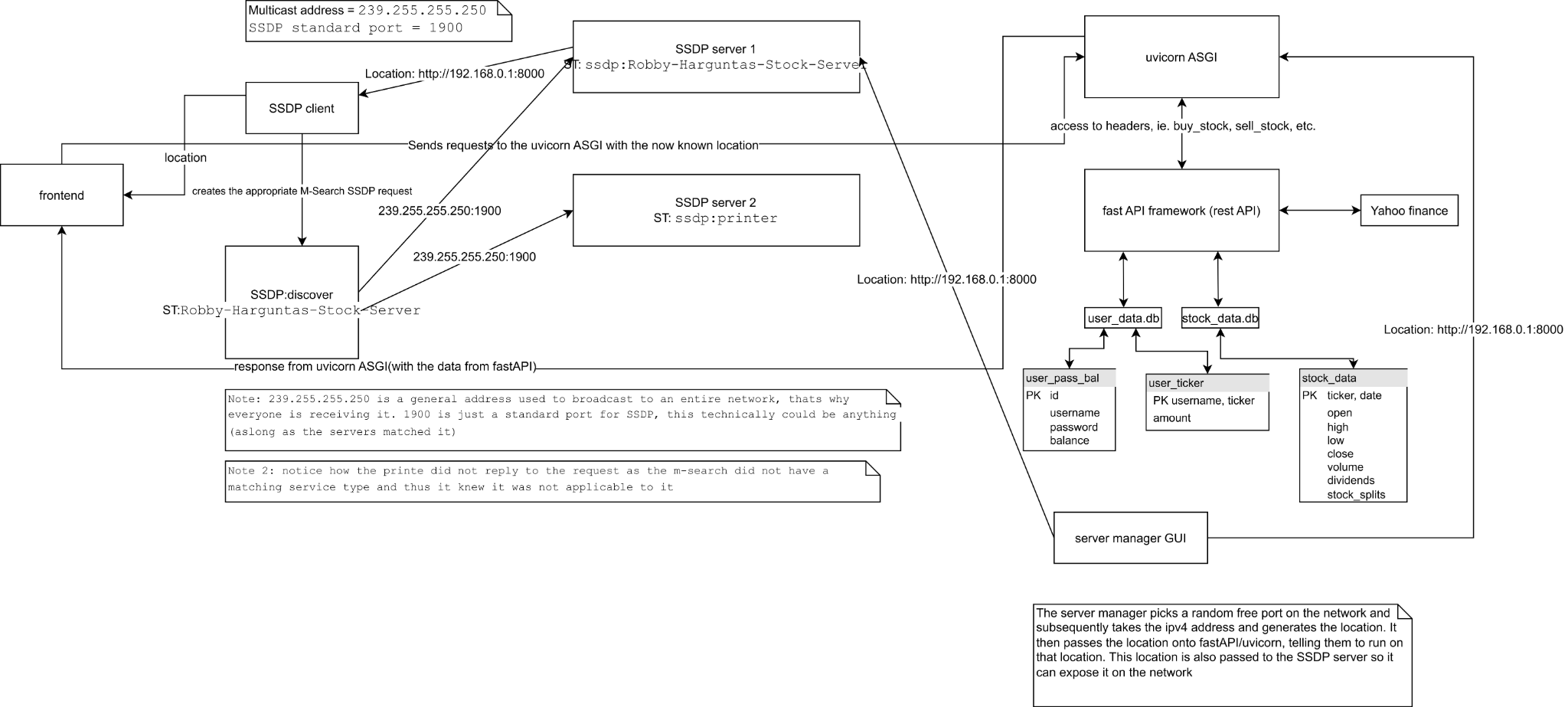
High level diagram:



The backend server can be broken down into four main parts.

-A SSDP (simple service discovery protocol) client/server

-a rest API (which has its ip + port exposed by the SSDP server)

-user\_data sqlite3 database for storing user credentials and owned stock data

-stock\_data sqlite3 database for caching the price history of stocks (optimization)

SSDP:

-SSDP is commonly used to allow IOT devices (a tv, printer, etc.) to be discovered on a network. A real world application, for example, could be a TV with the ability to use your phone as a remote through their app. (This is actually something I discovered on my network through messing around with SSDP). You could probably also find devices like this using my SSDP\_client.java and changing the service type to search for (“ssdp:all” will give you all the SSDP compliant devices on your network)

-In this program, SSDP is used to allow anyone on a network to host their own servers and have it automatically be discovered by the client (in this case our frontend) application. The intended use case for this was to enable multiple teachers, on the same network connection, to host their own server so they could manage their own students.

-SSDP works broadcasting UDP packets across an entire network. (Technically it's multicast however, I don’t really understand the difference and in practice, they functioned exactly the same) These packets contain specific headers which allow them to be identified as SSDP and subsequently allows the client devices to determine if that is a service they want to work with. For instance, an SSDP packet would contain a man (manufacturer assigned unique identifier), service type (usually “ssdp:” followed by the purpose of the service, for instance, printer or mediaStreamingDevice) and a user agent (who is the device making the request, for example, chrome actually has SSDP built in [idk why, maybe casting content or sharing data] and subsequently includes a user agent stating that it is chrome, and what version it is running). There are many more headers that can be included which describe different properties of the service/hosting device, i.e mx, host, etc.

-In this case, the purpose of the SSDP server is to expose the location of the rest API (it's ip address + port) so it can be found and subsequently used to make requests to the fastAPI rest api and get/modify data on the two databases and yahoo finance.

-Clients (those looking for a server) broadcast M-search packets with the ST of the server they are looking for.

-The server waits for m-search requests and subsequently if the ST matches, they reply to it, in this case, replying with the location of the rest api, thus allowing the application to be discovered.

-Note, this only works because every single device on the network is receiving the m-search request, they’re all just choosing to ignore it except the correct SSDP server

-in this case, our frontend waits for the SSDP client to find a location and subsequently once received stores it and just interacts with the unicorn ASGI (rest api)

-SSDP is the first step within UPNP (Universal plug and play), see [this book](https://openconnectivity.org/upnp-specs/UPnP-arch-DeviceArchitecture-v2.0-20200417.pdf)(page 18-39)

Rest API:

The rest api can further be broken down into two parts:

*Fast API:*

-FastAPI is a rest API web framework

-It allows us to create a template for a web server (different web headers, in this case buy\_stock, sell\_stock, etc.) and subsequently implement functionality for them

-This in itself, is not a server and cannot be run (you can run it technically, but that's because it comes packaged with its own development server which is not very performant and not the intended use of the framework anyway)

*Unicorn:*

-Unicorn is an ASGI (Asynchronous Server Gateway Interface)

-The ASGI is a web server intended to manage and balance the load for fastAPI (through multiple workers, which in this case we do not have)

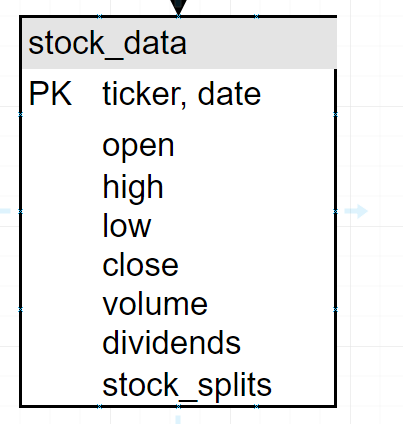
-uvicorn and its web server receive all of the http requests, take the headers and subsequently pass it to the corresponding methods in the fastAPI web server template for it to be executed. The response is then sent to unicorn where it can be passed back to the client.

-manages fastAPI and its state, so for example, if the fastAPI template crashes (my code sucks), it will acknowledge that and restart it.

Databases, their schemas and purpose:

*Stock\_data.db:*

Stock\_data table:

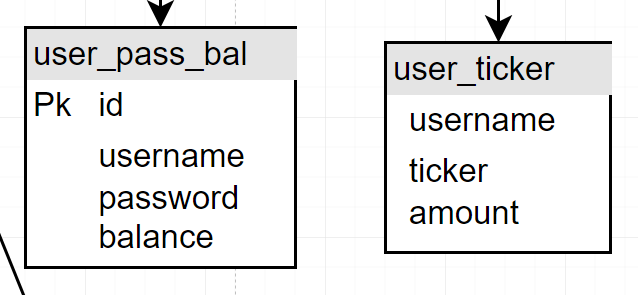


-this database serves a cache. Whenever we want to retrieve the price of a stock we first check the database. If the database does not have what we are looking for, we call the yahoo finance api, retrieve the data and then write it to the database in case we need to make that request again. We can do this because historical stock prices don’t change, so the only thing we need to worry about is whether or not we have the latest sets of dates.

-See database\_manager.py get\_stock\_history\_by\_ticker method for the logic I am using to determine whether the database is up to date or not

*User\_data.db:*

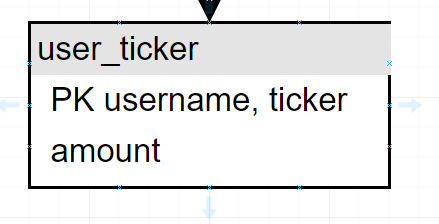
User\_pass\_bal table

**

-This table is responsible for holding username, password, user balance and a unique id. This id is used to authenticate users without storing their username/password( more secure, but technically it should be a refreshing session key)

One to one relationship, for every id/username there will be one password and one balance

User\_ticker table



One to many relationship. A username can have multiple entries, each with their own unique ticker. And there can be multiple users all owning the same stock but with different amounts.