GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN IN PUBLICA COMMODA SEIT 1737

HPS

Seminar Report

Comparing different BaaS solutions and their performance

João Pedro Soares

MatrNr: 23346583

Supervisor: Timon Vogt

Georg-August-Universität Göttingen
Institute of Computer Science

March 24, 2024

Abstract

In this report, we go over the main differences between Supabase and Pocketbase and the advantages of using these tools over creating your own backend or using a proprietary solution like Firebase. Developing an app with either of these platforms comes with its specific challenges, which is only natural since the technologies behind them differ. In our report, we determined that the performance between the two is different in some situations, but ultimately, for the purpose of developing simple applications, the differences are negligible. Both are very simple to set up and use but Supabase has some more advanced features(better data importing tools, edge functions, and vector embeddings for similarity search) that can be useful in bigger projects. Supabase is the most popular of the two platforms, but they are both very effective.

Statement on the usage of ChatGPT and similar tools in the context of examinations

In this work I have used ChatGPT or a similar AI-system as follows:

- ☐ Not at all
- ☐ In brainstorming
- ☐ In the creation of the outline
- ☐ To create individual passages, altogether to the extent of 0% of the whole text
- ☑ For proofreading
- ☑ Other, namely: In the creation of parts of the code for the web development project, ChatGpt was used to help create better looking UI for the React application. It was also used for creating SQL queries for creating and updating tables in the databases.

I assure that I have stated all uses in full.
Missing or incorrect information will be considered as an attempt to cheat.

Contents

List of Tables

List of Figures

List of Listings

List of Abbreviations

GWDG Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

VM Virtual Machine

Baas Backend as a Service

1 Introduction

The software industry has fully embraced the cloud and made it the norm for developing web applications. With services provided by companies such as Amazon, Google, Microsoft, and others, the process of getting your application to your user base has never been more convenient. However, there are still many other individuals and companies developing open-source tools for web development. This scientific report embarks on a journey to explore two open-source tools: Supabase and Pocketbase, comparing their unique features and advantages and how they stand apart from their proprietary counterparts.

Firebase[Goo20] is a series of products and solutions that take advantage of Google's cloud resources to provide developers with tools for developing their applications, namely authentication, storage, databases, analytics, and much more, all in one platform.

Supabase[Sup20a] is an open-source alternative to Firebase. Supabase sets itself apart by offering a suite of services for web development, which include a Hosted Postgres Database, robust Authentication and Authorization mechanisms, auto-generated APIs, Edge Functions, and File Storage. What distinguishes Supabase is its open-source nature, which grants developers the freedom to deploy it locally or on servers they own, in contrast to Firebase, which is intrinsically tied to the Google Cloud Platform.

Another noteworthy contender to Firebase is Pocketbase[Poc23b], another open-source Backend as a Service (Baas) solution. Pocketbase, unlike Supabase, does not have its proprietary cloud service, necessitating self-deployment. We will examine the deployment process of both solutions and compare their user-friendliness to provide a comprehensive assessment of the two platforms.

The primary objective of this report is to construct a small-scale web application, implementing Supabase as its primary backend technology, alongside React and Express.js. Subsequently, we will assess the performance of the deployed application by hosting it on both Supabase's cloud service and the cloud Virtual Machine (VM)s provided by Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG) and doing the same for Pocketbase(comparing local deployment with deploying the services on a remote server). This comparative analysis aims to shed light on the practical implications of using these tools, with a particular emphasis on ease of use and performance.

By the end of this report, readers will have gained valuable insights into the capabilities of Supabase, as well as a comparative understanding of its deployment challenges and performance compared to Pocketbase. In an age where open-source solutions are rapidly reshaping the web hosting landscape, this exploration promises to contribute to the knowledge base of developers and technology enthusiasts alike.

1.1 Contributions

In this report, we built a React application and two different backends, each using one of the different Baas solutions, and deployed locally(using different machines that were available) and also remotely in the cloud. As we developed these applications we performed several tests. We also came up with what we thought was a good basis for evaluation in creating two hypothetical developers that would benefit from the use of these tools and understanding how the needs of these users could lead them to choose one of the tools over the other.

1.2 Report outline

This report undertakes a comprehensive comparison of three prominent backend service providers: Firebase, Supabase, and Pocketbase. In the state-of-the-art section, the report delves into a thorough examination of each platform, juxtaposing their features, and performance benchmarks, and exploring alternatives to Firebase. Following this, the practical comparison section details the practical application of these platforms, from designing the web application and its database to the development process and utilization of Supabase and Pocketbase, including their admin dashboards and self-hosting capabilities. The subsequent section presents the results of the evaluation, covering aspects such as complexity of use, security, scalability, and other pertinent criteria like pricing, support for advanced features such as vector embeddings and similarity search, performance, and community support. Concluding remarks synthesize the findings, offering insights and recommendations. Through this structured analysis, readers gain a nuanced understanding of the strengths and limitations of each platform, aiding informed decision-making in backend service selection.

2 State-of-the-art

2.1 Comparing Firebase vs Supabase vs Pocketbase

As was mentioned in the Introduction, Supabase is marketed as an open-source alternative to Firebase. Supabase has been compared to Firebase in [Sup20f] and [Jag23]. In the table in Figure 1 we can observe the main differences between the two that were pointed out in these articles and also the features from Pocketbase.

Figure 1: Firebase vs Supabase vs Pocketbase

	Firebase	Supabase	Pocketbase
Database technology	Firestore(NoSQL)	PostgreSQL(Relational)	SQLite (Relational)
Data Import	Integrations with Google products	Simple .sql file import option	No built-in importing solution
Pricing	Free tier with limited features	Open-source and free to use	Open-source and free to use
Real-time capabilities	Excellent real-time feautures	Real-time events and streaming changes	Real-time subscriptions
Custom Functions	Cloud functions for backend logic	PostgreSQL-based PL/pgSQL functions	No Custom Cloud Functions. Extendable with Javascript or Go
Authentication	Built-in authentication service	JSON web token authentication	Uses JWT for authentication
Data Migration	Limited data migration options	Powerful data migration tools	Limited data migration options
Extensions	Wide range of pre built extensions	Marketplace with 3rd party integrations	Currently no pre-built extensions
Community Support	Strong community and resources	Growing community support	Growing community support
Learning Curve	Generally user-friendly	Requires familiarity with PostgreSQL	Requires familiarity with SQLite
Scalability and Security	Relatively scalable, good security	Scalable with robust PostgreSQL security	Can only scale vertically. Allows for creating API rules to limit access to records
Hosting	Firebase hosting	Self-hosted with Supabase	Self-hosted
Flexibility	Flexible, document-based storage	More structured with sql capabilities	More structured with SQL capabilities

The most significant differences between Firebase and its open-source counter-parts are:

- you can use them for free.
- they use relational databases instead of noSQL, which makes querying and retrieving information faster and allows for more complex queries.
- they have smaller communities but growing communities(fewer extensions and learning resources).

2.2 Performance benchmarks

There are benchmarks made available by Supabase on Github[Sup20b] comparing it to Firebase. Another benchmark by Supabase [Sup20c] compares its performance while running on different machines: AWS t3a.micro (vCPU 2, RAM 1GB), Fly.io micro-1x (shared vCPU, RAM 128MB) and a Fly.io micro-2x (shared vCPU, RAM 512MB).

The most recent results performed by their developer team show that Supabase outperforms Firebase by up to 4x on the number of reads per second, and 3.1x on writes per second.

Pocketbase also has Github repository [Poc23c] with benchmarks and test made on different machines comparing their respective performance: Hetzner CAX11 (2vCPU ARM64, 4GB RAM), Hetzner CAX41 (16vCPU ARM64, 32GB RAM), Hop.io (1vCPU, 512MB RAM) and Fly.io (1VCPU, 256MB RAM).

After analyzing these benchmarks, both Hetzner CAX11(machine with the best specifications) and Fly.io(machine with the worst specifications) demonstrate competitive performance in various scenarios(results of each one of the test didn't vary much despite the differences in available resources). The choice between them may depend on factors such as specific use cases, pricing, and additional features offered by each provider.

2.3 Other alternatives to Firebase

Other tools in the world of BaaS solutions that are worth exploring are:

- Amplify from AWS - build full-stack web and mobile apps in hours. Easy to start, easy to scale.
- NHost - an open-source backend and development platform that enables developers to build and scale their web and mobile apps.
- Appwrite - an open-source platform for building applications at any scale, using your preferred programming languages and tools.
- Fastgen - a low-code API and workflow builder with an integrated Postgres DB.

It's important to keep in mind the support and community around each tool, as any developer would not want to invest time in using a tool just to find a problem/bug that won't be fixed in the future. It's crucial, when choosing a tool for your next big project to understand the capabilities and the available community resources. Choosing the right BaaS solution is important as it can impact your project in the long run.

3 Practical comparison

3.1 Designing the web application

In this report, we wanted to review these platforms from a broad perspective, but it's important to keep in mind all the different types of developers. Engineers, designers, and product managers utilize "user personas" to understand their user needs, prioritize features, and design user-friendly products that cater to specific user requirements. We developed two "user personas" that are trying to create the same application, using the same data.

- **User A** - is an inexperienced developer who is creating a simple project for a Hackathon. This person values ease of use and development speed. However, the solution must scale in case the project is successful.
- **User B** - is an experienced developer who can build the back-end from the ground up but wants to find a possible workaround for some of the complicated issues that come with back-end development, such as security, scalability, performance, API development, and authentication.

To evaluate tools for the different user personas described, you'll want to consider criteria that align with their specific needs and preferences. Here are some criteria tailored to each user persona:

For User A:

- Ease of Use: Intuitiveness of the platform/interface.
- Development Speed: How quickly can they create the required reports and dashboards without extensive coding or technical skills?
- Customization: Ability to easily edit the data.
- Visualization Options: Variety and quality of visualization tools available. Will help in communicating the project's goals to judges or stakeholders.

For User B:

- Security: Capabilities for implementing robust security measures.
- Scalability: Ability to handle growth and increased data loads.
- Performance: Tools and features to enhance application performance.
- Flexibility: Customizability and flexibility in implementing complex solutions.

To test these solutions, creating an example web application is necessary. The goal was to create an app that displayed data from a relational database in a user-friendly interface. This project uses React, Tailwind CSS, and a database and it's a comparison shopping website for cosmetic products that uses content creators as the main source for product reviews. In Figure 4, we can see what the products page looks like. Users can go on the website and search for products by category and, when they click on a product, they can see a list of content creators who recommended that product and the video they made about that product. Additionally, a user can follow a content creator and see all the videos they made and all their product recommendations.

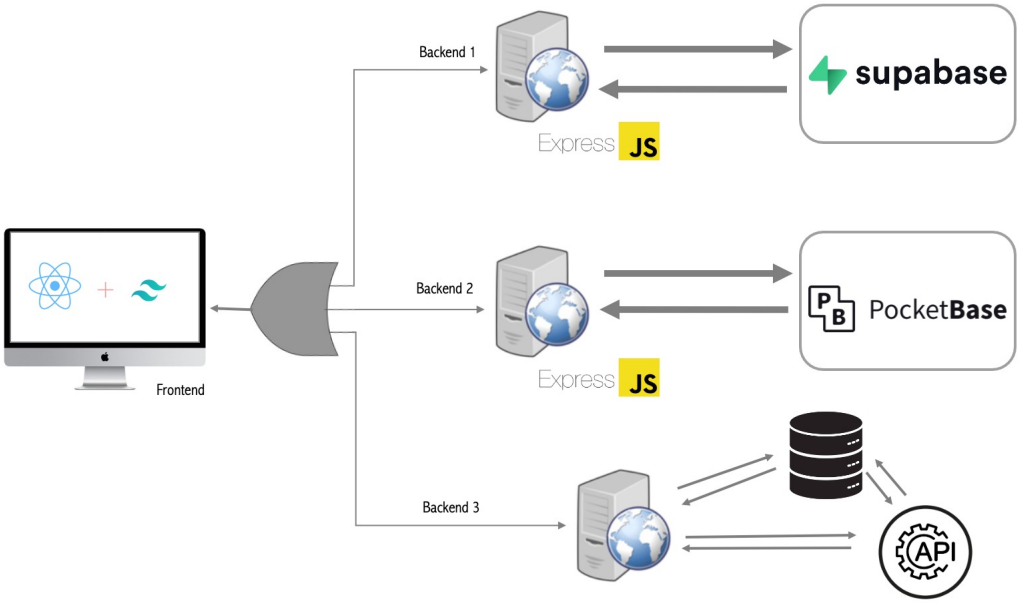
Essentially, we will develop the same application in two different ways:

- Using Supabase as the backend
- Using Pocketbase as the backend

To do this we needed to develop the code in a way that we could reuse as much of the software as possible. Therefore, the front end should make a call to an API that handles all the interactions with the database, authentication, etc. That way we can build two different APIs with the same endpoint and parameters(so the code on the front end

stays the same). We included a graph that demonstrates our intention in Figure 2(this graph includes a 3rd backend that would be developed from scratch that we ended up removing from the report due to the added complexity that developing this would bring to the report). This repository includes all the code developed during the research for this report and instructions on how to run/deploy it.

Figure 2: Graph showing App structure



3.1.1 Designing the database

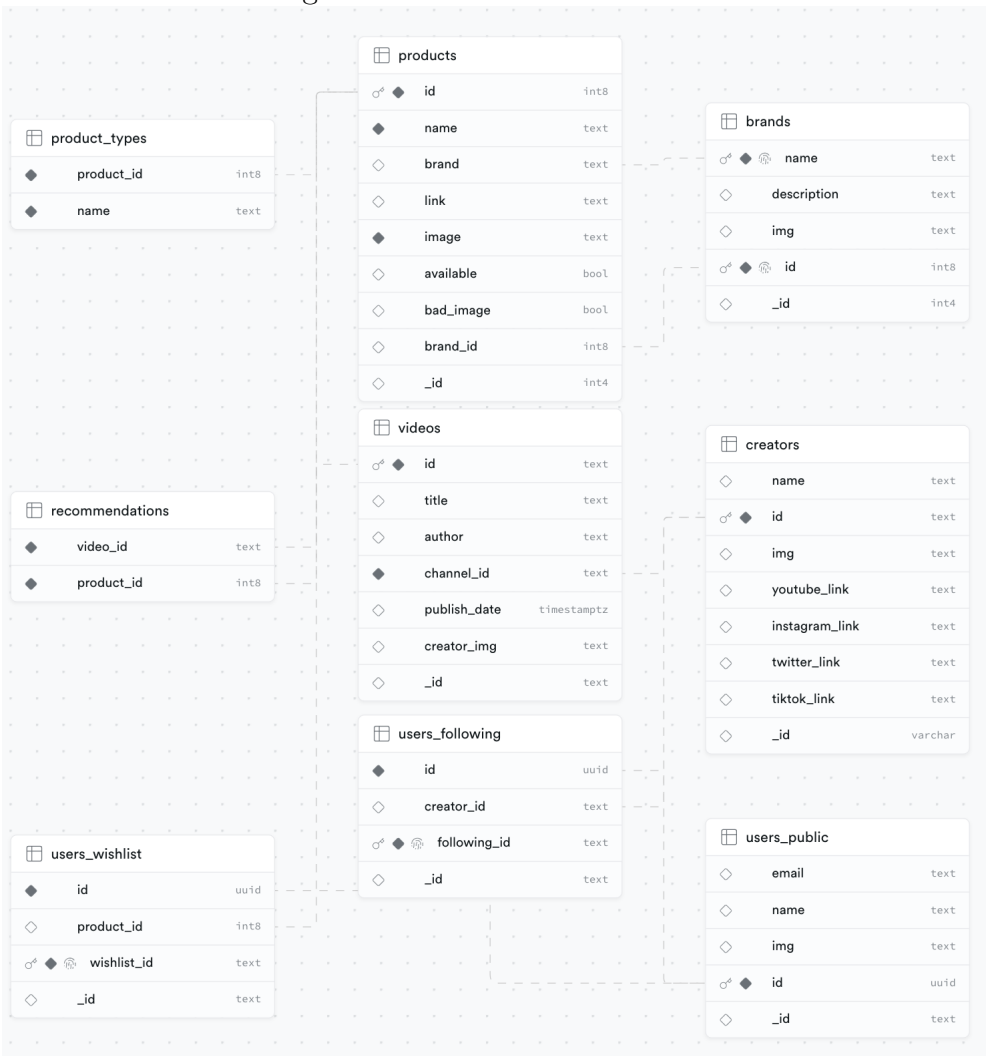
In total, there are 7 tables: **types**, **brands**, **creators**, **products**, **product-types**, **videos**, and **recommendations**. You can understand the different relations between tables in the database schema visualization in Figure 3.

3.2 Developing the application

Before creating the front end, we designed the whole application around the user's actions that we wanted to implement. The user should be able to easily search the app's page and answer questions like:

- What are the available products from brand _ ?
- What are the creators that recommend the product _ ?
- What products does the creator _ recommend?
- What are the Top Creators of the website?
- What are the Top Brands on the website?

Figure 3: DB Schema Visualizer



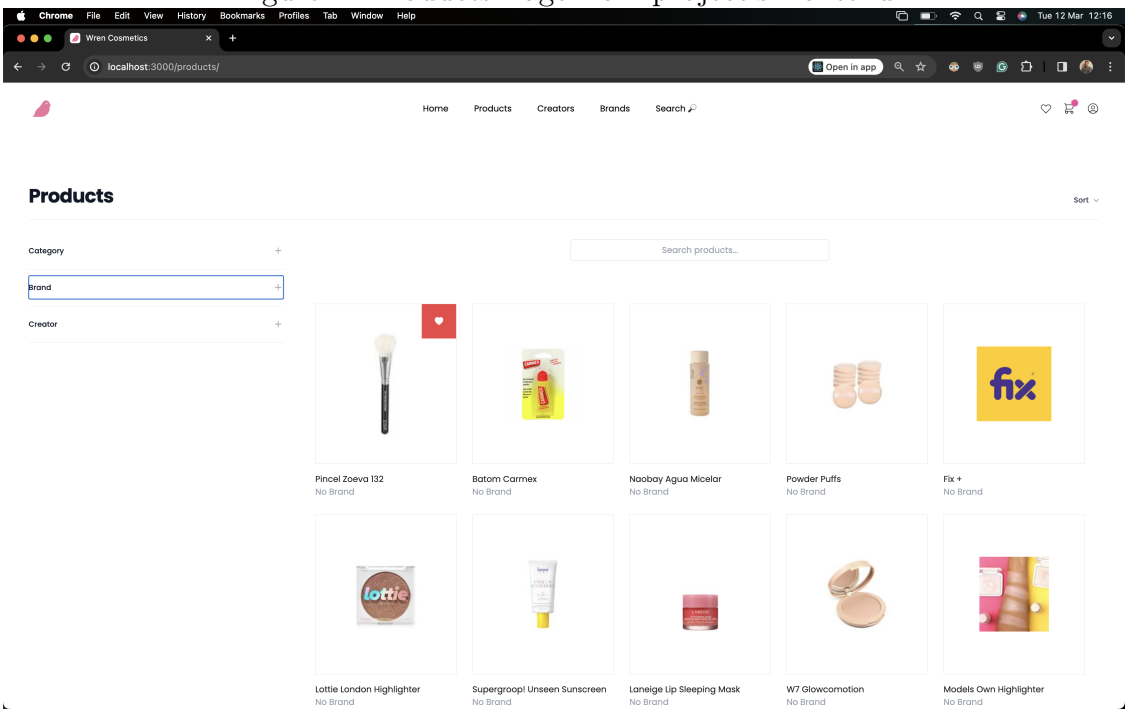
To answer these questions we designed the database and relation between the different tables in a way that optimized the queries that get the answer to those questions. This included the creation of certain "view" tables. A view is a virtual table based on the result set of an SQL statement(basically a table whose structure cannot be changed). We used views to avoid writing complex select queries using the platform's SDKs(writing code for Supabase/Pocketbase that selects all rows from a view table only takes 2 to 3 lines of code) and also because it's more optimal than running the full query every time we want to get this information.

3.3 Using Supabase

Supabase offers a collection of products, which include:

- database - Supabase provides a full Postgres database for every project with Real-time functionality, database backups, extensions, and more.
- authentication - add and manage email and password, passwordless, OAuth, and mobile logins to your project through a suite of identity providers and APIs.

Figure 4: Products Page from project's frontend



- storage - store, organize, transform, and serve large files—fully integrated with your Postgres database with Row Level Security access policies.
- AI and vectors - use Supabase to store and search embedding vectors. Useful for creating search engine-like functionality in your web applications.
- edge functions - globally distributed, server-side functions to execute your code closest to your users for the lowest latency.

These products can be used through Supabase's client libraries, which are available in several different programming languages: Javascript, Flutter, Python, C#, Swift, and Kotlin. Only Javascript and Flutter are officially supported, while the other client libraries are developed by the community.

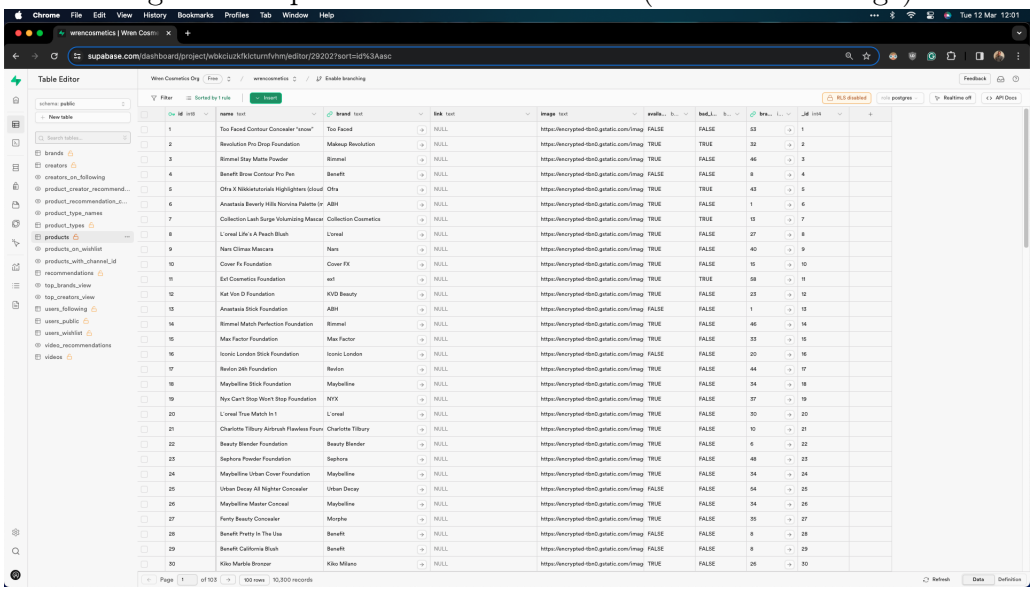
3.3.1 Admin Dashboard

Supabase's dashboard provides insightful statistics and offers tools to simplify data manipulation and the platform's different products. When you open the dashboard and create a new project you will come to the "Project Page". Here there are several tabs accessible from the left-side menu, with the most relevant being the following:

- Table Editor - allows the user to visualize the data in the tables as a spreadsheet and edit single rows. Here you can create new tables and even import data from a CSV or TSV file. This makes it so that the user doesn't need to specify the table's structure(What are its fields? What are the field types?). Here you can also edit a table's fields by changing their type or default value, making them unique, adding foreign key relations, and "check" constraints. We can see what this page looks like from the screenshot in Figure 5.

- SQL Editor - allows the user to run SQL queries on the database. Just like you would do when using Oracle's SQL Developer or PostgreSQL's PGadmin, you can query tables to get quick results, update existing tables(add or delete rows), and create new tables or views. The views you create can then be visualized in the "table editor" tab.
- Database - offers several tools to help administrate your database. Here you can see all your tables through a Schema Visualizer, create roles to manage access control, create triggers or functions, and create data backups.
- Authentication - allows the user to manage the app's authentication rules. Here the user can see all of the app's registered users, select each table's security policies, turn on their desired 3rd party OAuth providers, and create email templates for auth email confirmations.
- Reports - displays statics such as API request, response speed, network traffic, database health(memory usage, swap usage, CPU usage), and query performance(Most time-consuming, Most frequent, Slowest execution time).

Figure 5: Supabase Admin Dashboard(Table Editor Page)



3.3.2 Self-hosting

To self-host Supabase, we can use this docker-compose script, and these detailed instructions. To run Supabase on my GWDG instance, we cloned the repo to our local machine and copied it over to the instance using this command:

```
scp -i hpcsa-course-vm-key.pem /MY_LOCAL_FOLDER/supabase/*
    cloud@141.5.105.244:/home/cloud/supabase
```

Then we connected to the instance and forwarded port 8000 to port 8000 on my machine using the following command:

```
ssh -L 8000:127.0.0.1:8000 -o ServerAliveInterval=60 -i hpcsa-
    course-vm-key.pem cloud@INSTANCE_FLOATING_IP
```

Inside the instance, we navigated to **supabase/docker/** and ran this command:

Listing 1: Bash code for hosting Supabase

```
# Copy the fake env vars
cp .env.example .env
# Pull the latest images
docker compose pull
# Start the services (in detached mode)
docker compose up -d
```

After that, we just needed to import my data through the admin dashboard, and the app was ready to be used. It was that simple.

3.4 Using Pocketbase

PocketBase is an open-source backend consisting of an embedded database (SQLite) with real-time subscriptions, built-in auth management, convenient dashboard UI, and simple REST-ish API.

Pocketbase offers a different set of products from Supabase:

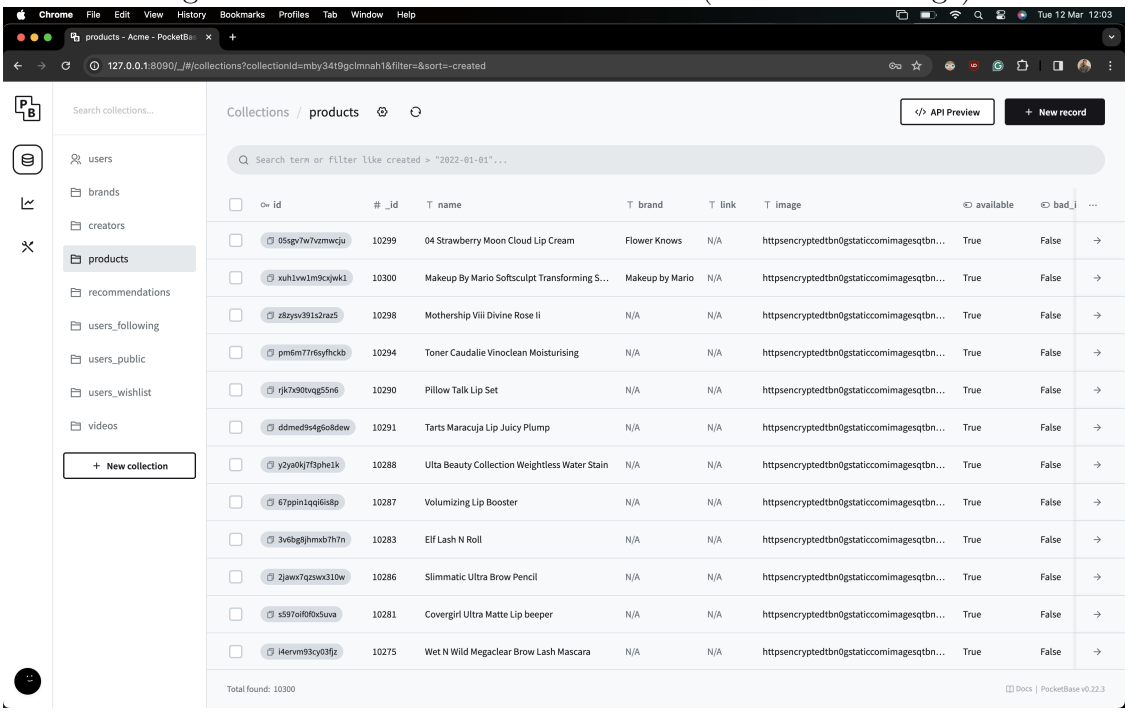
- database - PocketBase uses embedded SQLite (in WAL mode). For the majority of the queries, SQLite (in WAL mode) outperforms traditional databases like MySQL, MariaDB, or PostgreSQL (especially for read operations).
- authentication - just like Supabase, add and manage email and password, passwordless, OAuth, and mobile logins to your project.
- no edge function support - PocketBase differs from the other similar backend solutions like Firebase, Supabase, Nhost, etc., and doesn't support running server-side cloud functions without manual resource allocation. Instead, PocketBase could be used as a Go or JS framework that enables you to build your own custom app-specific business logic and still have a portable backend at the end.

3.4.1 Admin Dashboard

Pocketbase has a more basic dashboard, compared with Supabase, as we can see from the screenshot in Figure 6. There are 3 tabs accessible from the left-side menu:

- Collections - allows the user to visualize the data in the collections as a spreadsheet and edit single records. Here you can create new collections but you cannot import data from a CSV or TSV file. Here you can also edit a table's fields by changing their type or default value, making them unique, adding foreign key relations, and "check" constraints.
- Logs - displays HTTP API logs, including timestamps.
- Settings - general project settings: set up OAuth, create and manage app administrators, create backups, set up email templates, and export/import collection configurations.

Figure 6: Pocketbase Admin Dashboard(Table Editor Page)



3.4.2 Self-hosting

One of the best PocketBase features is that it's completely portable. This means that it doesn't require any external dependency and could be deployed by just uploading the executable on your server. To do this on GWDG, we first created a m1.large instance and downloaded the prebuilt minimal PocketBase app, and copied the files from my computer to the instance over SSH, using the following command:

```
1 scp -i hpcsa-course-vm-key.pem /MY_LOCAL_FOLDER/pocketbase_0.21.1
    _linux_amd64/* cloud@141.5.105.244:/home/cloud
```

Then, we connected to the instance and forwarded port 8090 to port 8090 on our machine using the following command:

```
1 ssh -L 8090:127.0.0.1:8090 -o ServerAliveInterval=60 -i hpcsa-
    course-vm-key.pem cloud@INSTANCE_FLOATING_IP
```

Then inside the instance, we ran **./pocketbase serve**. After that, we just needed to import the data and Pocketbase was ready to be used. It was that simple.

4 Results

After spending the duration of the app's development evaluating the experience of using the different tools provided by Supabase and Pocketbase, we decided to evaluate them according to the criteria that seemed more relevant to the "user personas" A and B.

4.1 Complexity/Ease of use

Supabase's documentation is well-structured and easy to read and navigate through. There are code examples for all the available functions, including parameters and expected results. On the other hand, Pocketbase also includes code examples for all available functions, including parameters and expected results, but the documentation is not as well structured.

The ability to import data from a CSV or TSV file in the Supabase dashboard is a big time saver and could be a deciding factor for some users(User A). This feature is very popular, as many users have requested something similar for Pocketbase [Poc22]. There are even some unofficial solutions developed by the community, like the one we used in our project, developed by Github user michal-kapala [mic23]. We did notice some issues with this solution, mainly with handling some strings with special characters(some CSV files had URLs to images or videos and these had to be edited to be able to import the data).

Overall, there is no clear winner for this section. Both Supabase and Pocketbase are very intuitive and easy to use, but the addition of a proper, official data import tool for Pocketbase would be appreciated.

4.2 Security

When using Supabase, you can set different RLS policies for each table. Row-level security, or RLS, refers to the practice of controlling access to data in a database by row so that users are only able to access the data they are authorized for. This contrasts with database-level or table-level security, which controls access to entire databases or tables. This means that we can select what tables we want our app users to be allowed to edit and which ones they should not be able to edit.

RLS is not supported in Pocketbase(uses table-level security). Instead, the developer can set up API Rules, which serve as access controls and data filters for collections. Each collection is governed by five specific API action rules:

- **listRule:** Defines access control for listing data.
- **viewRule:** Dictates access to viewing specific data entries.
- **createRule:** Specifies access control for creating new data entries.
- **updateRule:** Governs access to updating existing data entries.
- **deleteRule:** Controls access to deleting data entries.

For collections related to authentication (Auth collections), an additional rule called **options.manageRule** exists. This rule enables a designated user, even from a different collection, to have full management control over another user's data. This includes tasks such as changing their email, password, etc.

The rules can be configured with the following settings:

- **"locked" (null):** This setting restricts the action to be performed only by an authorized admin. It serves as the default configuration.
- **Empty string:** This configuration allows anyone, including admins, authorized users, and guests, to perform the specified action.

- **Non-empty string:** When set, only users, whether authorized or not, that meet the criteria defined by the rule filter expression will be able to perform the corresponding action. This provides a fine-grained control over access based on specific conditions.

Overall, there is no clear winner for this section. Both Supabase and Pocketbase offer reliable security mechanisms that allow developers to set restrictions on who can and can't alter the tables in the database.

4.3 Scalability

When using Firestore(from the Firebase toolkit), as it is a NoSQL document database, it scales automatically and handles relational data fairly well. However, it does have some limitations, and scaling certain types of data relationships can be extremely difficult(not ideal for full-text search). Supabase uses PostgreSQL, which allegedly is expensive and difficult to scale(in its vanilla form, is not inherently designed as a distributed database). Scaling PostgreSQL typically involves various manual interventions and optimizations, and it may require significant expertise to set up and manage in a distributed, high-performance, or highly available environment. Supabase addresses these issues by handling the scaling for you automatically, whether by providing your project with more disk space[Sup20e] or by partitioning tables for distribution. In addition, Supabase offers read-only replicas which help developers scale to millions of users. We have customers now with millions of users, so it definitely can be done.

Pocketbase only scales on a single server, aka. vertical. Most of the time, you may not need the complexity of managing a fleet of machines and services just to run your backend. PocketBase is a great choice for small and midsize applications - SaaS, mobile API backend, intranet, etc. Even without optimizations, PocketBase can easily serve 10,000+ persistent real-time connections on a cheap $4 Hetzner CAX11 VPS (2vCPU, 4GB RAM). You can explore the official benchmarks repo for more details.

In summary, if scalability is a major concern for the project and we are planning for more than 10,000+ users to use our application, it's best to pick Supabase over Pocketbase.

4.4 Other important evaluation criteria

4.4.1 Price

Self-hosting Supabase is free. If the user wishes to use Supabase's cloud platform, they provide simple, predictable pricing. Pocketbase is self-host only, meaning there is no official cloud platform, but they suggest some free options for small POC and hobby apps:

- Fly.io (*the free tier comes with 1vCPU, 256MB RAM, and 1GB disk storage /up to 3GB but requires card details/*).
- PocketHost.io (*shared single VM instance, maintained and provided by benallfree*).

Other than those options, the price for running Supabase/Pocketbase ultimately depends on the price for the instances needed to run the necessary services, whether these instances are provided by a cloud provider or by the user itself.

4.4.2 Vector embeddings and similarity search

Vector embeddings represent different dimensions of certain data that are essential for understanding patterns, relationships, and underlying structures. In the case of our application, vector embeddings are important for creating the "search-engine-like" features that enable the users to search for different products according to several criteria(name, type, color, and brand). For example, the user could wish to search for "Brown Mascara from MAC".

To help in this task, Supabase uses pgvector and other tools to allow the developer to store, index, and query your vector embeddings at scale., using their client libraries[Sup20d].

In the case of Pocketbase, there are no features that allow for the creation of vector embeddings. There are third-party tools that allow the use of SQLite to store and query vector embeddings[Ste23], but when asked if the developers of the tool would add it in the future, they said that "there are no plans implementing it since it will be hard to make it portable"[Poc23a].

Therefore, if we intend to build search features into our application(like in the case of our app where we want to develop a search engine for cosmetics), we would want to pick Supabase over Pocketbase to save time on implementing vectorization from scratch.

4.4.3 Performance

Figure 7 shows performance comparisons of Supabase and Pocketbase running on a MacBook Pro 2018(16 GB RAM) and a GWDG m1.large(8GB RAM) instance running CentOS 8. We ran each one of the API calls exactly 10 times and averaged the response time for each of them.

Figure 7: Comparing elapsed time of API calls on different hardware

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	AVG
Supabase – running in the cloud(provided by supabase)											
Get All Brands(107 rows)	147	424	462	144	256	296	113	288	130	170	243
Get All Creators(294 rows)	211	101	243	165	106	137	265	93	106	169	159.6
Get All Products (10,300 rows)	222	156	133	125	484	411	164	198	365	227	248.5
Supabase(Macbook Pro 2018 16GB RAM)											
Get All Brands(107 rows)	26	44	17	32	17	21	45	14	15	17	24.8
Get All Creators(294 rows)	29	20	16	57	33	17	25	20	15	15	24.7
Get All Products (10,300 rows)	218	184	242	204	184	200	196	207	192	183	201
Pocketbase(Macbook Pro 2018 16GB RAM)											
Get All Brands(107 rows)	17	14	27	14	13	11	14	14	13	12	14.9
Get All Creators(294 rows)	22	27	21	19	21	31	19	19	21	22	22.2
Get All Products (10,300 rows)	354	480	449	457	445	435	444	440	448	437	438.9
Pocketbase on m1.large											
Get All Brands(107 rows)	69	50	57	50	46	48	49	44	30	36	47.9
Get All Creators(294 rows)	145	48	73	50	189	227	67	47	50	44	94
Get All Products (10,300 rows)	3610	2830	3500	2380	1377	5490	3580	3620	2940	3150	3247.7
Supabase on m1.large											
Get All Brands(107 rows)	32	30	71	26	129	34	211	108	29	37	70.7
Get All Creators(294 rows)	38	30	30	40	41	36	170	84	115	63	64.7
Get All Products (10,300 rows)	528	313	814	519	442	645	744	318	292	294	490.9

We can see that, the API calls on Supabase cloud service tend to take the same amount of time, despite the different sizes of the data being transferred. When running locally, Supabase and Pocketbase perform very similarly when retrieving fewer rows (between 100 and 300). However, when retrieving 10300 rows, when running on the Macbook, Pocketbase takes 2x longer than Supabase and, when running on m1.large, Pocketbase takes 6x longer than Supabase.

4.4.4 Community support

As of January 2024, Supabase has more than 62.1k stars on Github with 1049 contributors and Pocketbase has 30.7k stars with 46 different contributors. Supabase has an average open-issue age of less than 30 days, while Pocketbase has an average open-issue age of between 30 and 90 days(this is an important metric as it lets us know how long it takes for an issue to be addressed by the contributors).

You can get advanced current repository statistics and insights from RepoTracker here:

- **Supabase** - https://repo-tracker.com/r/gh/supabase/supabase?utm_source=github-stats
- **Pocketbase** - https://repo-tracker.com/r/gh/pocketbase/pocketbase?utm_source=github-stats

5 Discussion

In this section, we connect the information from Section 2 to the description of the project from Section 3 and evaluation from Section 4 to form a conclusion in what area which of the products is ahead.

In Figure 8 we can see the main takeaways from Section 4. We can see that Pocketbase lacks some specific features that put it at a disadvantage. These could be added in the future, however as we are evaluating these tools as they currently are, Supabase is just more versatile. On the matter of how does this affect users A's and B's development, we can say that picking Supabase over Pocketbase gives them more options over the kind of applications they can build(see Section 4.4.2) and they both will appreciate the scalability capacity of Supabase(see Section 4.3). However, if user A just wanted to build a simple application that doesn't require them to use Supabase's additonal feautures then they would be more than fine using Pocketbase.

Figure 8: Comparing Results from our conducted research between Supabase and Pocketbase

	Supabase	Pocketbase
Documentation	✓	✓
Import data from CSV	✓	~
Security Controls	✓	✓
Vector embeddings and similarity search	✓	×
Scalability	✓	~
General Performance	✓	✓
Community Support	✓	~

✓- natively supported; good; ~ - supported; decent × - not supported; bad

6 Conclusion

Our goal for this report was to compare the feautures and capabilies of Supabase and Pocketbase. We built a React application and two different backend, each using one of the different Baas solutions. We evaluated the two on complexity/easy-of-use, security, scalability, price, community-support and additional feautures.

Both Supabase and Pocketbase are strong candidates for BaaS solutions and can be a great fit depending on the requirements of your project. If you're looking for a versatile platform with great real-time capabilities and a vast feature set, Supabase may be the way to go. However, Pocketbase is an excellent open-source alternative that offers a reliable relational database and a well-designed user interface.

References

[Goo20] Google. *Firebase*. Accessed on 27/10/2023. 2020. URL: https://firebase.google.com/.

[Jag23] Jagrat Patel. *Supabase vs Firebase2*. Accessed on 27/10/2023. 2023. URL: https://www.linkedin.com/pulse/firebase-vs-supabase-which-platform-best-building-your-jagrat-patel/.

[mic23] michal-kapala. *Pocketbase import - Github*. Accessed on 27/12/2023. 2023. URL: https://github.com/michal-kapala/pocketbase-import.

[Poc22] Pocketbase Github. *Pocketbase Issues - Import/Export CSV/JSON*. Accessed on 29/12/2023. 2022. URL: https://github.com/pocketbase/pocketbase/issues/48.

[Poc23a] Pocketbase. *Inclusion of Vector Database in PocketBase's Roadmap*. Accessed on 22/01/2024. 2023. URL: https://github.com/pocketbase/pocketbase/discussions/2990.

[Poc23b] Pocketbase. *Pocketbase*. Accessed on 27/10/2023. 2023. URL: https://pocketbase.io/.

[Poc23c] Pocketbase. *pocketbase-Benchmarks*. Accessed on 27/12/2023. 2023. URL: https://github.com/pocketbase/benchmarks.

[Ste23] Stephen Collins. *How to use SQLite to store and query vector embeddings*. Accessed on 27/10/2023. 2023. URL: https://stephencollins.tech/posts/how-to-use-sqLite-to-store-and-query-vector-embeddings.

[Sup20a] Supabase. *Supabase*. Accessed on 27/10/2023. 2020. URL: https://supabase.com/.

[Sup20b] Supabase. *Supabase Benchmarks Github*. Accessed on 27/10/2023. 2020. URL: https://github.com/supabase/benchmarks/issues/8.

[Sup20c] Supabase. *Supabase Benchmarks Github 2*. Accessed on 27/10/2023. 2020. URL: https://github.com/supabase/benchmarks/issues/6.

[Sup20d] Supabase. *Supabase Docs - AI Vectors*. Accessed on 27/10/2023. 2020. URL: https://supabase.com/docs/guides/ai.

[Sup20e] Supabase. *Supabase Docs - Database Size*. Accessed on 27/10/2023. 2020. URL: https://supabase.com/docs/guides/platform/database-size.

[Sup20f] Supabase. *Supabase vs Firebase*. Accessed on 27/10/2023. 2020. URL: https://supabase.com/alternatives/supabase-vs-firebase.