TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

KHOA CÔNG NGHỆ THÔNG TIN

Báo cáo nghiên cứu hệ quản trị cơ sở dữ liệu

Chủ đề: FireBase

Giảng viên hướng dẫn : Ths Dư Phương Hạnh

Nhóm sinh viên thực hiện (Nhóm 7) :

Nguyễn Hoàng Long – 15021752
Trần Minh Chiến - 15021359
Nguyễn Trung Đức - 15021436

Phần 1. Đặt vấn đề:

Mọi cơ sở dữ liệu (CSDL – database) đều được tạo ra theo cách không giống nhau, mỗi loại đều có ưu và nhược điểm riêng. Thực tế cho thấy những loại database như MySQL, MongoDB đôi khi bị "lạm dụng" vì tính phổ biến của nó, bất chấp tính tương thích với dự án / nhu cầu hiện tại. Không ai có thể phủ nhận sự lấn áp đến hoàn toàn của RDBMS vì tính phổ biến và cấu trúc, tuy nhiên dữ liệu dạng SQL không xử lý tốt các dữ liệu dạng phi cấu trúc, tốc độ xử lí dữ liệu trên nhiều bảng tính cũng khá chậm, và khó để mở rộng. Do đó, cơ sở dữ liệu dạng NoSQL tỏ ra vô cùng linh hoạt, có thể xử lí dữ liệu nửa cấu trúc và không cấu trúc rất tốt, dễ dàng cài đặt và mở rộng. Tuy nhiên với giới hạn sử dụng của máy tính cá nhân, nhiều người đã lựa chọn sử dụng các cơ sở dữ liệu trên cloud.

Phần 2. Firebase

I. Giới thiệu về Firebase - Realtime Database (iOS, Android, Web, C++ and Unity support)

Firebase là gì ?

- Firebase thực chất không phải là một hệ quản trị cơ sở dữ liệu truyền thống như các HQTCSDL khác, bản chất của Firebase là BAAS (Back-end as a service) trong đó có một tính năng là Realtime Database. Tính năng này của

Firebase dùng để lưu trữ và đồng bộ dữ liệu với một hệ quản trị cơ sở dữ liệu trên mây dạng NoSQL (NoSQL cloud database). Dữ liệu được đồng bộ giữa mọi client theo thời gian thực, và có thể sử dụng ngay cả khi offline.

- Firebase Realtime Database là một cơ sở dữ liệu trên mây. Dữ liệu được lưu trữ theo dạng JSON và đồng bộ theo thời gian thực với mọi client kết nối với nó. Khi xây dựng ứng dụng đa nền tảng, tất cả các clients sử dụng chung một cơ sở dữ liệu và tự động nhận được sự thay đổi với dữ liệu mới nhất.

Các tính năng nổi bật của Firebase

- Realtime: Thay vì sử dụng HTTP request, Firebase Realtime Database sử dụng dữ liệu đồng bộ - mỗi khi có sự thay đổi về dữ liệu, mọi kết nối với Firebase sẽ tự động nhận được cập nhật trong vài ms. Cung cấp trải nghiệm thời gian thực một cách nhanh chóng mà không cần quan tâm về phương thức cài đặt phức tạp nào cả.
- Offline: ứng dụng Firebase vẫn phản hồi khi offline vì Firebase Realtime Database SDK lưu trữ tạm thời dữ liệu trên đĩa. Khi kết nối được tái khởi tạo, client sẽ đồng bộ với trạng thái của server và cập nhập sự thay đổi về dữ liệu một cách tự động.
- Accessible from Client Devices: Firebase Realtime Database có thể trực tiếp truy cập từ một client như mobile hoặc web app mà không cần thông qua một ứng dụng server nào cả. Bảo mật và xác thực dữ liệu cũng có thể chỉnh sửa thông qua Firebase Realtime Database Security Rules có sẵn trên giao diện console của Google.
- Scale across multiple databases: tính năng này chỉ có trên phiên bản Firebase trả phí nên tạm thời không bàn tới.

Khi nào không sử dụng Firebase Realtime Database:

- Firebase Remote Config: lưu trữ các thiết lập về ứng dụng sẽ hiển thị như thế nào mà không yêu cầu users phải tải về cập nhật.
- Firebase Hosting: lưu trữ mã HTML, CSS và JavaScript, graphics, fonts và icons.
- Cloud Storage: lưu trữ ảnh, video và âm thanh.

Firebase hoạt động như thế nào ?

Firebase Realtime Database giúp bạn xây dựng một ứng dụng nhanh, thời gian thực với việc cho phép truy cập database trực tiếp từ phía mã nguồn của client. Dữ liệu được lưu trữ tạm thời bằng ổ đĩa, và kể cả khi đã offline, những sự kiện thời gian thực vẫn được "fire" hay gọi bình thường, mang đến cho người dùng trải

nghiệm realtime mượt mà. Khi thiết bị được kết nối lại với mạng Internet, Realtime Database đồng bộ dữ liệu trên local với server và tự động cập nhật các sự thay đổi khi ứng dụng offline, hợp nhất tất cả các xung đột một cách tự động. Realtime Database cung cấp một ngôn ngữ linh động, có quy tắc dựa trên biểu thức với cái tên Firebase Realtime Database Security Rules, để định nghĩa dữ liệu sẽ được cấu trúc như thế nào và khi nào dữ liệu có thể đọc hoặc ghi. Khi tích hợp với Firebase Authentication, lập trình viên có thể định nghĩa ai có thể truy cập dữ liệu và cách họ sử dụng dữ liệu đó.
Realtime Database là một cơ sử dữ liệu dạng NoSQL có cách tối ưu hoá và chức năng khác với cơ sở dữ liệu quan hệ. Realtime Database API được thiết kế cho các phương thức được cho phép để thực hiện một cách nhanh chóng. Nó cho phép người lập trình xây dựng một trải nghiệm thời gian thực có thể phục vụ được hàng triệu người sử dụng mà không ảnh hưởng tới sự phản hồi.

Các bước cài đặt cơ bản

Có 5 bước cài đặt cơ bản trên mọi nền tảng để sử dụng Firebase:

1. Tích hợp Firebase Realtime Database SDKs (bằng cách sử dụng Gradle, CocoaPods hoặc thẻ script).
2. Tạo Realtime Database References.
3. Tạo dữ liệu và theo dõi sự thay đổi.
4. Kích hoạt chế độ Offline cho phép dữ liệu có thể ghi vào ổ đĩa khi ứng dụng offline.
5. Bảo mật dữ liệu.

So sánh Realtime Database và Cloud Firestore

Firebase cung cấp 2 cơ sở dữ liệu trên mây và có thể truy cập từ phía client và hỗ trợ đồng bộ dữ liệu thời gian thực:

- Realtime Database là phiên bản đầu tiên của cơ sở dữ liệu trên Firebase. Nó hiệu quả, thời gian trễ thấp cho ứng dụng mobile yêu cầu đồng bộ dữ liệu trong thời gian thực với nhiều thiết bị.

- Cloud Firestore là một phiên bản mới của Firebase dành cho phát triển ứng dụng trên di động. Nó thừa kế sự thành công của Realtime Database với một cấu trúc dữ liệu mới và trực quan hơn. Cloud Firestore cũng có nhiều chức năng hơn, tốc độ truy xuất dữ liệu nhanh hơn và khả năng mở rộng cũng tốt hơn Realtime Database.

Tuy nhiên do Cloud Firestore đang ở trong giai đoạn beta nên chúng ta sẽ chỉ tìm hiểu sâu về Realtime Database. Các sự khác nhau chi tiết giữa 2 hệ quản trị cơ sở dữ liệu của Firebase có thể đọc thêm ở trên trang chủ.

Tổ chức dữ liệu trong Firebase

Tất cả Firebase Realtime Database được lưu dữ theo định dạng JSON. Chúng ta có thể hiểu rằng cơ sở dữ liệu sẽ được lưu dữ trên đám mây theo dạng cây JSON. Khác với cơ sở dữ liệu SQL, không có các bảng hay các ghi chú. Khi chúng ta thêm thông tin vào cây JSON, nó trở thành node của cấu trúc JSON hiện tại với khóa liên quan. Chúng ta có thể tự cung cấp khóa như ID người dùng hoặc tên ngữ nghĩa.

II. Hướng dẫn sử dụng Firebase

Trong dự án này chúng ta chỉ tập trung vào cách sử dụng Firebase Realtime Database với nền tảng Web và REST API. Còn cách sử dụng trên các nền tảng khác có thể tra cứu trên trang chủ của Firebase.

Web

Khởi tạo Realtime Database Javascript SDK

Chúng ta phải xác định Realtime Database URL(trong Firebase console, URL sẽ theo dạng https://<databaseName>.firebaseio.com) khi khởi tạo Javascript SDK.

Khởi tạo SDK theo đoạn mã 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);

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

Truy cập cơ sở dữ liệu:
Để đọc hoặc ghi dữ liệu từ cơ sở dữ liệu, bạn cần truy cập `firebase.database.Reference`:

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

Đọc và viết dữ liệu:
Dữ liệu của Firebase được lấy bởi gắn nhiều listener không đồng bộ với `firebase.database.Reference`. Listener được kích hoạt vào trạng thái ban đầu của dữ liệu và bất cứ lúc nào dữ liệu thay đổi

Lệnh viết đơn giản:
Với lệnh viết đơn giản, chúng ta có thể sử dung `set()` để lưu dữ liệu vào tham chiếu chỉ định, thay đổi bất cứ dữ liệu nào đang tồn tại tại đường dẫn đó. Ví dụ một mạng xã hội blog có thể thêm người dùng dùng lệnh `set()` như sau:

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

Sử dụng `set()` sẽ chèn tất cả các dữ liệu tại một vị trí rõ ràng, bao gồm cả các node con.

Nghe các giá trị của sự kiện:
Để đọc dữ liệu tại mộ đường dẫn và nhận sự thay đổi, sử dụng `on()` hoặc `once()` của `firebase.database.Reference` để theo dõi sự kiện

Event	Typical usage
`value`	Read and listen for changes to the entire contents of a path.

Chúng ta có thể sử dụng `value` để đọc nội dung tĩnh theo đường dẫn cho trước nếu nó tồn tại tại thời điểm sự kiện. Phương pháp này kích hoạt khi listener được sử dụng và sử dụng lại mỗi khi dữ liệu, kể cả nhánh con thay đổi. Callback của sự kiện thông qua việc lấy giữ liệu chưa tất cả các dữ liệu tại vị trí đó, bao gồm cả dữ liệu con. Nếu không có dữ liệu, chương trình sẽ trả về `null`.

Ví dụ sau mô tả một mạng xã hội blog nhận số sao nhận xét tại một bài viết từ cơ sở dữ liệu:

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

Listener nhận một chứa dữ liệu ở một `snapshot` tại vị trí cố định ở cơ sở dữ liệu tại thời điểm của sự kiện. Chúng ta có thể lấy dũ liệu ở trong bằng `val()`.

Đọc dữ liệu một lần:
Trong một số trường hợp chúng ta muốn dữ liệu nhận được mà không đợi thay đổi, chẳng hạn như đang nhận giá trị UI mà chúng ta không muốn thay đổi. Chúng ta có thể sự đụng `once()` để đơn giản tình huống này: nó kích hoạt một lần và không kích hoạt lại.
Việc này có ích cho dữ liệu chỉ được sử dụng một lần và không có nhu cầu thay đổi thường xuyên hay liên tục lắng nghe. Ví dụ, ứng dụng mạng xã hội blog từ ví dụ trước sử dụng phương pháp này để sử dụng dữ liệu người dùng khi học bắt đầu viết bài đăng mới:

```
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';
  // ...
});
```

Cập nhật hoặc xóa dữ liệu:
Cập nhật vùng định trước:
Để đồng thời viết vào nhánh con đã xác định trước mà không ghi đè lên các nhánh con khác, sử dụng `update()`.
Khi sử dụng `update()`,chúng ta có thể cập nhật giá trị nhánh con thấp hơn bằng cách chỉ rõ đường dẫn của khóa. Nếu dữ liệu được chứa ở nhiều vị trí để tiện, bạn có thể cập nhật tất cả dữ liệu sử dụng data fan-out.
Ví dụ, một mạng xã hội blog có thể tạo một bài báo và đồng thời cập nhật lên hoạt đông thường xuyên và nó được sử dụng theo đoạn mã như sau:

```
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 báo ở nhánh chứa các bài báo cho tất cả các người dùng tại `/posts/$postid` và đồng thời lấy khóa. Khóa có thể sử dụng để tạo ra cổng thử hai trong bài báo của người dùng ở `/user-posts/$userid/$postid`.

Sử dụng những đường dẫn này, chúng ta có thể thể hiện đồng thời các cập nhật đến với nhiều vị trí trong cây JSON với một câu gọi `update()`, chẳng hạn ví dụ này tạo bài báo ở cả hai vị trí. Đồng thời cập nhật theo cách này được gọi là atomic: tất cả các cập nhật thành công hoặc tất cả các bản cập nhật đều không thành công.

Xóa dữ liệu:
Các đơn giản nhất để xóa dữ liệu là sử dụng `remove()` trên một tham chiếu đến vị trí của dữ liệu đó.

Chúng ta cũng có thể xóa bằng cách đặt như là `null` giá trị cho các lệnh viết như là `set()` hoặc `update()`. Chúng ta có thể sử dụng kỹ năng này với `update()` để xóa nhiều nhánh con trong một lần gọi API.

Nhận một Promise:
Để biết khi nào dữ liệu của bạn được gửi lên server Firebase Realtime Database, bạn có thể sử dụng Promise. Cả hai và có thể trả về `Promise` mà dùng để biết khi nào việc gửi lên cơ sở dữ liệu được thực hiện.

Tách các listener:
Các callback được xóa bởi sử dụng phương thức `off()` ở trong Firebase database. Chúng ta có thể loại bỏ listener đơn bằng cách đưa vào như một biến vào `off()`. Sử dụng `off()` tại vị trí mà không có đối số nào xóa tất cả listener tại vị trí đó.
Sử dụng `off()` trên các listener cao hơn không tự động xóa bỏ các listener con trong nhánh con của nó; `off()` phải được gọi trên bất cứ nhánh con listener để loại bỏ callback. 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 ghi.

Lưu dữ liệu dưới dạng giao dịch:
Khi làm việc với dữ liệu có thể bị hỏng do sửa đổi đồng thời, chẳng hạn như bộ đếm gia tăng, chúng ta có thể sử dụng hoạt động giao dịch. Chúng ta có thể cung cấp cho hoạt động này một chức năng cập nhật và hoàn thành tùy chọn callback. Nếu một ứng dụng khách khác ghi vào vị trí trước khi giá trị mới của bạn được ghi thành công, hàm cập nhật của chúng ta sẽ được gọi lại với giá trị hiện tại mới và ghi sẽ được thử lại.
Ví dụ, trong ứng dụng mạng xã hội blog, bạn có thể cho phép người dùng lưu và bỏ lưu bài báo và theo dõi đã bao nhiêu người lưu bài báo như sau:

```
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;
  });
}
```

Sử dụng giao dịch ngăn không cho số lượng sao không chính xác nếu nhiều người dùng gắn dấu sao cùng một bài đăng cùng một lúc hoặc khách hàng có dữ liệu cũ. Nếu giao dịch bị từ chối, máy chủ trả về giá trị hiện tại cho người dùng, sẽ chạy lại giao dịch bằng giá trị được cập nhật. Điều này lặp lại cho đến khi giao dịch được chấp nhận hoặc chúng ta hủy giao dịch.

Sắp xếp dữ liệu:
Để lấy dữ liệu đã sắp xếp, hãy bắt đầu bằng cách chỉ định một trong các phương thức theo thứ tự để xác định cách các kết quả được sắp xếp:

Phương thức	Tính năng
orderByChild()	Sắp xếp kết quả bằng giá trị của khóa cho trước
orderByKey()	Sắp xếp kết quả bằng khóa con
orderByValue()	Sắp xếp kết quả bằng giá trị khóa con

Chúng ta chỉ có thể sử dụng một phương thức đặt hàng tại một thời điểm. Gọi một phương thức đặt hàng nhiều lần trong cùng một truy vấn sẽ đưa ra một lỗi.

Ví dụ sau minh họa cách bạn có thể truy xuất danh sách các bài đăng hàng đầu của người dùng được sắp xếp theo số lượng lưu của họ:

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

Lọc dữ liệu:
Để lọc dữ liệu, chúng tacó thể kết hợp bất kỳ phương pháp giới hạn hoặc phạm vi nào với phương thức theo thứ tự khi tạo truy vấn.

Phương thức	Tính năng
limitToFirst()	Đặt số lượng mục tối đa để trở về từ đầu danh sách kết quả đã sắp xếp
limitToLast()	Đặt số mục tối đa cần trả lại từ cuối danh sách kết quả đã sắp xếp.
startAt()	Trả về các mục lớn hơn hoặc bằng khóa hoặc giá trị được chỉ định, tùy thuộc vào phương thức theo thứ tự đã chọn.
endAt()	Trả lại các mục nhỏ hơn hoặc bằng khóa hoặc giá trị được chỉ định, tùy thuộc vào phương thức theo thứ tự đã chọn.
equalTo()	Trả về các mục bằng khóa hoặc giá trị được chỉ định, tùy thuộc vào phương thức theo thứ tự đã chọn.

Không giống như các phương thức theo thứ tự, chúng ta có thể kết hợp nhiều hàm giới hạn hoặc phạm vi. Ví dụ: chúng ta có thể kết hợp `startAt()` và `endAt()` để giới hạn kết quả tại vùng được xác định.

REST

Xác thực Rest request

Xác thực thường được sử dụng một trong 2 cách sau:

- Google OAuth2 access tokens: Thông thường , khả năng đọc và ghi dữ liệu vào cơ sở dữ liệu thời gian thực được quy định bởi các quy tắc đã được định rõ. Tuy nhiên , cũng có thể truy cập dữ liệu từ server và cung cấp cho server đo quyền truy cập đọc và ghi vào dữ liệu thông qua token được cung cấp từ Google OAuth2 được tạo từ dịch vụ của tài khoản

- Firebase ID tokens: API REST chấp nhận các mã thông báo ID Firebase giống nhau từ các ứng dụng SDK của client

Google OAuth2 access tokens:

- Tạo access token:
  Có thể sử dụng một trong các thư viện Google API sau đây để tạo access tokens
  - https://www.googleapis.com/auth/userinfo.email
  - https://www.googleapis.com/auth/firebase.database
- Xác thực với access token:
  Sau khi có được token , khi tạo truy vấn với cớ sở dữ liệu sẽ có dạng:
  -curl "https://<DATABASE_NAME>.firebaseio.com/users/ada/name.json?access_token=<ACCESS_TOKEN>"

Firebase ID tokens

- Tạo ID token
  Sau khi người dùng hoặc thiết bị đăng nhập thành công, Firebase sẽ tạo mã thông báo ID tương ứng để nhận dạng duy nhất và cấp quyền truy cập vào một số tài nguyên
- Xác thực với ID token
  Để gửi các yêu cầu được xác thực với API REST , cần truyền ID token ở đuôi truy vấn auth=<ID_TOKEN>:

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

Sử dụng API

- GET - Đọc dữ liệu
  - curl 'https://[PROJECT_ID].firebaseio.com/[PATH].json’
- PUT- Ghi dữ liệu
  -curl -X PUT -d '{ JSON object }' \
  'https://[PROJECT_ID].firebaseio.com/[PATH].json'
- POST - Đẩy dữ liệu
  - curl -X POST -d '{ JSON object }' \
  'https://[PROJECT_ID].firebaseio.com/[PATH].json'
- PATCH - Cập nhật dữ liệu
  -curl -X PATCH -d '{ JSON object }' \
  'https://[PROJECT_ID].firebaseio.com/[PATH].json'
- DELETE - Xóa dữ liệu
  -curl -X DELETE \

'https://[PROJECT_ID].firebaseio.com/[PATH].json'

III. Security and Rules

Giới thiệu chung về Rule

Firebase Realtime Database Rules định nghĩa xem ai có thể đọc hoặc ghi dữ liệu, dữ liệu được tổ chức như thế nào và đánh chỉ mục ra sao. Những quy tắc này được lưu trên Firebase server và được thực thi một cách tự động. Mọi yêu cầu đọc và ghi chỉ được thực hiện nếu thoả mãn điều kiện. Mặc định, với user được xác thực sẽ có quyền đọc và ghi trên cơ sở dữ liệu. Việc này có tác dụng bảo vệ cơ sở dữ liệu khỏi bị phá hoại đến khi mình có thời gian để tự thiết lập các quy tắc và xác thực.
Firebase Database Rules có 4 loại:
.read - Khi nào dữ liệu được đọc bởi người dùng
.write - Khi nào dữ liệu được phép ghi
.validate - Định nghĩa format của giá trị nằm ở dạng nào
.indexOn - Chỉ ra một child để index nhằm hỗ trợ cho việc sắp xếp và truy vấn dữ liệu.
Firebase Realtime Database cung cấp một bộ công cụ giúp cho việc quản lý về việc bảo mật của ứng dụng. Những công cụ này làm cho việc xác thực người dùng, thực thi quyền của người dùng và xác thực input một cách dễ dàng hơn.

Authentication

Bước đầu tiên để bảo mật ứng dụng đó là xác định người dùng. Quá trình này gọi là authentication. Lập trình viên có thể sử dụng Firebase Authentication để yêu cầu người dùng đăng nhập vào ứng dụng. Firebase Authentication bao gồm các phương thức đăng nhập như email-password, Google, Facebook hay đăng nhập ẩn danh và nhiều hơn thế nữa.
Định danh của người dùng là một vấn đề quan trọng trong việc bảo mật. Những người dùng khác nhau có những dữ liệu khác nhau và có những quyền truy cập khác nhau. Ví dụ, trong một ứng dụng chat, mỗi tin nhắn được người dùng tạo ra. Khi đó người dùng chỉ có thể xoá được tin nhắn của mình chứ không thể xoá được tin nhắn của người khác.

Authorization

Định danh người dùng là một phần của bảo mật dữ liệu. Khi bạn biết được người dùng họ là ai, bạn cần phải tìm cách để kiểm soát việc họ truy cập vào cơ sở dữ liệu. Firebase Database Rules cho phép bạn quản lý quyền truy cập của từng user. Ví dụ, đây là một bộ quy tắc cho phép mọi người đọc được địa chỉ /foo/ và các địa chỉ bên trong như /foo/bar/baz nhưng không ai có thể ghi vào nó.

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

Firebase Database Rules bao gồm cả những biến dựng sẵn và các hàm cho phép bạn gọi đến địa chỉ khác, server-sida timestamps và authentication information…
Ví dụ một quy tắc cho phép ghi với các người dùng được xác thực thông qua /users/<uid>/, với <uid> là ID của user được lấy từ Firebase Authentication.

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

Data validation

Quy tắc .validate giúp bạn kiểm tra logic trước khi dữ liệu được thêm vào database.
Ví dụ quy định dữ liệu được thêm vào là chuỗi ký tự có độ dài nhỏ hơn 100 ký tự.

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

Định nghĩa database index

Firebase Realtime Database cho phép sắp xếp và truy xuất dữ liệu. Với dữ liệu nhỏ, cơ sở dữ liệu việc đánh index không thực sự cần thiết trong lúc phát triển. Trước khi khởi chạy ứng dụng, công việc indexing rất quan trọng trong việc tăng trưởng của ứng dụng.
Ví dụ về index về height và length của một danh sách khủng long:

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

IV. Usage and Perfomance

Realtime Database Billing

Do Firebase chỉ cho phép miễn phí một lượng lưu trữ và download nhỏ nên chúng ta cần tìm hiểu về các khoản phí mà Google yêu cầu trả để có thể phát triển ứng dụng của chúng ta.

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. Storage thu phí $5 mỗi GB/month, được tính hằng ngày. 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 lấy dữ liệu từ cơ sở dữ liệu, Firebase tính phí cho việc tải về dữ liệu.
- Protocol Overhed: 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 Realtime Database và ước tính số tiền phải trả, người dùng phải 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. Firebase hiển thị số liệu sử dụng theo các số liệu sau:

- Connections: Số lượng kết nối đang mở và kết nối vào cơ sở dữ liệu, bao gồm WebSocket, server-sent events. Không bao gồm RESTful request.
- Storage: Lượng dữ liệu được lưu trữ trong cơ sở dữ liệu. Không bao gồm Firebase hosting và dữ liệu lưu trữ trong các project Firebase khác.
- Downloads: Toàn bộ dung lượng download từ cơ sở dữ liệu, bao gồm là protocol và encryption overhead.
- Load: Biểu đồ này thể hiện cơ sở dữ liệu được sử dụng như thế nào, thực hiện yêu cầu 1 phút 1 lần. Dùng để theo dõi các vấn đề về hiệu năng.

Các cách tối ưu lượng 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.
- Giảm thiểu số lượng kết nối

- Sử dụng TLS session ticket
- Indexing: Việc indexing sẽ giảm được lưu lượng sử dụng để query, có 2 lợi ích trong việc giảm thiểu lượng truy cập và tăng hiệu năng của cơ sở dữ liệu.
- Tối ưu listener
- Giảm lượng lưu trữ trong storage: dọn dẹp cơ sở dữ liệu với những dữ liệu dư thừa hoặc trùng lặp.
- Sử dụng Rules.
- Kiểm tra các lỗi trong ứng dụng.

Realtime Database Limits

Hoạt động	**Giới hạn**	**Mô tả**
Kết nối đồng thời	100,000	Một kết nối được tính bằng mỗi thiết bị di động, tab của trình duyệt và server app đang kết nối với cơ sở dữ liệu.
Phản hồi đồng thời được gửi từ database	~100,000/ second	Phản hồi bao gồm broadcast và các phép đọc được gửi từ server từ một cơ sở dữ liệu trong một khoảng thời gian. Giới hạn này ứng với các gói dữ liệu được gửi đi bồm đọc, broadcast, push notification được gửi từ database.
Số lượng cloud function được trigger trong một lệnh ghi dữ liệu.	1000	Số lượng phép đọc và ghi không bị giới hạn trong một function, tuy nhiên trong một phép ghi dữ liệu và database chỉ cho phép trigger tối đa 1000 function.
Độ lớn của một event trigger bởi một phép ghi dữ liệu.	1 MB	Độ lớn của 1 event bao gồm các giá trị sau: 1. Dữ liệu đã tồn tại 2. Giá trị cập nhật
Dữ liệu truyền tới Cloud Functions	10MB/sec	Lượng dữ liệu có thể truyền tới Cloud Functions

Data tree

Tính chất	**Giới hạn**	**Mô tả**

Độ sâu tối đa của cây	32	Mỗi nhánh của cây dữ liệu phải nhỏ hơn 32 level deep.
Độ dài của 1 key	768 bytes	Key được encode dưới dạng UTF-8 và không bao gồm xuống dòng và các kí tự sau .$#[] hoặc ASCII control characters.
Độ lớn tối đa của string	10 MB	Dữ liệu dưới dạng UTF-8

Read

Mô tả	**Giới hạn**	**Ghi chú**
Kích cỡ của 1 response từ database	256MB	Kích cỡ của dữ liệu tải về từ cơ sở dữ liệu phải nhỏ hơn 256MB trong mỗi phép đọc.
Tổng số node trong 1 địa chỉ.	75 triệu	
Thời gian của một query	15 phút*	Khi query chạy quá 15 phút sẽ bị thất bại * Một query chạy trên Firebase console chỉ có thể chạy tới 5 giây trước khi thất bại

Write

Mô tả	**Giới hạn**	**Ghi chú**
Độ lớn của 1 yêu cầu ghi	256MB từ REST API và 16MB từ SDKs	
Số byte được ghi	64MB/minute	

Tối ưu hoá hiệu năng của cơ sở dữ liệu

- Tối ưu đường truyền

RESTful requests bao gồm GET và PUT yêu cầu kết nối Internet, mặc dù chỉ là một kết nối rất ngắn. Tuy nhiên, khi có nhiều kết nối nhỏ được tạo thành, sẽ tiêu tốn một lượng đường truyền không hề nhỏ, truy xuất cơ sở dữ liệu, băng thông gửi đi trong thời gian thực và các kết nối hiện tại với cơ sở dữ liệu.

Khi có thể, sử dụng native SDKs cho ứng dụng thay vì REST API. SDKs sẽ duy trì kết nối luôn mở, giảm thiểu được SSL encryption và tải cơ sở dữ liệu mà sẽ tiêu tốn khi sử dụng REST API.

- Phân chia dữ liệu trên nhiều cơ sở dữ liệu

Chia dữ liệu trên nhiều cơ sở dữ liệu khác nhau của Realtime Databases có các lợi ích sau:

- Tăng số lượng kết nối
- Cân bằng tải giữa các cơ sở dữ liệu
- Nếu có một nhóm các người dùng chỉ cần truy cập vào một nhóm các bản ghi, sử dụng các cơ sở dữ liệu khác nhau cho thông lượng cao hơn và độ trễ thấp hơn.

* Tính năng này chỉ có thể sử dụng vời người dùng sử dụng Blaze plan của Firebase

- Xây dựng một cấu trúc dữ liệu hiệu quả
- Chặn các kết nối không được xác thực
- Sử dụng query-based rule để giới hạn tải về
- Index queries
- Sử dụng lại SSL session
- Tối ưu listener
- Dọn dẹp dữ liệu không sử dụng

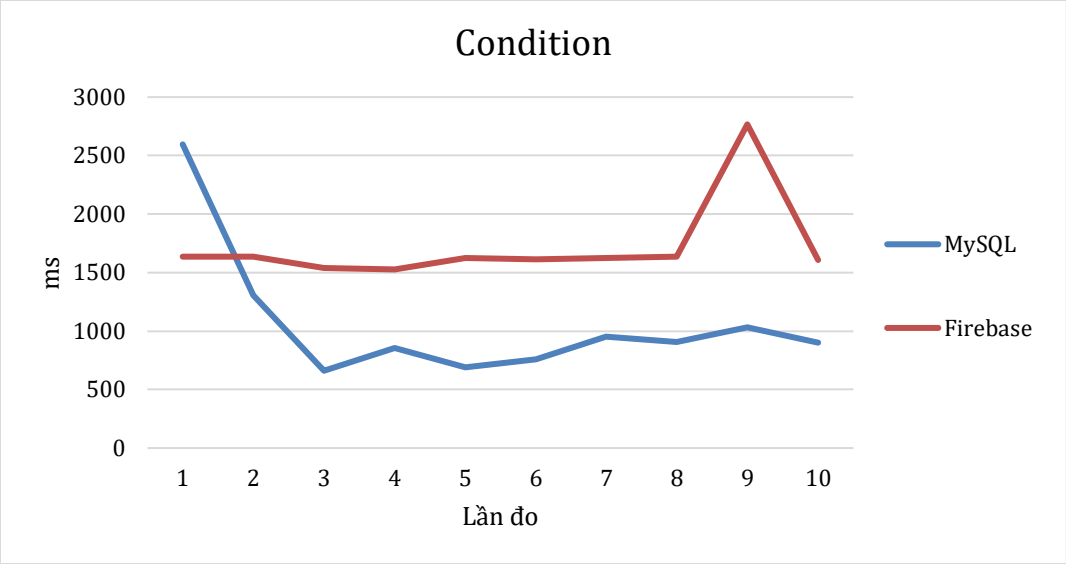
Phần 3: So sánh hiệu năng giữa Firebase và MySQL

Các yếu tố dẫn đến sai số:

- 2 DBMS không chạy cùng trên một cấu hình máy
- 2 DBMS không deploy trên cùng một server
- Tiến hành đo qua REST API nên không thể tránh được độ trễ của mạng.
- Google không khuyến cáo sử dụng Firebase thông qua REST API thay vì Native SDKs vì sẽ tốn một lượng lưu lượng và thời gian cho việc SSL encryption.
- Số liệu trình bày sau đây chỉ mang tính chất tham khảo

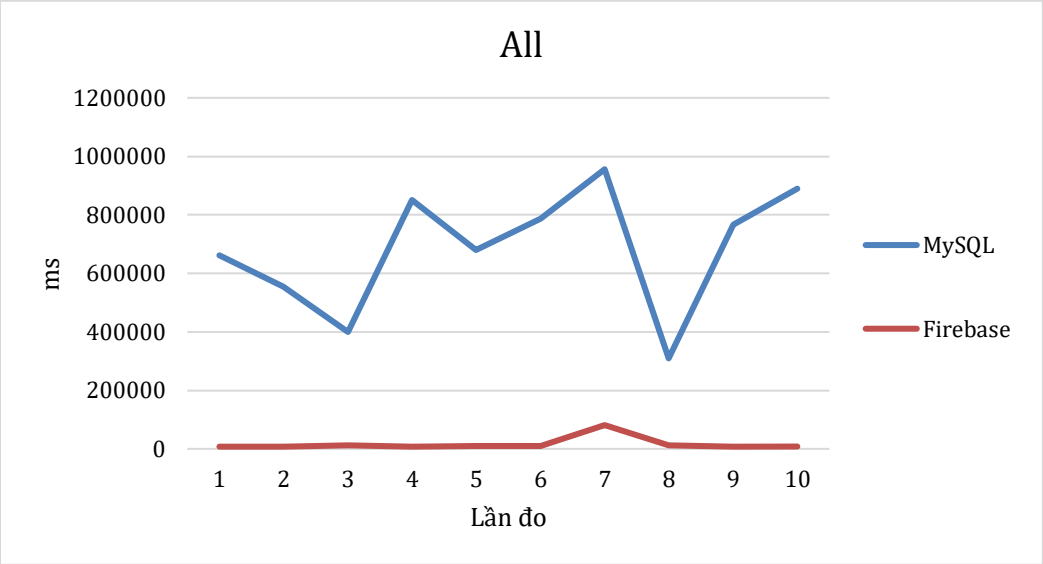
Cơ sở dữ liệu tiến hành test: Lưu thông tin của tên nhân viên và tên công ty của họ.

Truy vấn 1: Truy vấn có điều kiện



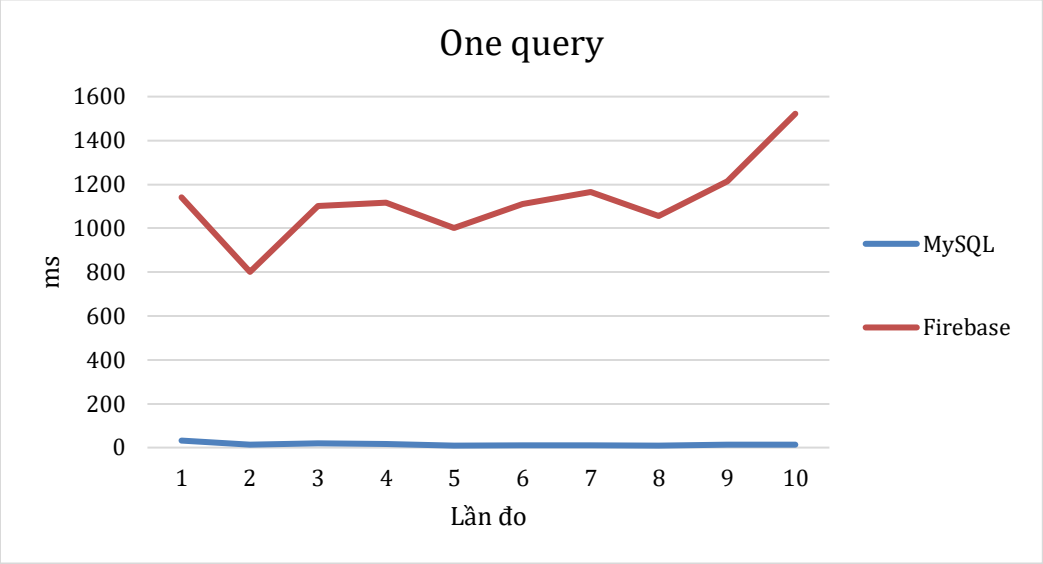
Trung bình: MySQL: 1065.2ms < Firebase: 1720.7 ms

Truy vấn 2: Truy vấn toàn bộ bản ghi



Trung bình: MySQL: 685980.9 ms >> Firebase: 16603.6

Truy vấn 3: Truy vấn 1 bản ghi



Trung bình: MySQL: 14.6 ms < Firebase: 1122.7 ms

Phần 4: Tổng kết

Firebase Realtime Database là một hệ quản trị cơ sở dữ liệu mới, giúp cho việc phát triển ứng dụng nhanh chóng ở phía client-side, tuy nhiên chúng ta sẽ tốn một khoản phí không nhỏ nếu ứng dụng mà mở rộng lớn. Tuy nhiên ở khía cạnh nghiên cứu, do Firebase cung cấp dưới dạng Service nên chúng ta khó có thể tìm hiểu được rõ cơ chế hoạt động cũng như làm thế nào để tối ưu được hiệu năng. Nói chung đối với phát triển ứng dụng nhỏ dạng di đông, yêu cầu đồng bộ thời gian thực Firebase cho chúng ta một trải nghiệm tuyệt vời mà không cần phải biết lập trình về Back-end. Kết hợp với các service khác của Google Cloud, Firebase cung cấp cho nhà phát triển mọi công cụ cần thiết để lập trình và theo dõi ứng dụng và thu thập thông tin tương tác của người dùng.

Phần 5: Tài liệu tham khảo

- Firebase Offical Documents