Deadline: 11.05

Team members

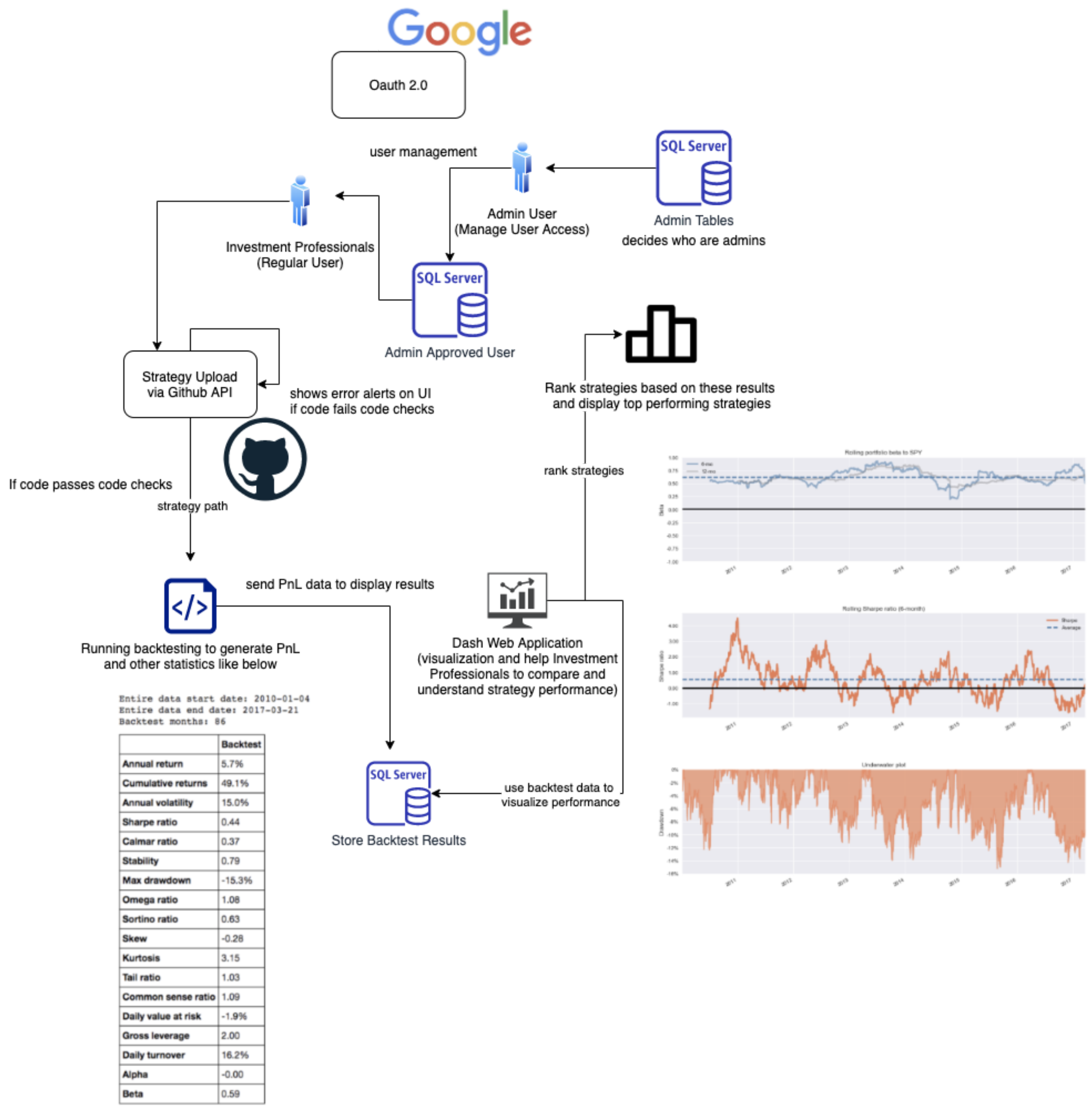
1. Zihan Guo: zg2391
2. Linxiao Wu: lw2944
3. Jialiang Zhang: jz3283
4. Erica Chenchen Wei: cw3137

Team Name
Alchemist

Part 0:

We met with Rohit on Nov 5th, 8:00 PM on discord. From our meeting, Rohit has provided us the following recommendations:

1. Change current login method to OAUTH 2.0 authentication to make sure that our service is secure.
   a. Solution: we have followed this advice and plan to refactor code to OAUTH 2.0
2. Consider adding CI/CD to conform best engineering practice.
   a. Solution: We will leave this as an optional feature for now, but might create CI/CD deployment between 1st iteration and final presentation.
3. Make sure that we always raise a PR when merging into master. (create feature, debug, fix branches) e.g. https://guides.github.com/introduction/flow/
   a. Solution: Roger that! We will make sure to do so.
4. Add ERD (entity relational diagram) and Class Diagram to gauge complexity and help everyone stay on the same page. (some kind of architecture diagram would be useful to visualize what our application would look like at its final stage).
   a. Solution: Added below.


Google
Oauth 2.0
user management
SQL Server
Admin User
(Manage User Access)
Admin Tables
decides who are admins
Investment Professionals
(Regular User)
SQL Server
Admin Approved User
Strategy Upload
via Github API
shows error alerts on UI
if code fails code checks
Rank strategies based on these results
and display top performing strategies
rank strategies
If code passes code checks
strategy path
send PnL data to display results
Running backtesting to generate PnL
and other statistics like below
Dash Web Application
(visualization and help Investment
Professionals to compare and
understand strategy performance)
SQL Server
Store Backtest Results
use backtest data to
visualize performance
Rolling portfolio beta to SPY
Rolling Sharpe ratio (6-month)
Underwater plot

```
Entire data start date: 2010-01-04
Entire data end date: 2017-03-21
Backtest months: 86
```

	Backtest
Annual return	5.7%
Cumulative returns	49.1%
Annual volatility	15.0%
Sharpe ratio	0.44
Calmar ratio	0.37
Stability	0.79
Max drawdown	-15.3%
Omega ratio	1.08
Sortino ratio	0.63
Skew	-0.28
Kurtosis	3.15
Tail ratio	1.03
Common sense ratio	1.09
Daily value at risk	-1.9%
Gross leverage	2.00
Daily turnover	16.2%
Alpha	-0.00
Beta	0.59

Part 1: Overview

1. What will your project do?

We are building a backtesting[1] platform that allows quantitative analysts and investment professionals to test their systematic trading strategies.

[1] https://www.investopedia.com/terms/b/backtesting.asp

a. Login feature for users (registration, login); Registration requires Admin approval. (using relational database)
b. A User will be able to upload a strategy (.py) file to test his/her strategy.
c. Run strategy backtest and display statistics and graphs.
d. (ML component) will select the optimal portfolio allocation and strategy recommendation to the user.

2. **Who or what will be its users? Your project must impose registration, authenticated login and timeout/explicit logout, so your answer should say something about this.**

We have three types of users:

a. Our users will be quantitative analysts or algorithmic traders who write strategy codes in Python and want to quickly test their strategy performance.
b. In addition, our users will be engineers who are in support roles to these investment professionals.
c. Admins can manage users in the admin page.
d. Authentication:
Use OAuth 2.0 authorization framework to authenticate user access. Users should register a valid google account before log in to our application, and users should be able to log in the backtesting platform through *Google login* page.
e. Access control:
   i. Use the Flask-Login library for session management.
   ii. Use Flask-Bcrypt for hashing passwords.
   iii. Use Flask-SQLAlchemy to create a user model.
   iv. Create login forms for admin users to control user log in.
~~f. Registration:~~ (Not necessary when using third party identity provider)
   - ~~User needs to provide a unique username, a unique email address, a password and the confirmation of the password to register an account.~~
g. Log in:
   - Users can log in to the application with the google email and password.
   - Resources can only be accessed after a user successfully login.
h. Log out:
   - When a user logs out. Next time a user comes, he/she has to login again.
i. Admin Control:
   - Admin can approve/disapprove a user's request to log in the application.

- Once the admin approves the user's login request, that user would then be able to login.

3. **Your project must be demoable, but does not need a GUI if there's a command line console or some other way to demonstrate. (All demos must be entirely online, there will be no in-person demos.) What do you think you'll be able to show in your demo?**
    a. We will demonstrate the workflow of a user from login (step 1) to uploading a strategy (step 2) to running backtesting (on server, step 3) to displaying strategy performance results (step 4) to strategy recommendation and portfolio recommendation using ML (step 5). In addition, we will show how the interface differs among different user types.

4. **Your project must store and retrieve some application data persistently (e.g., using a database or key-value store, not just a file), such that when your application terminates and starts up again some arbitrary time later, the data is still there and used for later processing. What kind of data do you plan to store?**

    There are several kinds of data that we need to store:
    a. User Data (~~UserName,~~ UserId, ~~Password,~~ Email, ~~Image,~~ Strategy): Using OAuth, we do not need to store user password our own
    b. UserStrategy (StrategyId, UserId, StrategyLocation)
    c. UserBacktest (StrategyId, StrategyLocation, BacktestId, BacktestS3Url)

    We store the above three types of data in SQL server/AWS RD. Strategies are uploaded to Github so we only need to keep track of its file location. When pushed to master, we need to set-up an automatic process to update the StrategyLocation. When a strategy finishes a backtest, we would store the backtest result in a S3 bucket. (e.g. s3:..../strategy_id/backtest_id.csv)

5. **Your project must leverage some publicly available API beyond those that "come with" the platform; it is acceptable to use an API for external data retrieval instead of to call a library or service. The API does not need to be a REST API. There are many public APIs linked at https://github.com/public-apis/public-apis (Links to an external site.) and https://github.com/n0shake/Public-APIs (Links to an external site.)**

What API do you plan to use and what will you use it for?

We will use REST API. Below are some example endpoints that we will build:

a. /login
b. /admin
c. /home
d. /upload_strategy
e. /save_strategy
f. /view_strategy
g. /run_backtest
h. /view_backtest
i. /save_backtest
j. /recommend_strategy
k. /recommend_portfolio
l. /view_ranked_strategy
m. /view_optimal_portfoli
n. /logout

To store files in GitHub programmatically, we also use GitHub API https://developer.github.com/v3/ with PyGithub module.

a. **create_git_ref**(*ref*, *sha*) -> **POST /repos/:owner/:repo/git/refs**
b. **get_branch**(*branch*) -> **GET /repos/:owner/:repo/branches**
c. **Branch.branch.delete()**-> DELETE /repos/:owner/:repo/branches/:branch
d. **merge**(*commit_message=NotSet*, *commit_title=NotSet*, *merge_method=NotSet*, *sha=NotSet*) -> **PUT /repos/:owner/:repo/pulls/:number/merge**

Part 2: User Stories

Write three to five user stories for your proposed application, constituting a Minimal Viable Product (MVP); registration/login/logout should not be included among these user stories, nor should 'help' or other generic functionality. That is, your application should do at least three application-specific things. Use the format

< label >: As a < type of user >, I want < some goal > so that < some reason >.

My conditions of satisfaction are < list of common cases and special cases that must work >.

The type of user (role) does not need to be human. You may optionally include a wishlist of additional user stories to add if time permits. Keep in mind that the type of user, the goal, the reason (if applicable within the system), all the common cases and all the special cases must be testable and demoable.

Users: Investment Professionals

<Strategy Upload>: As an investment professional, I want to easily upload all trading strategies so that I can get results and compare them smoothly. My conditions of satisfaction are <

Common case:

Investment professionals upload the valid trading strategies(.py) file and then they have options to save or not. Then they can choose to use which strategy to run backtesting. After backtesting is done, then can view the result and save to compare with others.

Special cases:

1. Investment professionals upload an invalid strategy, which can't be compiled, a warning message will prompt up to let them know.
2. Investment professionals upload a valid strategy but forget to save it, there's a window prompted up to ask if save or not.
3. Investment professionals forget to save results, there's automatic windows to ask if to save or not, all the results will be backed up for one week if not saved. >

Admins: security; manager users.

<**Access Control**>: As an <admin>, I want to <manage users' access to the system> so that this user can log in to the application and access relevant information. My conditions of satisfaction are <

common case:

Admin approves the user's login and adds the user to the User table. Admin disapproves a user's login by removing the user from the User table. ~~with valid information such that the user name is unique in the current user table, and the password is valid[2].~~

[2] We will display rules that define a valid password (minimal length, special character requirements, if can contain capitalized letters and at least how many, etc.)

Special cases:

1. Admin tries to approve a user's login request when the email matches an existing record in the user database.
2. ~~Admin tries to approve a user's registration without a username.~~
3. ~~Admin tries to approve a user's registration with an invalid user password.~~ >.

~~<**Access Control**>: As an <admin>, I want to <manage users' access to other users' strategy data> so that this user can access strategy data uploaded by other users and run his/her own backtest. My conditions of satisfaction are <~~
~~common case:~~
~~Admin allows the target user to access other user's strategy data. Otherwise, the user can only access his/her own strategy data and backtest results.~~
~~Special cases:~~

1. ~~Admin tries to grant the target user access to a non-existent strategy.~~
2. ~~Admin disallows a user to access a specific strategy that the user used to be able to access.>.~~

(The above is not needed because we use OAuth 2.0 framework with third party identity provider)

Engineers: easy to support IPs(investment professionals) and maintain system health.

< **System Analytics**>: As an Engineer, I want to know more about user activity and system health in an intuitive interface, so that I can support admins and IPs easily.

My conditions of satisfaction are <

common case:
When engineers log in, they should be able to see the UI of system output logs sorted by date, grouped user information (without private information) and system health . The system health is also illustrated by query per second, response speed, and other reasonable metrics.
Special cases:

1. User(IP or Admin) submits a report from the front end.
2. Service quality like response rate decreases.

>.

Part 3: Acceptance Testing

We discussed with the TA about the data and procedure of running strategies, users should upload the corresponding data along with the strategy to perform the backtesting.

Acceptance testing for <***Strategy Upload***>:

Common cases:

1. Investment professionals should be able to upload valid strategies and corresponding data with the strategy successfully.
2. Investment professionals should choose which strategies to run backtesting.
3. Investment professionals should be able to compare multiple backtesting results(the result will include a cumulative return graph to illustrate the profit and a table of statistical results) after running all strategies.
4. Investment professionals should be able to delete any strategies/data/results they don't want anymore.

Special cases:

1. Investment professionals can't upload invalid strategies and empty data/invalid data files.
2. Investment professionals can't view any results without strategies and data uploaded.

Acceptance testing for <***Access Control***>:

Common cases:

1. A user tries to login to the application using a valid google email account. After the admin approves the login, test success if a new user record adds to the user database, otherwise the test fails.

Special cases:

1. The application should prevent the admin from approving a user's login request with an existing email.
2. ~~The application should prevent the admin from approving a user's registration with empty passwords.~~

~~Acceptance test for <**Access Control**>:~~

~~Common cases:~~

1. ~~A user should only be able to download his/her own strategy and backtest results as default.~~
2. ~~Admin grants user A to access strategy B from a different user; user A should be able to download strategy B and run it for backtesting.~~

~~Special cases:~~

1. ~~Admin can't approve a user to download a non-exists strategy if the strategy is no longer in the strategy database.~~
2. ~~A user should not be able to download a strategy that the admin disapproves of.~~

Acceptance testing for <**System Analytics**>:

1. Engineers should be able to see the system log sorted by date once logged in
2. System service log should be displayed the function stack trace if opened.
3. When IPs or Administrator submits a report, engineers should be alerted once logged in.
4. When system health gets worse (e.g. deliberately delay response for 10 seconds), the system health chart should display the information
5. When engineers click the log for a specific date, detailed information like function stack trace should appear.

Acceptance testing:

Erica part -

Zihan part -

Part 4: Technology Chosen

Language	Python 3.8
Framework	Flask
IDE	PyCharm
Package Manager	Pip
Style Checker	PyLint

Unit Testing	Pytest
Test Coverage	Coverage.py[3]
Bug Finder	PyChecker, PyLint

[3] https://coverage.readthedocs.io/en/coverage-5.3/