# 16:332:568:01 Spring2018 SOFTWARE ENGINEERING WEB APPLICATIONS

# Course Project

# Project Phase 1 - Data Collection Module

# Design Report

Group 7
Zhongze Tang(zt67)
Zichen Zhu(zz313)
Weijia Sun(ws368)
Huayu Zhao(hz274)
Weitian Li(wl436)

02MAR2018

## 1. Intro

This phase of course project is to implement a spider to collect stocks information and store to database. We decided to use Python as our programming language and MongoDB as our database engine considering the size of the project, the development time scale and the number of useful third-party libraries. Python is a popular programming language in lots of area such as web development, machine learning, mathematical computation etc. with loans of helpful and useful third-party libraries like Flask, arrow, numpy requests etc. Although there are disadvantages in python like multithreading and operating efficiency, those disadvantages are not obvious in this size of project and in total the advantages are outweigh the disadvantages. We are using MongoDB instead of RDBMS database base on the reason that in this circumstance that each record of stock information is not associated with others we want high efficiency, scalability and ability of parallel more than integrity and keeping the relationship between records. And MongoDB can work well with Python.

## 2. Overview

In this phase, we use the API provided by https://www.alphavantage.co/ and the third-part library *requests* to deal with the HTTP GET operation, *arrow* to converting date, time and time zone, *pymongo* to drive the database, *apscheduler* to handling the task scheduling. And in general, we use docker to warp up the application and make it platform-independent. That means users can run the application with docker and do not need to install any others program in their computer or server. The application read the relative configuration from environment variables instead of hardcode relative information in the application.

## 3. Implement

There are 5 models in this phase: init_db, run_app, Utils, GetStockData and DatabaseUtils.

### 3.1. init_db

This function reads the configuration from environment variables. Then connect to database. By judging the existence of 'init' in collection flag and if 'init' is true, decide whether to initialize the database. If database need to be initialized, the function first drop collections: daily, flag, realtime. Second, create the collections daily and realtime with valida restriction. Step 3, initialize the indexes of collections with index timestamp in descending then symbol in descending and index symbol in descending and timestamp in descending. Last, get the daily historical stocks information from API and store in database.

### 3.2. run_app[*]

This function is the main entry point of the application. In this function, there is a scheduler to schedule the tasks that are get daily historical stocks information every minute and get real time stocks information every 3 seconds. And pass the parameters to relative functions.

### 3.3. Utils

This class only has on static method called get_env. This method returns the value of an environment variable name from system environment variables and raise an EnvironmentError if the variable does not exist.

### 3.4. GetStockData

Both get_daily_data and get_real_time_data in the class receive a string or a list of string of stock name(s). Getting the stocks information from API and convert the data into a list of dictionary of result. Both methods convert price into Decimal128, volume into Int64 in BSON type and time into datetime.datetime type. The get_daily_data returns a list of {'timestamp': *datetime.datetime*, 'symbol': *string*, 'open': *bson.Decimal128*, 'high': *bson.Decimal128*, 'low': *bson.Decimal128*, 'close': *bson.Decimal128*, 'volume': *bson.Int64*} and the get_real_time_data returns a list of {'timestamp': *datetime.datetime*, 'symbol': *string*, 'price': *bson.Decimal128* , 'volume': *bson.Int64*}.

3.5. DatabaseUtils

This class deals with insert and update data into database and get data from database. The save_daily_history and save_realtime get a parameter as a list of data as the format of returning of get_daily_data and get_real_time_data. And the method traverses the list using update with the query symbol from database equals symbol from the element in the list and timestamp from database equals timestamp from the element in the list and the flag upsert=True which means if the record does not exist insert the record to the database for each record in the list. The get_daily_history_symbols and get_realtime_symbols will return a list of string of name of stocks exist in the collection daily or realtime. The get_daily_history_by_symbol and get_realtime_by_symbol will return a list of sorted by timestamp record with the format of get_daily_data and get_real_time_data returning.