**Memo**

To: Professor Pisano

From: Prithika Ganesh [pvganesh@bu.edu](mailto:pvganesh@bu.edu), Ignacio Nunez Gomez [inunezg@bu.edu](mailto:inunezg@bu.edu), Suyash Bhatia [suyashb@bu.edu](mailto:suyashb@bu.edu), Zen Mae Lee leezm@bu.edu

Team: Team 30, DevUp

Date: 11/20/22

Subject: The Art of Valuation: First Prototype Testing Report

1. **Summary of Equipment and Setup**

1.1 Required Materials

1.2 Set Up

1.3 Testing Procedure

**2.0 Measurements Taken**

2.1 Measurable Criteria

* 1. Measurements Taken Table Results

**3.0 Conclusion**

**1. Summary of Equipment and Setup**

***1.1 Required Materials***

Software:

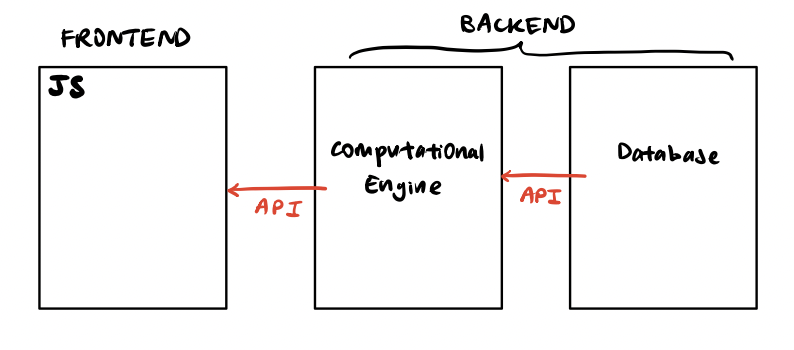
* Emulators: XCode or AndroidStudios
* Languages: ReactNative and Python 3
* Frameworks: Flask (Python)
* Data APIs and API keys: IEX Cloud Apperate

Hardware:

* Personal computing device i.e. laptop, PC etc.

***1.2 Set Up***

The setup requires running the frontend and backend code on an iOS/Android emulator. The Visual Studio Code IDE will be used as our main platform. Once the frontend is up and running on the emulator, the backend code runs in the same directory to fetch data from the database with an API, computed in our computational engine (e.g. using Price and Earnings to obtain P/E value), and then another API will fetch the computed data, which is currently displayed on the web. This mechanism will eventually be the basis of our application once a fully functional frontend is completed.



*Figure 1: Illustration of Data Flow from Backend to Frontend*

***1.3 Testing Procedure***

1. For the frontend, open VSC and navigate into the *workshopfinance* directory
2. Run ‘npm start’ → ‘a’/’i’ for Android/iOS.
3. Run the routes.py file (i.e. python routes.py) in a new terminal to initiate running the backend.
4. Open the localhost URL from Flask: shows the data fetched from the database, which has been computed in the computational engine and fetched by another API to be displayed on the web API URL.

**2. Measurements Taken**

***2.1 Measurable Criteria***

The criteria for successful running and output are as follows:

1. The frontend is able to successfully run on both IOS and Android emulators without crashing or any errors.
2. User input is able to be taken in by both the IOS and Android emulators, and can successfully be relayed back to the user through an alert.
3. Backend is able to take a set of comps, the target company, and the valuation method and fetch appropriate market data from API for public companies.
4. Backend computation engine takes in data from API calls as well as the valuation method, and is able to make accurate calculations, as manually verified.

***2.2 Measurements Taken Table Results***

| **Action** | **Category** | **Y/N?** |
| --- | --- | --- |
| Backend obtains data from database with IEX Cloud APIs | Backend | Y |
| Computational engine computes data correctly | Backend | Y |
| Application runs smoothly on iOS/Android emulator | Frontend | Y |
| Both iOS/Android systems are able to take in and display user input | Frontend | Y |
| **Result** | | 100 % |

**3. Conclusion**

Our first prototype's main goal was to establish our application's core components: the API fetch, computational engine, and front end.

The backend was systematically divided into data gathering/fetching and financial computation. The main method being tested was *get\_metrics*, whichtakes a list of company ticker symbols (Ex: AAPL) and returns various financial metrics in the form of a two-dimensional list. This list contains metrics like *share\_price, revenue, EBITDA, EBIT,* etc. for each company in our input list. In our testing, we were able to accurately gather this data from the dataframes. The second backend element tested was the computational engine which takes in a list of company data and computes the required financial multiples (i.e. price to earnings, enterprise value/EBITDA, enterprise value/EBIT, enterprise value/revenue), generating the comparable intrinsic values of the target company. By running this script several times with different inputs and displaying the results, we ensured the accuracy of the computational engine script. The team also manually calculated these values to ensure correct output.

On the frontend, the current state of the user interface involves a series of inter-connected screens that navigate to each other. The frontend also contains a page that takes in user input and saves the values upon a button press. For testing purposes, the user submitted values are displayed as an alert to verify appropriate storage. By testing that the frontend interface works consistently for both iOS and Android systems with a variety of input values, the team was able to ensure smooth app development for both platforms.

The criteria used to measure the first prototype essentially ensures that these components have no compilation issues as of date and function smoothly. Integrating the backend and frontend (i.e. displaying data from the computational engine on the frontend for the user to view) is currently a work in progress and the primary next step after completing the frontend for the application by internal deadlines set by our team.