TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Khoa Công Nghệ Thông Tin

Báo cáo nghiên cứu Hệ quản trị cơ sở dữ liệu

Giảng viên: Ths Dư Phương Hạnh

Chủ đề: **Firebase Database**

Nhóm sinh viên thực hiện:

Lê Công Thương

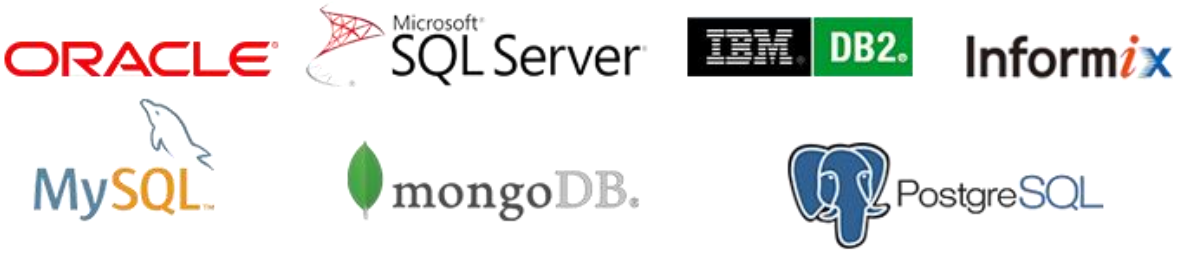
Phạm Văn Trọng

Vũ Ngọc Chi

Hoàng Anh Tuấn

Mục lục

Phần 1. Vấn đề của việc sử dụng DBMS hiện nay



Hiện nay, ngày càng có nhiều các Hệ quản trị Cơ sở dữ liệu (DBMS) được phát minh cũng như được nâng cấp trở nên hoàn thiện hơn. Điều này làm cho việc lựa chọn một DBMS cho dự án trở nên khó khăn. Nhất là khi mỗi loại DBMS lại có cách xây dựng và tổ chức khác nhau, cũng như có những ưu và nhược điểm riêng. Thực tế thì ở các nhà phát triển trẻ, chưa có kinh nghiệm, thường rất dễ gặp tình trạng lạm dụng các Hệ quản trị như MySQL, mongoDB, PostgreSQL vì độ phổ biến của chúng, mặc cho việc cách thức các DBMS này hoạt động có thể không phù hợp với mục tiêu dự án.

Xu thế mà các dự án lớn hiện nay phát triển đa phần đều theo hướng kiến trúc MicroService, tức là dự án sẽ chia thành nhiều module, mỗi module sẽ đảm nhận một chức năng riêng, các module này có thể được phát triển bằng các công nghệ khác nhau, trong đó bao gồm cả việc lựa chọn DBMS. Việc dùng nhiều DBMS sẽ đem lại ưu điểm là tận dụng được các điểm mạnh của từng DBMS cho việc phát triển từng module. Tuy nhiên, điều này vẫn đặt ra một vấn đề là lựa chọn DBMS thế nào cho hợp lý? Đòi hỏi các nhà phát triển phải có hiểu biết về các DBMS trước khi bắt tay vào chọn một DBMS cho dự án của mình.

Về các loại Hệ quản trị Cơ sở dữ liệu, RDBMS (Relational DBMS) với cơ chế lưu trữ SQL đã từng thống trị gần như toàn bộ hoạt động lưu trữ dữ liệu trong các thập kỉ trước. Điển hình cho sự thống trị này đó chính là độ phổ biến của MySQL như hiện nay. Thế nhưng, nhược điểm của RDBMS lại là ở tốc độ xử lý và khả năng làm việc với các kiểu dữ liệu phi cấu trúc. Đó chính là điểm mà Cơ sở dữ liệu kiểu

NoSQL làm khá tốt. Nhất là khi công nghệ phát triển, lượng dữ liệu này càng lớn thì việc lưu trữ ở “local” là điều rất bất tiện. Điều này dẫn đến việc ra đời của của các Hệ quản trị CSDL trên nền tảng Cloud.

Phần 2: Giới thiệu chung về Google Firebase

1. Tổng quan:

Google Firebase là một dịch vụ cơ sở dữ liệu thời gian thực hoạt động trên nền tảng đám mây được cung cấp bởi Google nhằm giúp các lập trình viên phát triển nhanh các ứng dụng bằng cách đơn gian hóa các thao tác với cơ sở dữ liệu. Nói một cách dễ hiểu hơn, thay vì trực tiếp cung cấp các ứng dụng, Firebase cung cấp các dịch vụ nền tảng cho các lập trình viên, sử dụng để xây dựng ứng dụng cũng như hỗ trợ các lập trình viên tối ưu hóa, tối đa hóa ứng dụng của mình. Với nhiều dịch vụ chất lượng cao đi kèm mức giá phải chăng. Firebase đã và đang, không chỉ là sự lựa chọn hàng đầu cho các lập trình viên đơn thân (single dev) hay các công ty khởi nghiệp (start ups), mà các công ty, tổ chức lớn có tên tuổi cũng sử dụng “Ngọn lửa” để xây dựng các tính năng, các chương trình mới, cũng như chuyển đổi các dịch vụ trước đây sang hệ thống của Firebase. Chẳng hạn như Shazam, Fabulous và cả chính Google nữa,

khi nền tảng nhắn tin Allo được xây dựng trên nền tảng Firebase Realtime Database.

2. Lịch sử phát triển

Về mặt lịch sử, Firebase (tiền thân là Evolve) trước đây là một start up được thành lập vào năm 2011 bởi Andrew Lee và James Tamplin. Ban đầu, Evolve chỉ cung cấp cơ sở dữ liệu để các lập trình viên thiết kế các ứng dụng chat (và hiện tại thì để làm quen với realtime database thì bạn cũng làm ứng dụng chat đó thôi). Tuy nhiên, họ nhanh chóng nhận ra tiềm năng sản phẩm của mình khi nhận thấy các khách hàng không sử dụng CSDL để làm ứng dụng chat, mà thay vào đó, để lưu các thông tin như game progress. Bộ đôi Lee và Tamplin quyết định tách mảng Realtime ra để thành lập một công ty độc lập – chính là Firebase – vào tháng 4 năm 2012. Sau nhiều lần huy động vốn và gặt hái được những thành công nổi bật, Firebase đã được Google để ý. Vào tháng 10 năm 2014, Firebase gia nhập gia đình Google.

Cả Google và Firebase đều như hổ mọc thêm cánh. Firebase có điều kiện để phát triển thần tốc, mở rộng số lượng các dịch vụ con, còn Google có được một đội ngũ nhân lực chất lượng cao, năng động, cũng như cơ sở hạ tầng và sự hiệu quả mà các dịch vụ của Firebase mang lại, mà không phải xây dựng lại từ đầu. Hiện tại, Google đã chuyển các dịch vụ nền tảng hỗ trợ các lập trình viên bên ngoài về cho Firebase quản lí, chẳng hạn như Cloud Messaging, AdMob và Analytics.

Firebase, theo hướng đi của Google, chính thức hỗ trợ Android, iOS và Web. Thực tế, macOS cũng được hỗ trợ vì macOS chia sẻ nhiều dòng code với iOS, song vì Google và Firebase muốn sử dụng web cho ứng dụng desktop thay vì native, nên có khá ít tài liệu chính thức nói về Firebase cho macOS, cũng như các thư viện cho macOS có thể kém chức năng và không ổn định lắm. Còn về Windows, hiện tại tôi chưa thấy họ lên tiếng nào về việc sẽ chính thức phát hành thư viện cho đứa con của Microsoft, nên nếu các bạn muốn làm ứng dụng cho Windows (UWP) thì chỉ nên (và cũng chỉ có mỗi con đường) làm web-based native apps mà thôi.

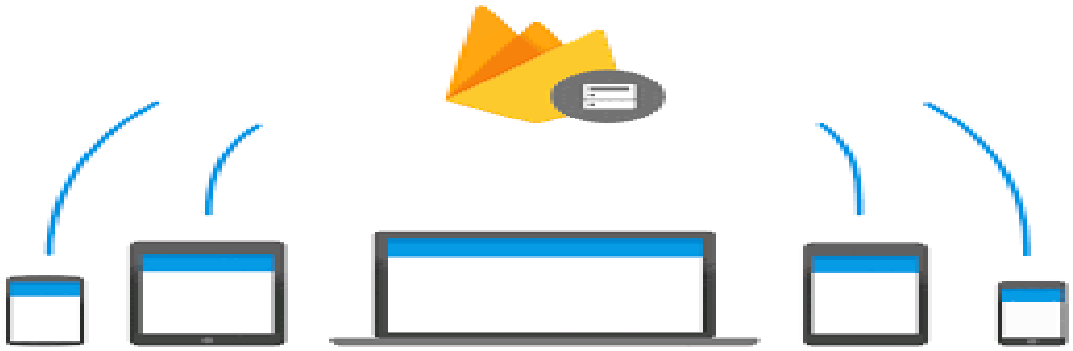
3. Các dịch vụ chính:

Bản chất của Google Firebase là ***Backend-as-a-service*** (BaaS), bao gồm một số dịch vụ cốt lõi cho việc phát triển ứng dụng:

- **Cloud Messaging** là một giải pháp tin nhắn đa nền tảng đáng tin cậy miễn phí. Mỗi tin nhắn dung lượng đến 4KB trong ứng dụng client.
- **Firebase Authentication** Hầu hết các ứng dụng cần xác thực quyền. Giúp ứng dụng lưu dữ liệu an toàn sử dụng trong các đám mây.
- **Firebase Realtime Database** cơ sở dữ liệu đám mây NoSQL đồng bộ hóa. Dữ liệu được đồng bộ hóa trên tất cả các client trong thời gian thực, và luôn có sẵn khi ứng dụng offline.
- **Firebase Storage** được xây dựng cho các nhà phát triển ứng dụng, để lưu trữ và phục vụ nội dung do người dùng tạo ra, chẳng hạn như hình ảnh hoặc video.
- **Firebase Hosting** nhanh chóng và an toàn lưu trữ tĩnh cho ứng dụng web.
- **Firebase Test Lab** cung cấp các thiết bị vật lý và ảo cho phép chạy thử nghiệm mô phỏng môi trường sử dụng thực tế
- **Firebase Crash Reporting** Thông tin toàn diện và hành động để giúp chẩn đoán và sửa chữa các vấn đề trong ứng dụng.
- **Cloud Firestore**: Bản nâng cấp của Realtime DB, hiện vẫn đang trong giai đoạn thử nghiệm (Beta)

Trong báo cáo này, chúng ta sẽ tập trung đi sâu và phân tích **Firebase Realtime Database**.

Phần 3: Firebase – Realtime Database



1. Giới thiệu:

Firebase Realtime Database là một cơ sở dữ liệu được lưu trữ trên Cloud (*Cloud-hosted database*). Dữ liêu được lưu trữ dưới dạng JSON và được đồng bộ hóa theo thời gian thực đối với mỗi client kết nối đến. Khi xây dựng một apps đa nền tảng như Android, IOS và JavaScript, tất cả các clients sẽ chia sẻ trên một cơ sở dữ liệu và tự động cập nhập mới nhất dữ liệu.

Các tính năng nổi bật của **Firebase Realtime Database**

- **Realtime**: Thay vì sử dụng cá HTTP requests đặc trưng, Firebase Realtime Database sử dụng dữ liệu đồng bộ - bất kì khi nào dữ liệu thay đổi, các thiết bị kết nối đến sẽ cập nhập sự thay đổi này trong vài ms. Cung cấp trải nghiệm thời gian thực một cách nhanh chóng mà không cần quan tâm quá nhiều về phương thức cài đặt phức tạp.
- **Offline**: Các Firebase apps vẫn phản hồi thậm trí khi bị offline bởi vì Firebase Realtime Database SDK duy trì tạm thời dữ liệu trên disk. Ngay sau khi kết nối được thiết lập trở lại, thiết bị client sẽ nhận những sự thay đổi đã diễn ra khi offline và đồng thời đồng bộ nó với trạng thái server hiện thời.
- **Accessible from Client Devices**: Các thiết bị di động hoặc trình duyện web có thể tiếp cận đến Firebase Realtime Database. Các Application server sẽ không còn cần thiết. Bảo mật và xác thực dữ liệu được cung cấp thông qua Firebase Database Security Rules, expression-based rules. Các rules này sẽ được thực thi khi dữ liệu được đọc hoặc viết.

- **Scale across multiple databases**: Với **Firebase Realtime Database** trên Blaze pricing plan, Firebase Realtime Database hỗ trợ scale dữ liệu của apps bằng cách lưu dữ liệu trên nhiều database instances trong cùng một dự án Firebase. Kiểm soát quyền truy cập đối với dữ liệu trên database được thực hiện bằng custom Firebase Realtime Database rules đối với mỗi database instance.

2. Cách thức hoạt động của Firebase Realtime Database:

Firebase Realtime Database giúp bạn xây dựng một ứng dụng thời gian thực trong một thời gian ngắn bằng cách cho phép client truy cập trực tiếp đến database một cách bảo mật. Dữ liệu được lưu trữ tạm thời ở local, thậm chí khi offline, các events realtime vẫn tiếp tục, đem lại cho người dùng trải nghiệm tốt nhất. Khi mà thiết bị được kết nối trở lại, Realtime Database đồng bộ những thay đổi ở dữ liệu local với những cập nhập xảy ra ở nơi khác khi mà client ở trạng thái offline, giải quyết tất cả dữ liệu một cách tự động.

Realtime Database cung cấp ngôn ngữ quy tắc linh hoạt expression-based, hay được gọi là *Firebase Realtime Database Security Rules*, giúp định nghĩa cách thức dữ liệu được cấu trúc và khi nào dữ liệu được đọc hoặc viết. Khi được tích hợp với *FireBase Authentication*, các nhà phát triển có thể định nghĩa ai có thể truy cập đến loại dữ liệu nào vào cách thức họ có thể truy cập đến chúng.

Realtime Database là một cơ sở dữ liệu NoSQL và có những chức năng và cách tối ưu khác so với các hệ cơ sở quan hệ. *Realtime Database API* được thiết kế dành riêng cho các thao tác cần thực hiện một cách nhanh chóng. Điều này giúp cho nhà phát triển có thể đem lại những trải nghiệm realtime tuyệt vời cho hàng triệu người dùng. Bởi vì vậy, nghĩ về cách thức mà người sử dụng cần để truy cập đến dữ liệu là một điều hết sức quan trọng và sau đó là cấu trúc nó tương ứng.

3. Mô hình dữ liệu:

Firebase Realtime Database có cấu trúc là một JSON Tree: Tất cả dữ liệu trong **Firebase Realtime Database** được lưu trữ như những đối tượng JSON. **Realtime Database** có thể được coi như một cây JSON – JSON tree được lưu trữ trên Cloud và khác với các mô hình dữ liệu quan hệ, sẽ không có bảng hay

các record. Khi thêm dữ liệu vào một cây JSON, nó trở thành một node trong cây JSON hiện tại theo cùng với khóa liên quan.

Lấy ví dụ, một ứng dụng chat cho phép người dùng lưu trữ hồ sơ thông tin cá nhân cơ bản và danh sách liên lạc. Một hồ sơ thông tin cá nhân người dùng đặc trưng được đặt trong một đường dẫn path giống như */users/uid.*

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { ... },
    "eclarke": { ... }
  }
}
```

4. Các bước cài đặt chung cho các nền tảng:

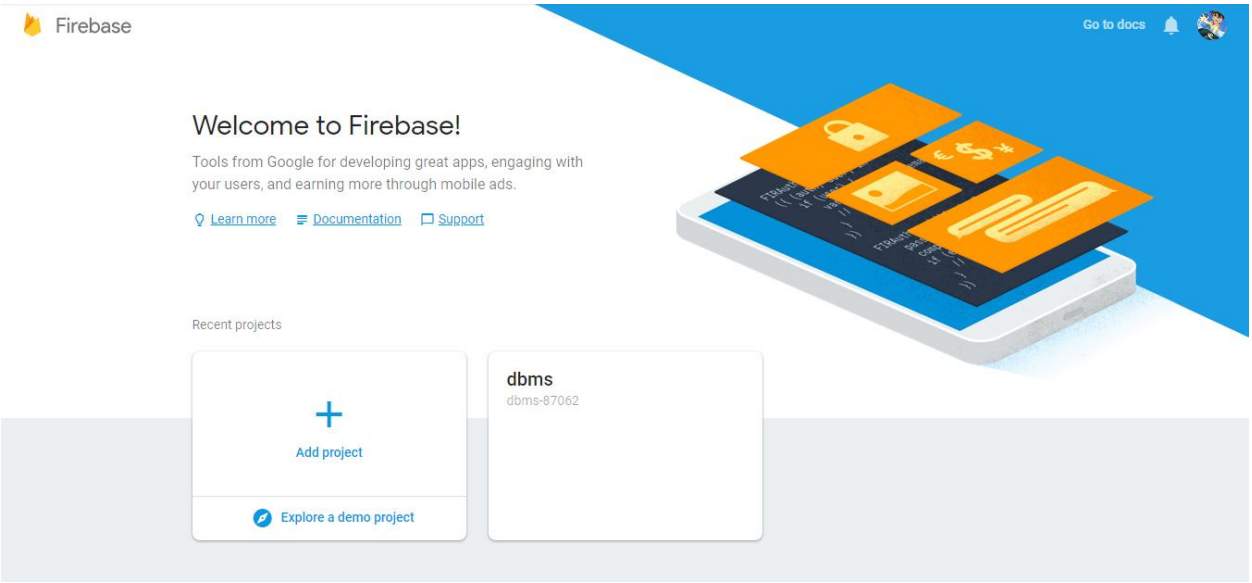
- Bước 1: Tích hợp *Firebase Realtime Database SDKs*
- Bước 2: Tạo *Realtime Database References*
- Bước 3: Tạo dữ liệu và lắng nghe sự thay đổi
- Bước 4: Kích hoạt *Offline Persistence* (cho phép dữ liệu được viết vào disk của thiết bị khi offline)
- Bước 5: Bảo mật dữ liệu: Sử dụng các quy tắc (rules) trong config để bảo mật dữ liệu của bạn.

Phần 4: Hướng dẫn sử dụng Firebase Realtime DB trên nền tảng Web

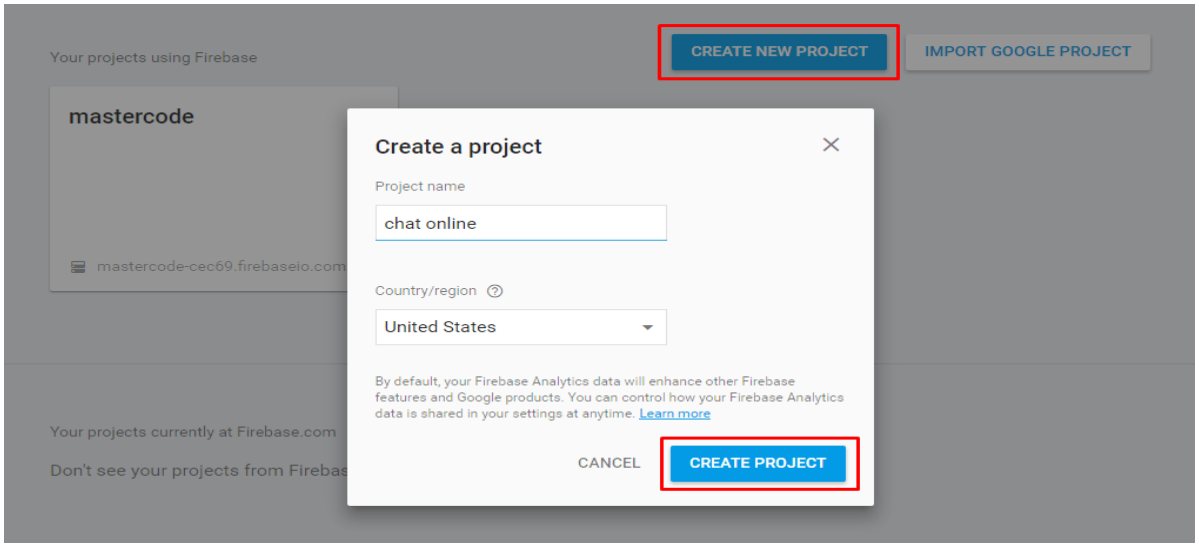
1. Đăng ký tài khoản sử dụng Firebase.

-- Truy cập *https://firebase.com* và đăng ký tài khoản hoặc đăng nhập bằng tài khoản Google đã có.

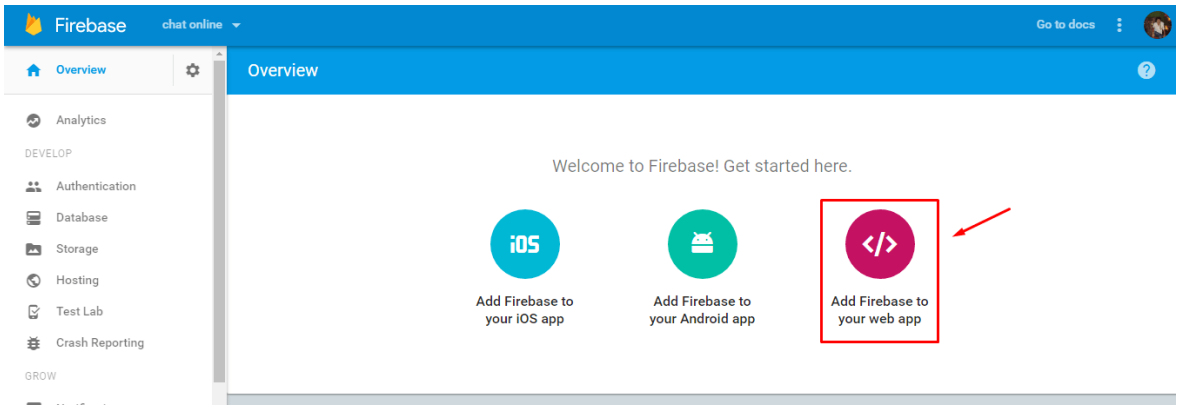
-- Đi tới bảng điều khiển tại địa chỉ: *https://console.firebase.google.com*



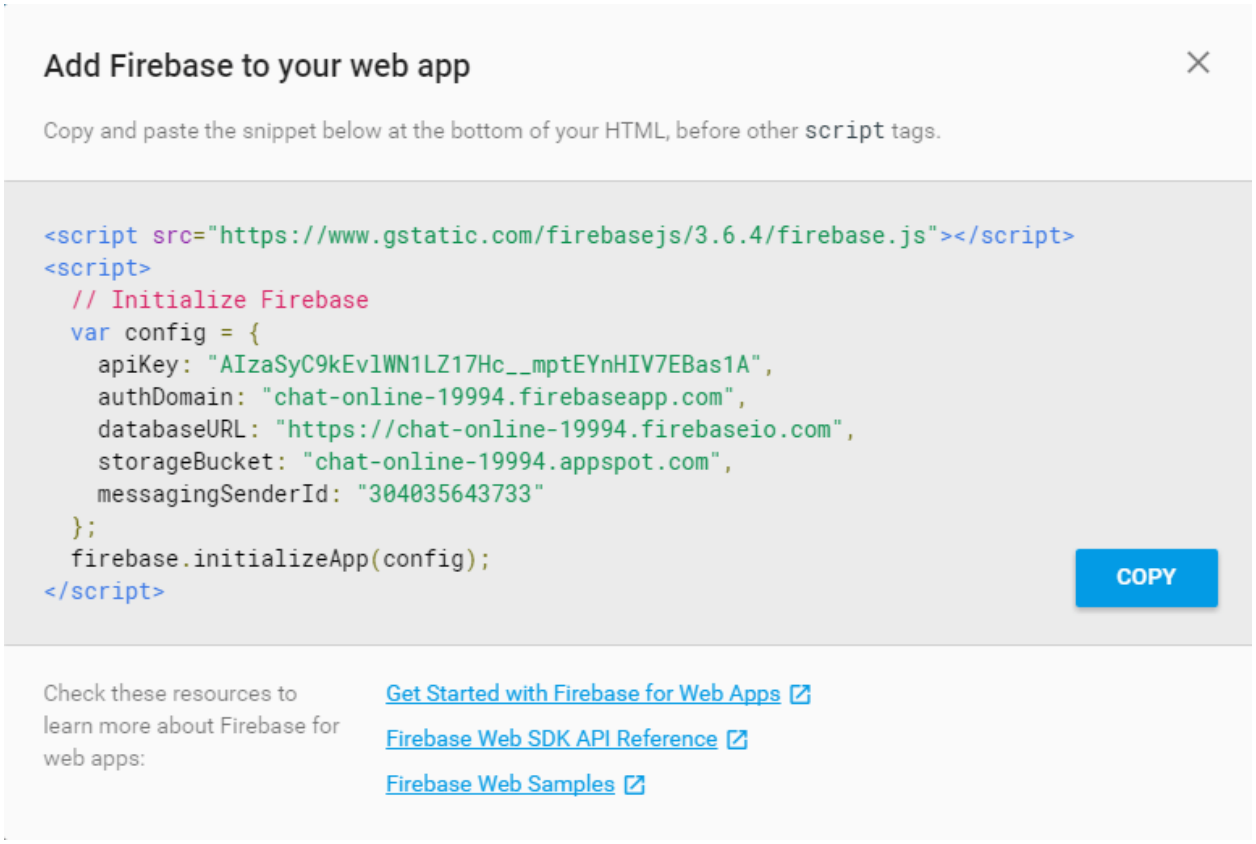
-- Tạo project mới



-- Chọn Add Firebase to your web app:



-- Copy đoạn mã config ở cửa sổ mới hiện lên:



2. Khởi tạo Realtime Database JavaScript SDK:

Bạn có thể tìm *URL Realtime DB* của mình trong tab "*Database*" trong bảng điều khiển *Firebase*. Nó sẽ ở dạng *https://<databaseName>.firebaseio.com.*

Khởi tạo SDK của bạn bằng đoạn đã copy bên trên, có dạng như sau:

```
// Set the configuration for your app
// TODO: Replace with your project's config object
var config = {
  apiKey: "apiKey",
  authDomain: "projectId.firebaseapp.com",
  databaseURL: "https://databaseName.firebaseio.com",
  storageBucket: "bucket.appspot.com"
};
firebase.initializeApp(config);
```

Tham chiếu cơ sở dữ liệu:

```
// Get a reference to the database service
var database = firebase.database();
```

3. Đọc và ghi

Sau khi đã có tham chiếu đến cơ sở dữ liệu, chúng ta có thể bắt đầu quá trình đọc, ghi.

a, Lệnh ghi dữ liệu: *set(value)*

Ví dụ: ghi username, email, imgUrl vào node *users/${user_id}*

```
function writeUserData(userId, name, email, imageUrl) {
  firebase.database().ref('users/' + userId).set({
    username: name,
    email: email,
    profile_picture : imageUrl
  });
}
```

Chú ý: Sử dụng *set()* sẽ ghi đè dữ liệu hiện có tại node tham chiếu.

b, Lệnh đọc dữ liệu:

-- Sự kiện *value* để đọc dữ liệu: Chúng ta có thể sử dụng sự kiện *value* để đọc nội dung tại một đường dẫn nhất định. Phương pháp này được kích hoạt khi dữ liệu, bao gồm cả node con có sự thay đổi. Kết quả trả về chứa tất cả dữ liệu tại vị trí đó, bao gồm cả dữ liệu ở node con. Nếu không có dữ liệu, lệch đọc sẽ trả về false khi gọi hàm *exist()* và null khi gọi *val()*.

```
var starCountRef = firebase.database().ref('posts/' + postId + '/starCount');
starCountRef.on('value', function(snapshot) {
  updateStarCount(postElement, snapshot.val());
});
```

-- Trong trường hợp, đọc dữ liệu một lần, chứ không quan tâm đến các lần thay đổi sau, thì chúng ta sử dụng *once()* thay vì *on()* như ví dụ trên

```
var userId = firebase.auth().currentUser.uid;
return firebase.database().ref('/users/' + userId).once('value').then(function(snapshot) {
  var username = (snapshot.val() && snapshot.val().username) || 'Anonymous';
  // ...
});
```

Kết quả trả về của ví dụ trong ảnh là giá trị *username* của node *users/${user_id}* tại thời điểm gọi hàm.

c, Cập nhật và xóa dữ liệu:

-- Cập nhật dữ liệu: bổ sung dữ liệu mà không làm ảnh hưởng tới các node con khác trên cùng node cha.

```
function writeNewPost(uid, username, picture, title, body) {
  // A post entry.
  var postData = {
    author: username,
    uid: uid,
    body: body,
    title: title,
    starCount: 0,
    authorPic: picture
  };

  // Get a key for a new Post.
  var newPostKey = firebase.database().ref().child('posts').push().key;

  // Write the new post's data simultaneously in the posts list and the user's post list.
  var updates = {};
  updates['/posts/' + newPostKey] = postData;
  updates['/user-posts/' + uid + '/' + newPostKey] = postData;

  return firebase.database().ref().update(updates);
}
```

Ví dụ này sử dụng push để tạo một bài đăng và cấp key cho bài đăng mới này. Lệnh *update()* sẽ cập nhật nội dung bài theo *key* đã lấy bên trên và cả *key* của bài đăng vào danh sách những bài đăng của *user* đó.

Trong trường hợp muốn biết khi nào dữ liệu được commit, chúng ta có thể sử dụng thêm hàm callback như sau (dùng cả với *set()* và *update()*):

```
  firebase.database().ref('users/' + userId).set({
    username: name,
    email: email,
    profile_picture : imageUrl
  }, function(error) {
    if (error) {
      // The write failed...
    } else {
      // Data saved successfully!
    }
  });
}
```

-- Xóa dữ liệu: remove() là cách đơn giản nhất để xóa toàn bộ dữ liệu ở một node.

4. Transaction trong Realtime DB.

Khi làm việc với Realtime DB, dữ liệu có thể bị gian đoạn hay hỏng do sửa đổi đồng thời từ nhiều người dùng, chẳng hạn như việc nhiều người cùng tạo mới mới một bài đăng trong ví dụ bên trên, sẽ dẫn đến bộ đếm hoặc postId gặp sự cố. Lúc này, chúng ta có thể nghĩ đến việc sử dụng một transaction. Hàm cập nhật nhận trạng thái hiện tại của dữ liệu làm đối số và trả về trạng thái mong muốn mới mà chúng ta muốn viết. Nếu một người dùng khác có thao tác tạo mới cùng lúc và vào cùng vị trí node trước khi giá trị mới của chúng ta được ghi thành công, hàm cập nhật sẽ được gọi lại với giá trị hiện tại mới và lệnh ghi được thử lại.

```
function toggleStar(postRef, uid) {
  postRef.transaction(function(post) {
    if (post) {
      if (post.stars && post.stars[uid]) {
        post.starCount--;
        post.stars[uid] = null;
      } else {
        post.starCount++;
        if (!post.stars) {
          post.stars = {};
        }
        post.stars[uid] = true;
      }
    }
    return post;
  });
}
```

5. Sắp xếp truy vấn

Các phương thức sắp sếp truy vấn:

-- *orderByChild(key)*: nhận tham số là một khóa, sắp xếp kết quả trả về theo giá trị của một khóa này.

-- *orderByKey()*: không nhận tham số, sắp xếp kêt quả trả về theo khóa.

-- *orderByValue()*: không nhận tham số, sắp xếp kêt quả trả về theo giá trị.

Tại một thời điểm, chỉ có thể sử dụng một trong các phương pháp sắp xếp trên.

Ví dụ: Truy vấn danh sánh bài đăng của một *User* và sắp xếp theo số lượng đánh giá của người đọc (*starCount*)

```
var myUserId = firebase.auth().currentUser.uid;
var topUserPostsRef = firebase.database().ref('user-posts/' + myUserId).orderByChild('starCount');
```

6. Lọc dữ liệu: Các phương thức lọc dữ liệu:

Phương thức	**Sử dụng**
limitToFirst()	Giới hạn kết quả trả về tính từ kết quả đầu tiên
limitToLast()	Giới hạn kết quả trả về tính từ kết quả cuối cùng
startAt()	Trả về các bản ghi bằng hoặc lớn hơn giá trị của khóa (*key*) hoặc giá trị (*value*), phụ thuộc vào phương thức sắp xếp (*orderby--*)
endAt()	Trả về các bản ghi bằng hoặc nhỏ hơn giá trị của khóa (*key*) hoặc giá trị (*value*), phụ thuộc vào phương thức sắp xếp (*orderby--*)
equalTo()	Trả về các bản ghi bằng giá trị của khóa (*key*) hoặc giá trị (*value*), phụ thuộc vào phương thức sắp xếp (*orderby--*)

Không như các phương thức sắp xếp, các phương thức giới hạn có thể sử dụng kết hợp với nhau để đạt được kết quả truy vấn cuối cùng.

7. Sử dụng database trong kiến trúc REST

a, Xác thực người dùng: Có 2 phương pháp cơ bản để xác thực:

-- **Google OAuth2 access tokens**: Thông thường, khả năng đọc và ghi vào cơ sở dữ liệu thời gian thực được điều chỉnh bởi các RULES. Tuy nhiên, bạn có thể truy cập dữ liệu của mình từ máy chủ và cấp cho máy chủ đó quyền truy cập đọc và ghi đầy đủ vào dữ liệu của bạn bằng mã thông báo truy cập Google OAuth2 được tạo từ tài khoản dịch vụ.

Hướng dẫn tạo *Oauth2 token*: Sử dụng một trong hai thư viện Google API:

- https://www.googleapis.com/auth/userinfo.email
- *https://www.googleapis.com/auth/firebase.database*

Code mẫu với Node Js:

```
var {google} = require("googleapis");

// Load the service account key JSON file.
var serviceAccount = require("path/to/serviceAccountKey.json");

// Define the required scopes.
var scopes = [
  "https://www.googleapis.com/auth/userinfo.email",
  "https://www.googleapis.com/auth/firebase.database"
];

// Authenticate a JWT client with the service account.
var jwtClient = new google.auth.JWT(
  serviceAccount.client_email,
  null,
  serviceAccount.private_key,
  scopes
);

// Use the JWT client to generate an access token.
jwtClient.authorize(function(error, tokens) {
  if (error) {
    console.log("Error making request to generate access token:", error);
  } else if (tokens.access_token === null) {
    console.log("Provided service account does not have permission to generate access tokens");
  } else {
    var accessToken = tokens.access_token;

    // See the "Using the access token" section below for information
    // on how to use the access token to send authenticated requests to
    // the Realtime Database REST API.
  }
});
```

Sử dụng **Access Token** với theo cú pháp:

```
curl
"https://<DATABASE_NAME>.firebaseio.com/users/ada/name.json?access_token=<
ACCESS_TOKEN>"
```

-- **Firebase ID tokens**: Bạn cũng có thể muốn gửi yêu cầu được xác thực là người dùng cá nhân, chẳng hạn như hạn chế quyền truy cập bằng *RULES* trên SDK

client. API REST chấp nhận các mã ID Firebase giống nhau được các SDK ứng dụng khách sử dụng.

Hướng dẫn tạo **ID token**: Khi người dùng hoặc thiết bị đăng nhập thành công, Firebase sẽ tạo **ID token** tương ứng để nhận dạng duy nhất chúng và cấp cho họ quyền truy cập vào một số tài nguyên. Bạn có thể sử dụng lại mã ID token đó để xác định người dùng hoặc thiết bị trên máy chủ phụ trợ tùy chỉnh của mình. Để lấy mã **ID token** từ ứng dụng client, hãy đảm bảo người dùng đã đăng nhập và thực hiện như sau (Javascript):

```
firebase.auth().currentUser.getIdToken(/* forceRefresh */ true).then(function(idToken) {
  // Send token to your backend via HTTPS
  // ...
}).catch(function(error) {
  // Handle error
});
```

Sử dụng **ID token**:

```
curl
"https://<DATABASE_NAME>.firebaseio.com/users/ada/name.json?auth=<ID_TOKEN
>"
```

b, Áp dụng REST với cURL:

-- Đọc dữ liệu -- GET:

```
curl 'https://<DATABASE_NAME>.firebaseio.com/<PATH>.json'
```

-- Ghi dữ liệu -- PUT:

```
curl -X PUT -d {data}
'https://<DATABASE_NAME>.firebaseio.com/<PATH>.json'
```

-- Cập nhật dữ liệu – PATCH:

```
curl -X PATCH -d
'https://<DATABASE_NAME>.firebaseio.com/<PATH>.json'
```

-- Xóa dữ liệu:

```
curl -X DELETE
'https://<DATABASE_NAME>.firebaseio.com/<PATH>.json'
```

8. Một vài điểm cần lưu ý trong cơ chế Security và Rules của Realtime DB.

a, Realtime DB Rules

-- *Realtime DB Rules* xác định xem người dùng nào có quyền đọc hoặc ghi, quyền tổ chức cũng như đánh chỉ mục (indexing) dữ liệu. Các quy tắc này được setup ngay trên server Firebase và tuân thủ ở mọi nơi. Theo mặc định, chỉ những người dùng đã được xác thực mới có quyền đọc và ghi dữ liệu. Chủ sở hữu của database có thể chỉnh sửa rules này để phục vụ mục tiêu của mình.

-- Các quyền cơ bản:

- **.read**: định nghĩa quyền đọc dữ liệu
- **.write**: định nghĩa quyền ghi dữ liệu
- **.validate**: định nghĩa format cho value
- **.indexOn**: xác định node con để đánh chỉ mục, phục vụ truy vấn và sắp xếp

b, Realtime DB Security

Realtime DB cung cấp một bộ công cụ đầy đủ để đảm bảo vấn đề Bảo mật cho Ứng dụng. Bao gồm các quá trình/ tính năng:

- Xác thực (*Authentication*): Bước đầu tiên trong quá trình bảo mật dữ liệu là xác thực người dùng. Một tùy chọn cho các nhà phát triển Ứng dụng đó là sử dụng Realtime DB kèm với dịch vụ bổ sung là *Firebase Authentication*. Dịch vụ này cho phép người dùng đăng nhập vào Ứng dụng của bạn. Ngoài ra, *Firebase Authentication* còn có các chế độ đăng nhập bằng tài khoản Google, Facebook, hoặc email, đăng nhập ẩn danh, …
- Ủy quyền (*Authozitation*): Sau khi xác thực, bước tiếp theo là quản lý khả năng truy cập tới dữ liệu của mỗi người dùng.

Ví dụ:

Cài đặt ủy quyền cho phép tất cả người dùng đều có quyền đọc nhưng không có quyền ghi đối với dữ liệu tại `/foo/:`

```
{
  "rules": {
    "foo": {
      ".read": true,
      ".write": false
    }
  }
}
```

Cài đặt chỉ cho phép những người dùng đã xác thực mới có quyền ghi tại `/users/<uid>/`, với <uid> là id của người dùng được xác thực thông qua *Firebase Authentication*

```
{
  "rules": {
    "users": {
      "$uid": {
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

- Xác thực dữ liệu (*Data Validation*): Dữ liệu sẽ được xác thực trước khi đưa vào Database, ví dụ, kiểm tra dữ liệu là chuỗi và có độ dài kí tự < 100:

```
{
  "rules": {
    "foo": {
      ".validate": "newData.isString() && newData.val().length < 100"
    }
  }
}
```

9. Indexing trong Realtime DB.

a, Giới thiệu: Firebase cho phép bạn thực hiện truy vấn đặc biệt trên dữ liệu của mình bằng cách sử dụng khóa con tùy ý. Nếu bạn biết các index trong database của bạn là gì, bạn có thể xác định truy vấn thông qua quy tắc .indexOn trong RULES của bạn để cải thiện hiệu suất truy vấn.

b, Sử dụng

-- Đánh chỉ mục với phương thức *orderByChild()*: Giả sử có bộ dữ liệu về khủng long như sau:

```
{
  "lambeosaurus": {
    "height" : 2.1,
    "length" : 12.5,
    "weight": 5000
  },
  "stegosaurus": {
    "height" : 4,
    "length" : 9,
    "weight" : 2500
  }
}
```

Với điều kiện giả định là sắp xếp dữ liệu trả về chỉ theo chiều *cao - height* và *chiều dài - length* chứ không theo *cân nặng – weight*, chúng ta có thể tối ưu truy vấn bằng cách dùng index thông qua *indexOn()*

```
{
  "rules": {
    "dinosaurs": {
      ".indexOn": ["height", "length"]
    }
  }
}
```

-- Đánh chỉ mục với *orderByValue()*: Giả sử có bộ dữ liệu

```
{
  "scores": {
    "bruhathkayosaurus" : 55,
    "lambeosaurus" : 21,
    "linhenykus" : 80,
    "pterodactyl" : 93,
    "stegosaurus" : 5,
    "triceratops" : 22
  }
}
```

Trong trường hợp muốn tạo bảng xếp hạng từ bộ dữ liệu trên, có thể sử dụng chỉ mục đối với '*.value*':

```
{
  "rules": {
    "scores": {
      ".indexOn": ".value"
    }
  }
}
```

10. Khả năng sử dụng

** Chi phí sử dụng **Realtime Database**: Do *Firebase* có yêu cầu trả phí và chỉ cho phép miễn phí một lượng lưu trữ và download nhất định nên chúng ta cần tìm hiểu về các khoản phí mà Google yêu cầu trả để có thể phát triển ứng dụng một cách hoàn thiện nhất

- *Firebase* thu phí cho dữ liệu mà ta lưu trong cơ sở dữ liệu và mọi kết nối tại tầng Session trong mô hình OSI.

- *Kho lưu trữ (storage)* thu phí $5 mỗi GB/tháng. Các kết nối bao gồm các kết nối và giải mã từ các phương thức truy cập cơ sở dữ liệu và dữ liệu được tải về sau khi được đọc từ cơ sở dữ liệu.

- Cả 2 phương thức đọc và ghi dữ liệu đều ảnh hưởng tới chi phí mà bạn phải trả.

- Các loại traffic phải trả phí khi vượt quá giới hạn miễn phí:
    - *Data downloaded*: Khi người dùng export dữ liệu từ cơ sở dữ liệu, Firebase tính phí cho việc tải về dữ liệu.
    - *Protocol Overhed*: bao gồm Firebase Realtime Database's realtime protocol, WebSocket, HTTP header
    - *SSL encryption overhead*: Trung bình tiêu tốn khoảng 3.5KB cho mỗi lần khởi tạo bắt tay và khoảng vài chục byte cho TLC record header trên mỗi message được gửi đi.
    - *Firebase console data*: Các dữ liệu phục vụ cho việc đọc và ghi từ Firebase console đều được tính vào chi phí.

- Ước tính lượng sử dụng phải trả: Để xem lượng sử dụng tài nguyên *Realtime Database* và ước tính số tiền phải trả, cá nhà phát triển có thể kiểm tra trong tab '*Usage*' trong Firebase console theo 3 khoảng thời gian có sẵn là trong khoảng thời gian hợp đồng, trong vòng 30 ngày hoặc trong vòng 24h.

- Các cách giảm chi phí sử dụng:
    - Sử dụng native SDKs: Khi có thể hãy sử dụng SDKs tương ứng với platform của ứng dụng, thay vì REST API. Việc này sẽ giảm thiểu lượng tiêu tốn để mã hoá SSL trong việc sử dụng REST API.
    - Sử dụng Indexing: Việc indexing sẽ giảm được lưu lượng sử dụng để query và tăng hiệu năng của cơ sở dữ liệu.
    - Giảm thiếu số lượng kết nối không cần thiết
    - Sử dụng TLS session ticket
    - Tối ưu listener (lắng nghe sự kiện)
    - Giảm lượng lưu trữ trong kho lưu trữ (storage): dọn dẹp cơ sở dữ liệu với những dữ liệu dư thừa hoặc trùng lặp.
    - Sử dụng Rules phù hợp với từng nhóm dữ liệu hoặc nhóm người sử dụng.
    - Kiểm tra các lỗi gây thất thoát tài nguyên trong ứng dụng.

Phần 5: So sánh, đánh giá hiệu năng của Realtime Database với Mysql