Architecture Ideas

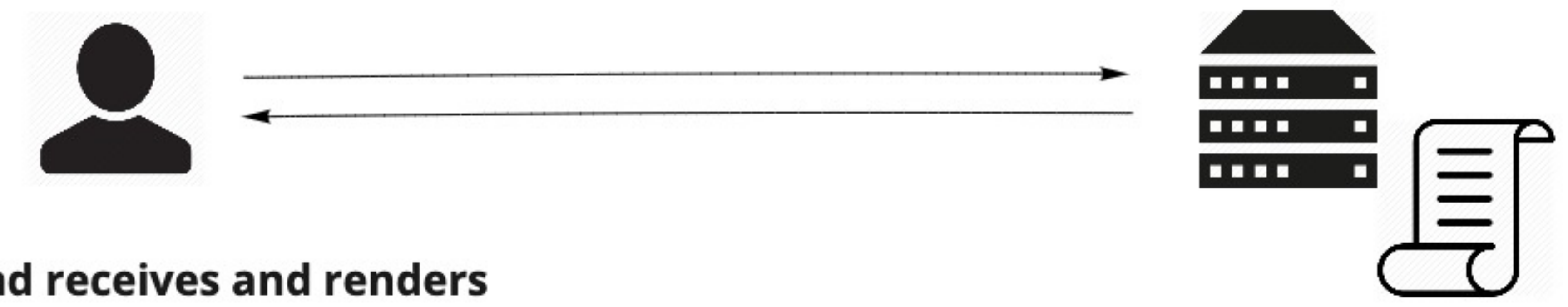
for The Sentimentalists

Original Miro Board: https://miro.com/app/board/o9J_klehSYw=/

Single Request / Server

1- Use submits a doc for analysis

2- Server is running an API endpoint, receives the request, and processes it (web scraping, analysis), returning an output



3- Front end receives and renders the result for the user.

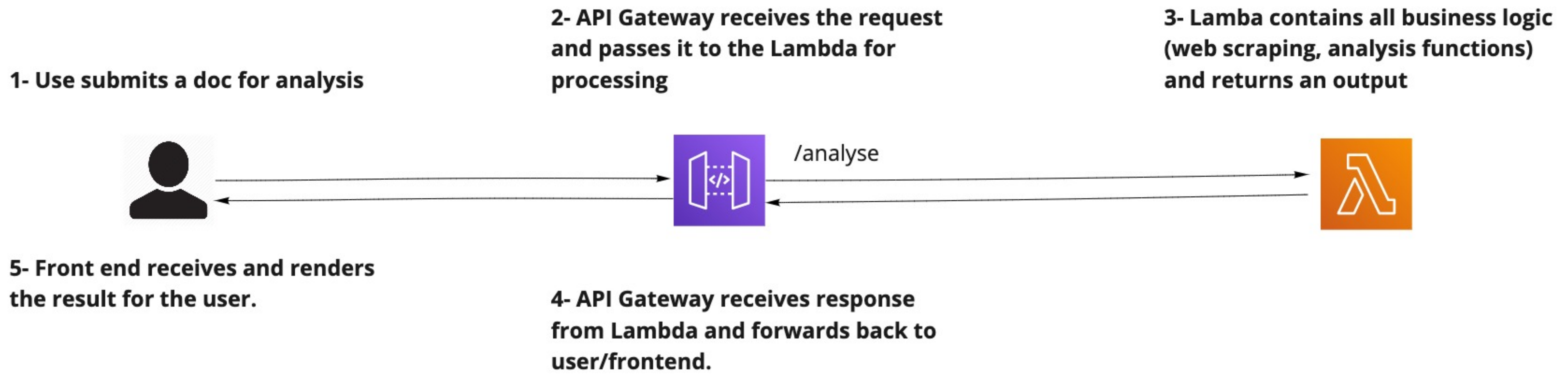
✅ Pros

- One script
- Easily deployable on other platforms

⛔ Cons

- Have to write API handlers in python/java
- One codebase makes it harder to test/deploy
- Have to setup server/
- Have to pay for the server running

Single Request / Lambda



1- Use submits a doc for analysis

2- API Gateway receives the request and passes it to the Lambda for processing

3- Lamba contains all business logic (web scraping, analysis functions) and returns an output

4- API Gateway receives response from Lambda and forwards back to user/frontend.

5- Front end receives and renders the result for the user.

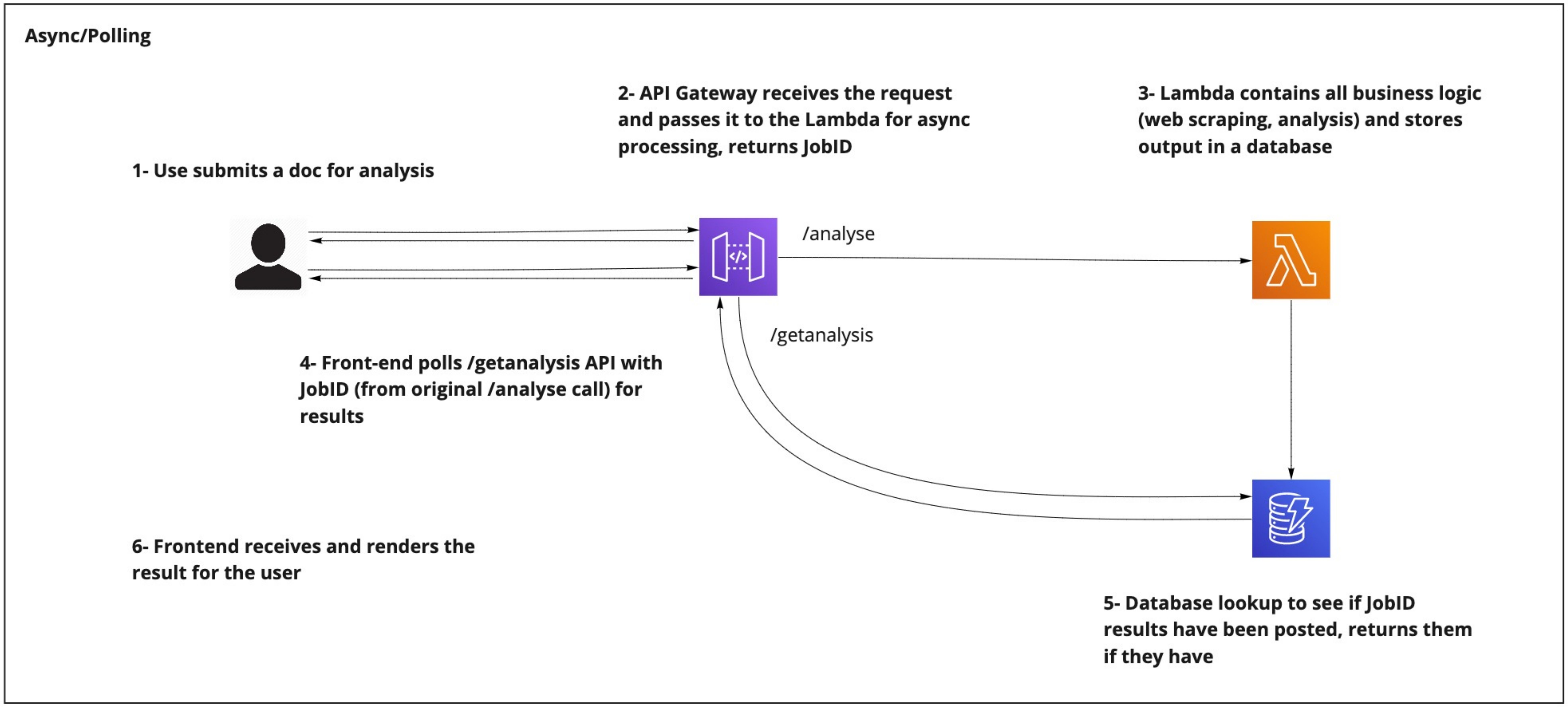
✅ Pros

- Easier to get going
- Can mock APIs with APIG
- Don't have to build a server/template

⛔ Cons

- All business logic is in one long-running lambda (not parallel)
- Connection from frontend might time-out

Async/Polling



1- Use submits a doc for analysis

2- API Gateway receives the request and passes it to the Lambda for async processing, returns JobID

3- Lambda contains all business logic (web scraping, analysis) and stores output in a database

4- Front-end polls /getanalysis API with JobID (from original /analyse call) for results

5- Database lookup to see if JobID results have been posted, returns them if they have

6- Frontend receives and renders the result for the user

✅ Pros

- Separates request submission from getting analysis data

⛔ Cons

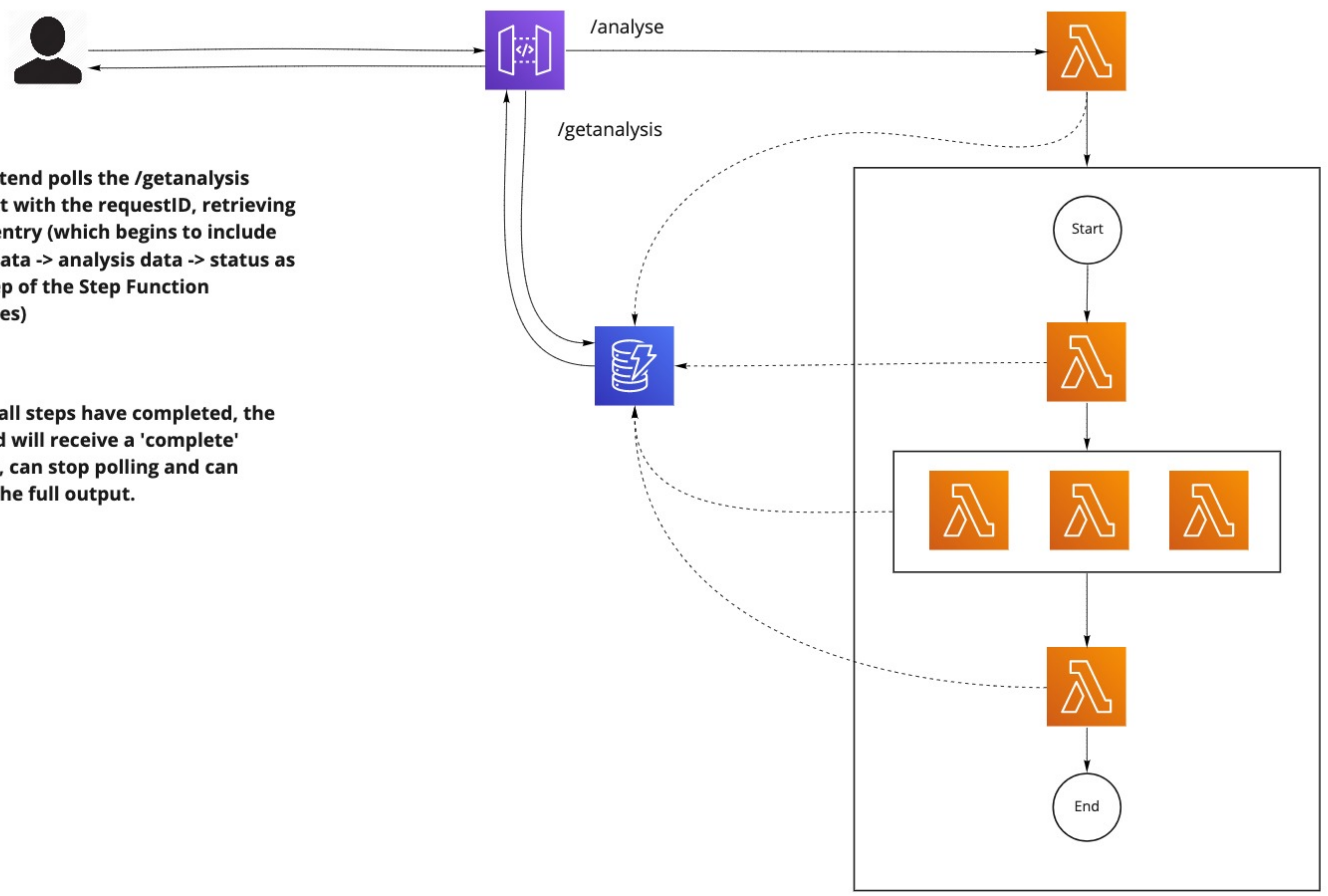
- All business logic is still in one long-running lambda (not parallel), meaning it might take awhile
- Polling connections are a bit gross, especially when scaling

Async/Polling with Step Function

1- Use submits a doc for analysis

2- API Gateway receives the request and passes it to the Lambda for async processing, returns JobID

3- Lambda creates request record, returns request ID to API, and triggers Step Functions to start processing.



4a- Frontend polls the /getanalysis endpoint with the requestID, retrieving the DB entry (which begins to include article data -> analysis data -> status as each step of the Step Function completes)

4b- 1st Step retrieves article and does preliminary processing, stores the article in the DB again the request ID

5- 2nd Step conducts analysis using one or more lambdas in parallel, storing the result of each in the DB again the request ID.

6- 3rd Lambda updates DB to record processing is complete.

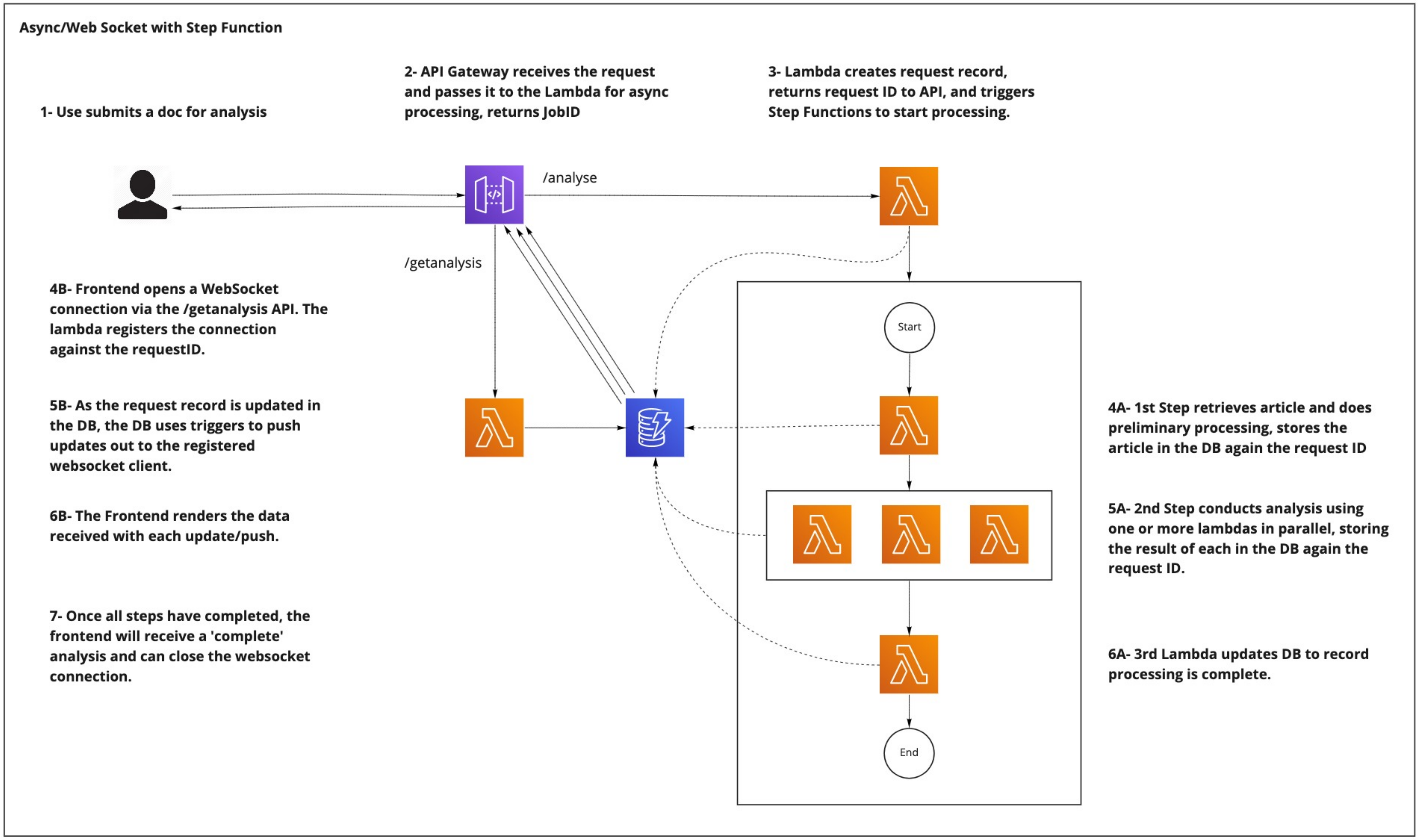
7- Once all steps have completed, the frontend will receive a 'complete' analysis, can stop polling and can render the full output.

✅ Pros

- Each step of business logic is separated into a small/compact module (easily testable, changeable without affecting whole platform)
- Easy to add/remove modules to step function
- Analysis modules can run in parallel

⛔ Cons

- Polling connections are still a bit gross, especially when scaling
- Slightly more complex to setup each component



Async/Web Socket with Step Function

1- Use submits a doc for analysis

2- API Gateway receives the request and passes it to the Lambda for async processing, returns JobID

3- Lambda creates request record, returns request ID to API, and triggers Step Functions to start processing.

4A- 1st Step retrieves article and does preliminary processing, stores the article in the DB again the request ID

5A- 2nd Step conducts analysis using one or more lambdas in parallel, storing the result of each in the DB again the request ID.

6A- 3rd Lambda updates DB to record processing is complete.

4B- Frontend opens a WebSocket connection via the /getanalysis API. The lambda registers the connection against the requestID.

5B- As the request record is updated in the DB, the DB uses triggers to push updates out to the registered websocket client.

6B- The Frontend renders the data received with each update/push.

7- Once all steps have completed, the frontend will receive a 'complete' analysis and can close the websocket connection.

✅ Pros

- As above
- No polling - websocket delivers updates to the front end as they happen

⛔ Cons

- More complex to setup/deploy