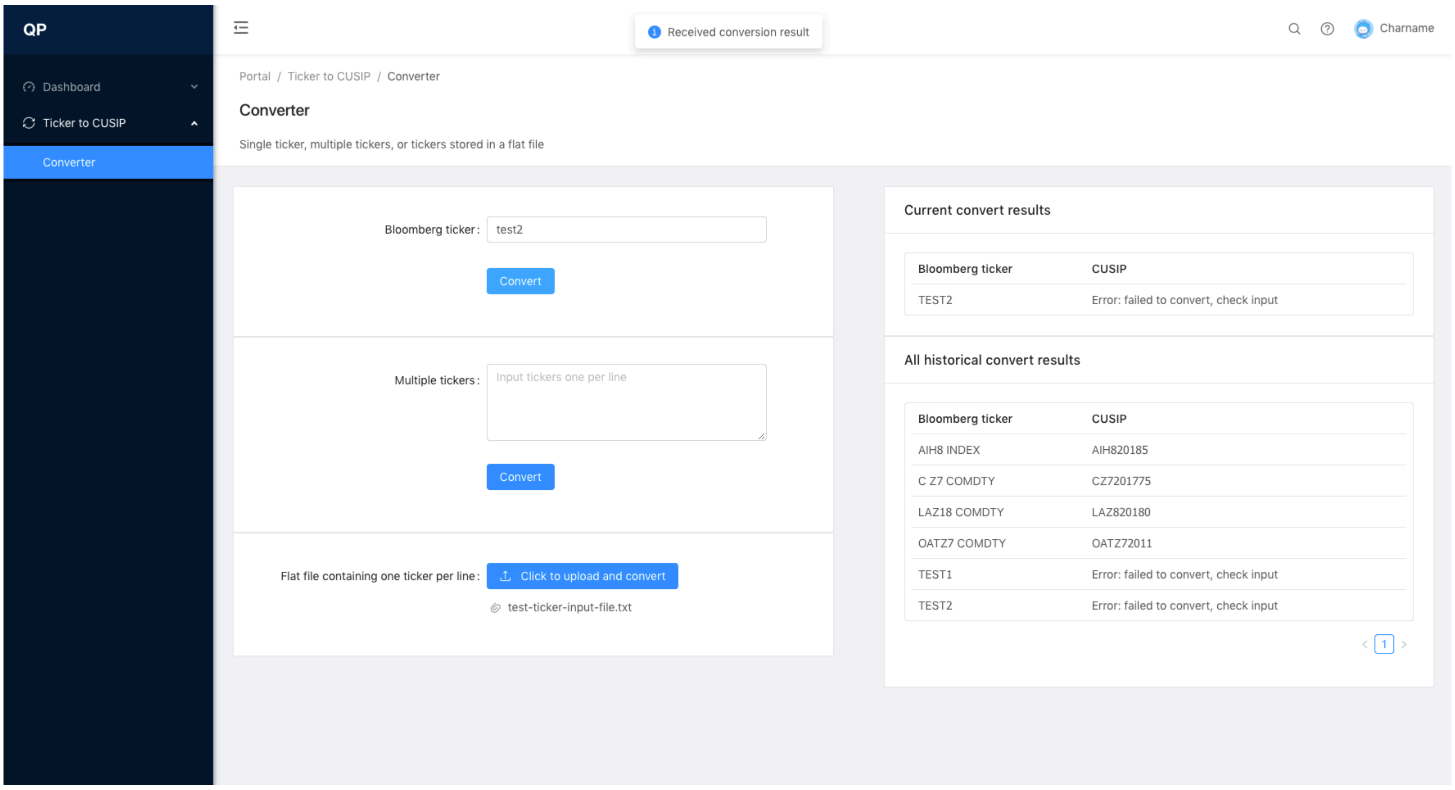
This is the writeup and response to Problem Statement. See `Build-and-run` section for instructions to get it up and running. Also see attached screenshot for a UI overview.

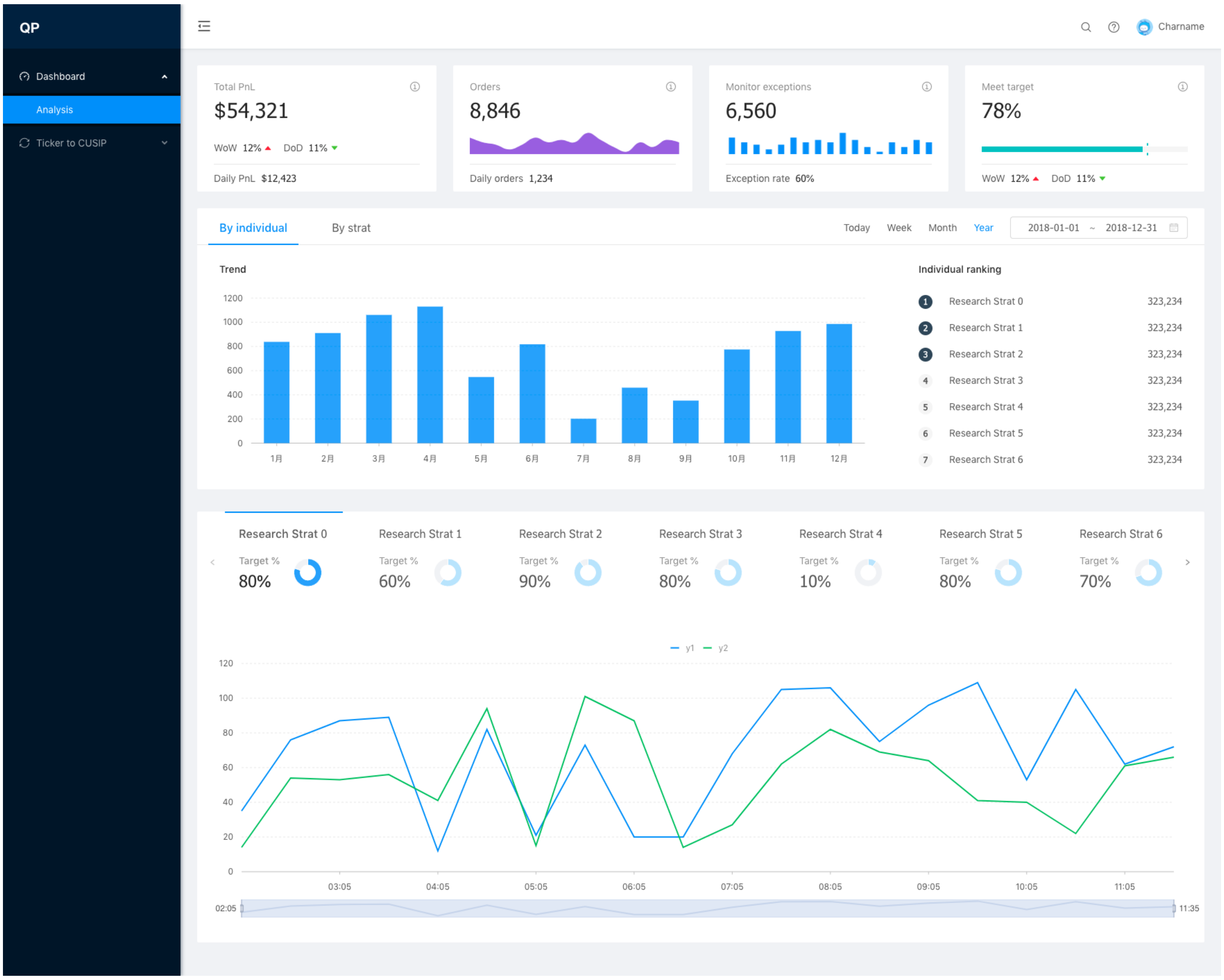

QP
Dashboard
Ticker to CUSIP
Converter
Received conversion result
Charname
Portal / Ticker to CUSIP / Converter
Converter
Single ticker, multiple tickers, or tickers stored in a flat file
Bloomberg ticker :
test2
Convert
Multiple tickers :
Input tickers one per line
Convert
Flat file containing one ticker per line :
Click to upload and convert
test-ticker-input-file.txt

Current convert results

Bloomberg ticker	CUSIP
TEST2	Error: failed to convert, check input

All historical convert results

Bloomberg ticker	CUSIP
AIH8 INDEX	AIH820185
C Z7 COMDTY	CZ7201775
LAZ18 COMDTY	LAZ820180
OATZ7 COMDTY	OATZ72011
TEST1	Error: failed to convert, check input
TEST2	Error: failed to convert, check input


QP
Dashboard
Analysis
Ticker to CUSIP
Charname
Total PnL
$54,321
WoW 12% DoD 11%
Daily PnL $12,423
Orders
8,846
Daily orders 1,234
Monitor exceptions
6,560
Exception rate 60%
Meet target
78%
WoW 12% DoD 11%
By individual
By strat
Today
Week
Month
Year
2018-01-01 ~ 2018-12-31
Trend
Individual ranking
1 Research Strat 0 323,234
2 Research Strat 1 323,234
3 Research Strat 2 323,234
4 Research Strat 3 323,234
5 Research Strat 4 323,234
6 Research Strat 5 323,234
7 Research Strat 6 323,234
Research Strat 0 Target % 80%
Research Strat 1 Target % 60%
Research Strat 2 Target % 90%
Research Strat 3 Target % 80%
Research Strat 4 Target % 10%
Research Strat 5 Target % 80%
Research Strat 6 Target % 70%
y1
y2

Do you agree with this proposal? Why? If not, what is a better solution?

Solution and alternatives

Overall, the problem is asking for a utility tool (i.e. Bloomberg to modified CUSIP converter). The proposal suggests a client-server model, where the RESTful API is used by both dev users via API access, and by UI users via web frontend.

One alternative is to have separate implementations of the converter on frontend and backend. Backend impl is used for API and frontend impl is used in browser. The pros and cons:

Client-server method pros:
- Consistency in return results. Better maintainability and extensibility.

- Finer control on the tool, e.g. permission and authentication, operation history logging.

Separate implementation method pros:
- Usable without network connection. Can be extended to become a PWA (Progressive Web Application), that's largely usable offline and only request for info when needed.
- Faster conversion result because HTTP request is not needed.

Consider the points above. For this particular tool, finer control doesn't seem to be necessary (why want to track how many tickers a colleague converted in a day, or restrict their access). However, the need to use it offline and the cost from network request is also low, assuming intranet environment. Especially performance is boosted by caching, rendering this gain even less.

Based on above, I favor and implemented the client-server method, where consistency and maintainability outweighs other factors.

Discussion on caching

This problem is an obvious candidate for caching. You can cache conversion results both on backend and frontend.

- On the backend, we can cache already converted tickers, and skip the calculation next time. It's using extra memory footprint to save CPU cycles. In my solution, this is implemented, to show its capability for more calculation-intensive problems. The gain in this CUSIP conversion case is of course little because calculation is small and fast.
- On the frontend, we cache already converted and received ticker results. For future requests, only request for unknown pairs, and return known pairs directly. This caching is of much value to user and is implemented.

The internal portal

It will be ridiculous if the frontend of this tool is in a separate webpage and user need to remember the url to access. Rather, user shouldn't need to remember and shouldn't need to open a new browser tab at all to use this tool. That brings the point: this tool should be part of the quant portal internal website. It's the intranet homepage where quant team members can find & search for all work information, and easily navigate to tools & utilities. Analogous to work.alibaba-inc.com (intranet access only) from Alibaba.

Here I presented a proof-of-concept of such quant portal (see it from http://localhost:8000). It's now frontend only but easy to envision how it can be implemented and extended.

Describe the solution in detail, e.g. the choice of programming languages, infrastructure, web server, etc.

The solution is isomorphic JavaScript architecture, React on the frontend and NodeJS on the backend.

Frontend

Frontend framework is known to be fragmented. This solution adopts state-of-the-art frontend components, tools and methodologies. Disadvantage is that it could take some time for dev to pickup.

- React, antd, for component
- Redux, dva, for state management
- roadhog, for mocking
- webpack+babel, for transpile and build
- lodash, moment, etc., for light-weight util libraries

Backend

This solution uses a NodeJS backend. This is not required and can be changed to Python/Java.

In the long run, it's recommended that we have a microservice architecture. Each service can be run & developed in isolation, as long as it exposes standard RESTful API. This way it's easier to collaborate and work in parallel within the team.

With that, we can easily have Python/Java/NodeJS services all running in harmony. For any new service, the tech stack will be chosen based on strength of programming language / developer skillset / team convention and standards.

Some advantages of Node are:

- Isomorphic architecture, same code / library / build process can be shared between

frontend and backend
- Easy interoperability with no-SQL databases like MongoDB, native literal support for JSON (which will be the primary network request body format)
- Better package manager and package management ecosystem (yarn and npm are better than pip / maven, by a small margin)

Engineering

Below are engineering / efficiency / QA related tools used in the solution

- Husky, pre-commit and pre-push hooks
- cross-env, env management
- git, for source control (consider adopt Github flow i.e. create own Github team and Github organization, public repo for open source projects, private repo for internal projects. This way it promotes team identity and brand name. Alternatively, setup internal Gitlab)
- eslint and prettier, for linting and code quality
- travis and appveyor, for continues integration (TBD)

Some notes on UI/UX

Several points were taken into consideration while implementing UI/UX:

- User can convert same ticker multiple times. Silently cached and accelerated, no change on usage
- User can easily copy results to excel. A underlying table format is used to enable this

Create a prototype for the solution that should minimally include the Bloomberg ticker to CUSIP generator. The prototype can also be used to showcase capabilities like client-server interaction and user experience via an interface.

See enclosed project. `node` and `npm` are prerequisites (install from https://nodejs.org/en/) to run the project

Instruction to build-and-run:

```
$ cd quant-portal-backend
# install can take quite some time... be patient
$ npm i
$ npm run dev

# test the API
$ curl -v -X POST -H "Content-Type: application/json" \
    -d '{"data": ["AIH8 Index"]}' \
    localhost:7001/api/ticker2cusip

$ cd ../quant-portal-frontend
# install deps zzz...
$ npm i
# compilation will again take ~30 seconds. Startup only - hot reload is fast
$ npm run start

# use the web UI
# Open http://localhost:8000/#/form/converter using browser
```

State the assumptions used in the prototype that deviate from the actual solution. (e.g. Flask's built-in server was used in the prototype in place of a WSGI server, or that modules were hardcoded or delivered as a concept due to unfamiliarity in that area)

- From my research, the future contract is valid in the current month. E.g. if now is Apr 2018, it's possible to have Apr 2018 contract. If it's May 2018 now, then Apr 8 contract

will mean "Apr 2028". Implemented according to this understanding
- Users of this tool will be employees
- Reverse conversion CUSIP -> ticker is not needed and thus not implemented