build.gradle

```
implementation "com.google.firebase:firebase-storage:$firebase_version"
```




```
{
  "project_info": {
    "project_number": "1062442537261",
    "firebase_url": "https://placemark-222108.firebaseio.com",
    "project_id": "placemark-222108",
    "storage_bucket": "placemark-222108.appspot.com"
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:1062442537261:android:634c4d908a4ce143",
        "android_client_info": {
          "package_name": "org.wit.placemark"
        }
      },
      "oauth_client": [
        {
          "client_id": "10624425372XXXXXXXXXXXXXq320l17eu0pdv.apps.googleusercontent.com",
          "client_type": 1,
          "android_info": {
            "package_name": "org.wit.placemark",
            "certificate_hash": "368ead570ae3aa95a69bd78936dc4f2123a7ed96"
          }
        },
        {
          "client_id": "1062442537261-uXXXXXXXXXXXXX0avhhp4l5eqc4.apps.googleusercontent.com",
          "client_type": 3
        }
      ],
      "api_key": [
        {
          "current_key": "AIzaSyBXXXXXXXXXXXXXXXX52I95o"
        }
      ],
      "services": {
        "analytics_service": {
          "status": 1
        },
        "appinvite_service": {
          "status": 2,
          "other_platform_oauth_client": [
            {
              "client_id": "106244253XXXXXXXXXXXXXXXhp4l5eqc4.apps.googleusercontent.com",
              "client_type": 3
            }
          ]
        },
        "ads_service": {
          "status": 2
        }
      }
    }
  ],
  "configuration_version": "1"
}
```


google-services.json

Uploading Image to Storage

gs://placemark-222108.appspot.com > fUZhVTJ1Z...enEzYWLwo2

Upload file

Name	Size	Type	Last modified
321	221....	image/jpeg	Nov 18, 2...

gs://placemark-222108.appspot.com

Upload file

Name	Size	Type	Last modified
9VCoy8TSQCU08vMKAMg1qYfi2RT2/	–	Folder	–
fUZhVTJ1ZhROOU0I1CenEzYWLwo2/	–	Folder	–
Ih7wuYj3vheuQvAAWejAtSWtH0i1/	–	Folder	–

321

Name
321

Size
226,346 bytes

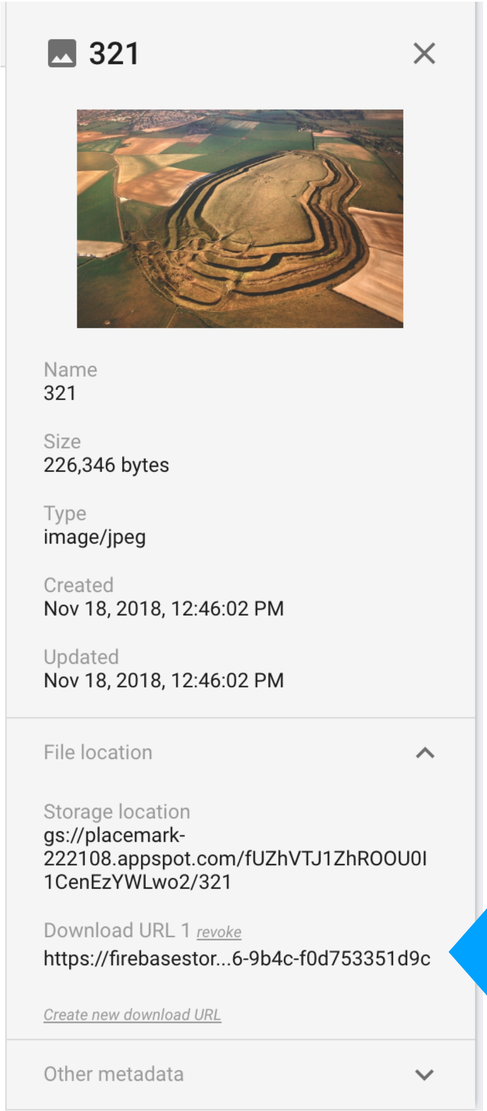
Type
image/jpeg

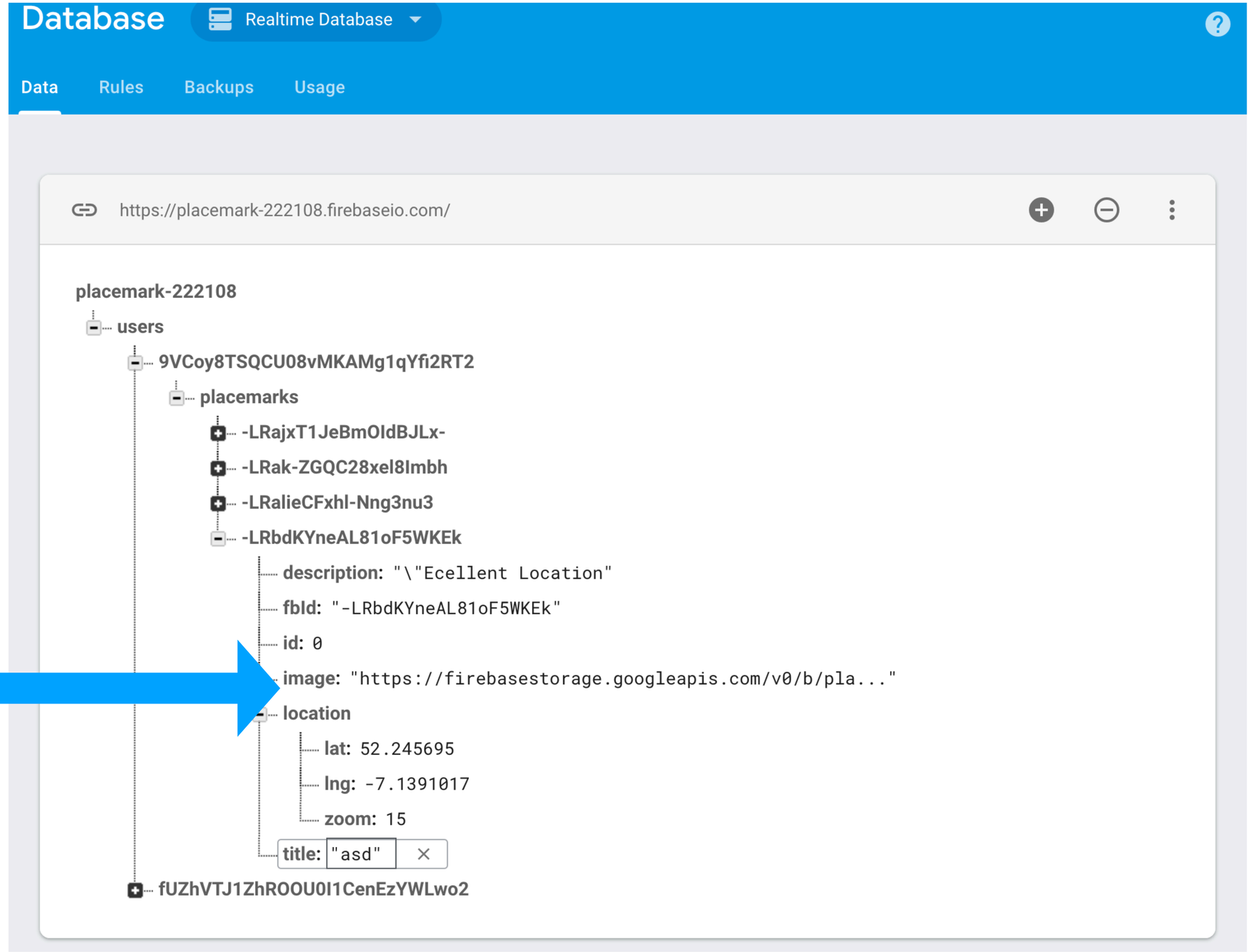
Created
Nov 18, 2018, 12:46:02 PM

Updated
Nov 18, 2018, 12:46:02 PM

File location

Other metadata





Store image url in Database

```
fun updateImage(placemark: PlacemarkModel) {
  if (placemark.image != "") {

    // get the full image file name



    // create storage object to be uploaded to the cloudstore (node name is id of user)

    // read the image into a bitmap object


      // compress the image into a byte array



      // put the bytes unto the object and start upload (asynchronous call)

        // failure

        // success, get full path of uploaded image (asynchronous call)

          // recover full path

          // store full path in database



  }
}
```

```
fun updateImage(placemark: PlacemarkModel) {
  if (placemark.image != "") {

    // get the full image file name
    val fileName = File(placemark.image)
    val imageName = fileName.getName()

    // create storage object to be uploaded to the cloudstore (node name is id of user)


    // read the image into a bitmap object



      // compress the image into a byte array




      // put the bytes unto the object and start upload (asynchronous call)


        // failure


        // success, get full path of uploaded image (asynchronous call)

          // recover full path

          // store full path in database




  }
}
```

```
fun updateImage(placemark: PlacemarkModel) {
  if (placemark.image != "") {

    // get the full image file name
    val fileName = File(placemark.image)
    val imageName = fileName.getName()

    // create storage object to be uploaded to the cloudstore (node name is id of user)
    var imageRef = st.child(userId + '/' + imageName)

    // read the image into a bitmap object



      // compress the image into a byte array




      // put the bytes unto the object and start upload (asynchronous call)


        // failure


        // success, get full path of uploaded image (asynchronous call)

          // recover full path

          // store full path in database




  }
}
```

```
fun updateImage(placemark: PlacemarkModel) {
  if (placemark.image != "") {

    // get the full image file name
    val fileName = File(placemark.image)
    val imageName = fileName.getName()

    // create storage object to be uploaded to the cloudstore (node name is id of user)
    var imageRef = st.child(userId + '/' + imageName)

    // read the image into a bitmap object
    val bitmap = readImageFromPath(context, placemark.image)

    bitmap?.let {
      // compress the image into a byte array




      // put the bytes unto the object and start upload (asynchronous call)


        // failure


        // success, get full path of uploaded image (asynchronous call)

          // recover full path

          // store full path in database




  }
}
```

```
fun updateImage(placemark: PlacemarkModel) {
  if (placemark.image != "") {

    // get the full image file name
    val fileName = File(placemark.image)
    val imageName = fileName.getName()

    // create storage object to be uploaded to the cloudstore (node name is id of user)
    var imageRef = st.child(userId + '/' + imageName)

    // read the image into a bitmap object
    val bitmap = readImageFromPath(context, placemark.image)

    bitmap?.let {
      // compress the image into a byte array
      val baos = ByteArrayOutputStream()
      bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
      val data = baos.toByteArray()

      // put the bytes unto the object and start upload (asynchronous call)


        // failure


        // success, get full path of uploaded image (asynchronous call)

          // recover full path

          // store full path in database




  }
}
```

```
fun updateImage(placemark: PlacemarkModel) {
  if (placemark.image != "") {

    // get the full image file name
    val fileName = File(placemark.image)
    val imageName = fileName.getName()

    // create storage object to be uploaded to the cloudstore (node name is id of user)
    var imageRef = st.child(userId + '/' + imageName)

    // read the image into a bitmap object
    val bitmap = readImageFromPath(context, placemark.image)

    bitmap?.let {
      // compress the image into a byte array
      val baos = ByteArrayOutputStream()
      bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
      val data = baos.toByteArray()

      // put the bytes unto the object and start upload (asynchronous call)
      val uploadTask = imageRef.putBytes(data)
      uploadTask.addOnFailureListener {
        // failure


        // success, get full path of uploaded image (asynchronous call)

          // recover full path

          // store full path in database




  }
}
```

```
fun updateImage(placemark: PlacemarkModel) {
  if (placemark.image != "") {

    // get the full image file name
    val fileName = File(placemark.image)
    val imageName = fileName.getName()

    // create storage object to be uploaded to the cloudstore (node name is id of user)
    var imageRef = st.child(userId + '/' + imageName)

    // read the image into a bitmap object
    val bitmap = readImageFromPath(context, placemark.image)

    bitmap?.let {
      // compress the image into a byte array
      val baos = ByteArrayOutputStream()
      bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
      val data = baos.toByteArray()

      // put the bytes unto the object and start upload (asynchronous call)
      val uploadTask = imageRef.putBytes(data)
      uploadTask.addOnFailureListener {
        // failure
        println(it.message)
      }.addOnSuccessListener { taskSnapshot ->
        // success, get full path of uploaded image (asynchronous call)

          // recover full path

          // store full path in database




  }
}
```

```
fun updateImage(placemark: PlacemarkModel) {
  if (placemark.image != "") {

    // get the full image file name
    val fileName = File(placemark.image)
    val imageName = fileName.getName()

    // create storage object to be uploaded to the cloudstore (node name is id of user)
    var imageRef = st.child(userId + '/' + imageName)

    // read the image into a bitmap object
    val bitmap = readImageFromPath(context, placemark.image)

    bitmap?.let {
      // compress the image into a byte array
      val baos = ByteArrayOutputStream()
      bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
      val data = baos.toByteArray()

      // put the bytes unto the object and start upload (asynchronous call)
      val uploadTask = imageRef.putBytes(data)
      uploadTask.addOnFailureListener {
        // failure
        println(it.message)
      }.addOnSuccessListener { taskSnapshot ->
        // success, get full path of uploaded image (asynchronous call)
        taskSnapshot.metadata!!.reference!!.downloadUrl.addOnSuccessListener {
          // recover full path

          // store full path in database




  }
}
```

```
fun updateImage(placemark: PlacemarkModel) {
  if (placemark.image != "") {

    // get the full image file name
    val fileName = File(placemark.image)
    val imageName = fileName.getName()

    // create storage object to be uploaded to the cloudstore (node name is id of user)
    var imageRef = st.child(userId + '/' + imageName)

    // read the image into a bitmap object
    val bitmap = readImageFromPath(context, placemark.image)

    bitmap?.let {
      // compress the image into a byte array
      val baos = ByteArrayOutputStream()
      bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
      val data = baos.toByteArray()

      // put the bytes unto the object and start upload (asynchronous call)
      val uploadTask = imageRef.putBytes(data)
      uploadTask.addOnFailureListener {
        // failure
        println(it.message)
      }.addOnSuccessListener { taskSnapshot ->
        // success, get full path of uploaded image (asynchronous call)
        taskSnapshot.metadata!!.reference!!.downloadUrl.addOnSuccessListener {
          // recover full path
          placemark.image = it.toString()
          // store full path in database




  }
}
```

```
fun updateImage(placemark: PlacemarkModel) {
  if (placemark.image != "") {

    // get the full image file name
    val fileName = File(placemark.image)
    val imageName = fileName.getName()

    // create storage object to be uploaded to the cloudstore (node name is id of user)
    var imageRef = st.child(userId + '/' + imageName)

    // read the image into a bitmap object
    val bitmap = readImageFromPath(context, placemark.image)

    bitmap?.let {
      // compress the image into a byte array
      val baos = ByteArrayOutputStream()
      bitmap.compress(Bitmap.CompressFormat.JPEG, 100, baos)
      val data = baos.toByteArray()

      // put the bytes unto the object and start upload (asynchronous call)
      val uploadTask = imageRef.putBytes(data)
      uploadTask.addOnFailureListener {
        // failure
        println(it.message)
      }.addOnSuccessListener { taskSnapshot ->
        // success, get full path of uploaded image (asynchronous call)
        taskSnapshot.metadata!!.reference!!.downloadUrl.addOnSuccessListener {
          // recover full path
          placemark.image = it.toString()
          // store full path in database
          db.child("users").child(userId).child("placemarks").child(placemark.fbId).setValue(placemark)
        }
      }
    }
  }
}
```

update()

```
suspend override fun update(placemark: PlacemarkModel) {
  var foundPlacemark: PlacemarkModel? = placemarks.find { p -> p.fbId == placemark.fbId }
  if (foundPlacemark != null) {
    foundPlacemark.title = placemark.title
    foundPlacemark.description = placemark.description
    foundPlacemark.image = placemark.image
    foundPlacemark.location = placemark.location
  }

  db.child("users").child(userId).child("placemarks").child(placemark.fbId).setValue(placemark)
  if ((placemark.image.length) > 0 && (placemark.image[0] != 'h')) {
    updateImage(placemark)
  }
}
```

Upload Image whenever update requested