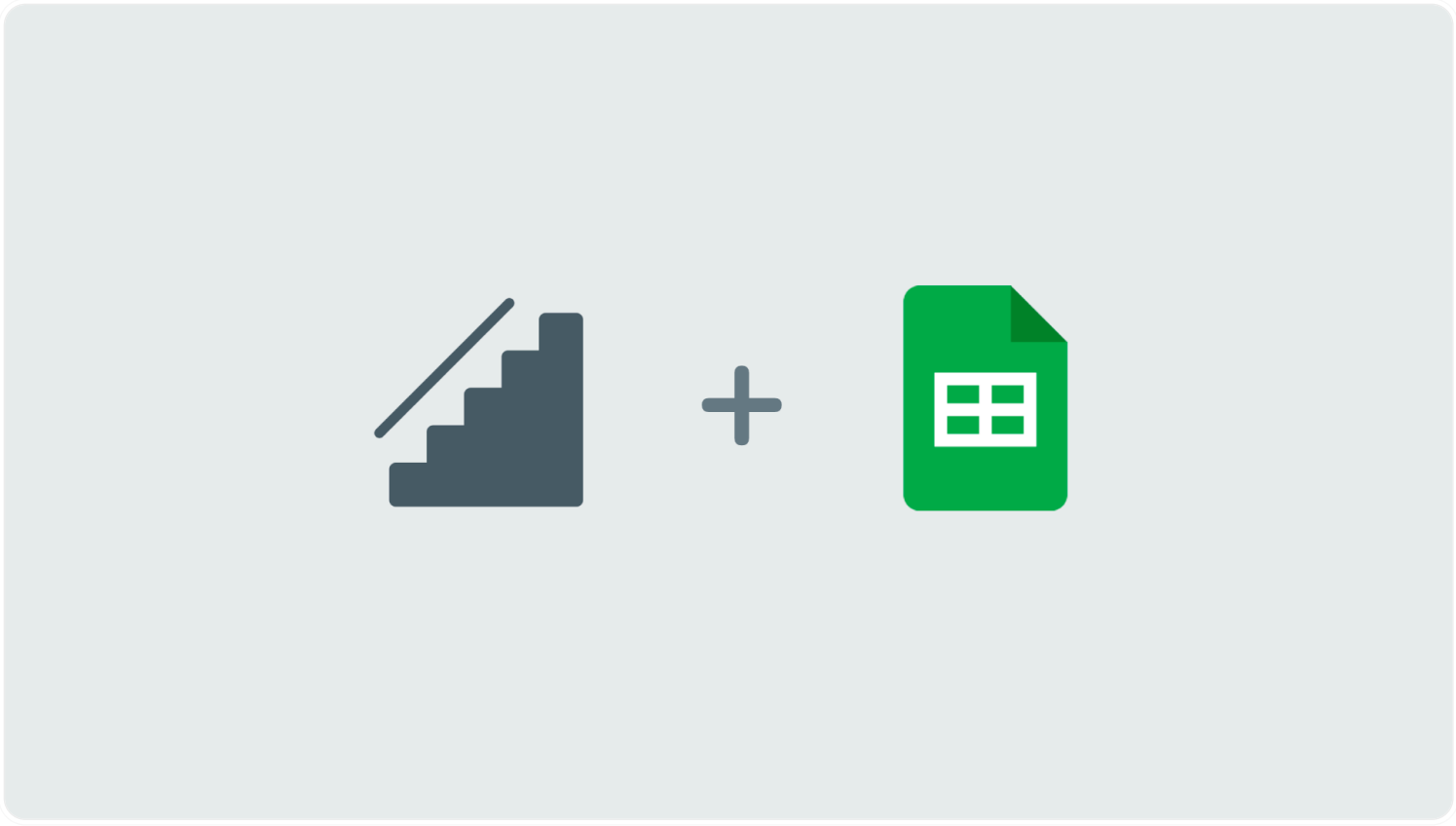
ENGINEERING

# How Levels.fyi scaled to millions of users with Google Sheets as a backend

Our philosophy to scaling is simple, avoid premature optimization

Dushyant Sabharwal
February 3, 2023



> Update (January 2024): While we continue to use Google Sheets throughout our operations, we now also rely on a Postgres DB.

Levels.fyi has become *the* career site for professionals. Our users today span the entire globe and as of now roughly 1-2 million unique users visit the site every month. We started back in 2017 with a humble vision of helping professionals *get paid, not played*, and within a couple of years, the site became a primary resource for US tech professionals looking for jobs, salaries and to get negotiation help.

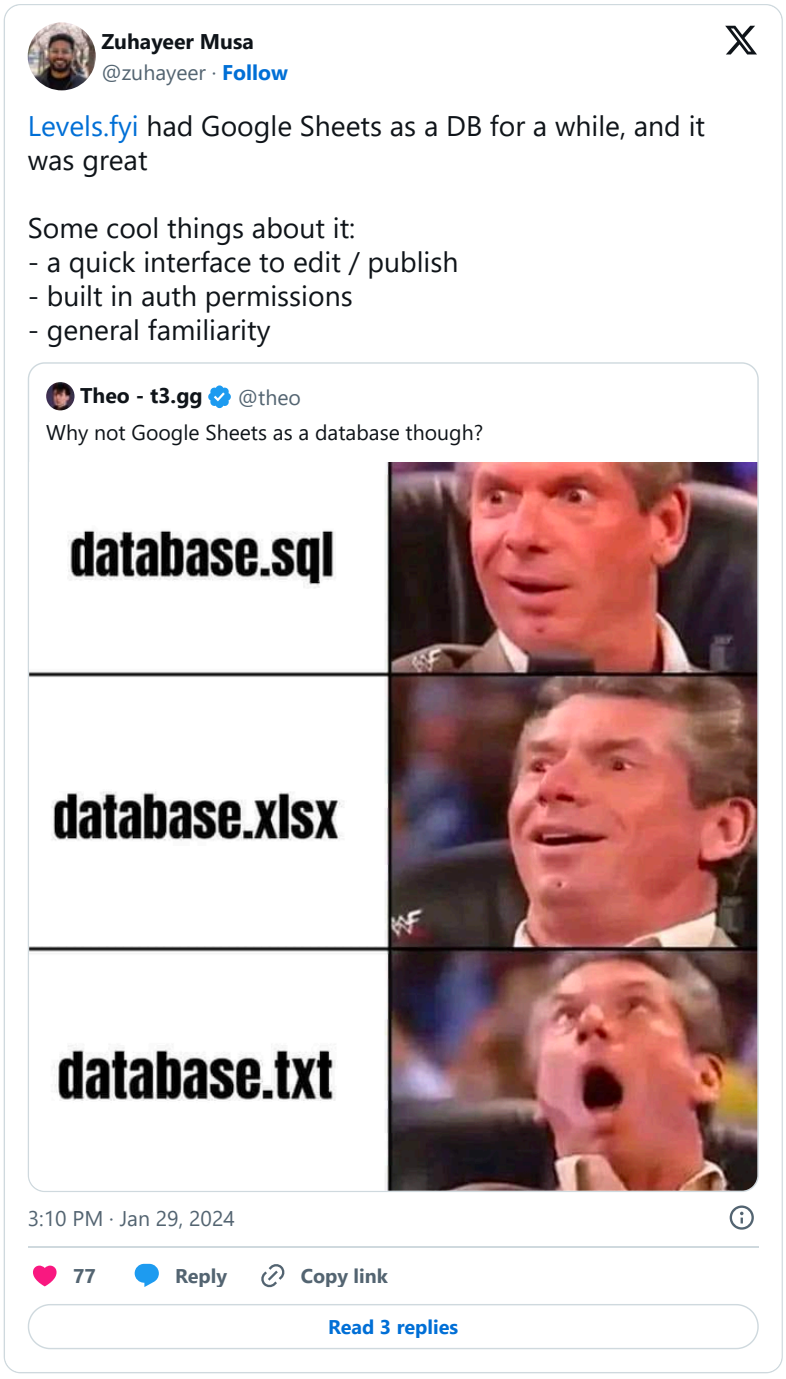
**But, what if I told you that the initial version of the site did not have a backend?**

It seems like a pretty counterintuitive idea for a site with our traffic volume to not have a backend or any fancy infrastructure, but our philosophy to building products has always been, *start simple and iterate*. This allows us to move fast and focus on what's important.


August 2024 Update: Describing how Levels.fyi used Google Sheets to scale to millions of users using this viral meme of Yusuf Dikec from the Olympics.

**Why did we start without a backend?**

- To focus more on the product/idea fit. Google Forms + Google Sheets allowed us to move fast in releasing the initial version
- Save effort and money on setting up a API and database server on AWS
- Save effort on operational maintenance of a API and database server



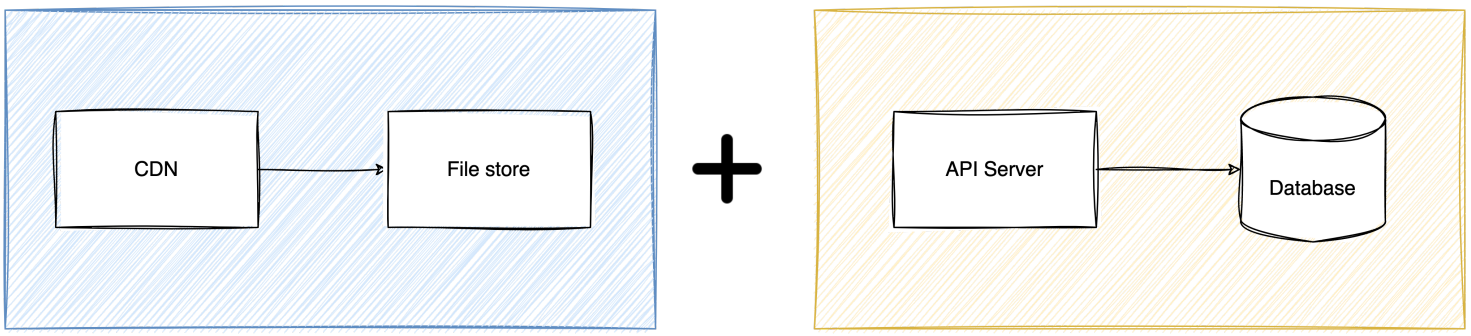

Through this blog post we will be sharing our strategy and learnings on building a dynamic site without a database and an API server. We hope this helps you bootstrap your products and ideas faster, as well!

### Static vs Dynamic site

Whenever someone thinks about building a dynamic site the first cause of mental fatigue is the backend. The backend is responsible for processing, storing & delivering the dynamic content. If we take the backend for any site out of the equation then it's mainly a static site.

Building a static site is not super challenging because you don't have to think about maintaining a server for APIs and a database for data storage.

To build a static site you only need the left part and a dynamic site requires both.




## Our approach

Our secret sauce to do away without a database and a server consisted mainly of:

- Google Forms
- Google Sheets
- AWS Lambda
- AWS API Gateway

> 🖍️ It may also be worth mentioning that you could start without any AWS stuff. Just that loading from Google Sheets directly has longer latency and isn't performant.

The user interface can be replaced by Google Forms. The database can be replaced by Google Sheets. And the API server can be replaced by AWS API Gateway + AWS Lambda.

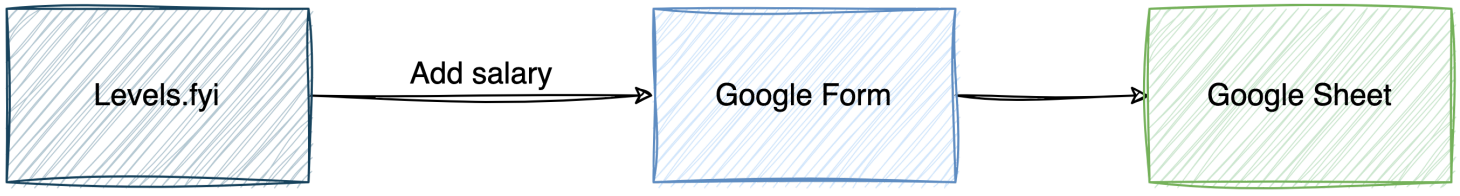
Google Forms, Google Sheets & API Gateway are *no-code* tools and they require zero amount of operational maintenance. It's Google's & AWS's job to keep them up and running 24x7.

## Write Flow

Example: User adds a new salary on the site.

### Version 0

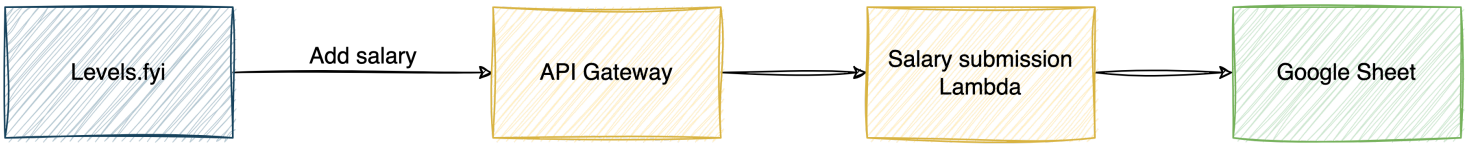
V0 had no UI, add salary form was a Google Forms UI 😱




### Version 1

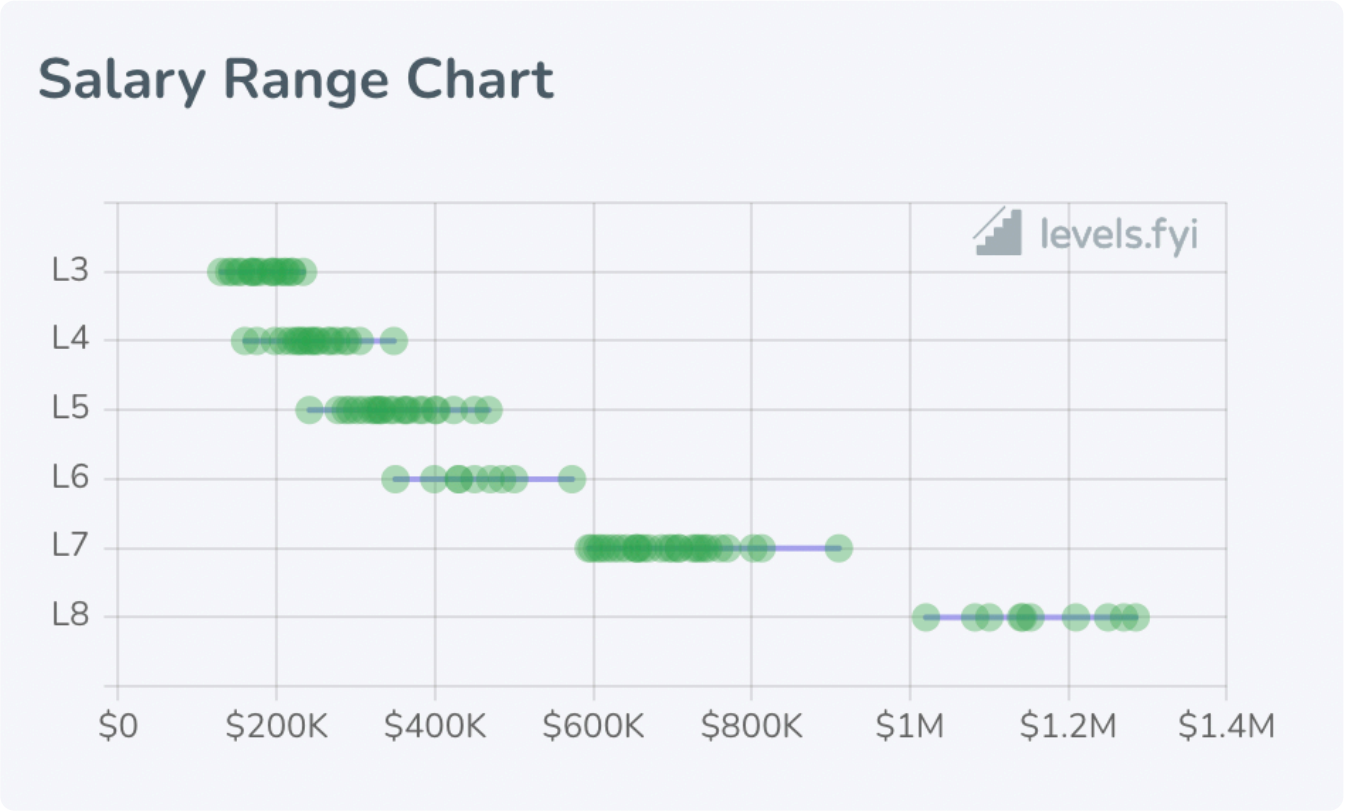
Our own UI with necessary validations on the frontend

1. Frontend calls publicly visible API Gateway
2. API Gateway triggers and spawns a lambda function
3. Lambda function processes and appends new salary to the sheet



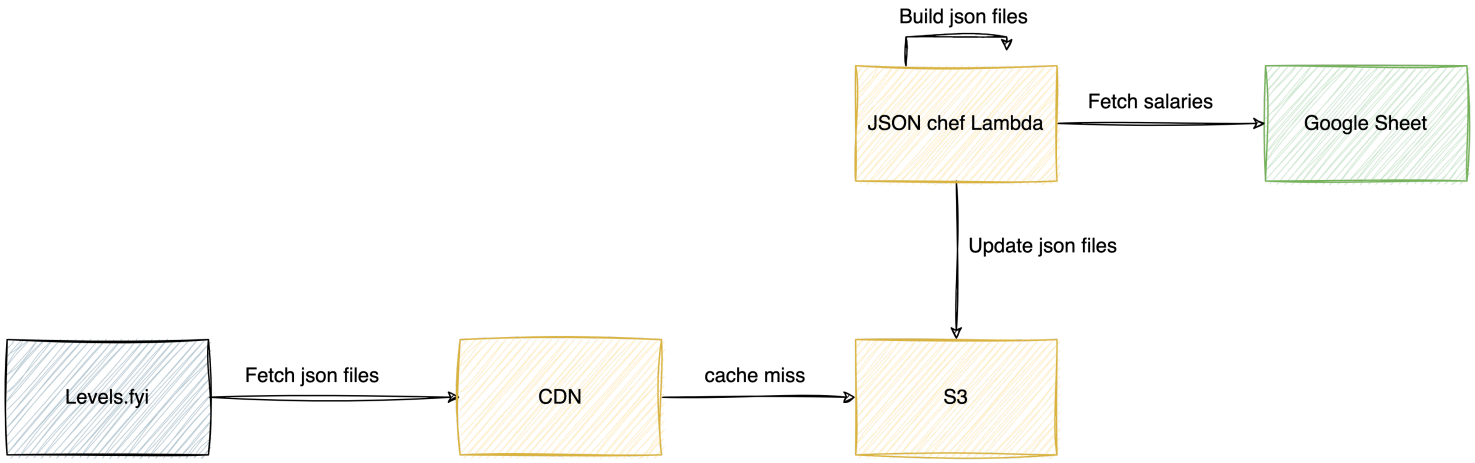

### Read Flow

Our read flows were more complex than write ones since beyond just showing salary data we also had use cases where we would show charts like below




Our recipe for building a read flow was as follows:

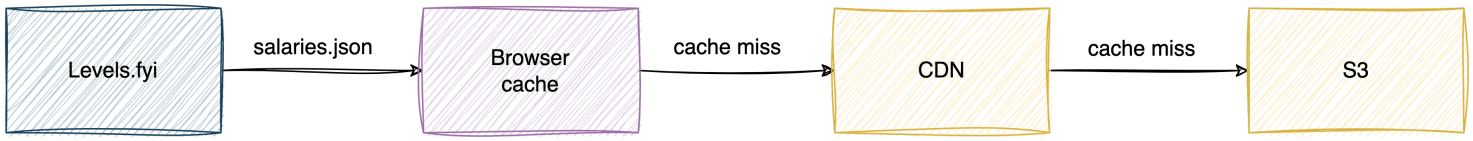
1. Process data from Google Sheet and create a JSON file
2. Use AWS Lambda for processing and creating new JSON files
3. Upsert JSON files on S3
4. Cache JSON files using a CDN like AWS Cloudfront




On initial page load the user's browser downloads all the needed JSON files and any further processing happens in the browser.

Yes, you read that right. In the first few years of Levels.fyi, every single salary (over 100k at some point) were downloaded in a single json. All graphs, statistics, calculations, etc were done in the browser.

We used the below caching strategy to reduce data transfer for the user.




## Drawbacks

The above architecture/design worked well for 24 months but as our users and data grew we started running into issues.

1. The size of json files grew to several **MBs**, every cache miss was a massive penalty for the user and also for the initial page load time
2. Our lambda functions started timing out due to the amount of data that needed to be processed in a single instance of execution
3. We lacked any SQL based data analysis which became problematic to make data driven decisions
4. Google Sheets API rate limiting is pretty strict for write paths. Our writes were scaling past those limits
5. Since our data was downloaded as json files it was easy to scrape and plagiarise

## Migration to a new backend

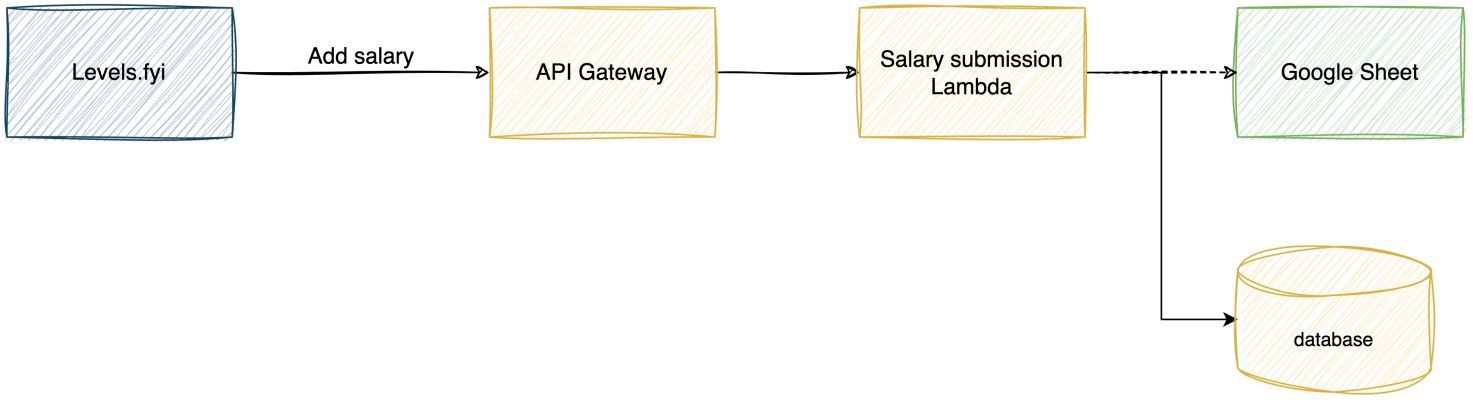
To mitigate the drawbacks we started migrating off of the Google Sheets and AWS lambda functions to databases and API servers.

Our migration goals were:

- Getting rid of all the JSON files
- Using APIs for all read & write paths

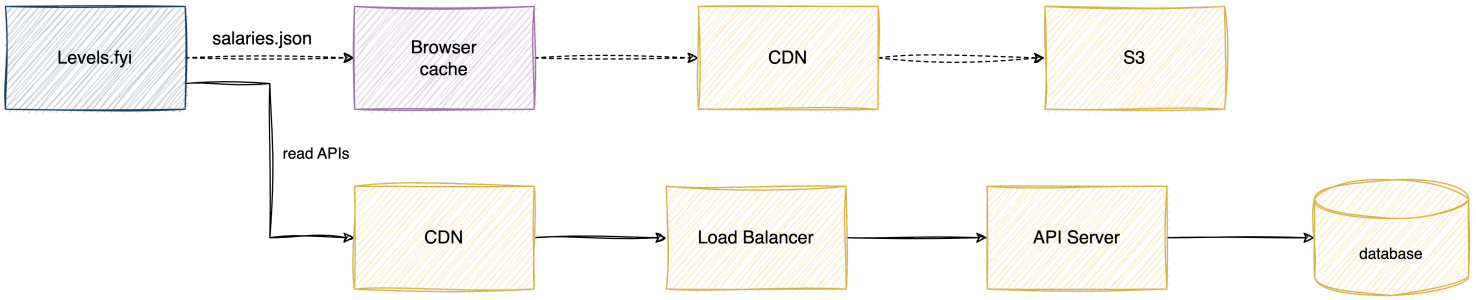
### Duplicating writes

In order to not break our current functionality dependent on Google Sheets and JSON files, we started duplicating writes to existing Google Sheets and our new database. This allowed us to start migrating read use cases to APIs

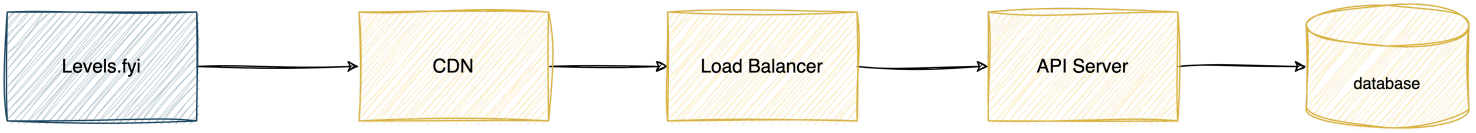



### Splitting JSON files

Started splitting JSON files into smaller chunks and deprecating each chunk by a new read API



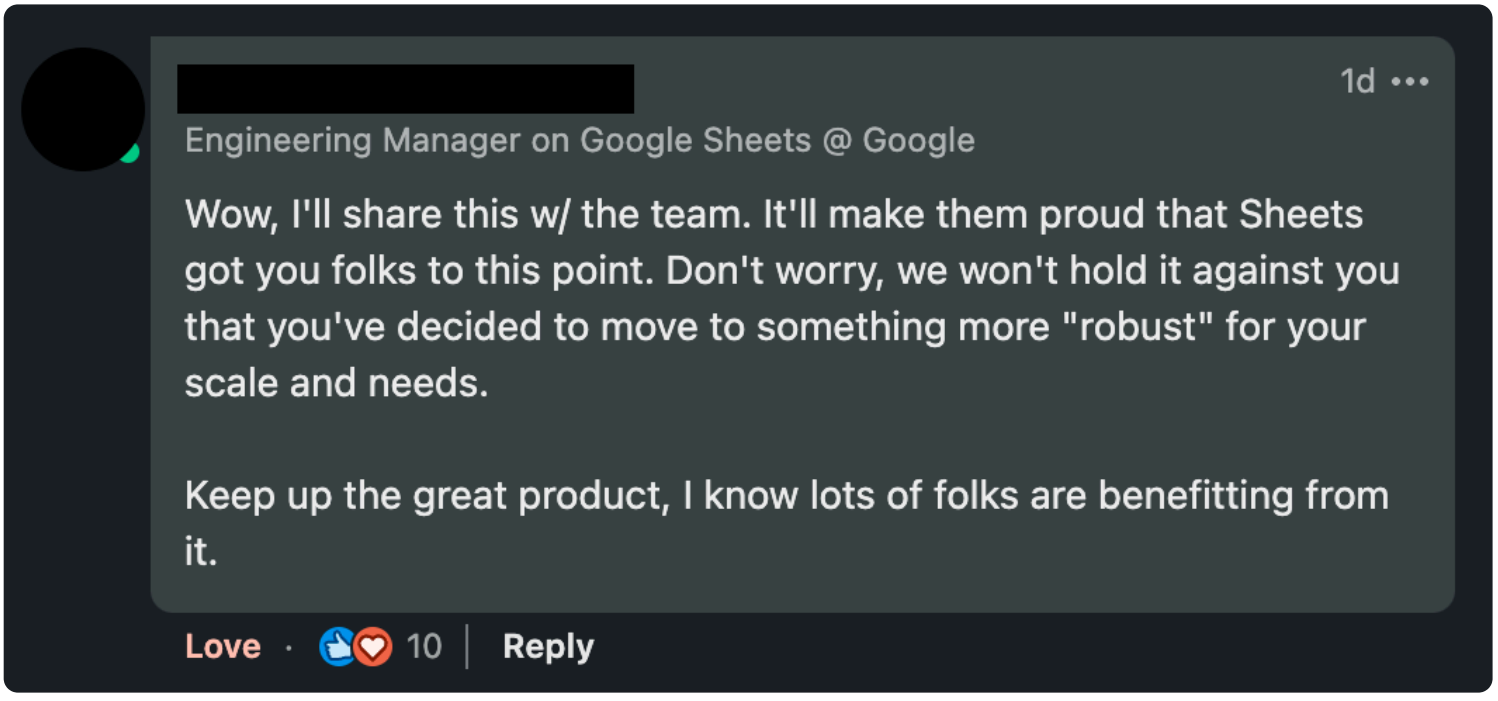

Moving all read & write paths to the API




## Conclusion

Google Forms & Sheets allowed us to launch & test ideas in a rapid manner rather than getting lost in setting up bits and pieces of the backend. After Levels.fyi achieved product market fit and our scale increased it made sense to move to a more robust and scalable backend infrastructure.

Our backend today is more sophisticated but our philosophy to scaling is simple, **avoiding premature optimization**. Even now, one of our most trafficked services today still has a single node.js instance serving **60K requests per hour** (topic for another blog post).




An engineering manager from the Google Sheets team commented on a LinkedIn post about Levels.fyi utilizing Google Sheets.

> Also check out: How Levels.fyi Built Scalable Search with PostgreSQL

levels.fyi

Helping people build better careers