**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA CÔNG NGHỆ PHẦN MỀM**





**BÁO CÁO ĐỒ ÁN**

**MÔN SEMINAR VÀ CÁC VẤN ĐỀ HIỆN ĐẠI**

**Đề tài: Xây dựng ứng dụng quản lý công việc tích hợp**

**mô hình Serverless Service**

GVHD: Đinh Nguyễn Anh Dũng

Nhóm sinh viên thực hiện:

1. Trần Thanh Hiền MSSV: 21520230

2. Nguyễn Phúc Bình MSSV: 21520638

🙡🙢 Tp. Hồ Chí Minh, 01/2025 🙠🙣

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

*……., ngày……...tháng……năm 20…*

**Người nhận xét**

*(Ký tên và ghi rõ họ tên****)***

**BẢNG PHÂN CÔNG, ĐÁNH GIÁ THÀNH VIÊN:**

| Họ và tên | MSSV | Phân công | Đánh giá |
| --- | --- | --- | --- |
| Trần Thanh Hiền | 21520230 |  |  |
| Nguyễn Phúc Bình | 21520638 |  |  |

**LỜI MỞ ĐẦU**

**MỤC LỤC**

[**Chương 1: Tổng quan 6**](#_heading=h.3znysh7)

[1.1 Giới thiệu 7](#_heading=h.3dy6vkm)

[1.2 Mục tiêu nghiên cứu 8](#_heading=h.1t3h5sf)

[1.2.1. Mục tiêu tổng quát 8](#_heading=h.hf1lqanctxez)

[1.2.2. Mục tiêu cụ thể 8](#_heading=h.4xsqo2hthknc)

[1.3 Tổng quan về Serverless 9](#_heading=h.4d34og8)

[1.3.1. Định nghĩa và đặc điểm 9](#_heading=h.zfapipsqxfks)

[1.3.2. Lợi ích và thách thức 10](#_heading=h.ibgynie0aame)

[1.3.3. Xu hướng phát triển 11](#_heading=h.35ob8ptrvdfc)

[1.4 Các nền tảng Serverless phổ biến 11](#_heading=h.2s8eyo1)

[1.4.1. AWS Lambda 11](#_heading=h.u7rv7llbdfq4)

[1.4.2. Google Cloud Functions 12](#_heading=h.kv5wla5f6t5a)

[1.4.3. Azure Functions 13](#_heading=h.90lcaruinxgx)

[1.4.4. Firebase Cloud Functions 13](#_heading=h.ei0k60ozgvnq)

[1.5 Phân tích lựa chọn Firebase 14](#_heading=h.17dp8vu)

[1.5.1. Ưu điểm của Firebase 14](#_heading=h.giadrfbygzix)

[1.5.2. Các tính năng nổi bật 15](#_heading=h.jsf9icmgrt6h)

[1.6 Kiến trúc Firebase 15](#_heading=h.3rdcrjn)

[1.6.1. Authentication 15](#_heading=h.9zjv14p8e8eq)

[1.6.2. Realtime Database 16](#_heading=h.t4qnfqsi012w)

[1.6.3. Cloud Functions 16](#_heading=h.xj2ckgfpjuol)

[1.6.4. Cloud Storage 17](#_heading=h.90cfk4vmjui0)

[1.6.5. Hosting 17](#_heading=h.w8liymib2xb8)

[1.7 Chính sách chi phí sử dụng của Firebase 18](#_heading=h.26in1rg)

[1.8 Chức năng ứng dụng demo 21](#_heading=h.lnxbz9)

[**Chương 2: Cơ sở lý thuyết và nguyên lý hoạt động của Firebase serverless 22**](#_heading=h.q8p53ucdx25n)

[2.1 Kiến trúc serverless 22](#_heading=h.4eew3ly1nx93)

[2.2 So sánh giữa kiến trúc serverless với một số kiến trúc phổ biển khác. 23](#_heading=h.wh64seebo8kf)

[2.3 Cách thức deploy của firebase function bên trong hệ thống serverless của firebase 25](#_heading=h.omcctf9x9oq)

[2.4 Quy trình deploy và phát triển của Firebase function trên hệ thống serverless 27](#_heading=h.cfymyzceuvf5)

[2.5 Quản lý và phát triển hệ thống firebase function. 29](#_heading=h.oak76h764i4j)

[2.6 Xử lí lỗi khi có function có lỗi 31](#_heading=h.r5w2zmdn4qtc)

[2.7 Cơ chế hoạt dộng của Instance bên trong firebase 33](#_heading=h.90bbp35omlz6)

[2.8 Cold start và cách xử lí bên trong serverless để tối ưu hệ thống 34](#_heading=h.s9zb3072x6jl)

[2.9 Cơ Chế Mở Rộng Của Firebase Functions Khi Quá Tải 35](#_heading=h.jgssh3nd66q7)

[2.10 CI/CD trong serverless. 35](#_heading=h.w9p5k9jn4z73)

[2.11 Bảo mật trong Firebase Functions 37](#_heading=h.obhy7hg40n2p)

[2.12 Cách mở rộng và ứng dụng những điểm mạnh yếu vào trong thực tiễn. 37](#_heading=h.mawjwmvzaumf)

[**Chương 3: Xây dựng ứng dụng demo 38**](#_heading=h.35nkun2)

[3.1 Cấu hình ứng dụng Firebase 38](#_heading=h.44sinio)

[3.2 Kết nối và khởi tạo các Firebase dưới dạng local 39](#_heading=h.4i7ojhp)

[3.3 Tạo tương tác giữa Client và Firebase 42](#_heading=h.2xcytpi)

[3.4 Giao diện Người dùng 45](#_heading=h.1ci93xb)

[3.5 Đánh giá việc triển khai ứng dụng web trên việc sử dụng dịch vụ Firebase Service 50](#_heading=h.3whwml4)

[**Chương 4: Kết luận 50**](#_heading=h.2bn6wsx)

[4.1 Kết quả đạt được 50](#_heading=h.qsh70q)

[4.2 Bài học kinh nghiệm 51](#_heading=h.3as4poj)

[4.3 Hướng phát triển 51](#_heading=h.1pxezwc)

# Tổng quan

## Giới thiệu

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, các phương pháp xây dựng và triển khai phần mềm cũng không ngừng được đổi mới để đáp ứng nhu cầu ngày càng cao về tốc độ, hiệu quả và khả năng mở rộng. Mô hình Serverless đã nổi lên như một xu hướng quan trọng trong ngành công nghệ thông tin, mang lại những lợi ích vượt trội cho cả nhà phát triển và doanh nghiệp.

Với mô hình này, các nhà phát triển không còn phải quản lý cơ sở hạ tầng phức tạp như trước đây. Thay vào đó, họ chỉ cần tập trung vào việc xây dựng và phát triển các tính năng ứng dụng, trong khi hạ tầng được tự động quản lý và mở rộng bởi các nhà cung cấp dịch vụ đám mây. Điều này không chỉ giảm gánh nặng vận hành mà còn tối ưu hóa chi phí và tăng năng suất phát triển phần mềm.

Trong xu thế áp dụng Serverless, Firebase là một trong những dịch vụ tiêu biểu, được thiết kế để hỗ trợ toàn diện cho việc xây dựng các ứng dụng web và di động. Firebase cung cấp nhiều công cụ và dịch vụ, từ cơ sở dữ liệu thời gian thực, xác thực người dùng, lưu trữ file, đến các chức năng như hosting và thông báo đẩy, giúp lập trình viên dễ dàng triển khai các ứng dụng hiện đại.

Việc nghiên cứu mô hình Serverless thông qua Firebase mang ý nghĩa quan trọng, đặc biệt trong bối cảnh nhu cầu xây dựng các ứng dụng hiện đại, hiệu quả và dễ bảo trì ngày càng tăng. Thông qua đề tài nghiên cứu này, nhóm chúng em sẽ không chỉ tìm hiểu lý thuyết về mô hình Serverless mà còn áp dụng thực tế bằng cách xây dựng một ứng dụng quản lý công việc dựa trên các dịch vụ của Firebase. Ứng dụng này sẽ giúp giải quyết bài toán về quản lý công việc, bao gồm tạo nhiệm vụ, phân công công việc, theo dõi tiến độ và thông báo tự động.

Quá trình nghiên cứu và thực hiện đề tài này không chỉ giúp nhóm hiểu rõ hơn về lợi ích của mô hình Serverless mà còn giúp nhận diện các thách thức, như chi phí sử dụng dịch vụ đám mây, quản lý tài nguyên hoặc tích hợp với các công nghệ khác. Bên cạnh đó, nhóm cũng có cơ hội nâng cao năng lực sử dụng các công cụ hiện đại như Firebase, mở rộng kiến thức về phát triển ứng dụng theo xu hướng công nghệ mới, từ đó chuẩn bị tốt hơn cho các dự án lớn hơn trong tương lai.

## Mục tiêu nghiên cứu

### **1.2.1. Mục tiêu tổng quát**

Mục tiêu tổng quát của đề tài là thiết kế và xây dựng một ứng dụng quản lý công việc dựa trên mô hình Serverless, sử dụng Firebase làm nền tảng phát triển chính. Ứng dụng này hướng tới việc giải quyết các vấn đề thường gặp trong quản lý công việc nhóm, chẳng hạn như khó theo dõi tiến độ công việc, thiếu sự minh bạch trong phân công nhiệm vụ, hoặc khó khăn trong giao tiếp và phối hợp giữa các thành viên. Tìm hiểu cách ứng dụng dịch vụ Firebase trong việc xây dựng ứng dụng serverless cho ứng dụng web.

Đề tài không chỉ tập trung vào việc tạo ra một sản phẩm hữu ích phục vụ cho nhu cầu thực tiễn mà còn giúp nhóm em nắm vững các khía cạnh kỹ thuật của mô hình Serverless. Thông qua việc ứng dụng Firebase – một nền tảng Serverless tiên tiến – nhóm có thể hiểu rõ cách các dịch vụ đám mây hoạt động và hỗ trợ trong quá trình phát triển phần mềm hiện đại. Quan trọng hơn, việc thực hiện đề tài còn trang bị cho nhóm kiến thức thực tiễn và kỹ năng lập trình cần thiết để đáp ứng những yêu cầu khắt khe trong ngành công nghệ thông tin hiện nay.

### **1.2.2. Mục tiêu cụ thể**

#### 1.2.2.1. Nghiên cứu về mô hình Serverless và dịch vụ Firebase

* Tìm hiểu các khái niệm cơ bản của mô hình Serverless, phân tích ưu điểm và hạn chế so với các mô hình truyền thống.
* Nghiên cứu chi tiết về các dịch vụ của Firebase, bao gồm Firestore, Authentication, Cloud Functions, Hosting, và Cloud Storage. Đây là các thành phần quan trọng để xây dựng một ứng dụng toàn diện và hiệu quả.
* Đánh giá tiềm năng của mô hình Serverless trong việc xây dựng ứng dụng quản lý công việc, đặc biệt trong bối cảnh các nhóm làm việc nhỏ cần các giải pháp nhanh chóng và tối ưu chi phí.

#### 1.2.2.2. Xây dựng ứng dụng demo

* Thiết kế giao diện và chức năng cho ứng dụng quản lý công việc nhóm, đảm bảo thân thiện với người dùng và dễ sử dụng.
* Phát triển các tính năng chính như: tạo nhiệm vụ, phân công công việc, theo dõi tiến độ, quản lý thành viên, và thông báo tự động.
* Tích hợp các dịch vụ của Firebase để đảm bảo ứng dụng có khả năng tự động mở rộng, xử lý thời gian thực và bảo mật thông tin người dùng.

#### 1.2.2.3. Đánh giá hiệu quả của mô hình Serverless

* So sánh hiệu quả triển khai ứng dụng bằng mô hình Serverless với các mô hình truyền thống, dựa trên các tiêu chí như tốc độ phát triển, chi phí vận hành, khả năng mở rộng và bảo trì.
* Phân tích các thách thức gặp phải trong quá trình phát triển, từ đó rút ra bài học kinh nghiệm cho việc triển khai các dự án tương tự trong tương lai.

## Tổng quan về Serverless

### **1.3.1. Định nghĩa và đặc điểm**

#### 1.3.1.1. Định nghĩa

Mô hình Serverless, hay còn gọi là "Không máy chủ", là một cách tiếp cận hiện đại trong phát triển phần mềm, nơi mà nhà phát triển không cần quản lý máy chủ vật lý hoặc ảo. Thay vào đó, các nhà cung cấp dịch vụ đám mây như Firebase, AWS Lambda, hoặc Azure Functions sẽ chịu trách nhiệm vận hành, mở rộng và duy trì cơ sở hạ tầng. Lập trình viên chỉ cần tập trung vào việc viết mã và triển khai chức năng.

#### 1.3.1.2. Đặc điểm

* Không cần quản lý máy chủ: Serverless cho phép nhà phát triển loại bỏ hoàn toàn việc cấu hình và bảo trì máy chủ, giảm thiểu thời gian dành cho các công việc vận hành.
* Tự động mở rộng: Hệ thống có khả năng tự động mở rộng dựa trên nhu cầu thực tế, giúp đảm bảo hiệu suất tốt ngay cả khi lượng truy cập tăng đột biến.
* Tính chi phí theo mức sử dụng: Người dùng chỉ phải trả tiền cho thời gian chạy thực tế của ứng dụng, thay vì phải duy trì một hệ thống lớn liên tục.
* Tích hợp sâu: Các dịch vụ như Firebase cung cấp một hệ sinh thái đồng bộ, giúp kết nối các thành phần như cơ sở dữ liệu, xác thực, và lưu trữ một cách liền mạch.

### **1.3.2. Lợi ích và thách thức**

#### 1.3.2.1. Lợi ích

* Tối ưu hóa nguồn lực: Không cần đội ngũ kỹ thuật chuyên trách để duy trì máy chủ, giúp tập trung nguồn lực vào phát triển tính năng.
* Nhanh chóng triển khai: Mô hình này giúp giảm thời gian đưa sản phẩm ra thị trường nhờ khả năng tích hợp sẵn các dịch vụ.
* Hiệu suất cao: Với tính năng tự động mở rộng, ứng dụng có thể đáp ứng tốt ngay cả trong các tình huống có lưu lượng truy cập lớn.
* Giảm chi phí vận hành: Chỉ trả phí cho thời gian thực thi mã, không tốn kém chi phí cố định như mô hình truyền thống.

#### 1.3.2.2. Thách thức

* Phụ thuộc vào nhà cung cấp dịch vụ: Khi sử dụng Serverless, ứng dụng có thể bị ràng buộc với một nhà cung cấp duy nhất, khó chuyển đổi hoặc mở rộng sang nền tảng khác.
* Giới hạn kiểm soát: Nhà phát triển không có quyền kiểm soát trực tiếp đối với cơ sở hạ tầng, có thể dẫn đến khó khăn khi khắc phục sự cố hoặc tối ưu hiệu suất.
* Độ trễ khởi động (Cold Start): Một số dịch vụ Serverless gặp phải độ trễ khi khởi động ứng dụng từ trạng thái không hoạt động, ảnh hưởng đến trải nghiệm người dùng.

#### 1.3.2.3. So sánh với kiến trúc truyền thống

| **Tiêu chí** | **Serverless** | **Kiến trúc truyền thống** |
| --- | --- | --- |
| Quản lý hạ tầng | Nhà cung cấp | Tự vận hành |
| Tính mở rộng | Tự động | Thủ công hoặc qua công cụ |
| Chi phí | Theo mức sử dụng | Chi phí cố định |
| Kiểm soát | Giới hạn | Hoàn toàn |

### **1.3.3. Xu hướng phát triển**

Xu hướng Serverless đang tập trung vào việc tích hợp AI/ML để tự động hóa và tối ưu quy trình, đồng thời mở rộng khả năng hoạt động trên nhiều nền tảng như IoT và blockchain. Các nhà cung cấp như Firebase liên tục cải thiện trải nghiệm phát triển thông qua công cụ trực quan và tự động hóa deployment. Vấn đề bảo mật và tuân thủ chuẩn quốc tế như GDPR cũng được chú trọng, đặc biệt trong các ngành nhạy cảm như tài chính và y tế.

## Các nền tảng Serverless phổ biến

Serverless đã trở thành một trong những xu hướng công nghệ quan trọng trong ngành công nghiệp điện toán đám mây, được các công ty hàng đầu thế giới như Amazon, Microsoft, và Google đẩy mạnh phát triển. Các dịch vụ serverless hiện đại được cung cấp bởi những "ông lớn" này không chỉ hỗ trợ việc triển khai ứng dụng nhanh chóng mà còn giúp giảm đáng kể chi phí và công sức quản lý hạ tầng.

### **1.4.1. AWS Lambda**

AWS Lambda là dịch vụ serverless do Amazon Web Services (AWS) cung cấp, cho phép người dùng chạy mã mà không cần phải quản lý các máy chủ. Lambda hỗ trợ nhiều ngôn ngữ lập trình, bao gồm Node.js, Python, Java, và Go, giúp các nhà phát triển dễ dàng tích hợp với các dịch vụ AWS khác như Amazon S3, DynamoDB và API Gateway.

Mô hình tính phí của Lambda dựa trên số lượng yêu cầu và thời gian thực thi của mã, cho phép doanh nghiệp tiết kiệm chi phí, chỉ phải trả tiền cho tài nguyên sử dụng thực tế. AWS Lambda cung cấp khả năng mở rộng tự động, giúp xử lý lượng lớn yêu cầu đồng thời mà không cần lo lắng về vấn đề tài nguyên. Tuy nhiên, điểm yếu của Lambda là độ trễ khởi động (cold start) khi phải xử lý các yêu cầu đầu tiên sau một khoảng thời gian không sử dụng.



### **1.4.2. Google Cloud Functions**

Google Cloud Functions là dịch vụ serverless do Google Cloud cung cấp, giúp lập trình viên dễ dàng xây dựng và triển khai các chức năng không máy chủ. Cloud Functions hỗ trợ các ngôn ngữ như JavaScript, Python, và Go, giúp ứng dụng dễ dàng kết nối với các dịch vụ khác của Google Cloud, như Google Cloud Storage và Google Cloud Pub/Sub.

Dịch vụ này lý tưởng cho các tác vụ sự kiện, từ xử lý HTTP requests cho đến việc quản lý các sự kiện từ các dịch vụ đám mây khác. Google Cloud Functions cũng cung cấp khả năng mở rộng tự động, giúp xử lý các yêu cầu theo thời gian thực mà không cần phải lo lắng về hạ tầng. Tuy nhiên, một điểm cần lưu ý là chi phí sử dụng Cloud Functions có thể tăng nhanh nếu ứng dụng có lưu lượng truy cập lớn và yêu cầu tính toán nhiều.



### **1.4.3. Azure Functions**

Azure Functions là dịch vụ serverless của Microsoft Azure, cho phép lập trình viên viết mã mà không phải lo lắng về việc quản lý máy chủ. Azure Functions hỗ trợ nhiều ngôn ngữ lập trình như C#, Java, Python, và JavaScript, và có thể tích hợp với các dịch vụ khác của Azure như Azure Storage, Azure Event Grid và Azure Service Bus. Điểm đặc biệt của Azure Functions là khả năng tùy chỉnh quy mô theo nhu cầu, từ việc xử lý các tác vụ đơn giản đến các ứng dụng phức tạp.

Mô hình tính phí của Azure Functions cũng dựa trên số lượng yêu cầu và thời gian chạy, giúp tối ưu hóa chi phí. Tuy nhiên, giống như các dịch vụ serverless khác, Azure Functions cũng gặp phải vấn đề về độ trễ khởi động và có thể không phù hợp với những ứng dụng yêu cầu độ phản hồi cực nhanh.



### **1.4.4. Firebase Cloud Functions**

Firebase Cloud Functions là một dịch vụ serverless mạnh mẽ được tích hợp trong Firebase, cho phép lập trình viên thực thi mã backend mà không cần quản lý cơ sở hạ tầng. Firebase Cloud Functions hỗ trợ các ngôn ngữ lập trình như JavaScript và TypeScript, và đặc biệt phù hợp với các ứng dụng web và di động sử dụng Firebase. Các chức năng này có thể dễ dàng tích hợp với các dịch vụ khác của Firebase như Firebase Realtime Database, Firebase Authentication, và Firebase Cloud Messaging.

Firebase Cloud Functions cung cấp khả năng tự động mở rộng và chi phí tính theo mức sử dụng thực tế, giúp các dự án tiết kiệm chi phí và tối ưu hóa hiệu suất. Một điểm mạnh của Firebase Cloud Functions là việc triển khai dễ dàng và nhanh chóng, đặc biệt là khi làm việc trong hệ sinh thái Firebase. Tuy nhiên, Firebase cũng có những hạn chế về độ trễ và tính linh hoạt khi so với các dịch vụ khác như AWS Lambda và Google Cloud Functions.

### 

## Phân tích lựa chọn Firebase

### **1.5.1. Ưu điểm của Firebase**

Firebase nổi bật với nhiều ưu điểm, khiến nó trở thành một nền tảng phổ biến cho việc phát triển ứng dụng di động và web. Một trong những điểm mạnh lớn nhất của Firebase là tính dễ dàng sử dụng và khả năng tích hợp sâu với các dịch vụ khác. Firebase giúp các nhà phát triển không phải lo lắng về việc quản lý cơ sở hạ tầng, vì tất cả các dịch vụ đều được quản lý tự động trên nền tảng đám mây. Điều này giúp giảm thiểu công sức và chi phí cho việc duy trì và nâng cấp hệ thống.

Bên cạnh đó, Firebase cung cấp khả năng mở rộng tự động, giúp ứng dụng của bạn có thể đáp ứng tốt với lượng người dùng ngày càng tăng mà không cần thay đổi cấu trúc hay can thiệp vào mã nguồn. Firebase cũng nổi bật ở việc tích hợp các tính năng bảo mật và xác thực người dùng thông qua Firebase Authentication, một dịch vụ cho phép xác thực người dùng dễ dàng mà không cần phải lo lắng về các vấn đề bảo mật phức tạp. Tính năng Realtime Database của Firebase giúp việc đồng bộ hóa dữ liệu giữa các người dùng và thiết bị trở nên mượt mà, giúp ứng dụng của bạn có thể phản hồi tức thì với bất kỳ thay đổi nào trong cơ sở dữ liệu.

Một ưu điểm nữa của Firebase là hệ sinh thái công cụ mạnh mẽ, bao gồm Firebase Analytics, Firebase Cloud Messaging, Firebase Crashlytics, và nhiều dịch vụ khác, cho phép các nhà phát triển theo dõi, phân tích và tối ưu hóa ứng dụng ngay trong quá trình phát triển. Firebase cũng có khả năng tích hợp tốt với các dịch vụ bên ngoài và có thể sử dụng cho cả ứng dụng web và di động, làm cho nó trở thành một giải pháp linh hoạt và hiệu quả cho các dự án phát triển ứng dụng hiện đại.



### **1.5.2. Các tính năng nổi bật**

Firebase cung cấp một loạt các tính năng nổi bật, giúp các nhà phát triển xây dựng và duy trì ứng dụng một cách hiệu quả hơn. Một trong những tính năng quan trọng nhất là Realtime Database, cho phép đồng bộ dữ liệu giữa các thiết bị và người dùng trong thời gian thực. Điều này cực kỳ hữu ích cho các ứng dụng yêu cầu cập nhật tức thời, như các ứng dụng trò chuyện, mạng xã hội, hay các dịch vụ chia sẻ thời gian thực.

## Kiến trúc Firebase

### **1.6.1. Authentication**

Firebase Authentication cung cấp một giải pháp xác thực người dùng đơn giản và an toàn cho các ứng dụng web và di động. Với Firebase Authentication, nhà phát triển có thể dễ dàng triển khai nhiều phương thức đăng nhập khác nhau như email/mật khẩu, Google, Facebook, Twitter, hoặc xác thực qua số điện thoại mà không cần xây dựng các hệ thống phức tạp từ đầu. Firebase Authentication hỗ trợ các tính năng bảo mật mạnh mẽ như mã hóa mật khẩu và bảo vệ thông tin người dùng, giúp đảm bảo an toàn dữ liệu cho ứng dụng.

Hơn nữa, Firebase Authentication còn hỗ trợ tính năng xác thực liên tục, cho phép người dùng đăng nhập vào ứng dụng của bạn một cách liên tục mà không phải nhập lại mật khẩu. Điều này giúp nâng cao trải nghiệm người dùng và giảm thiểu sự phiền phức khi sử dụng ứng dụng. Một điểm đặc biệt là Firebase Authentication còn tích hợp trực tiếp với các dịch vụ Firebase khác như Firebase Realtime Database và Firestore, giúp đồng bộ hóa và quản lý dữ liệu người dùng một cách dễ dàng và hiệu quả.

### **1.6.2. Realtime Database**

Firebase Realtime Database là một dịch vụ cơ sở dữ liệu NoSQL cho phép lưu trữ và đồng bộ dữ liệu trong thời gian thực giữa các thiết bị và người dùng. Điều này có nghĩa là khi một thay đổi được thực hiện trên một thiết bị, các thiết bị khác sẽ nhận được thay đổi ngay lập tức mà không cần phải tải lại dữ liệu. Firebase Realtime Database sử dụng cấu trúc dữ liệu dạng cây JSON, dễ dàng quản lý và truy cập.

Các ứng dụng có thể sử dụng Firebase Realtime Database để quản lý các ứng dụng yêu cầu sự đồng bộ hóa liên tục, chẳng hạn như ứng dụng chat, ứng dụng mạng xã hội, hay các trò chơi nhiều người chơi. Dịch vụ này cũng hỗ trợ việc quản lý quyền truy cập và bảo mật, cho phép nhà phát triển định nghĩa các quy tắc bảo mật để kiểm soát ai có thể đọc và ghi dữ liệu. Realtime Database có thể mở rộng linh hoạt theo nhu cầu của ứng dụng, giúp ứng dụng của bạn có thể xử lý hàng triệu kết nối đồng thời mà không gặp phải vấn đề hiệu suất.

### **1.6.3. Cloud Functions**

Firebase Cloud Functions là dịch vụ serverless mạnh mẽ giúp nhà phát triển có thể chạy các đoạn mã JavaScript hoặc TypeScript mà không cần phải lo lắng về việc quản lý máy chủ. Cloud Functions có thể được kích hoạt khi có các sự kiện xảy ra trong các dịch vụ Firebase khác như Firebase Authentication, Realtime Database, Firestore hoặc Firebase Cloud Storage.

Một trong những điểm nổi bật của Cloud Functions là khả năng mở rộng tự động dựa trên lưu lượng và nhu cầu sử dụng. Nhà phát triển không cần phải lo lắng về việc điều chỉnh máy chủ hoặc cơ sở hạ tầng khi có sự thay đổi về lượng truy cập. Cloud Functions lý tưởng để xử lý các tác vụ phức tạp mà không cần triển khai một hệ thống máy chủ riêng biệt, như gửi thông báo đẩy, xử lý dữ liệu, hay chạy các tác vụ phụ trợ như sao lưu và đồng bộ dữ liệu.

Với Firebase Cloud Functions, nhà phát triển cũng có thể dễ dàng viết và triển khai các chức năng backend của ứng dụng mà không cần xây dựng một hệ thống backend phức tạp, giúp tiết kiệm thời gian và chi phí phát triển.

### **1.6.4. Cloud Storage**

Firebase Cloud Storage là dịch vụ lưu trữ đám mây giúp lưu trữ và quản lý các tệp dữ liệu lớn như hình ảnh, video, và tài liệu một cách dễ dàng và an toàn. Dịch vụ này được thiết kế đặc biệt để xử lý các tệp có kích thước lớn, với khả năng mở rộng linh hoạt và bảo mật cao. Firebase Cloud Storage cung cấp tính năng mã hóa tệp khi truyền tải và khi lưu trữ, bảo đảm rằng dữ liệu của người dùng luôn an toàn.

Một điểm mạnh của Firebase Cloud Storage là khả năng tích hợp trực tiếp với các dịch vụ Firebase khác, giúp đơn giản hóa quá trình quản lý dữ liệu người dùng. Ví dụ, tệp tin tải lên có thể được kết hợp với dữ liệu người dùng trong Firebase Realtime Database hoặc Firestore, tạo ra một hệ sinh thái hoàn chỉnh cho ứng dụng. Hơn nữa, Firebase Cloud Storage hỗ trợ việc quản lý quyền truy cập tệp, cho phép nhà phát triển xác định quyền đọc/ghi tệp cho từng người dùng hoặc nhóm người dùng cụ thể.

### **1.6.5. Hosting**

Firebase Hosting là một dịch vụ mạnh mẽ và dễ sử dụng cho việc triển khai các ứng dụng web. Dịch vụ này cho phép nhà phát triển triển khai các trang web tĩnh và động với khả năng tải nhanh và bảo mật cao. Firebase Hosting sử dụng các CDN (Content Delivery Network) để phân phối nội dung từ các máy chủ trên toàn cầu, giúp giảm độ trễ và đảm bảo rằng người dùng có thể tải trang nhanh chóng, bất kể vị trí địa lý.

Firebase Hosting cũng hỗ trợ việc triển khai các ứng dụng SPA (Single Page Application) như ứng dụng React, Angular hoặc Vue.js, với các tính năng như điều hướng động và xử lý các yêu cầu API. Dịch vụ này cung cấp chứng chỉ SSL miễn phí cho mọi ứng dụng, bảo vệ dữ liệu người dùng khi truyền tải. Nhà phát triển có thể dễ dàng triển khai các bản cập nhật ứng dụng qua Firebase CLI mà không cần phải lo lắng về cấu hình máy chủ hoặc quản lý cơ sở hạ tầng.

## Chính sách chi phí sử dụng của Firebase

* Với việc là nền tảng dịch vụ serverless nên đi kèm theo đó là chi phí sử dụng mà người dùng phải trả khi dùng các dịch vụ của Firebase. Tuy nhiên với Firebase, Google cung cấp hai loại plan mà người dùng có thể sử dụng tùy vô nhu cầu sử dụng của người dùng gôm: **Spark plan** (Không tốn phí) **và Blaze plan** (Tốn phí).
  + **Spark plan:** Người dùng sẽ không tốn phí dịch vụ khi sử dụng nhưng đồng thời việc truy cập và sử dụng sẽ chỉ nằm ở một giới hạn nhất định nào đó.
  + **Blaze plan:** Người dùng sẽ tốn phí dịch vụ dựa vô số lượng truy cập sử dụng của ứng dụng mà sẽ tính phí dựa trên đó.

A blue screen shot of a blue screen

Description automatically generated

A screenshot of a computer

Description automatically generated

*Firebase Authentication Pricing*

*A screenshot of a computer

Description automatically generated*

*Cloud Firestore Pricing*

*A screenshot of a computer

Description automatically generated*

*Firebase Cloud Function Pricing*

*A screenshot of a computer

Description automatically generated*

*Firebase Cloud Storage Pricing*

*A screenshot of a computer

Description automatically generated*

*Firebase App Hosting Pricing*

Thông qua việc tiếp cận bảng giá trên giúp nhóm có thể ước chừng và nhắm trong việc tính toán các chi phí khi xây dựng ứng dụng thông qua Firebase Service mà không mất quá nhiều chi phí không cần thiết.

## Chức năng ứng dụng demo

Ứng dụng demo là một hệ thống quản lý công việc, có các chức năng cơ bản như

* Đăng nhập / Đăng ký qua email: Cho phép người dùng đăng ký tài khoản bằng email và xác thực tài khoản trước khi đăng ký. Sử dụng Firebase authentication trong nhóm các service của firebase để thực hiện chức năng authentication cho ứng dụng.
* CRUD (Create, Read, Update, Delete) Boards: Cho phép người dùng có thể tạo các bảng quản lý công việc đồng thời có thể thay đổi, thêm xóa các thành viên hoạt động trong bảng và có thể xóa bảng nếu cần thiết
* CRUD (Create, Read, Update, Delete) và realtime update cho List: List là danh sách các công việc bên trong bảng và phân ra các quá trình khác nhau trong Board như list “Đang làm”, list “ Đã hoàn thành” và cho phép người dùng có thể thêm, xóa, sửa các danh sách đồng thời các thành viên bên trong board có thể nhìn thấy sự thay đổi đó một cách tức thời.
* CRUD (Create, Read, Update, Delete) và realtime update cho Task: Task là công việc và người dùng có thể thêm xóa sửa hoặc chuyển sang các list khác nhau phù thuộc vào trạng thái của của task đó và cho phép assign các task đó cho 1 thành viên trong bảng đó, đồng thời hiển thị cập nhật sự thay đổi đó một cách tức thời thời cho các thành viên.
* CRUD (Create, Read, Update, Delete) và realtime update cho Comment: Người dùng có thể comment hoặc góp ý các thông tin cho từng task đó.
* UI: Giao diện được tối ưu để phản hồi nhanh và thân thiện với người dùng.
* Xử lý lỗi: Hệ thống sẽ xử lý và ghi nhận các lỗi có thể xảy ra trong quá trình sử dụng.

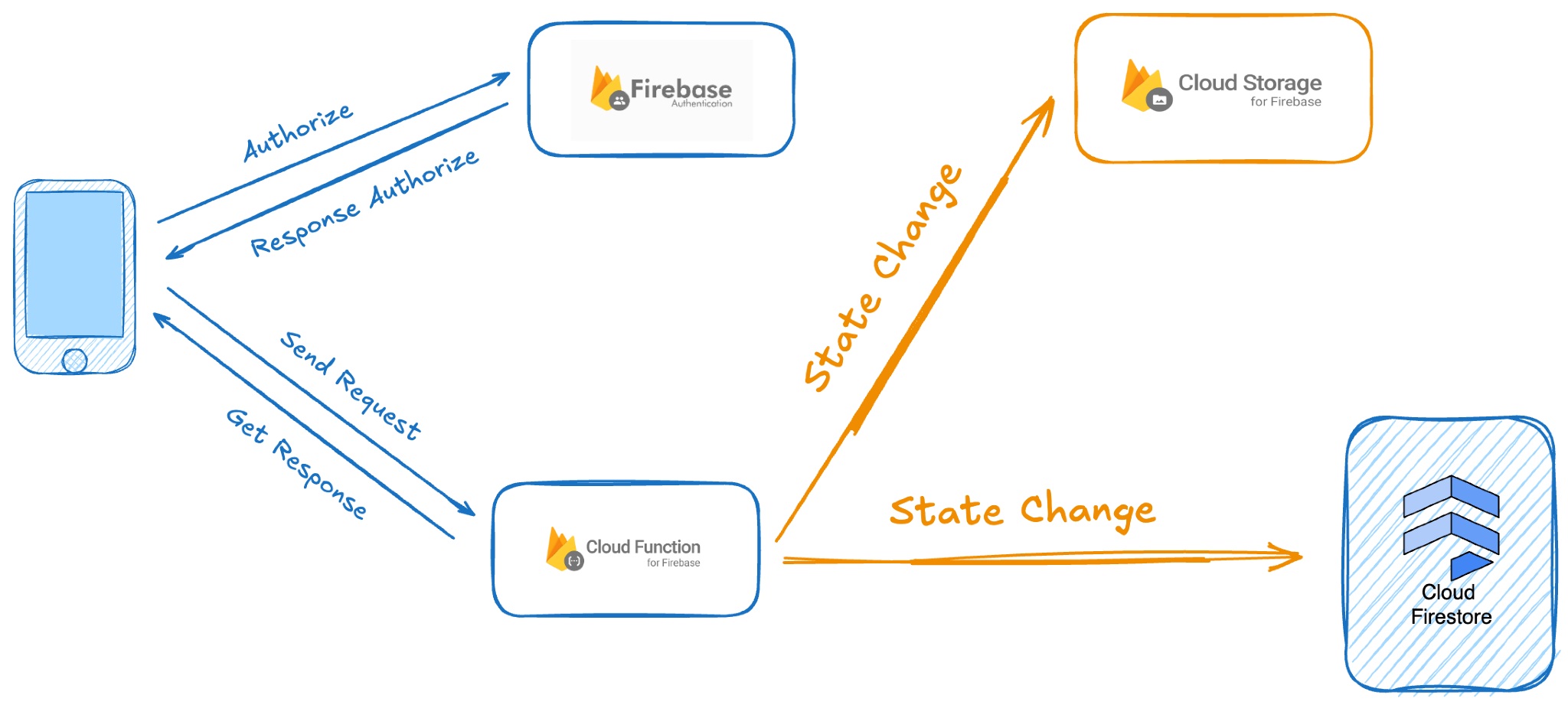
Ứng dụng các service serverless của Firebase trong việc xây dựng ứng dụng:

* Đối với việc đăng nhập và xác thực người dùng thì nhóm sẽ ứng dụng Firebase Authentication kết hợp với Firebase Function để tạo ra function xử lí đăng nhập phù hợp cho người dùng.
* Về tính năng realtime và cập nhật nhanh chóng các task trong ứng dụng cũng như quản lý nó thì nhóm sử dụng Firebase Firestore dù việc cập nhật có thể không nhanh bằng Firebase realtime nhưng với Firestore thì dễ dàng quản lý dữ liệu người dùng hơn.
* Việc lưu trữ dữ liệu thì nhóm cũng sử dụng Firestore để lưu trữ phù hợp cho ứng dụng này.
* Để có thể test, dev mà không bị ảnh hưởng nhiều chi phí thì nhóm sử dụng Firebase Emulators để có thể mô phỏng các dịch vụ của Firebase trên chính máy local nhằm gia tăng hiệu suất cũng như giảm chi phí người dùng khi không cần deploy các function mới lên firebase liên tục.

## Cơ sở lý thuyết và nguyên lý hoạt động của Firebase serverless

## Kiến trúc serverless

* Với *serverless* thì nhóm đã có nói đến một số vấn đề cơ bản của serverless ở trên nhưng ở đây nhóm sẽ nói rõ hơn về kiến trúc này. Để hiểu rõ hơn về kiến trúc này hơn thì ta hãy tìm hiểu rõ hơn về cụm từ serverless ở đây.
* Mặc dù tên gọi là *serverless*, điều này không có nghĩa là không có máy chủ để vận hành hệ thống. Thay vào đó, serverless đề cập đến việc các máy chủ đã được quản lý hoàn toàn bởi các nhà cung cấp dịch vụ đám mây hàng đầu như Google Cloud, AWS, và Azure. Các nhà cung cấp này xây dựng và duy trì cơ sở hạ tầng máy chủ, giúp người dùng chỉ cần tập trung vào việc triển khai ứng dụng hoặc chức năng mà không phải lo lắng về việc quản lý hoặc thiết lập hạ tầng.
* Điều này không chỉ tiết kiệm thời gian và công sức mà còn giúp giảm đáng kể chi phí vận hành và quản lý server cho người dùng.
* Thiết kế và ứng dụng kiến trúc serverless vô trong thực tiễn sẽ khá giống phần nào với kiến trúc micro service được sử dụng rộng rãi trên thế giới, tuy nhiên ở mặt nào đó thì việc thiết kế và sử dụng kiến trúc serverless sẽ dễ dạng hơn nhiều. Việc kiến trúc serverless cung cấp các thành phần **Baas** (Backend as service) với **Faas** (Function as service) đã hỗ trợ việc xây dưng nhũng kiến trúc này một các dễ dàng hơn.
* Với kiến trúc serverless thì khi xây dựng ta sẽ dựa trên các function nhỏ và được gọi khi được kích hoạt event thông qua việc call của firebase, trigger hoặc Https Request nên khi xây dựng và thiết kế kiến trúc serverless ta cũng phải để ý đến các các biến state tồn tại bên trong các function và hoạt động của nó.



Quy trình thực hiện của kiến trúc Serverless

## So sánh giữa kiến trúc serverless với một số kiến trúc phổ biến khác.

Kiến trúc serverless mới được trở nên khá thịnh hành hơn vài năm gần đây nhờ sự phát triển của cloud computing của các công ty cloud lớn nên serverless trở nên thịnh hành hơn nhưng vẫn có những sự so sánh với các kiến trúc truyền thống khác đã được xây dựng và triển khai lâu đời:

1. **So sánh với Micro-service**

Tổng quan cơ bản của Microservice:

* **Microservices** là một kiến trúc phần mềm trong đó các ứng dụng được chia thành nhiều dịch vụ nhỏ độc lập, mỗi dịch vụ thực hiện một chức năng cụ thể.
* Giao tiếp giữa các service thông qua thông qua HTTP, gRPC, hoặc các giao thức tương tự.
* Các service sẽ tồn tại độc lập với nhau và không ảnh hưởng gì nhau nếu như xảy ra vấn đề ở 1 service nào đó.

Microservice và serverless có giống nhau phần nào ở việc đối với microservice thì các service sẽ hoạt đọng riêng lẻ và không có sự liên quan ảnh hưởng gì đến nhau nếu gặp sự cố. Tương tự đối với serverless thì các function bên trong đó cũng tồn tại độc lập và không ảnh hưởng gì đến nhau trong việc triển khai

Khi so sánh với kiến trúc serverless được nói ở trên thì đối với microservice cũng sẽ có những điểm mạnh và điểm yếu khác nhau và có thể xem một số so sánh như sau:

| Tiêu chí | Microservices | Serverless |
| --- | --- | --- |
| Cách triển khai | Triển khai trên các máy chủ riêng lẻ | Triển khai lên nền tảng cloud computing Faas |
| Quản lý hạ tầng | Quản lý container các service | Nền tảng cloud Computing đảm nhiệm |
| Khả năng mở rộng | Mở rộng theo dịch vụ (manual hoặc auto-scaling). | Mở rộng tự động theo từng hàm dựa trên số lượng yêu cầu. |
| Thời gian phát triển | Cần nhiều thời gian để thiết lập, triển khai và tối ưu. | Tập trung vào logic ứng dụng, giảm thời gian thiết lập hạ tầng. |
| Trường hợp sử dụng phổ biến | Ứng dụng lớn, có các phần độc lập và yêu cầu tương tác phức tạp. | Ứng dụng nhỏ, quy mô vừa, yêu cầu xử lý nhanh và không liên tục. |

Dựa trên một số so sánh thì ta nhận thấy dù 2 kiến trúc này có thể tồn tại độc lập và không phụ thuộc nhưng vai trò vẫn khác nhau ở các trường hợp sử dụng khác nhau. Vơi Microservices phù hợp với các hệ thống lớn, phức tạp, cần kiểm soát chi tiết và yêu cầu hoạt động ổn định liên tục và Serverless lại lý tưởng cho các ứng dụng nhỏ hơn, có khối lượng công việc không đều, hoặc khi mục tiêu chính là tối ưu chi phí và thời gian phát triển.

1. **So sánh với Container Architecture**

Tổng quan cơ bản:

**Container Architecture** là một phương pháp triển khai và vận hành ứng dụng phần mềm thông qua các "container"—môi trường ảo hóa nhẹ, độc lập và nhất quán. Containers gói gọn tất cả các phụ thuộc (thư viện, file cấu hình, mã nguồn) cần thiết để chạy một ứng dụng, đảm bảo tính nhất quán trên mọi môi trường từ phát triển đến triển khai.

Cả serverless architecture và container architecture đều cho phép các nhà phát triển deploy ứng dụng bằng cách trừu tượng hóa môi trường máy chủ, nhưng giữa chúng có những điểm khác biệt. Khi sử dụng container architecture thì lập trình viên phải update lại từng container khi có sự thay đổi còn với serverless thì việc đó được tự động xử lí dựa trên môi trường cloud computing của chính nó

So sánh giữa Serverless architechture và container architecture:

| Tiêu chí | Container Architecture | Serverless Architecture |
| --- | --- | --- |
| Quản lý môi trường | Lập trình viên tự duy trì hệ điều hành, runtime và container. | Dịch vụ cloud computing đảm nhận toàn bộ việc bảo trì. |
| Mở rộng | Yêu cầu sử dụng nền tảng như Kubernetes để mở rộng. | Tự động mở rộng dựa trên số lượng yêu cầu. |
| Ứng dụng phù hợp | Ứng dụng có lưu lượng truy cập cao liên tục hoặc yêu cầu kiểm soát chi tiết. | Ứng dụng dựa trên sự kiện, tác vụ ngắn hạn hoặc không thường xuyên. |

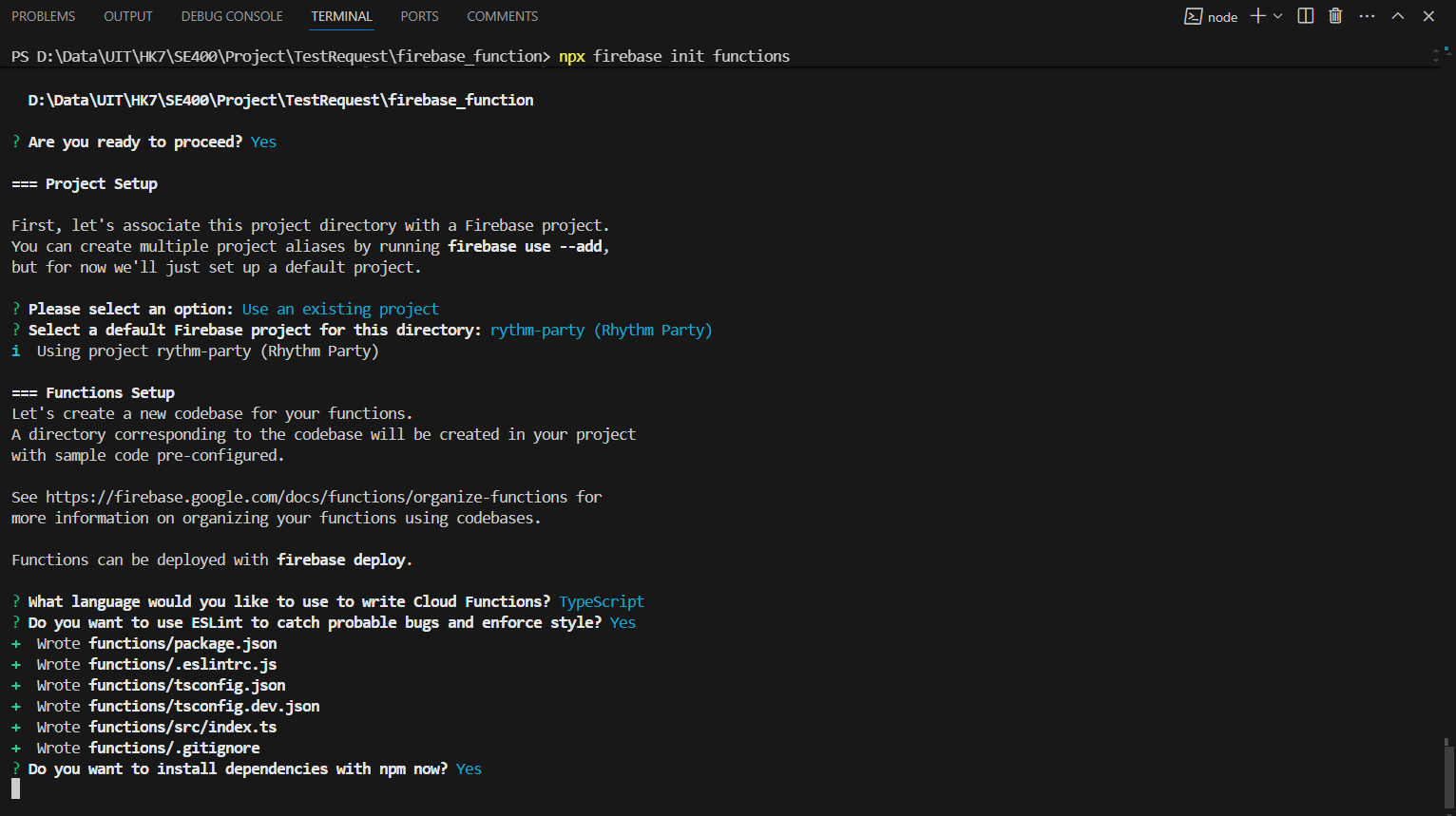
## Cách thức deploy của firebase function bên trong hệ thống serverless của firebase

Đối với firbase function thì firebase đã hỗ trợ và cung cấp các hàm và function để việc xây dựng và triển khai các function nó trở nên dễ dàng hơn.

Các bước thực hiện để tạo 1 function như sau:

B1. Khởi chạy xây dựng dự án firebase function:

Chạy lệnh: *npx firebase init function* để khởi tạo project function cho hệ thống

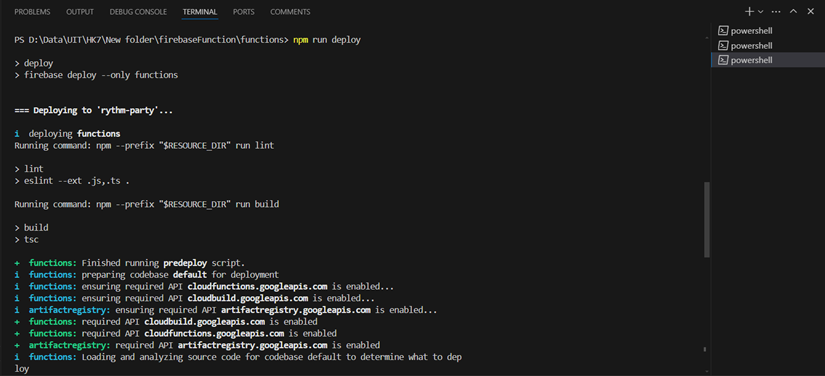


B2: Tạo 1 vài function để triển khai lên hệ thống



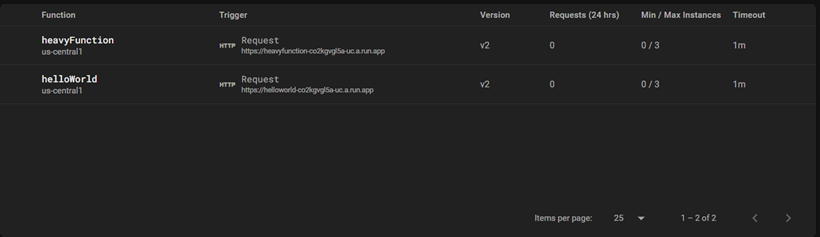
B3: Deploy function lên trên hệ thống server

Dùng lệnh “**npm run deploy**” có sẵn và được define như sau: "*deploy*": "firebase deploy --only functions",



B4. Kiểm tra function trên hệ thống

Đây là 2 function mà ta đã deploy khi nãy gồm heavyFunction và helloWorld



Dựa trên tất cả các bước trên thì ta đã có thể xây dựng 1 vài function mà ta mong muốn có ở trên firebase function.

## Quy trình deploy và phát triển của Firebase function trên hệ thống serverless

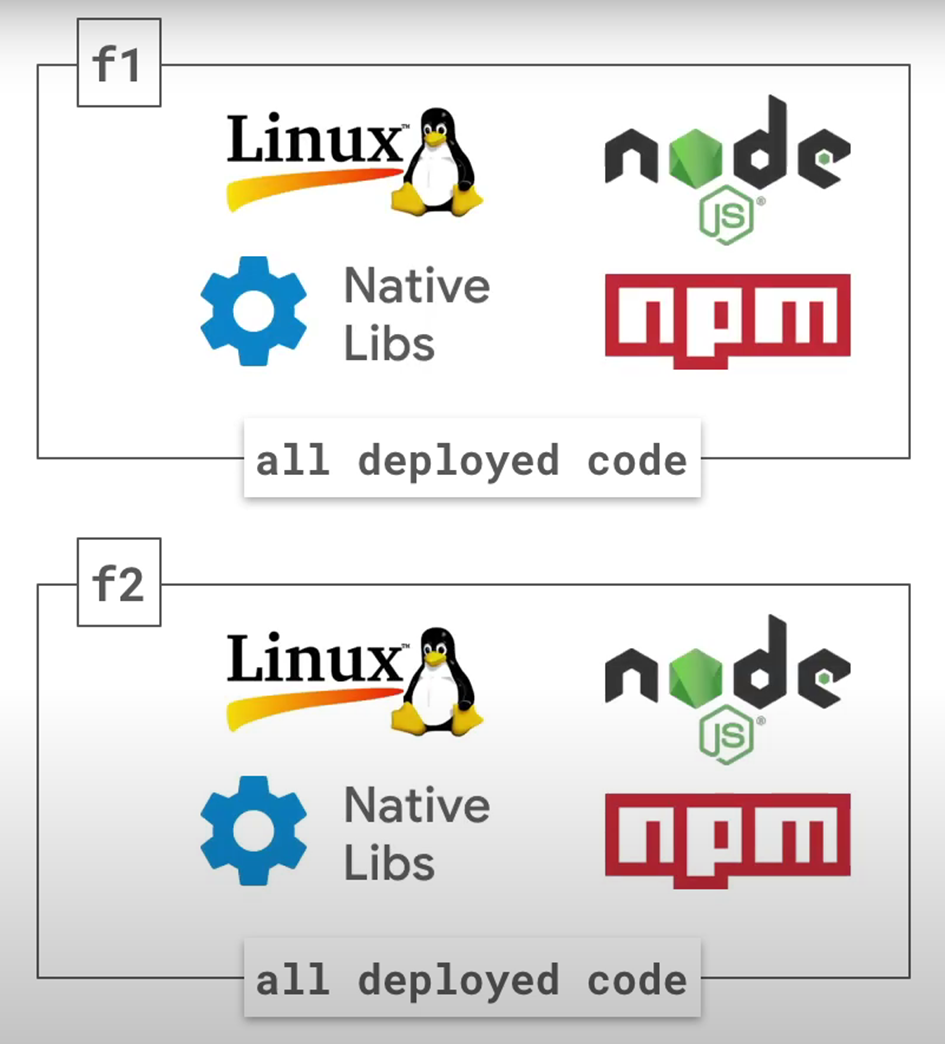
Dù firebase đã tối ưu cho việc deploy 1 function lên trên firebase rồi nhưng nó vẫn có quy trình hoạt động để có thể deploy thành công 1 dự án với 1 hoặc nhiều function bên trong đó gồm các bước kiểm tra và triển khai sau:

**B1:** Ban đầu Firebase CLI sẽ kiểm tra xác thực tài khoản đang được đăng nhập local có chính xác không và kiểm tra quyền hạn tài khoản với dự án firebase đó có đủ quyền để có thể deploy function lên trên dự án không.

**B2:** Firebase CLI sẽ chạy lệnh check eslint để kiểm tra tĩnh mã nguồn được xây dựng về lỗi cú pháp, lỗi logic, và những vi phạm quy tắc về coding style trong các dự án sử dụng JavaScript hoặc TypeScript ( tùy ngôn ngữ lập trình viên chọn xây dựng). Nếu lỗi thì việc deploy sẽ failed và đưa ra các lỗi cú pháp cần chỉnh trong dự án.

**B3:** Tiếp đến sẽ build và compile mã nguồn, nếu đang sử dụng TypeScript, mã nguồn sẽ được biên dịch sang JavaScript. Firebase CLI sẽ xử lý và chuẩn bị các dependency (trong nodemodules) cần thiết. Nếu có ở bước này có lỗi thì việc deploy sẽ failed và đưa ra các lỗi code bên trong dự án.

**B4:** Firebase CLI nén toàn bộ mã nguồn (bao gồm các dependency) đóng gói chúng vô trong 1 container với mỗi function sẽ được lưu vô trong 1 container khác nhau và nếu được kêu ra sử dụng thì sẽ được đem ra sử dụng. Và container này sẽ được tải lên Firebase Functions để triển khai.



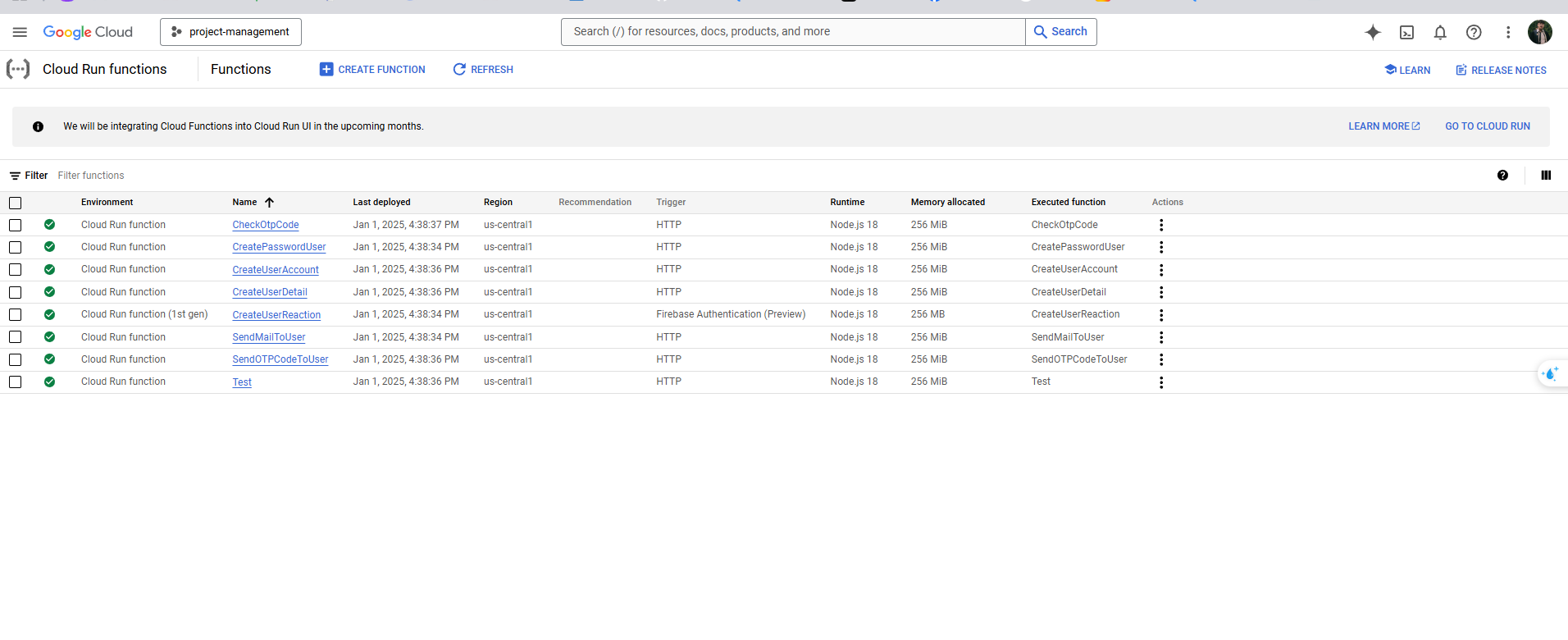
**B5:** Firebase upload container và các file nén lên máy chủ, giải nén và cài đặt các dependency. Sau đó sẽ thiết lập môi trường Node.js (theo phiên bản được chỉ định trong package.json hoặc mặc định).

**B6:** Firebase Functions thực hiện kiểm tra tính khả dụng của hàm (ví dụ: kiểm tra lỗi, xác minh các trigger). Sau khi xác nhận thành công, các hàm được kích hoạt và sẵn sàng xử lý yêu cầu từ người dùng hoặc hệ thống.

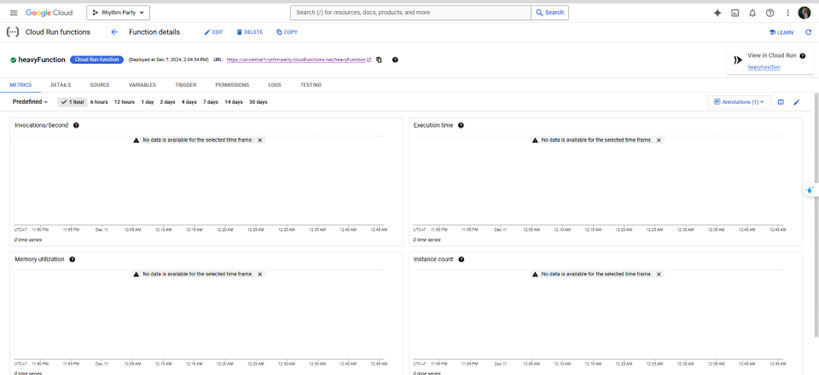
**B7:** Nếu tất cả các bước trước đó hoàn tất mà không gặp lỗi, Firebase CLI sẽ thông báo triển khai function thành công. Nếu trong 1 dự án triển khai 1 lúc nhiều function khác nhau thì có thể thông báo 1 số function nào được build thành công và nếu nào build thất bại

## Quản lý và phát triển hệ thống firebase function.

Sau khi đã triển khai thành công các Function lên trên Firebase thì ta có thể quản lý các function đó bên trong Google console ở mục Cloud Run Function của project đó



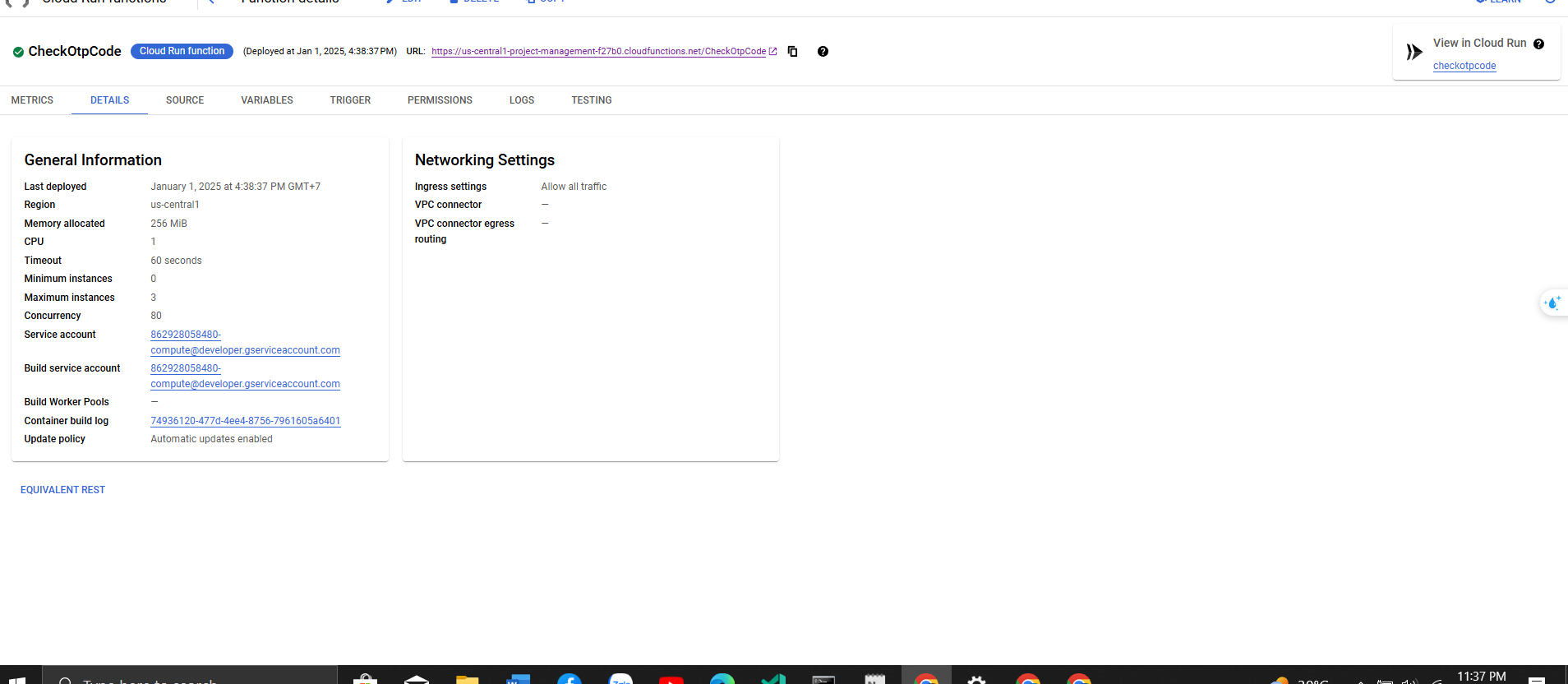
Ta có thể xem chi tiết của các function đó như thế nào



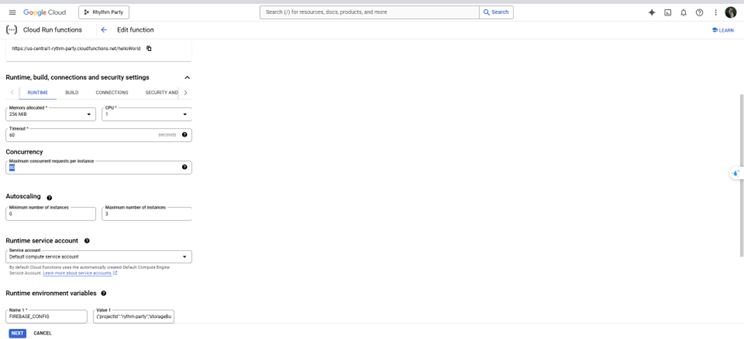
Ở đây tacó thể xem chi tiết hoạt động của function bao gồm:

* **Innovation/second:** Số lần truy cập đến api trong 1 giây
* **Execution time:** Thời gian thực thi của một function từ lúc được kích hoạt đến khi hoàn thành xử lý.
* **Memory utilization:** Lượng bộ nhớ (memory) mà một instance sử dụng khi thực thi function, thường được đo bằng megabytes (MB)
* **Instance count:** Số lượng instance (môi trường thực thi độc lập) được tạo ra để xử lý các yêu cầu hoặc sự kiện trong một thời điểm cụ thể.

Ngoài ra ta có thể xem các cấu hình chi tiết các thông số khi sử dụng function đó trên hệ thống. Các thông số được ghi ở đây là những thông số tối đa mà ta tạo ra cho các function để đảm bảo các function chạy ở những mức độ phù hợp với yêu cầu vs mong muốn của người dùng và không bị quá giới hạn mong muốn của lập trình viên khi xây dựng các function này.



Tùy chỉnh thông số cấu hình tối đa function theo mong muốn của mình



Với việc tùy chỉnh cấu hình này thì ta có thể cấu hình các thông số như:

* **Memory allocated:** Dung lượng RAM cho mỗi instance khi được sử dụng và tồn tại
* **CPU**: Số lượng CPU chạy trên mỗi instance đó
* **Timeout**: Thời gian tối đa 1 request
* **AutoScaling**: Số lượng instance tối thiểu hoặc tối đa mà 1 function có thể sử dụng để scale lên 1 cách phù hợp và nếu như function không hoạt động thì nó trả về mặc định là 0 nếu không sử dụng hoặc nếu muốn giảm thời gian cold start thì ta có thể để min instance là 1 để giảm thiểu thời gian cold start ( thời gian cold start là thời gian để request đến function và không có instance nào sẵn sàng xử lý).

**=> Nhờ những tùy chỉnh này mà ta có thể chọn và cấu hình phù hợp với nhu cầu cũng như mong muốn của nhóm cũng như developer để chọn và sử dụng các tác vụ phù hợp với từng ứng dụng và kinh phí sử dụng.**

## Xử lí lỗi khi có function có lỗi

Khi function lỗi thì tùy loại function mà ta có thể cho function đó retry khi gặp lỗi hay không.



Ta có thể tùy chỉnh việc retry on failure dựa trên các function dựa trên các thông số này khi check retry on failure tuy nhiên việc retry này cũng sẽ dễ bị lặp vô tận nếu như function lỗi liên tục và không thể thoát ra làm tiêu tốn tài nguyên

Chính vì vậy trong mỗi lần chạy 1 event thì function sẽ sinh ra eventId nếu như đã chạy và ta sẽ dùng 1 bảng collection riêng là RetryCount để có thể tính số lần retry của dự án rồi và từ đó sẽ check được là function này đã được retry mấy lần rồi giúp ta có thể handle được số lần retry của hệ thống đó và cancel nếu đã quá số lần retry.



## Container bên trong Firebase Function

Khi 1 function được deploy lên trên Firebase thì nó được đóng gói và nằm ở trong 1 container. Mỗi một function khi được gọi đến thì nếu chưa có thông tin gì thì Firebase sẽ khởi chạy container đó và build nên 1 Instance để từ đó có thể khởi chạy Function. Các dependencies (như thư viện, runtime, hoặc môi trường cần thiết) cũng được bao gồm trong container.

## Cơ chế hoạt dộng của Instance bên trong firebase

Instance ở trong Firebase nó giống như 1 máy ảo chạy độc lập với nhau và được khởi tạo với một runtime riêng biệt và nó sẽ không chia sẻ bộ nhớ, biến toàn cục, hoặc dữ liệu in-memory với các instance khác. Đồng thời instance này không biết đến trạng thái của các instance khác.

Các instance nó hoạt động khá giống như là máy ảo nhỏ được cách ly hoàn toàn về tài nguyên CPU, RAM, và môi trường runtime và Firebase Functions sử dụng các container nhẹ để chạy các instance này.

**Một số cơ chế hoạt động của Instance gồm:**

* *Khởi tạo instance:*
* Nếu chưa có instance(hoặc sau một thời gian dài không sử dụng), Firebase sẽ khởi chạy một container mới từ hình ảnh (image) của function. Đây được gọi là **"cold start"** và thường mất thêm thời gian để khởi động (vài trăm milliseconds đến vài giây, tùy thuộc vào cấu hình).

Quá trình này bao gồm:

* Tải mã nguồn của hàm từ môi trường lưu trữ.
* Cài đặt các dependency được liệt kê trong package.json.
* Khởi tạo runtime (Node.js, Python, v.v.).
* Thực thi đoạn mã khởi tạo trong hàm.
* *Tái Sử Dụng Instance:*
* Sau khi một instance được khởi tạo, Firebase có thể tái sử dụng nó để xử lý các yêu cầu tiếp theo nhằm tối ưu hóa tài nguyên và giảm thời gian phản hồi. Việc này giúp giảm thời gian khởi tạo khi dùng lại những instance cũ. Tuy nhiên dữ liệu tạm thời trong bộ nhớ instance (in-memory) có thể không đáng tin cậy vì instance có thể bị xóa bất kỳ lúc nào khi Firebase cần tài nguyên.
* *Tự Động Mở Rộng:*
* Firebase Functions tự động mở rộng (scale-out) bằng cách tạo thêm các instance mới khi lưu lượng tăng cao, tuy nhiên nó chỉ tạo đến mức tối đa được setup trong options.
* Cách thức hoạt động của nó là nếu 1 instance không thể xử lí thêm yêu cầu thì Firebase sẽ cung cấp thêm 1 instance mới để có thể xử lí các yêu cầu mới đồng thời dữ liệu in memory
* *Tự động thu hẹp:*
* Khi lưu lượng giảm, Firebase sẽ tự động thu hẹp số lượng instance (scale-in) để tiết kiệm tài nguyên.
* Các instance không hoạt động trong một khoảng thời gian (khoảng 15 phút) sẽ bị xóa.
* Chính vì vậy giúp tối ưu hóa chi phí vì bạn chỉ trả tiền cho tài nguyên được sử dụng.

## Cold start và cách xử lí bên trong serverless để tối ưu hệ thống

Vậy cold start ở đây là gì?

**Cold start** là hiện tượng xảy ra khi một hàm serverless được khởi chạy lần đầu tiên hoặc sau một khoảng thời gian không hoạt động. Trong giai đoạn này, môi trường thực thi (runtime environment) cần được khởi tạo, dẫn đến thời gian phản hồi lâu hơn so với các lần gọi hàm sau đó (warm invocation).

Thời gian cold start xảy ra khi instance đó đang được cho nghỉ ngơi ở trường hợp không có function nào hoạt động và số lượng instance đều tắt hoạt động nhằm tối ưu lưu lượng sử dụng của instance đó.

Quy trình khởi tạo cold start bao gồm:

* Cấp phát tài nguyên (container, bộ nhớ, CPU).
* Tải mã nguồn và các dependency (thư viện) cần thiết.
* Khởi tạo runtime (Node.js, Python, v.v.).
* Chạy các mã khởi tạo (initialization code) trong hàm.

Vậy làm sao để ta có thể giải quyết vấn đề này khi function không được gọi trong 1 thời gian dài? Thì như đã đề cập ở trên khi nhóm so sánh giữa việc sử dụng 1 instance hoặc 0 instance bên trong việc setup thì nếu ta để min instance bằng 0 thì số lượng instance chạy ít nhất là 0 nên hệ thống sẽ tự động tắt hết nếu không có hoạt động chính vì vậy ta có thể set mặc định là 1 instance để luôn luôn tồn tại 1 instance luôn tức trực khi cần thiết. Tuy nhiên việc để 1 instance như vậy cũng làm gây rủi ro tốn dung lượng và hoạt động của function làm tăng chi phí không cần thiết.



## Cơ Chế Mở Rộng Của Firebase Functions Khi Quá Tải

Firebase Functions hoạt động theo cơ chế mở rộng tự động (auto-scaling). Cụ thể:

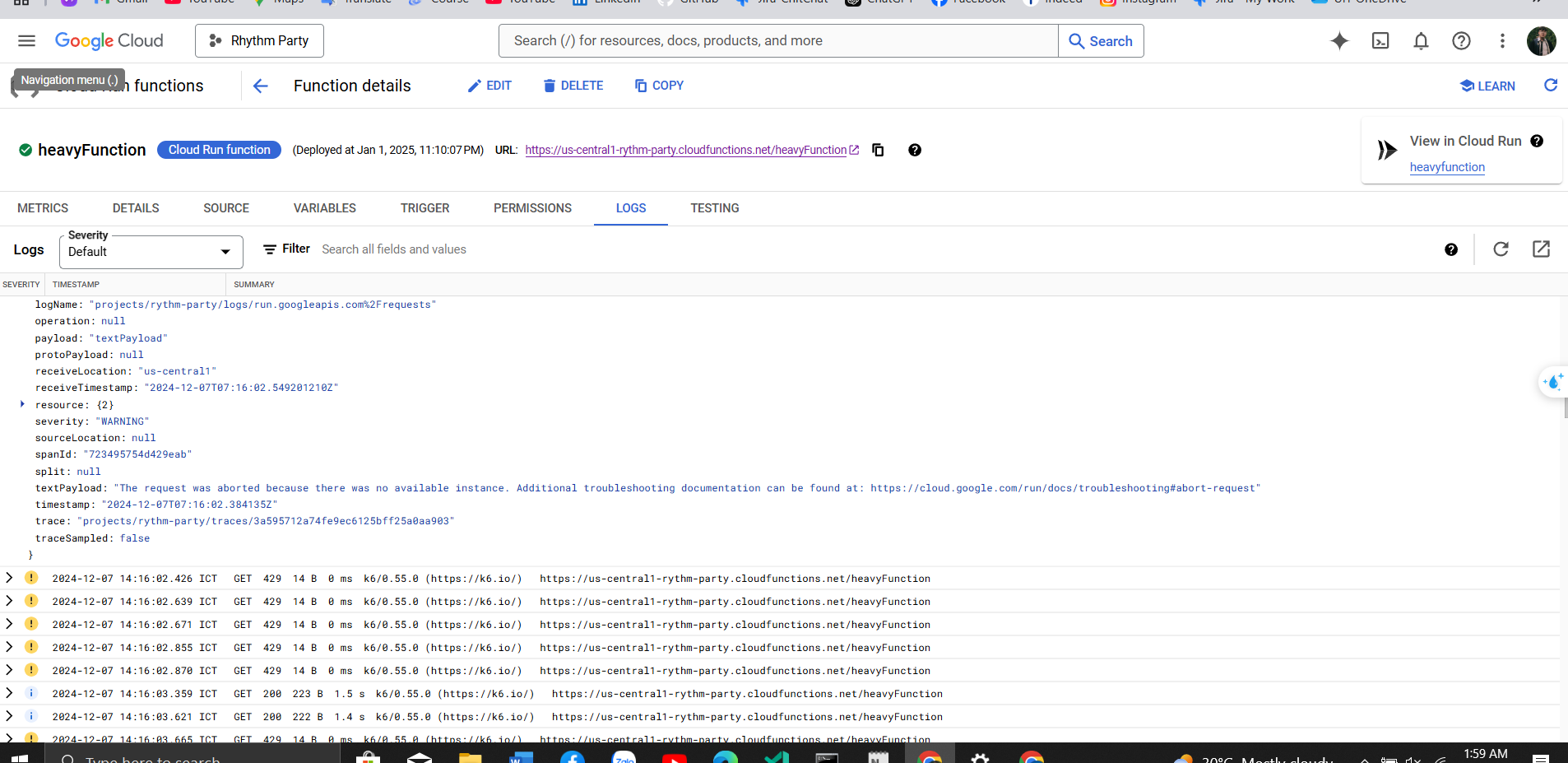
* Khi số lượng yêu cầu (request) vượt quá khả năng xử lý của một instance, Firebase sẽ tự động tạo thêm các instance mới để đáp ứng.
* Số lượng instance có thể được mở rộng lên đến **giới hạn tối đa** được cấu hình (hoặc mặc định của Firebase).

Tuy nhiên, nếu đã đạt giới hạn tối đa về số lượng instance và các request tiếp theo không được xử lý:

* **Hàng đợi yêu cầu (Request Queue):**Các request chưa được xử lý sẽ được đưa vào hàng đợi. Firebase cố gắng xử lý các request này tuần tự khi các instance hiện tại hoàn thành công việc của chúng.
* **Timeout của request:**Nếu một request trong hàng đợi không được xử lý kịp thời và vượt quá thời gian timeout của function (thường là 60 giây đối với HTTP-triggered functions), nó sẽ thất bại và trả lỗi về phía client.

Nhóm đã chạy thử nghiệm benchmark để kiểm tra giới hạn của Firebase Functions khi xử lý lượng lớn request đồng thời.

* Khi số lượng request vượt quá khả năng xử lý tối đa của các instance, Firebase sẽ đưa ra **cảnh báo** (warning) rằng không đủ instance để đáp ứng.
* Nếu tình trạng quá tải kéo dài và tài nguyên (RAM hoặc CPU) của các instance không đáp ứng được, các request sẽ bị **failed** và trả lỗi về cho client sau một thời gian chờ đợi.



## CI/CD trong serverless.

Nhằm đảm bảo function luôn được deploy khi push hoặc merge code từ trong nhánh khác vô nhánh master thì nhóm đã xây dựng CI/CD cho hệ thống bằng git action



Với Git action thì thì sẽ được kích hoạt khi được push hoặc merge vào master.

Các bước run của CI này gồm:

B1: Checkout code để lấy code về build

B2: Install package của Firebase CLI trong thư mục BE

B3: Install các package được sử dụng bên trong firebase function bên trong BE/functions

B4: Chạy deploy bằng hàm npm run deploy để deploy function mới lên trên server

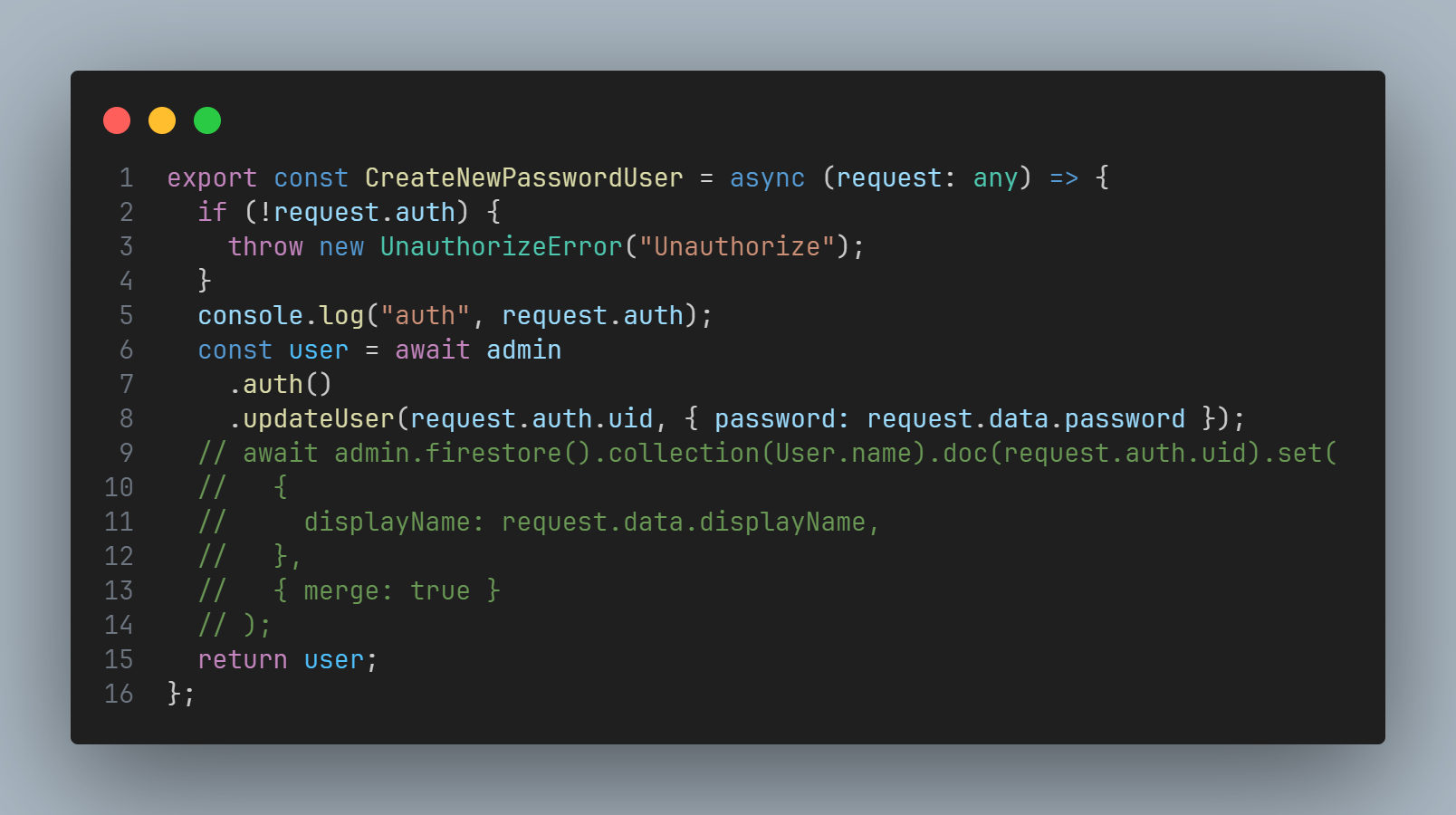
FIREBASE\_TOKEN là enviroment ci để cho firebase có thể authorize user đó trước khi deploy lên sản firebase

SERVICE\_ACCOUNT\_KEY là enviroment chứ thông tin key service của firebase khi deploy

Từ đó có thể xây dựng 1 git action chứa ci/cd cơ bản để có thể deploy function lên trên firebase.

## Bảo mật trong Firebase Functions

Khi sử dụng Firebase function ta có thể kết hợp cùng với Firebase authentication để nhờ đó có thể bảo mật và kiểm tra user khi gọi các hàm trên firebase và check thông tin request đó có xác thực hay đăng nhập hay chưa.



## Cách mở rộng và ứng dụng những điểm mạnh yếu vào trong thực tiễn.

**Điểm mạnh:**

* Nhờ sự kết hợp giữa FaaS (firebase function) và BaaS (firebase auth, firestore) mà ta có thể xây dựng ứng dụng dựa trên firebase và hoạt động hoàn toàn trên serverless mà không cần phụ thuộc vào việc xây dựng 1 server. Việc này giúp cho việc triển khai, xây dựng ứng dụng trở nên nhanh và gọn hơn. Đồng thời có thể kiểm tra đăng nhập, authen một cách dễ dàng khi áp dụng và sử dụng chúng khi xây dựng ứng dụng.
* Firebase Authentication và Firestore cung cấp các tính năng xác thực và cơ sở dữ liệu dễ dàng tích hợp. Firebase function giúp xây dựng các lớp backend để xử lí các API đó.
* Có thể tối ưu chi phí do Firebase chỉ tính phí dựa trên tài nguyên sử dụng (số lượng gọi hàm, lưu trữ, số lần xác thực, v.v.) và không cần tốn phí dù không sử dụng như server truyền thống.

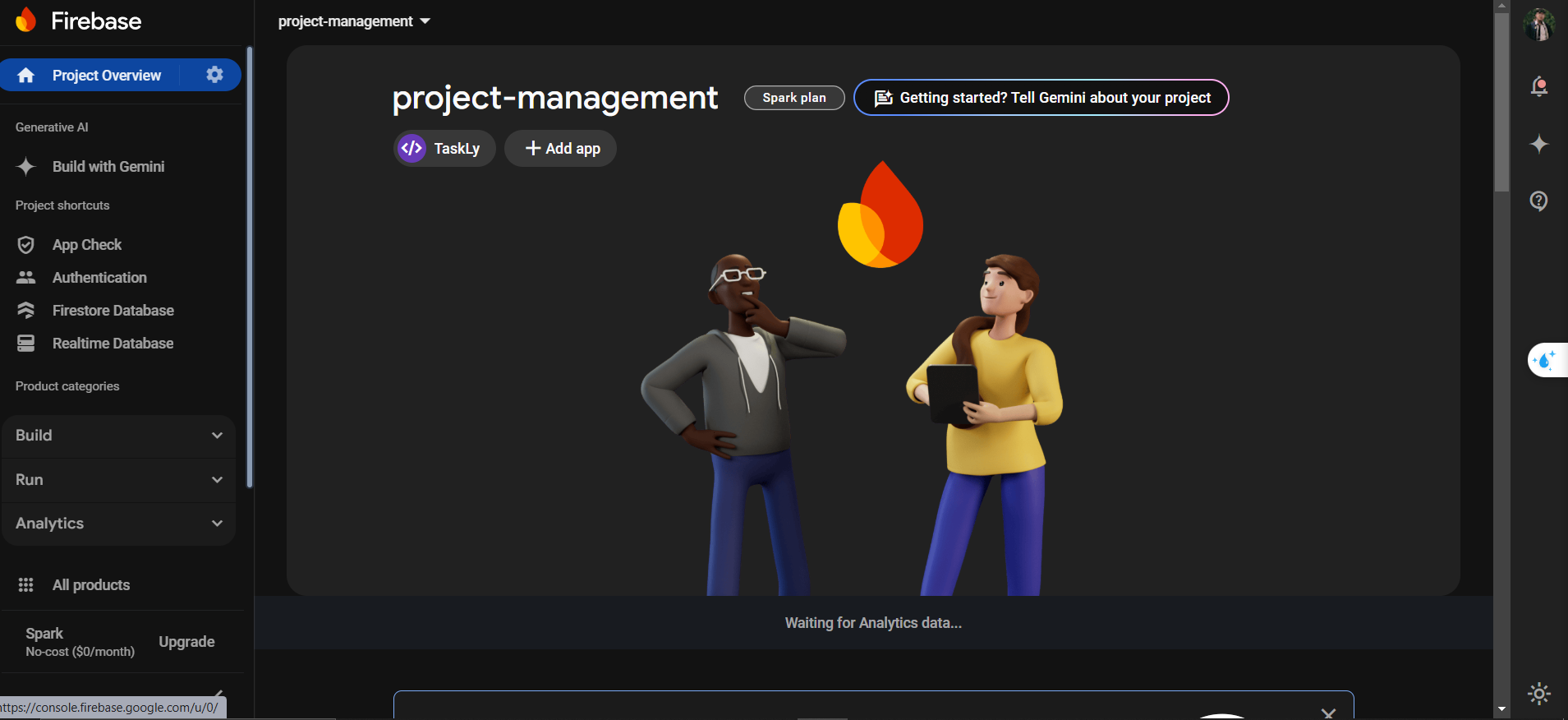
**Điểm yếu:**

* Dù có nhiều điểm mạnh như vậy nhưng việc sử dụng serverless và phụ thuộc vào 1 nền tảng như Firebase như vậy sẽ ảnh hưởng ít nhiều khi ta không nắm hoàn toàn server như cách truyền thống và sẽ phụ thuộc vào bên trong nền tảng cung cấp serverless.
* Có thể các chính sách sẽ thay đổi trong tương lai làm gia tăng các chi phí.
* Firebase là hệ sinh thái độc quyền của Google, gây khó khăn khi cần di chuyển sang nền tảng khác.
* Không thích hợp khi xây dựng 1 ứng dụng lớn do sẽ có thể tốn nhiều chi phí hơn so với cách truyền thống và không thể handle được nhiều trường hợp khác nhau khi cần

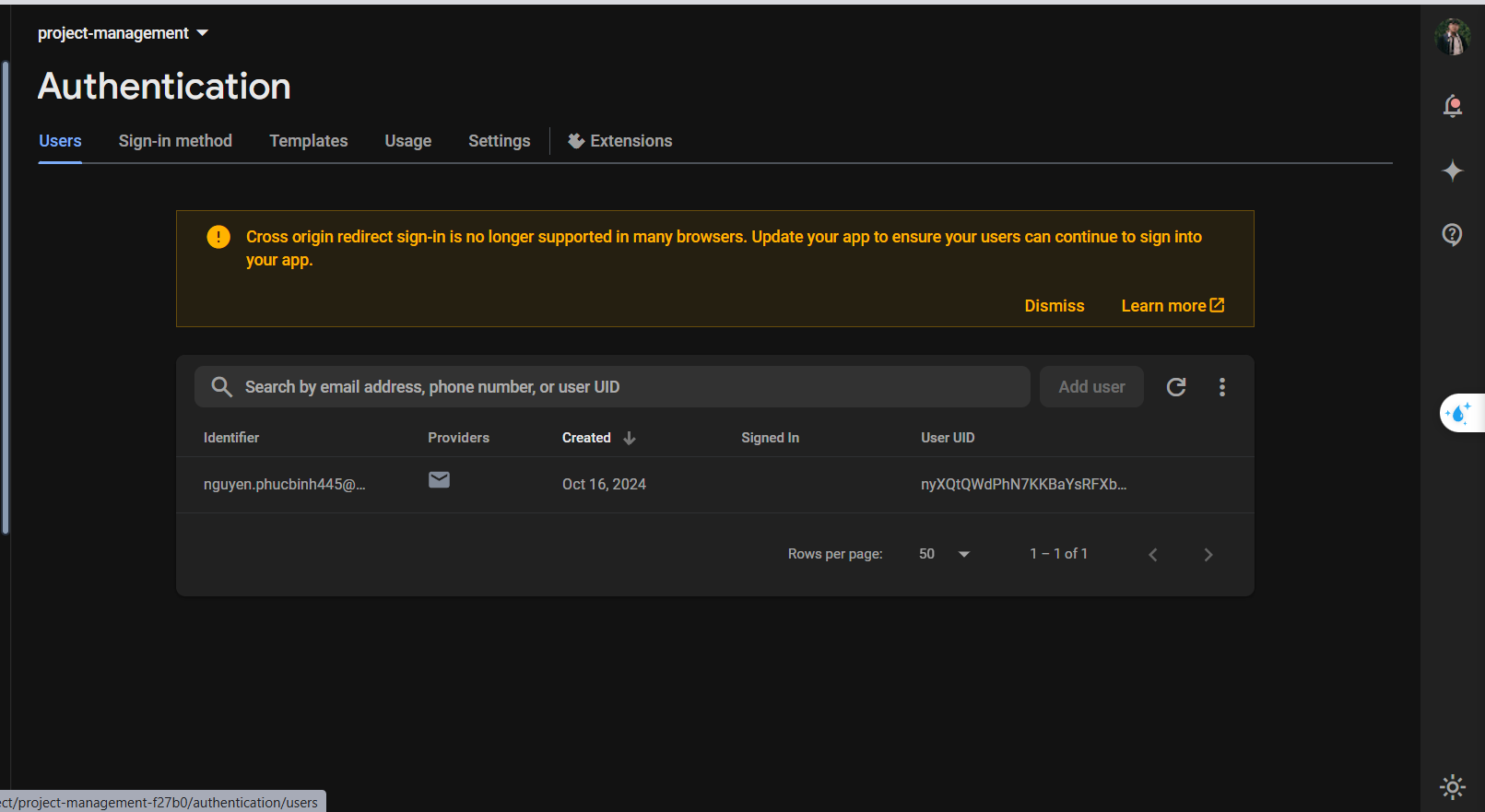
## Xây dựng ứng dụng demo

## Cấu hình ứng dụng Firebase

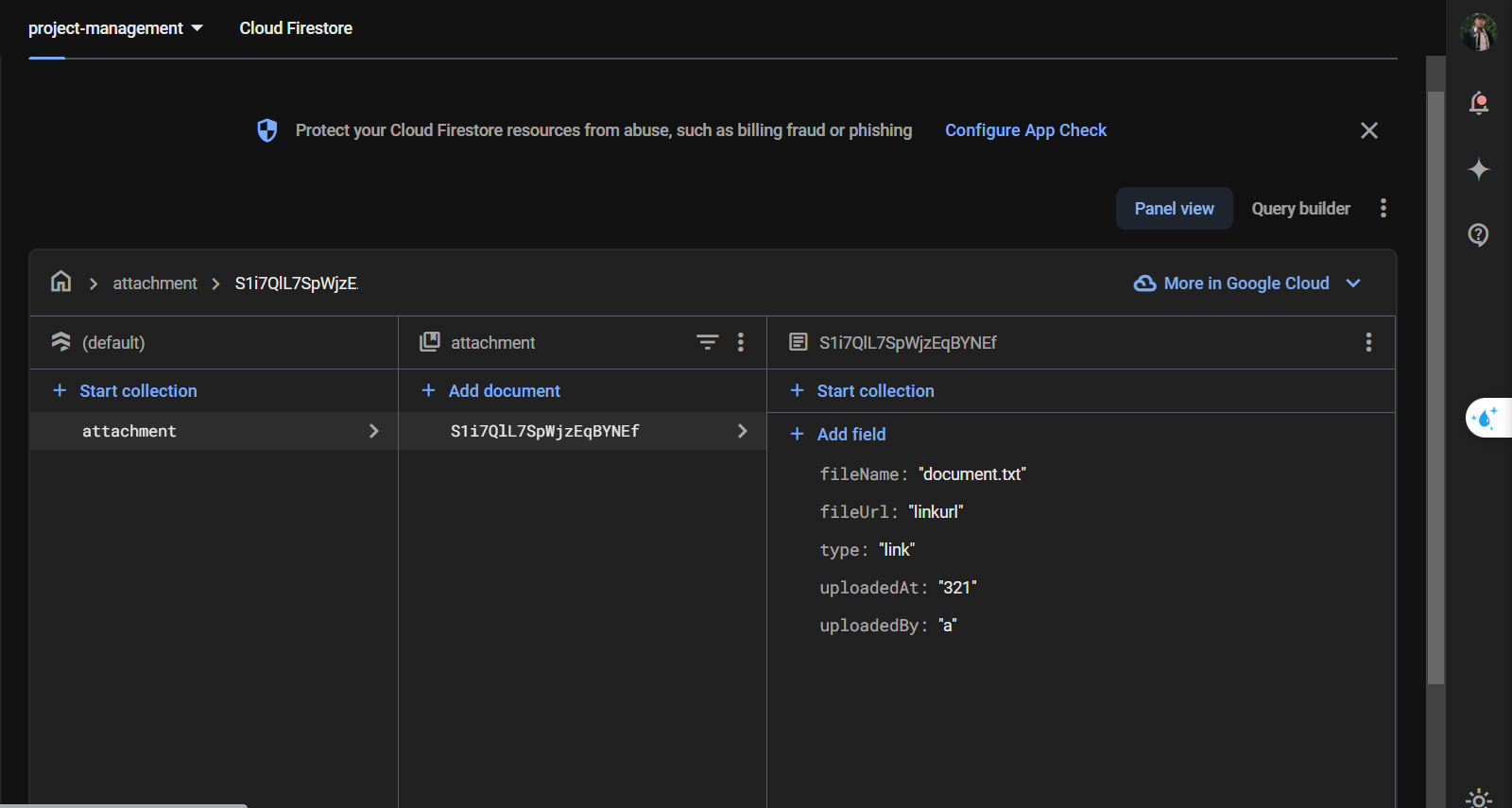
* Vô trang Firebase và tạo dự án bao gồm các việc đặt tên dự án chọn vùng của Cloud và Pricing Plan dựa trên nhu cầu sử dụng của người dùng



* Khởi tạo các Service sử dụng trong ứng dụng bao gồm Firebase authentication, Firestore, Storage,



*Firebase Authentication Service*

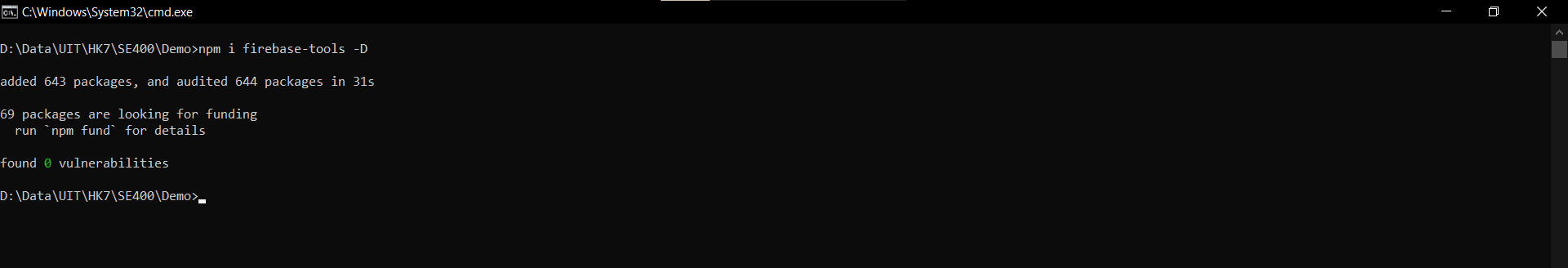
**

*Firestore Service*

Qua đó ta có thể sử dụng các service của Firebase thông qua các service này.

## Kết nối và khởi tạo các Firebase dưới dạng local

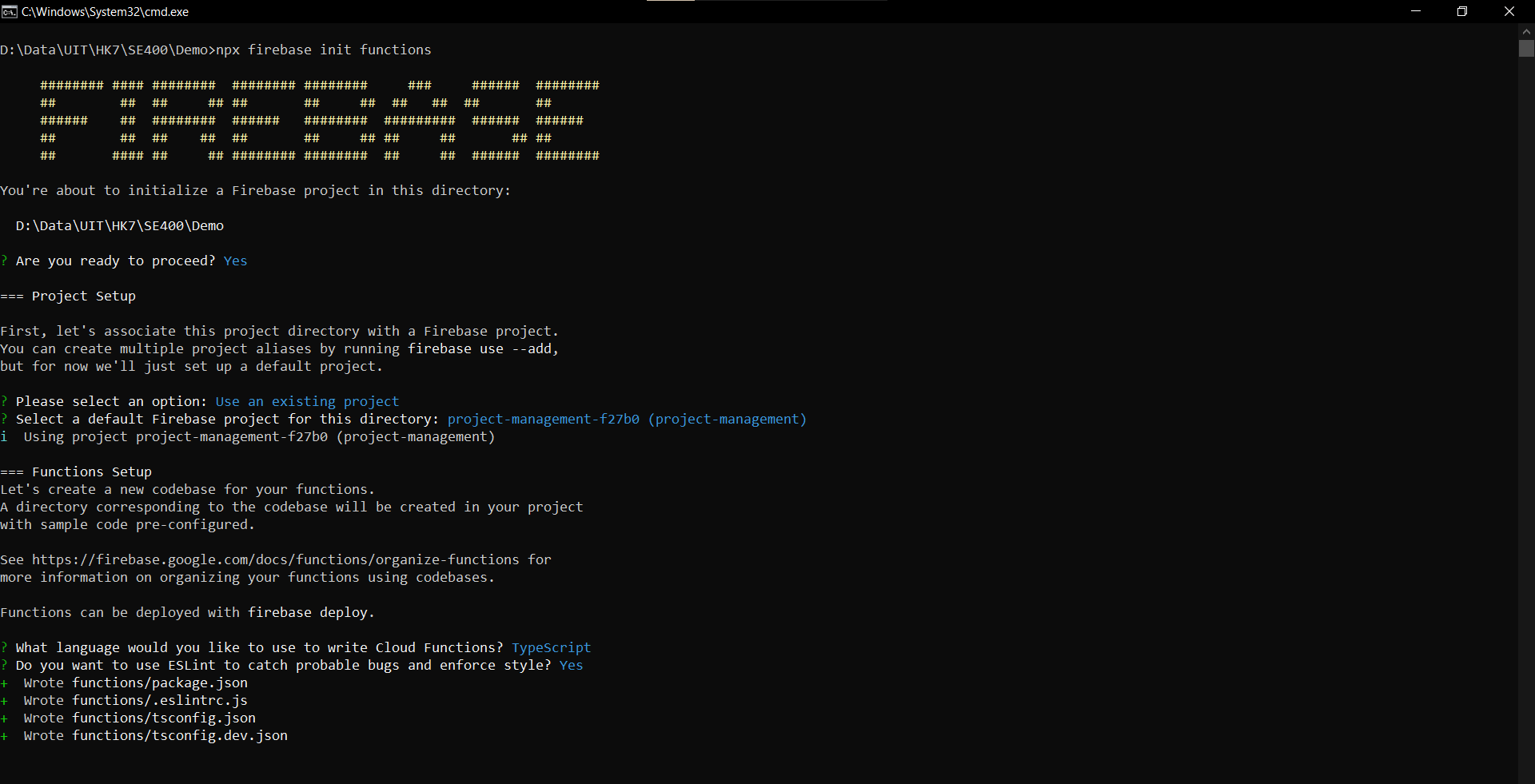
1. Tải Firebase-CLI về máy local thông qua câu lệnh: npm i Firebase-tools -D



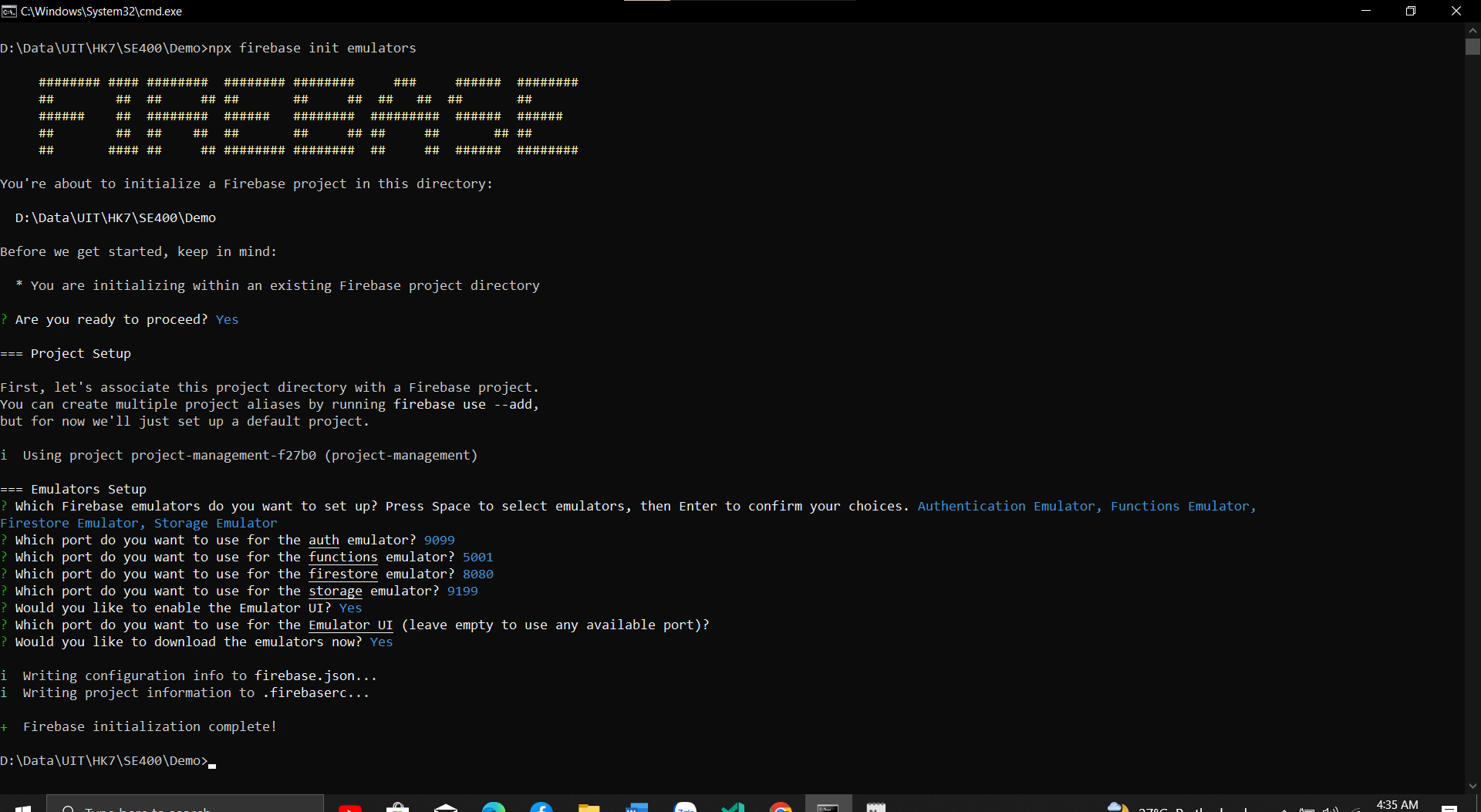
1. Kết nối với Firebase-CLI trên server bằng cách đăng nhập thông qua login tài khoản Firebase



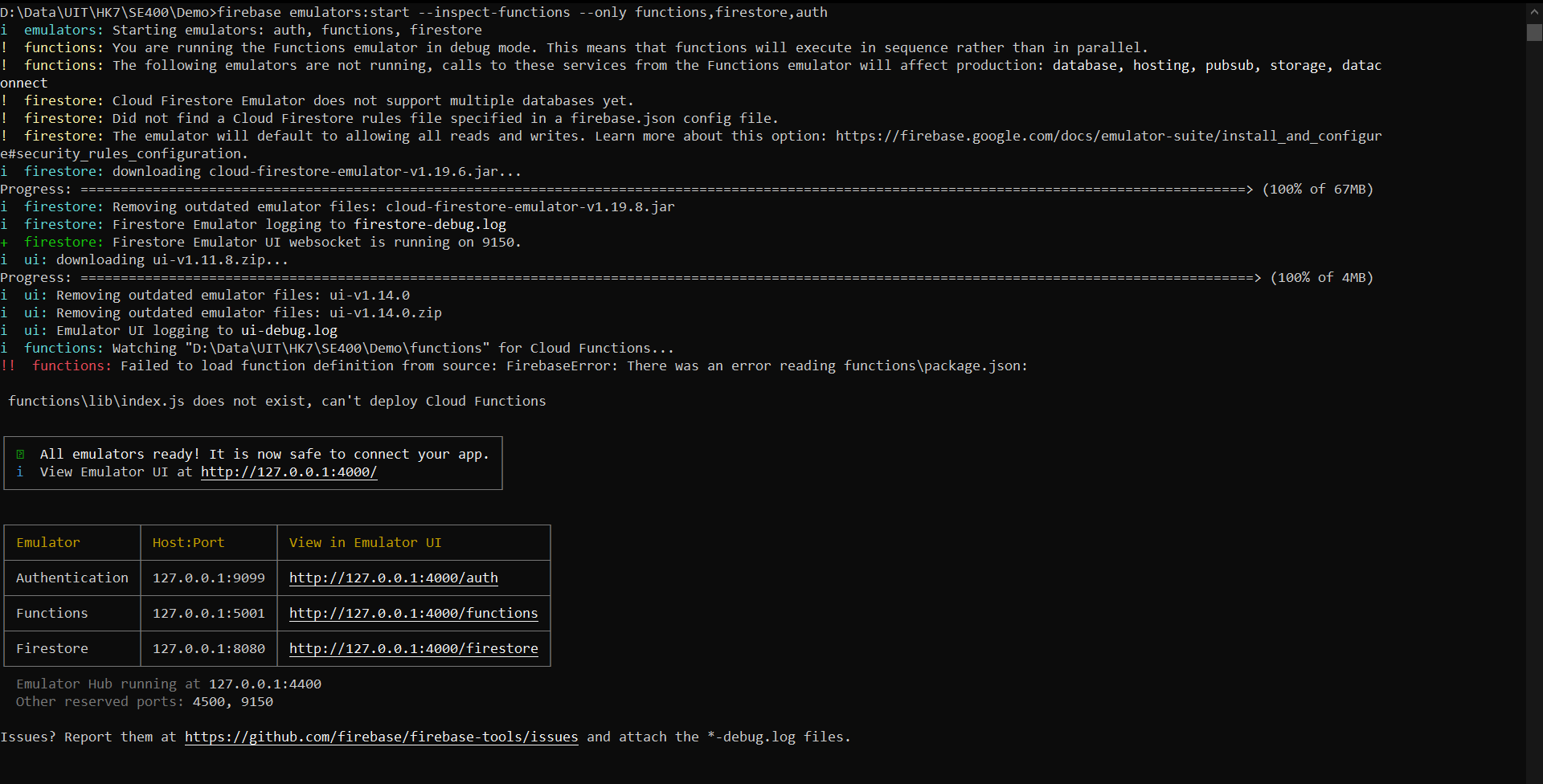
1. Tạo project function ở phía local và kết nối chúng với project đã có trên Firebase



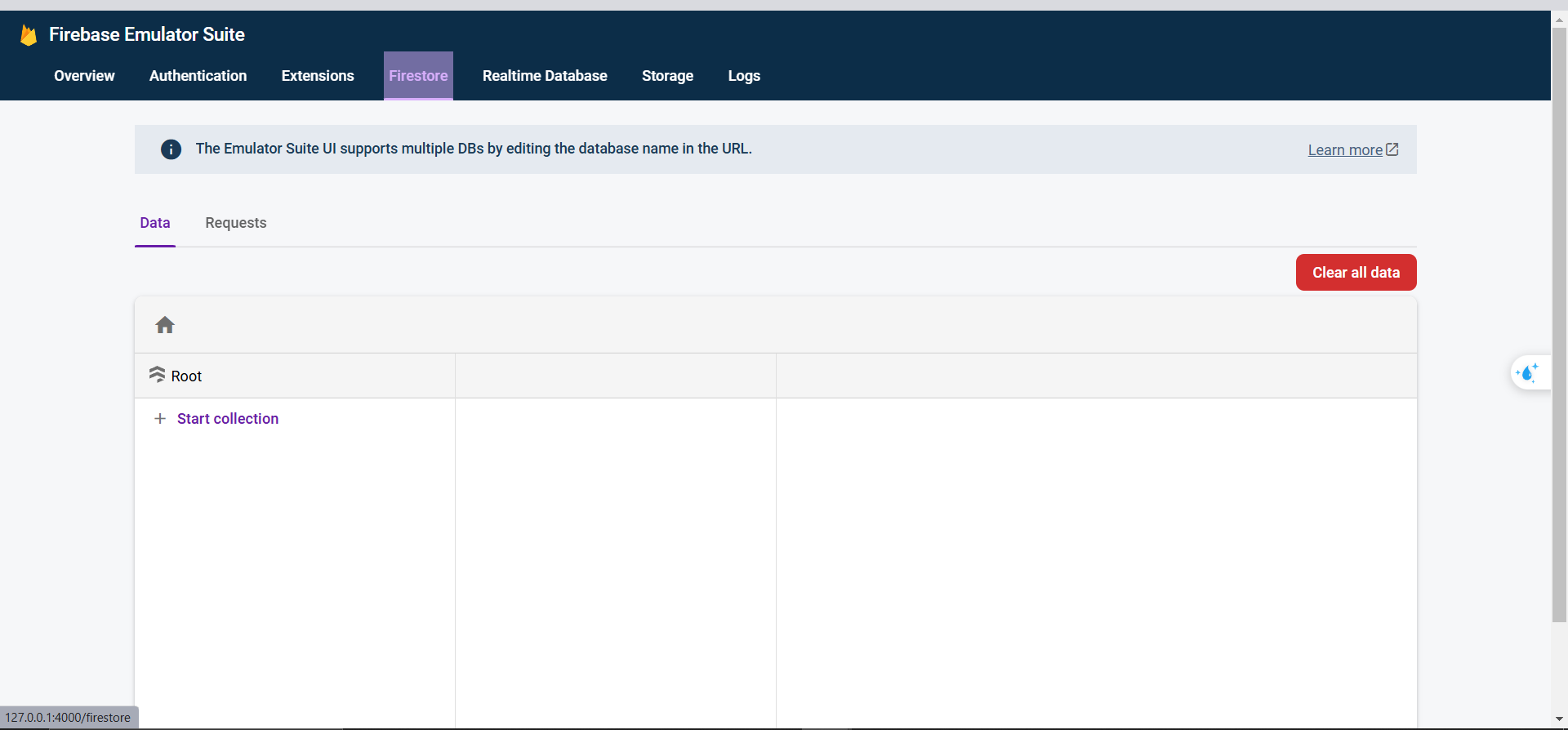
1. Dựa trên Firebase Emulators và tạo các máy ảo cho các service Authentication, Firestore, Firebase Function

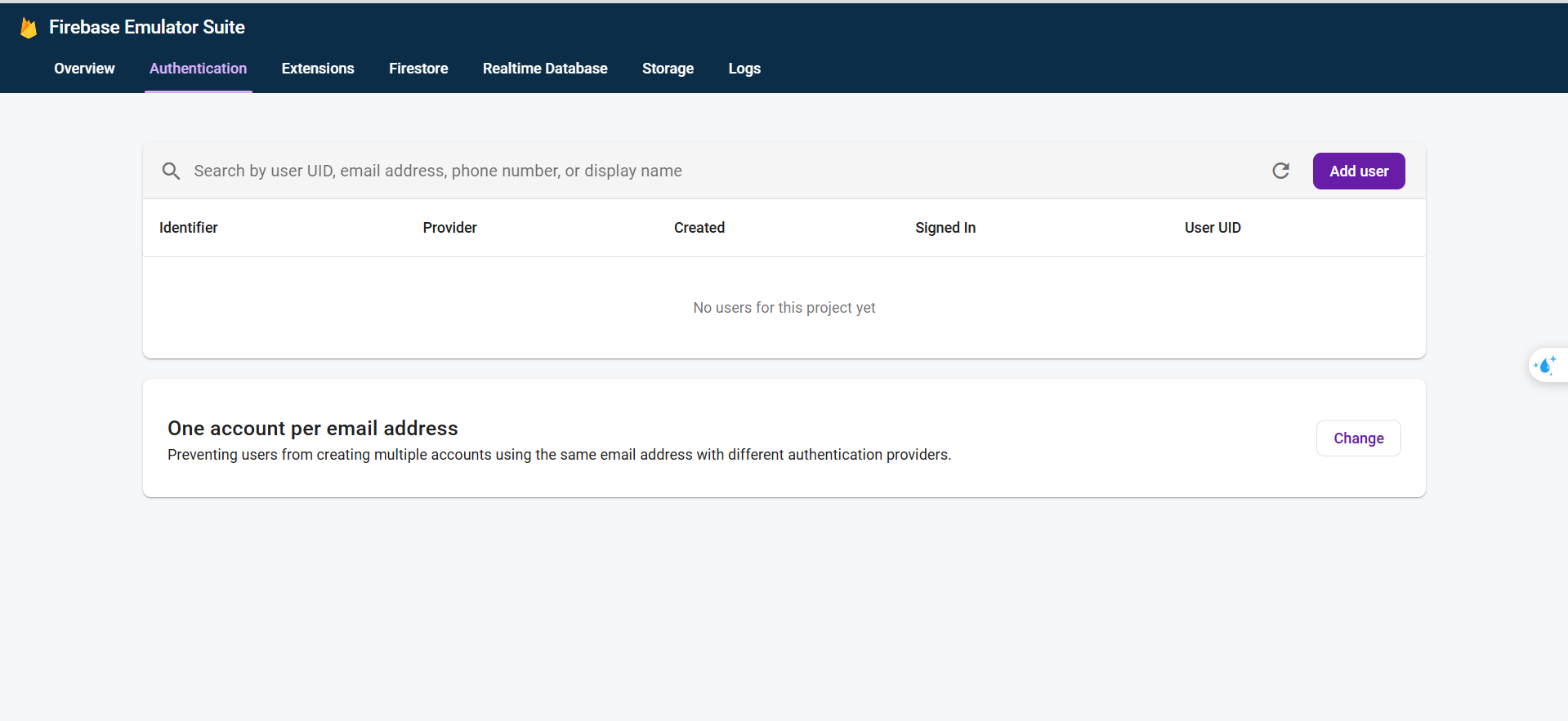


1. Ta có thể khởi chạy các service bằng lệnh: firebase emulators:start --inspect-functions --only functions,firestore,auth



1. Sau đó ta có thể tương tác và sử dụng các Firebase dưới dạng local mà không cần tương tác trực tiếp với server chính.





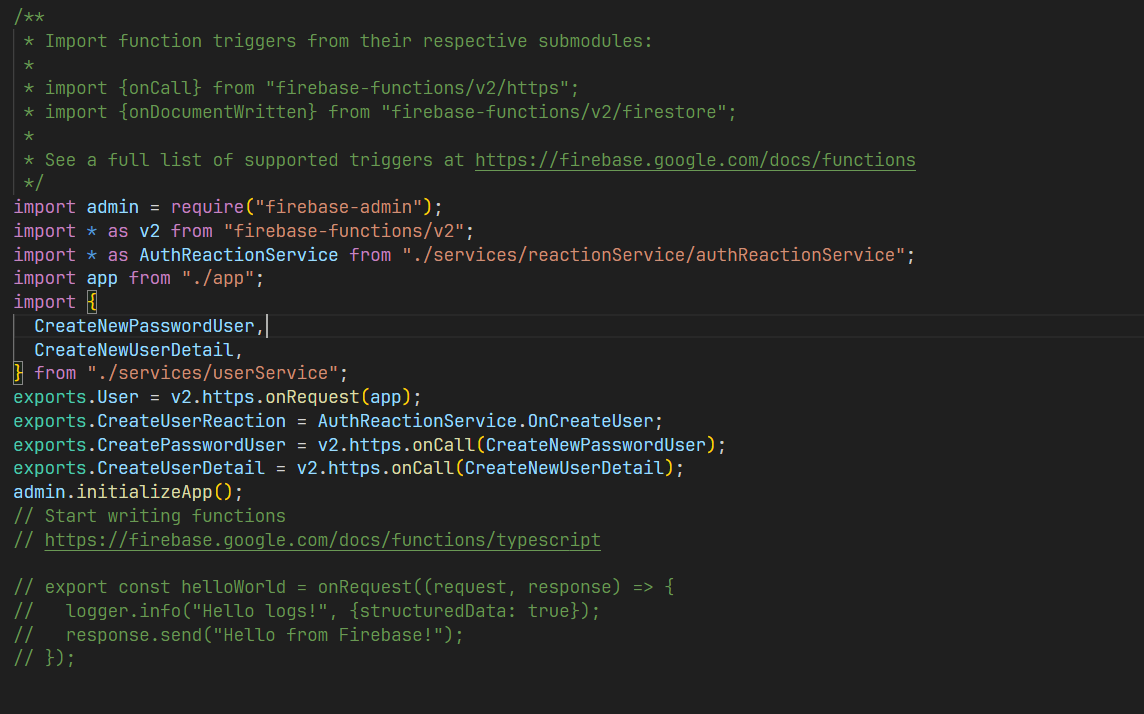
## Tạo tương tác giữa Client và Firebase

1. **Code các function serverless thông qua Firebase Function**

Đối với các Function của Firebase sẽ tồn tại 2 loại Function là HttpRequest Function và Reaction Function:

* HttpRequest Function là loại function mà người dùng chỉ cần tương tác thông qua request response như cách truyền thống tùy nhiên việc quản lý và mở rộng khi call các function đều được Firebase hỗ trợ thay vì tạo một server truyền thống
* Reaction Function là loại Function trigger mà khi người dùng tương tác với một trong các service của Firebase thì function này sẽ làm việc nếu như trường hợp 1 trigger nào đó phù hợp với yêu cầu diễn ra: Ví dụ khi ta up ảnh lên Firebase Storage thì Reaction Function sẽ chỉnh sửa các hình ảnh đó ở một độ phân giải phù hợp.

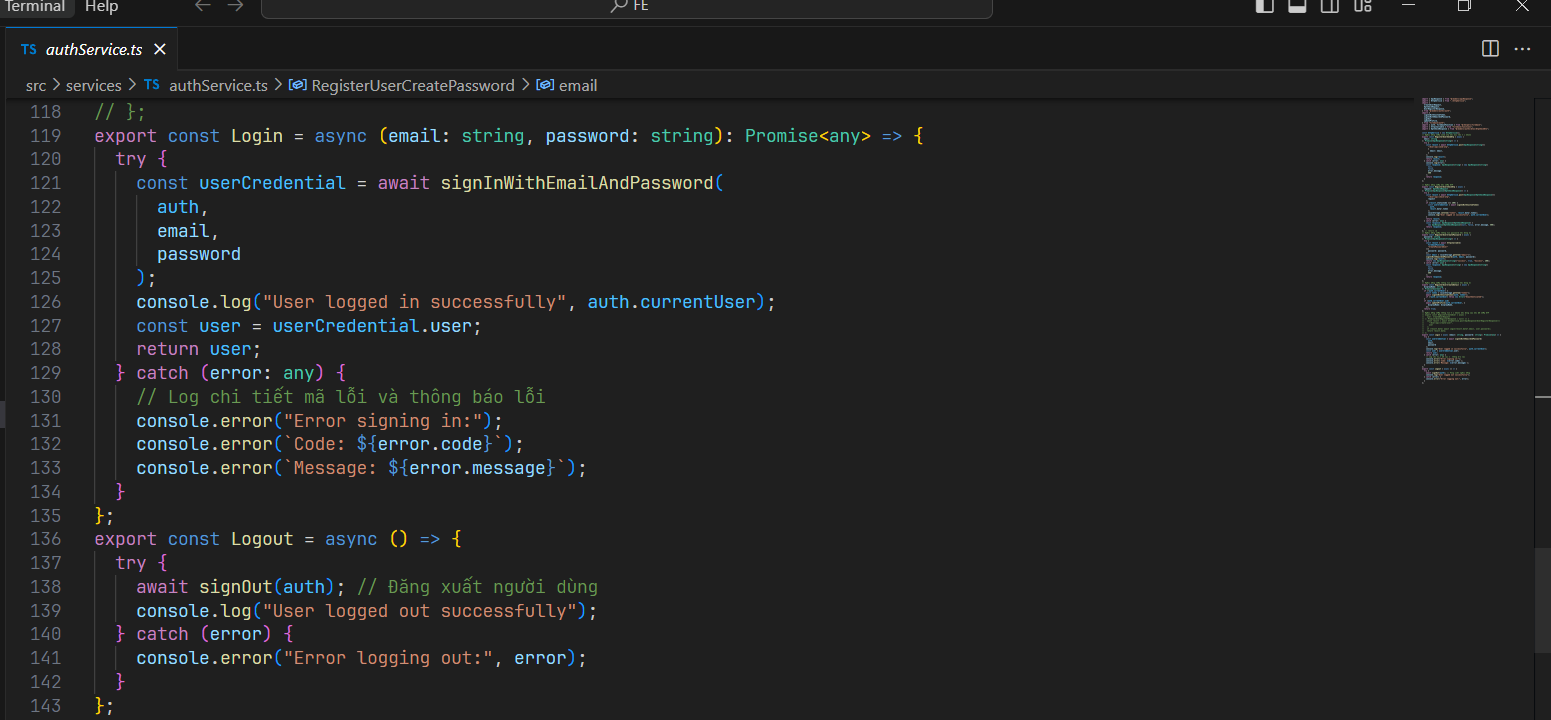
Với Firebase Function ta chỉ cần export các function cần thiết và phù hợp sau đó Firebase Function sẽ tự setup và build nó thay vì cần deploy như server truyền thống.



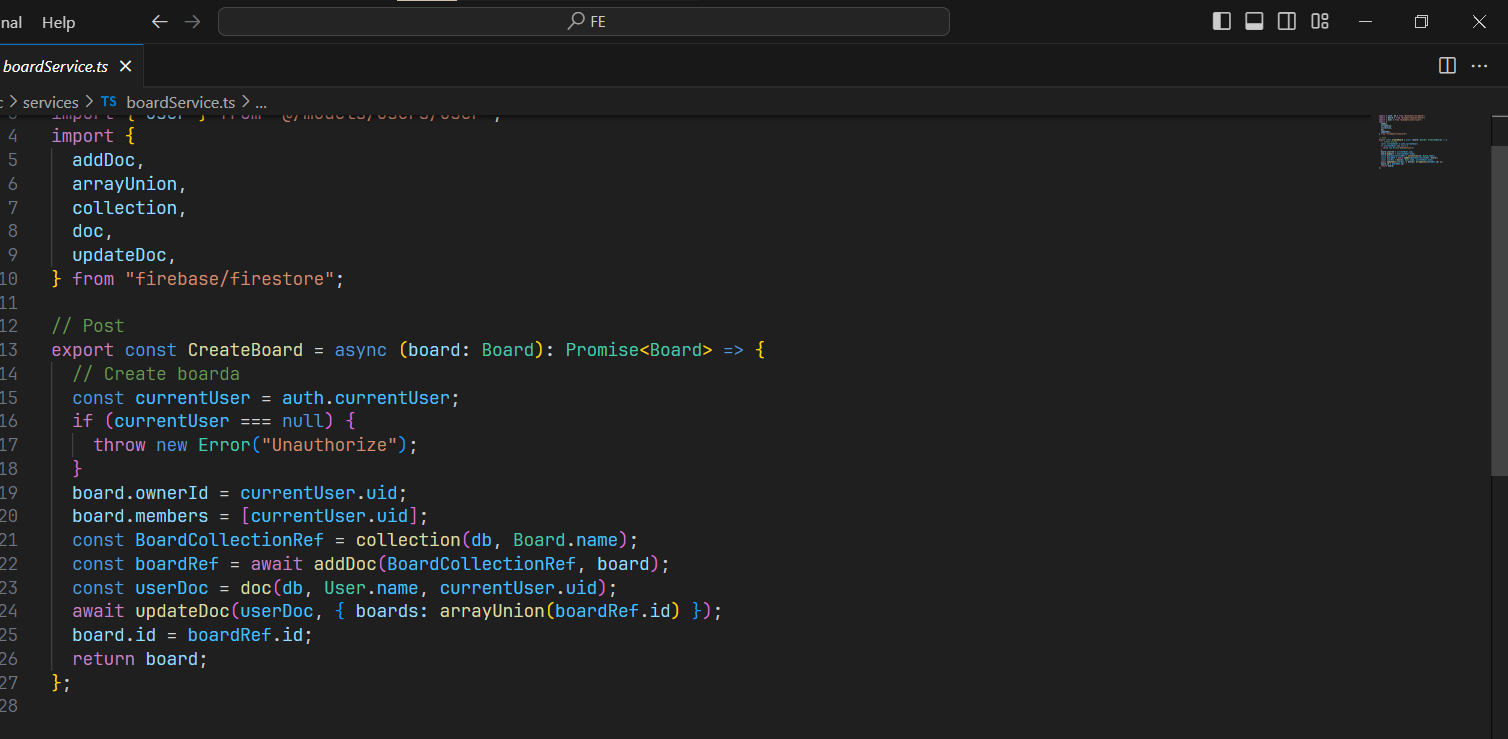
1. **Web Client trực tiếp tương tác với Firebase Service mà không cần thông qua server**

* Khi làm việc với Serverless Service thì người dùng ở phía web client không cần tương tác với phía backend mà có thể tương tác và thay đổi dữ liệu trực tiếp thông qua Firebase

Ở đây với các service ở phía front end có thể thực hiện các việc truy vấn hoặc đăng nhập dựa trên các hàm đã được hỗ trợ tốt từ phía Firebase mà ta không cần implement lại .



*Tương tác với Firebase Authentication*



*Tương tác dữ liệu trực tiếp với Firestore*

**Ưu điểm:**

* **Tương tác realtime:** Việc tương tác trực tiếp với dữ liệu của Firebase giúp cho phía client nhận biết sự thay đổi dữ liệu một cách nhanh nhất và thay đổi dữ liệu một cách nhanh chóng.
* **Triển khai ứng dụng nhanh chóng:** Có thể triển khai một cách nhanh chóng vì nhờ việc truy cập trực tiếp và sử dụng các service khác của firebase từ phía client giúp loại bỏ nhu cầu viết và duy trì backend server. Điều này giảm đáng kể thời gian phát triển và chi phí bảo trì server.
* **Quản lý và mở rộng quy mô:** Do hoạt động dưới dạng Serverless Service nên ta không cần quan tâm đến việc mở rộng quy mô ứng dụng dù ứng dụng có mở rộng ra như nào nữa.

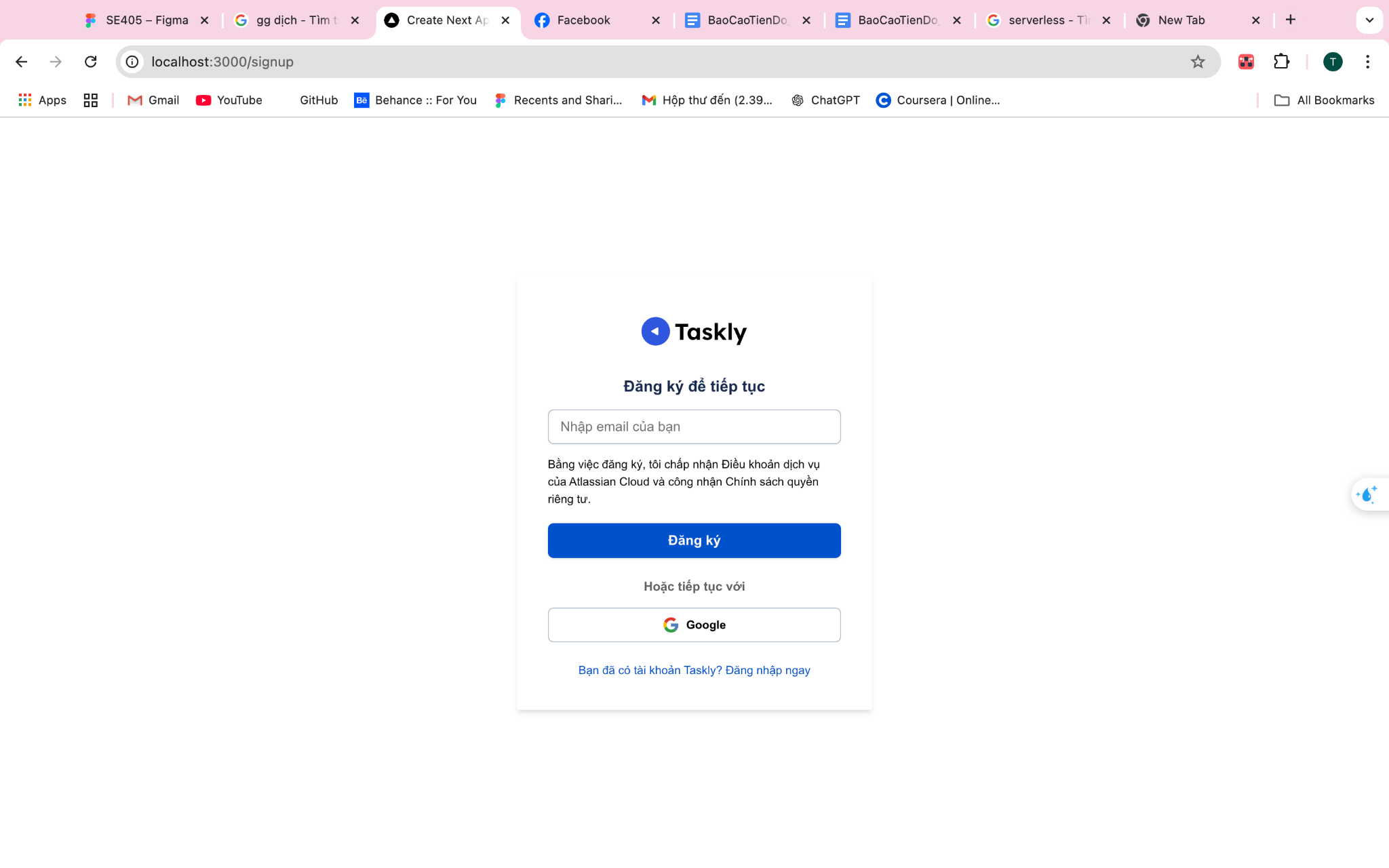
**Nhược điểm:**

* **Về bảo mật:** Việc tương tác trực tiếp từ client với cơ sở dữ liệu có thể làm lộ một số cấu trúc hoặc thông tin về database. Làm gây ra vấn đề bảo mật cho hệ thống nếu như không được quản lý một cách chặt chẽ.
* **Việc tái sử dụng kém:** Tuy việc xây dựng với firebase có thể triển khai ứng dụng một cách nhanh chóng nhưng nếu ta mở rộng ứng dụng trên các nền tảng khác nhau như web, android app, ios app thì ta sẽ tốn thời gian implement lại các phương thức từ đó gây tốn nhiều chi phí.

## Giao diện Người dùng

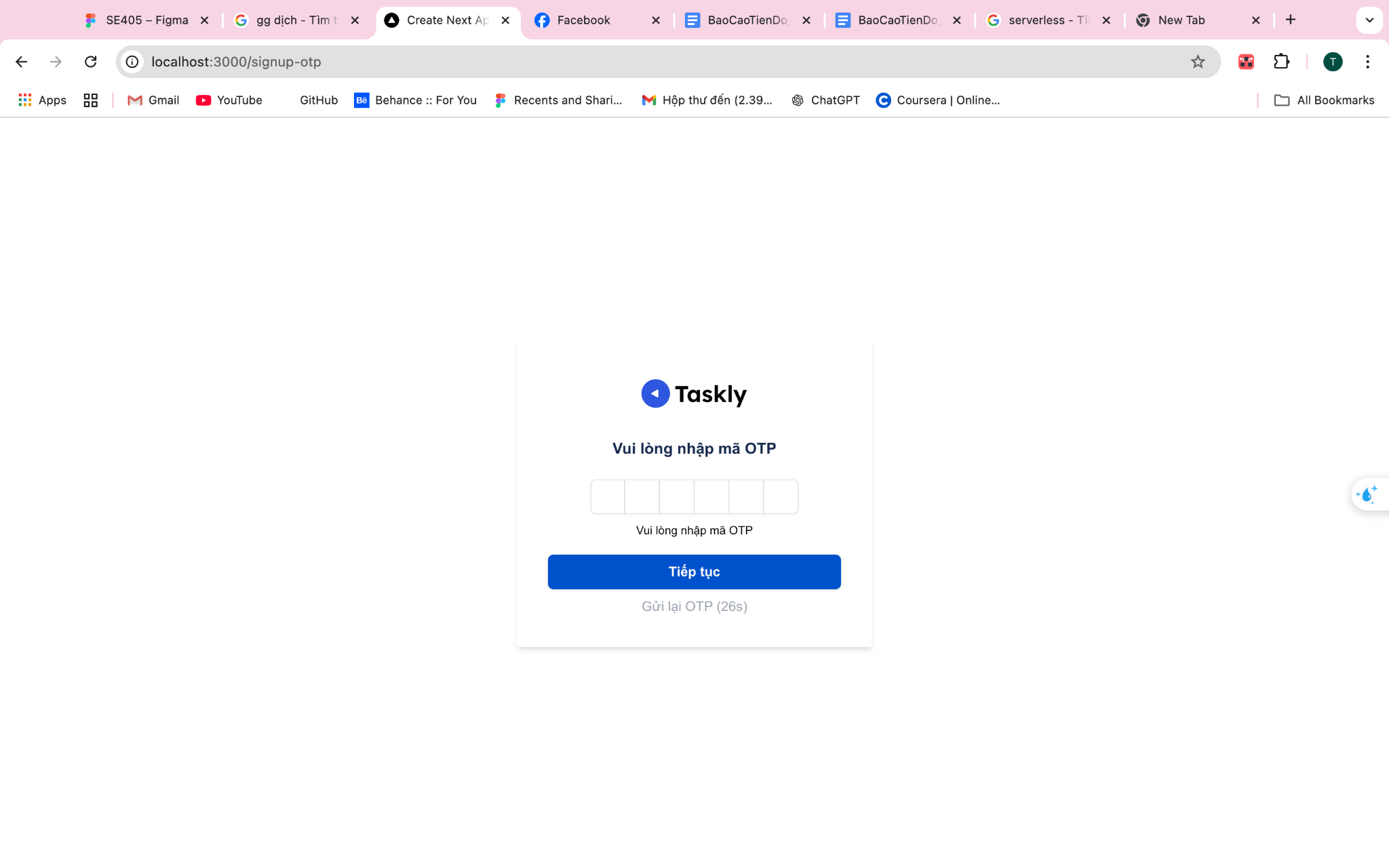
* + 1. **Tạo account**

Bạn có thể nhập email hoặc chọn đăng nhập bằng Google

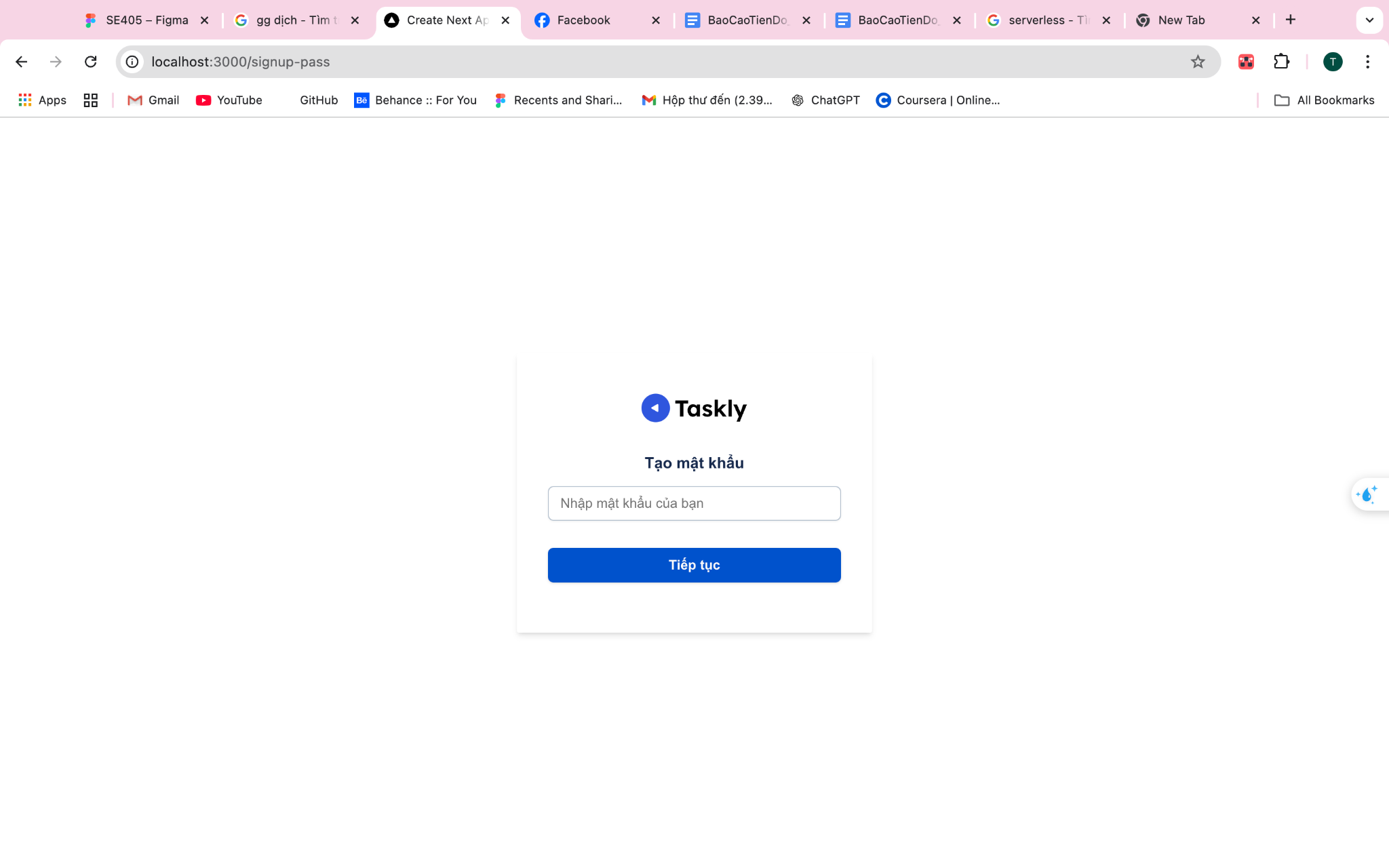


* + 1. **Nhập mã OTP**

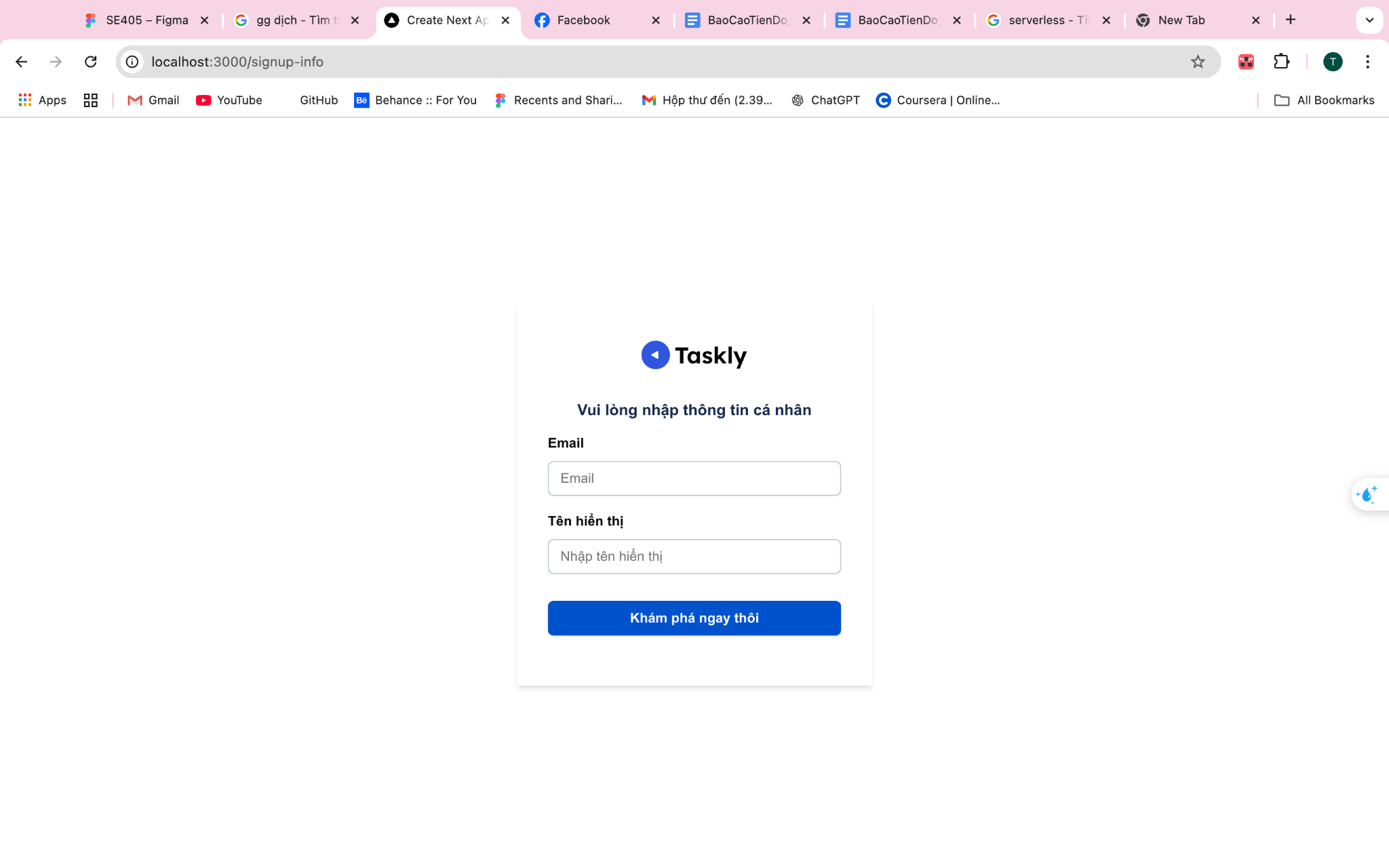
Nhập mã OTP để xác thực email



* + 1. **Tạo mật khẩu**

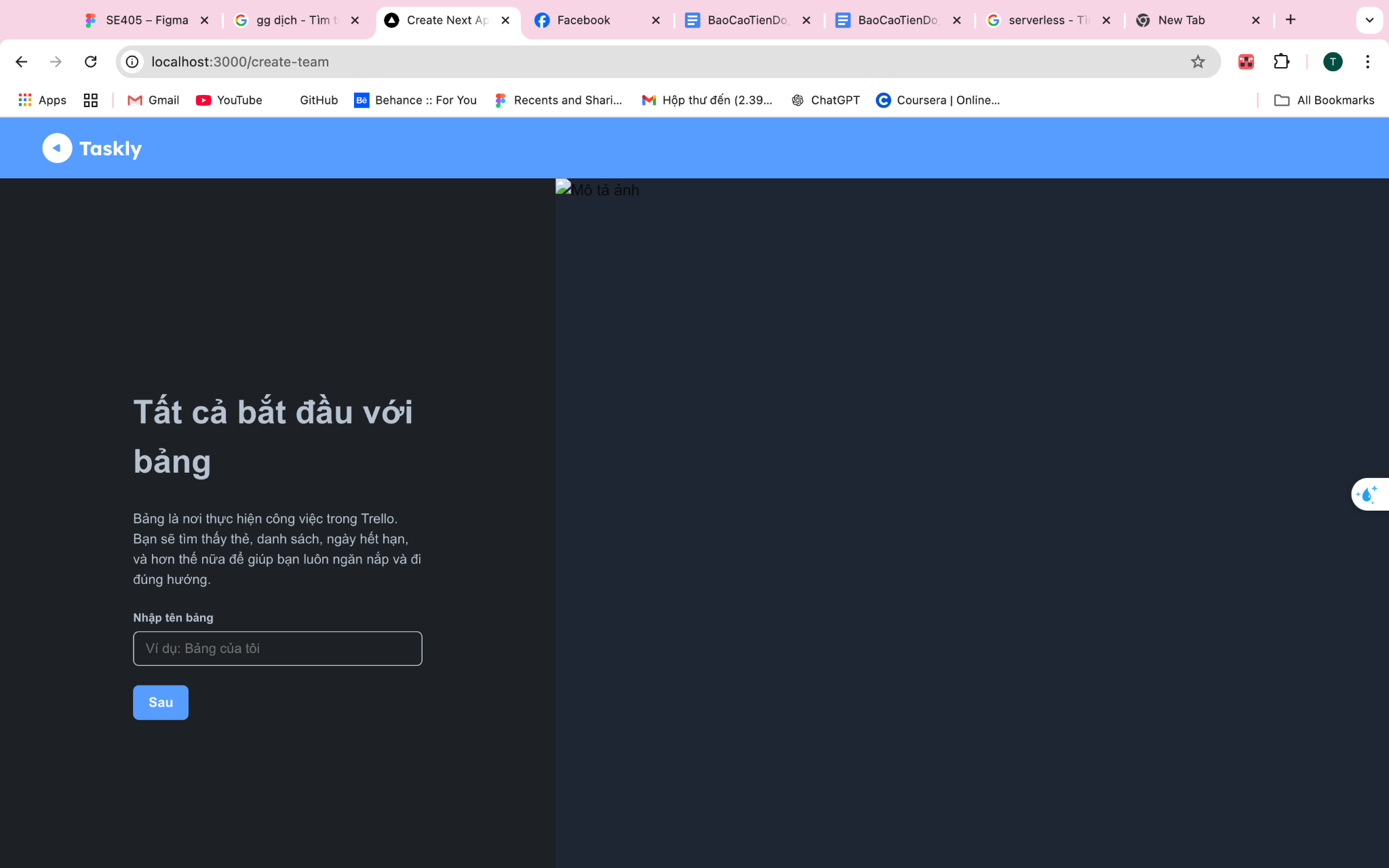


* + 1. **Nhập thông tin chi tiết**

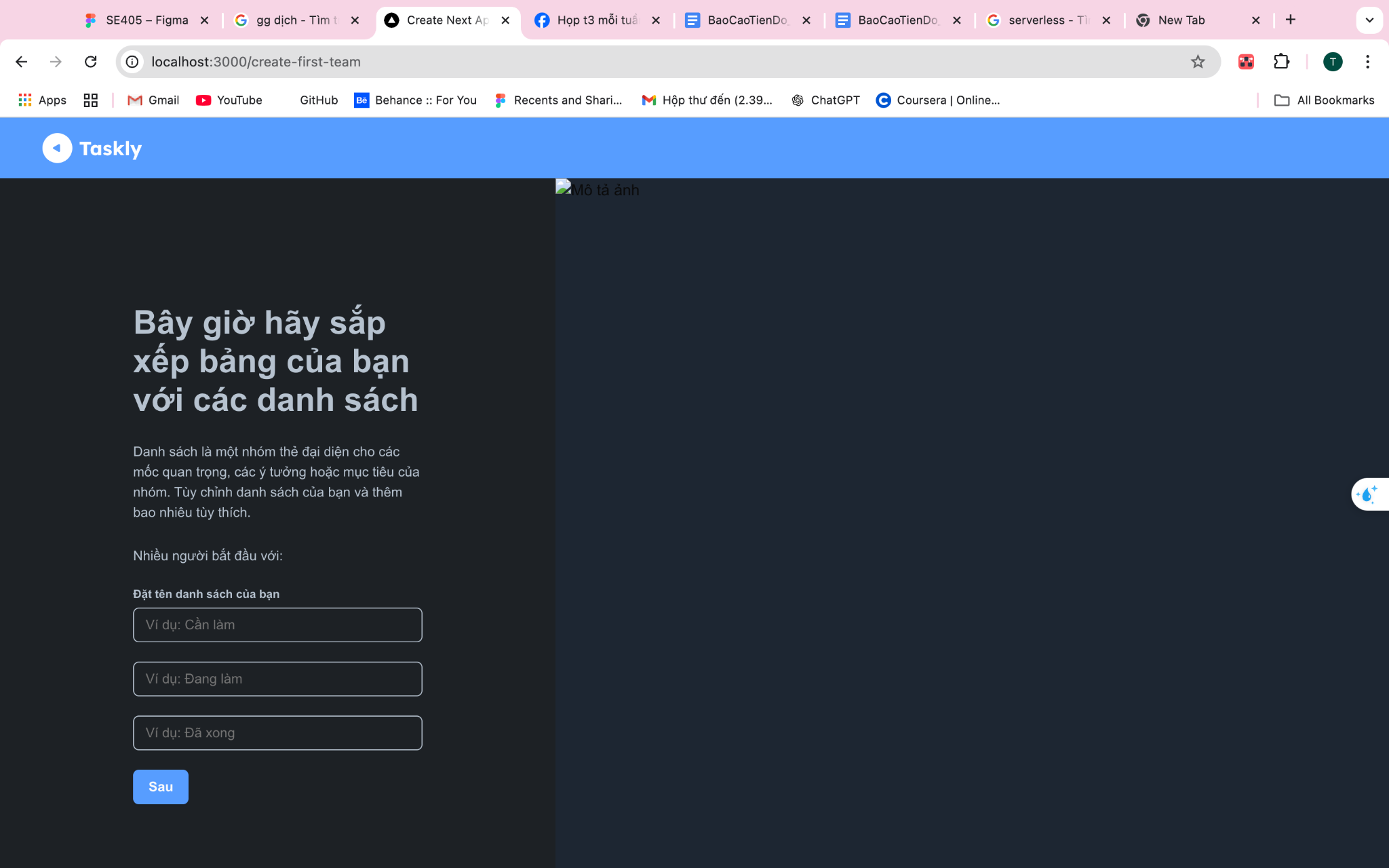


* + 1. **Tạo bảng**

Để bắt đầu sử dụng, bạn hãy tạo bảng

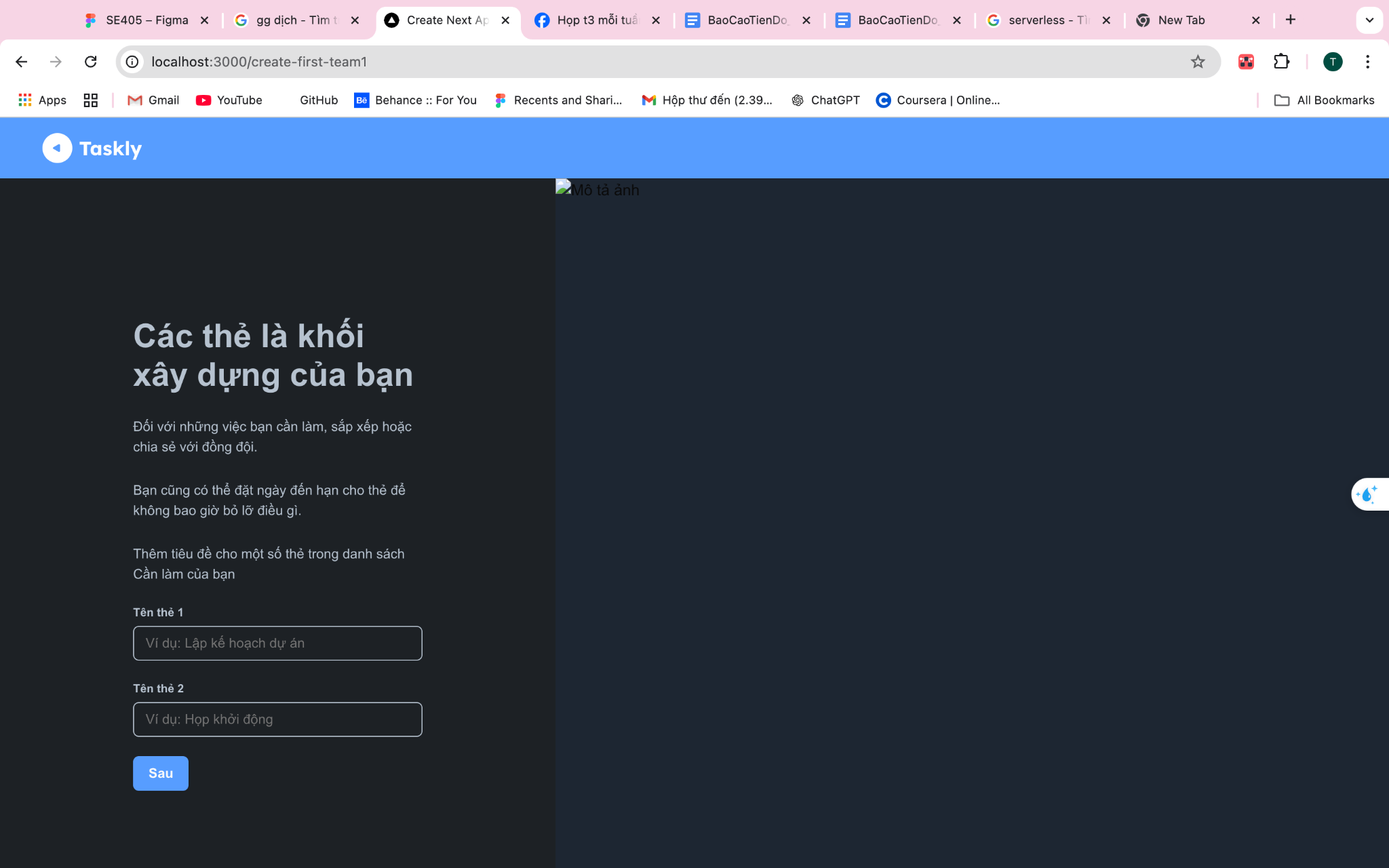


* + 1. **Đặt tên và tạo các task**



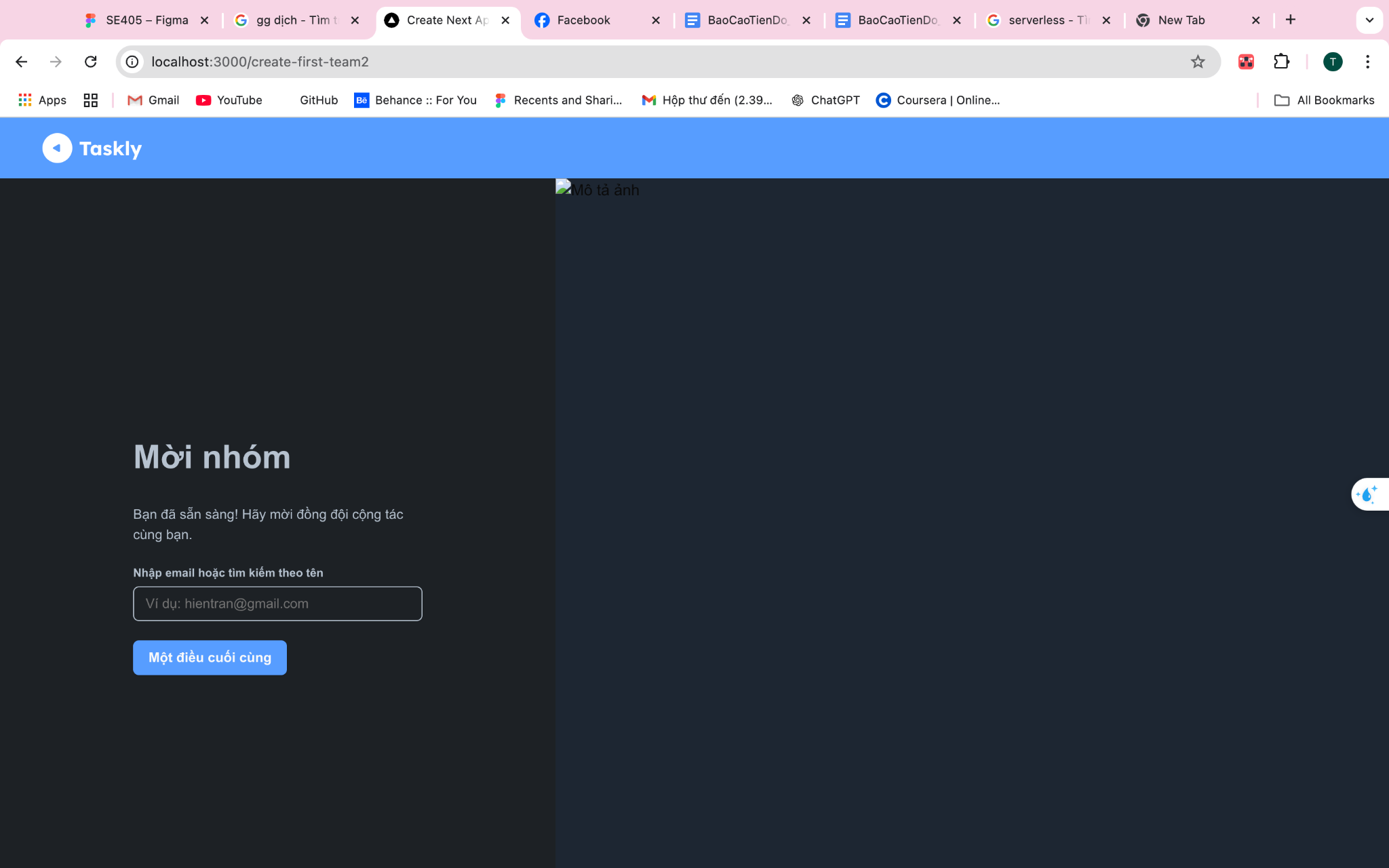
* + 1. **Đặt tên và tạo các thẻ trong card task**

Ví dụ như: SEM-12 code home page



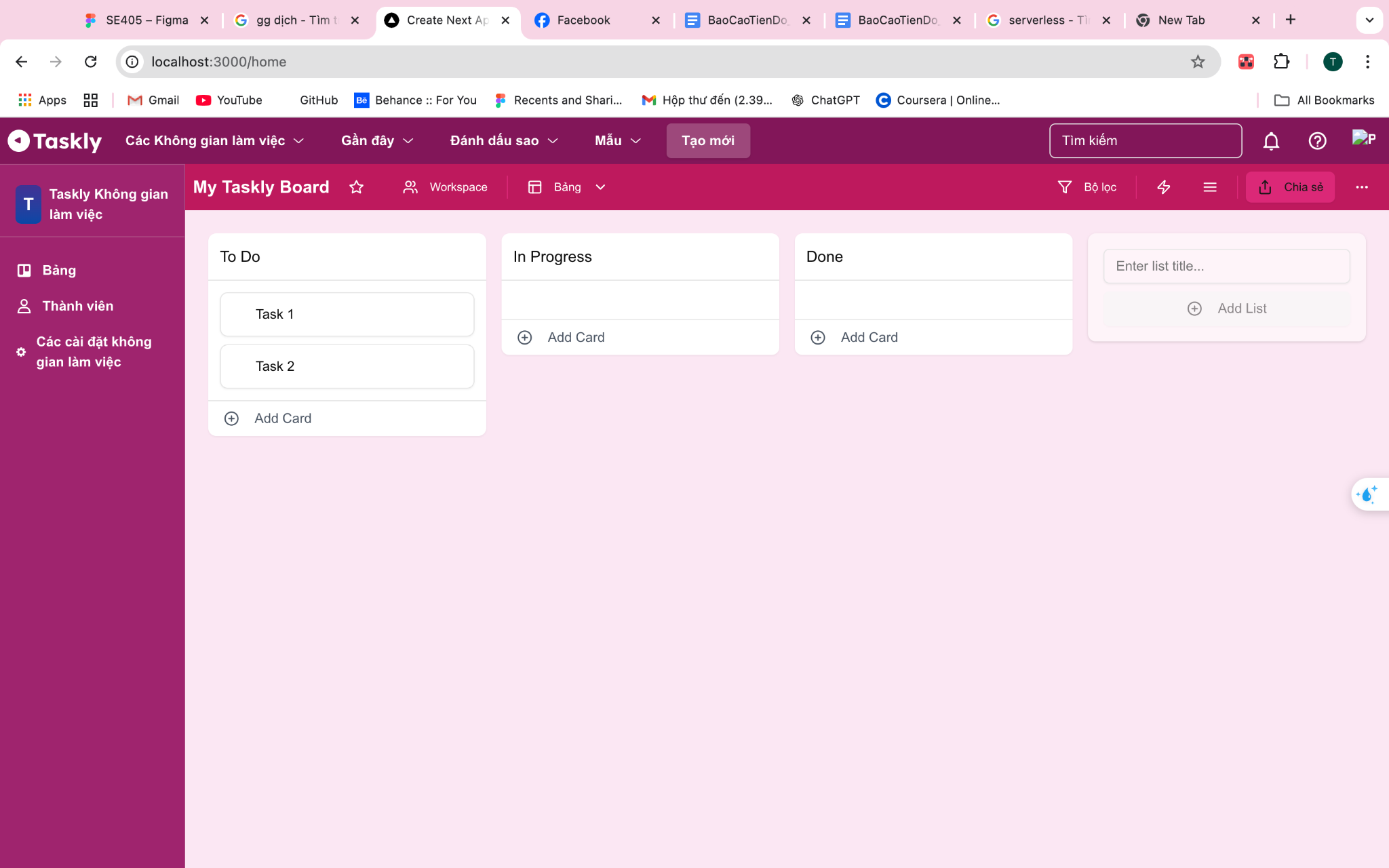
* + 1. **Mời nhóm người dùng**

Bạn có thể mời bạn bè join vào group Taskly để phân chia công việc



* + 1. **Trang home page**

Tại đây, chúng ta có thể create task, xoá task, chuyển vị trí task), chúng ta cũng có thể làm như vậy với các cột



## 

## Đánh giá việc triển khai ứng dụng web trên việc sử dụng dịch vụ Firebase Service

**Thuận lợi:**

* Việc triển khai ứng dụng với Firebase ban đầu khá ổn trong việc sử dụng và implement các phương thức mà nhóm mong muốn.
* Việc sử dụng các phương thức Firebase cung cấp dễ dàng và dễ dàng xử lí các tác vụ cơ bản.
* Việc sử dụng Firestore giúp việc phát hiện và xử lí realtime các đối tượng một cách dễ dàng hơn.
* Nhờ có Firebase Authentication mà việc kiểm tra và xác thực user và lưu trữ thông tin đăng nhập đã được Firebase cung cấp hỗ trợ nên việc xử lí các luồng kiểm tra Authentication một cách dễ dàng.

**Khó khăn:**

* Có thể gây tốn chi phí ít nhiều khi deploy và sử dụng các firebase function ở trên server chính của firebase tuy nhiên hiện tại nhóm chỉ triển khai ở mức local và chạy emulator nên vẫn chưa ảnh hưởng lắm.
* Do cấu trúc CSDL của Firestore khá khác với các kiểu cấu trúc CSDL mà nhóm đã quen sử dụng nên khi tiếp xúc với cấu trúc mới vẫn chưa tìm ra cách truy vấn nào có tốc độ thực thi tốt nhất cho hệ thống.
* Còn phát sinh một số lỗi kết nối khi kết nối từ client đến phía firebase emulators trong việc thực thi và tương tác với máy ảo.

## Kết luận

# 

## Kết quả đạt được

* Việc triển khai ứng dụng còn khá chậm so với kế hoạch dự tính ban đầu tuy nhiên vẫn đang ở tiến độ hợp lý.
* Tìm hiểu và nắm cách thức hoạt động của Firebase từ đó có thể triển khai ứng dụng chạy trên Firebase và kết nối với ứng dụng web của nhóm.
* Xây dựng và triển khai web dựa trên sự phát triển của Firebase

## 

## Bài học kinh nghiệm

* Từ việc tìm hiểu và ứng dụng Firebase Service Serverless vào trong ứng dụng thực tế nhóm nhận ra việc xây dựng ứng dụng của Firebase giúp dễ dàng và nhanh chóng xây dựng ứng dụng tuy nhiên chỉ phù hợp với các ứng dụng vừa và nhỏ vì các service của Firebase cung cấp các tính năng cơ bản, truy vấn và xử lí dữ liệu ở một mức nào đó.
* Cách tốt nhất để ứng dụng Firebase Service là chỉ nên ứng dụng ở một hoặc một số tính năng nào đó cần việc hỗ trợ đặc thù từ các service của Firebase ở mức độ cơ bản và dễ dàng.

## 

## Hướng phát triển

* Ứng dụng các tính năng khác của Firebase như FCM để thông báo cũng như nhận thông báo các task trong quá trình hoạt động.
* Sử dụng Firebase Analytics để theo dõi hiệu suất task thông qua việc sử dụng Firebase Analytics theo dõi hành vi người dùng, từ đó giúp nhóm phát triển cải thiện trải nghiệm quản lý công việc. Chúng ta có thể theo dõi các sự kiện như khi người dùng tạo task mới, hoàn thành task, hoặc thay đổi trạng thái task. Điều này sẽ giúp chúng ta hiểu rõ hơn về cách người dùng tương tác với ứng dụng và tối ưu hóa tính năng.TÀI LIỆU THAM KHẢO

1. **Firebase Documentation. "Build with Firebase." Firebase.** <https://firebase.google.com/docs/build>.
2. **Talbot, Mike. "Creating a Serverless Application with Firebase." Dev.to, 26 tháng 10, 2020.** <https://dev.to/miketalbot/creating-a-serverless-application-with-firebase-5d31>.
3. **"Tìm hiểu về Serverless (FaaS) - Deploy Functions lên Firebase Cloud Functions." Viblo, 5 tháng 5, 2021**. <https://viblo.asia/p/tim-hieu-ve-serverless-faas-deploy-functions-len-firebase-cloud-functions-yMnKMw0El7P>.
4. **Cloudflare. "Backend-as-a-Service (BaaS)." Cloudflare, 2024.** <https://www.cloudflare.com/learning/serverless/glossary/backend-as-a-service-baas/>.
5. **Cloudflare. "Serverless vs PaaS." Cloudflare, 2024.** <https://www.cloudflare.com/learning/serverless/glossary/serverless-vs-paas/>.

[Link github](https://github.com/leesoonduck3009/seminar-cnpm)