CENG 491 - Design Overview Document

Group 15 - AlgoTrading People

Algorithmic Trading Framework

Supervisor: Asst. Prof. Selim Temizer

Assistant: Çağlar Seylan

Members: Enes Aldemir, Bahar Aydemir, Ünal Akünal, Doruk Çetin

**Product Description**

Firstly, algorithmic trading is basically using computers to aid us in the actual trading process. We may know what to buy or sell and a computer can help us on when and how to buy it. It is the efficient actualization of a trading strategy, as an algorithm. Quantitative trading is number crunching to identify trading strategies, learn what to buy, determine which stocks would be profitable in the future.

Our project is a web-based algorithmic trading framework. A user can create trading algorithms using it, by visual interfaces. The user can backtest the created algorithms against historical and real time data. Our product also offers some simple tools for portfolio management. Additionally, we offer visualization in forms of plots and graphs to bring meaning to data. Lastly, users will be able to employ machine learning to optimize their algorithms.

Algorithmic trading has many advantages over traditional trading such as elimination of human emotions from the trading process, increased accuracy and speed, comfort and scalability. However, current trading and backtesting tools exceptionally offer solutions to people using programming practices. They are also tailored for people who already familiar with the concept of algorithmic trading. These two are what we are trying to solve within our project. Our product is approachable for the inexperienced. A person without any programming knowledge or familiarity to algorithmic trading should be able to start right away and experiment with their ideas. Therefore, our main goal is being user friendly by self explanatory, minimal interfaces. We are also designing it as a web application, so it is also lightweight. That means our product is easily accessible from practically anywhere connected to internet without spending any time for installing software. Lastly, it is compact as we offer a bundle of useful functionalities such as algorithm creation, stock monitoring and portfolio management in just one package.

**High Level System View**

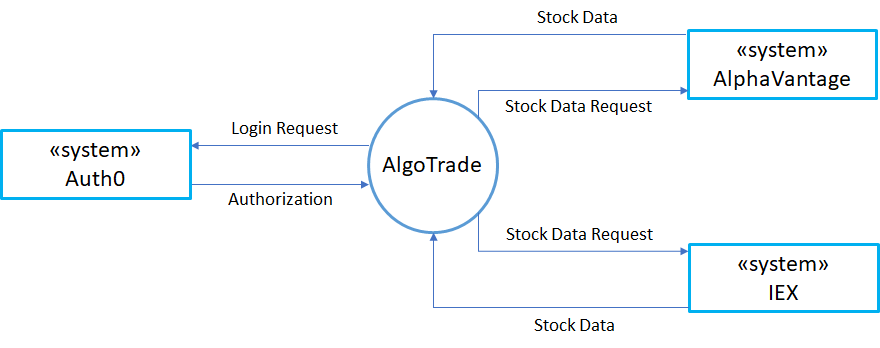


Figure 1 - Context diagram

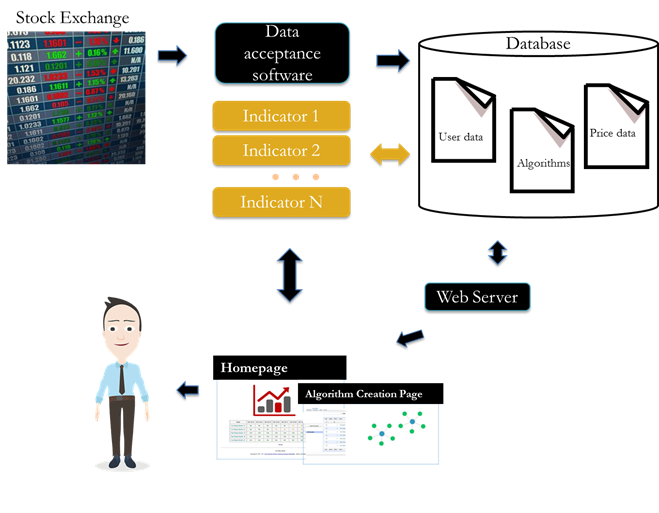
Context diagram presented above illustrates our product’s relationship with its environment by defining the external systems it communicates with. We do not directly implement a login/authorization feature and instead use Auth0, which is token based authentication platform. IEX and AlphaVantage are the APIs we use to get stock market data accordingly to our request. We can specify a particular time period and frequency for a given stock with our request.

In order to focus more on the functionalities for an algorithmic trading framework, we used these third party libraries to abstract authentication and data fetching operations. So the limitations that could arise from the usage of those tools are not in the scope of our development process. For instance the available markets to fetch stock prices from, and their frequency of data etc. is not our main concern, as long as we could create logical algorithms with our indicators and perform backtesting on top of currently available ones. Likewise, if we could detect a particular user is logged in and fetch their previous data from the database using their token and/or id; then the token system and the user login logic is considered good enough for this project.

**Overall Design**

The application will be web-based for being easily accessible, user-friendly, and easy to start for the users. Roughly, the application can be divided into three main components and can be further defined as follows:

* Algorithm Creation Interface: This supports composing an algorithm from different indicators, using graphical user interface bindings.
* Back Testing Client: This allows users to test their algorithms against previous real-world data, fetched from various sources.
* Optimization & Feedback: The parameters used inside the indicator are optimized using various machine learning techniques to give the user feedback about his/her system.

Figure 2 - Graphical representation of overall design

Majority of the data is taken from stock market and put into the database. The database consists of user, algorithm and price data. Then, it can be accessed from a web browser. There are two main pages namely homepage and algorithm creation interface. Live algorithm, comparison of investment options, self made algorithms and their graph visualizations are presented on the homepage. On the algorithm creation page, from a list of existing indicators, which are placed on the left side of the screen, a user is able to create new algorithms by drag and drop. After the creation of the algorithm, the user is able to test them by using historical or real time data. The newly created indicators and algorithms will be written back to the database for future uses.

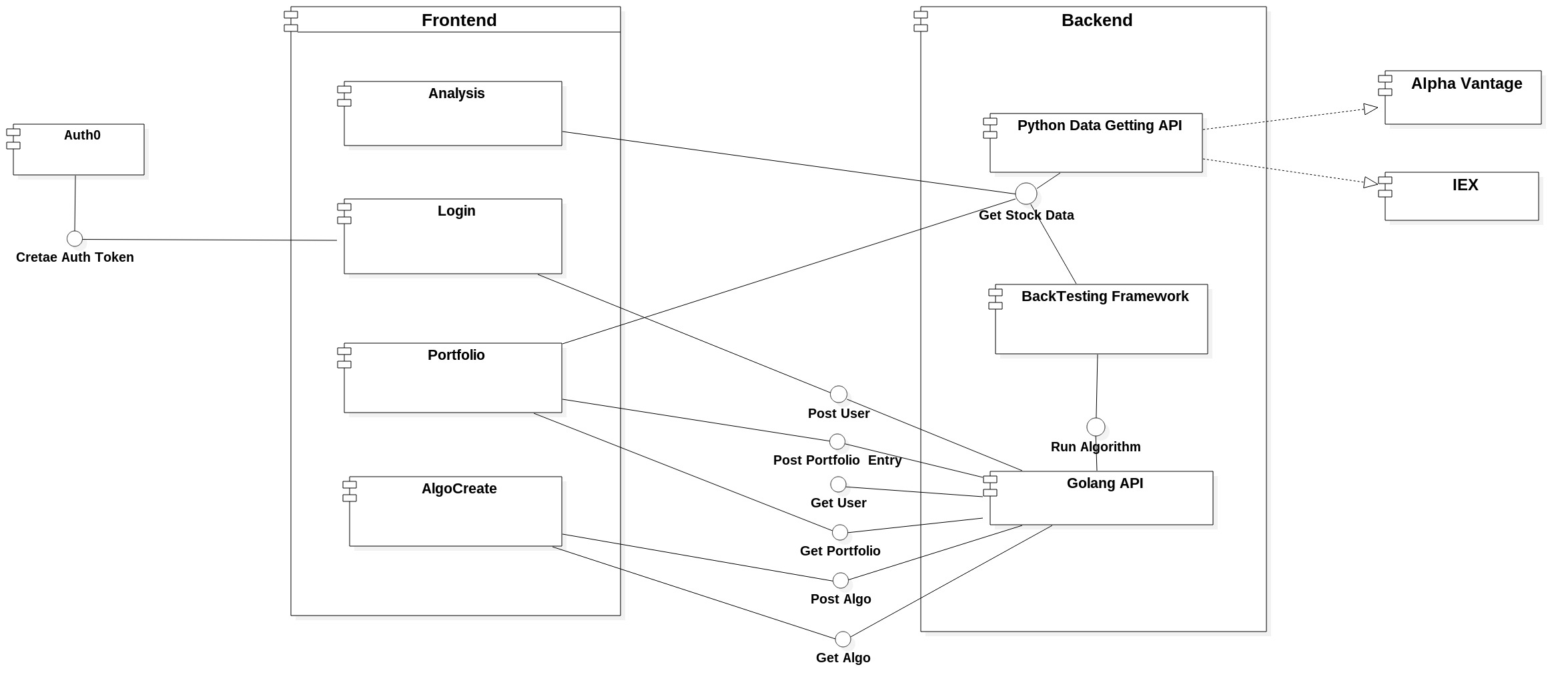


Figure 3 - Component diagram

As can be seen from the component diagram above, our design basically contains several back-end services to implement the microservice infrastructure and to support the varying needs of a comprehensive, lightweight and interactive framework on the front-end side. The four main functionalities of the system can be described as follows:

**Analysis:** Using our analysis tool, the users will be able to test their algorithms on historical datasets (this procedure is called backtesting, hence the name of the component in back-end side). Note that the analysis operation is also available against a stream of real time data, and all of these operations are done via data getting API and backtesting framework that lies inside of the backend code. It would be appropriate to state once again that the data we are fetching completely relies on the capabilities of the third party libraries; which means neither the support for different markets nor the available frequency for that market’s data is not in the scope of our product.

**Login:** The user logic depends mainly on the front-end code which not only gets the token and constructs the user for the first time but also consists of multiple ways to support signing up from different kinds of account types. If the user is not signing up but signing in, then we check if the token for that particular user already exists in the database or not.

**Portfolio:** In the context of our project, portfolio means a list of stocks that a particular user selects to keep track of the changes. Considering the variety of tools available for portfolio management, we only limit ourselves to provide these basic functionalities, nevertheless, the necessity for this convenient helper tool derives from the fact that the users may want to find the optimal input for their algorithms by looking through the current stock prices, without leaving our platform.

**AlgoCreate:** Our main focus in terms of productivity and usage for the product is the algorithm creation aspect. This interface provides an easy way to come up with different approaches to algorithmic trading and creating them by dragging and dropping basic building blocks. Every user is able to save their own algorithms and the database keeps them in a flat way to support a variety of indicators which can be bound together to implement the functionality in mind.

Note that those four parts are distributed amongst the website’s different tabs, and as a whole constitute the front-end code. The only third party library used here is Auth0, which is for the user authentication. From a higher level perspective, the system can be seen as an array of distinct components for different functionalities, but those abstractions are provided by defining a user class, which has a portfolio (list of assets) and a list of graphs (the serialized form of previously created algorithms). All of the stocks and stock price datasets are saved inside different collections, and can be fetched by different client processes. The back-end service **GoLang API** provides all of those functionalities for stocks, users, graphs and datasets; while **Data Getting API** uses third party libraries to fetch stock price data and convert it into necessary forms for different kinds of tasks, **Backtesting Framework** connects to both of those services and by fetching the graph structure implements a series of mathematical operations on top of the datasets in a topologically sorted order. There is also an **optimization** choice during backtesting framework, in which we use machine learning algorithms to optimize the parameters of the indicator and other building blocks. Note that this is an optional operation for the backtesting process, and it is not called unless explicitly defined by the user. The resultant buy or sell flag can be used to detect when to do which operation for a particular asset given as an input node to the graph.

**Alternative Design Options**

In the early stages of our discussions regarding the overall design of our project we weighed the options of developing our product to desktop, mobile and web platforms. Our primary aim has been to provide accessibility and ease of use from the very beginning so we discarded the option of developing a desktop application as it would not be approachable for our intended audience. Web applications have proven to be versatile on various platforms and they are not additionally demanding by means of use if the interfaces kept self explanatory, so we had chosen to develop our project as a web application. The idea of a mobile application also seemed to us as a comparably appropriate one and we have chosen to indicate it as an optional feature as we would like to have native mobile applications as well, if time allows.

Another design choice was about determining the stock exchange which is the main data gathering point of our project. Firstly, we wanted to use data from “Borsa Istanbul” which is the primary and only stock exchange of Turkey. However, to obtain real time data there are only professional services in exchange of high costs about 10000 TL per month. Among the free choices, we could obtain 15 minutes delayed data per minute as the best option. Therefore, we have decided to use “NASDAQ”, the New York based American stock market, in order to gather higher frequency data. Currently, we can refresh the prices up to ten times per minute and this rate is sufficient to develop and test our created algorithms.