**CS673 Software Engineering**

**Team 3 - StockPickers**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Sylvie Xiang | Team Leader | *sylvie* | 05/24 |
| Serguei | Design Leader | *Serguei* | 06/12 |
| Duy | QA Leader | *Duy Nguyen* | 5/28/22 |
| Jing | Security Leader  Requirement & Configuration Leader | *Jing Tian* | 5/24/22 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| V1 | Serguei | **5/28/22** | Initial version |
| V2 | Serguei | **6/12/22** | Updated version, incorporating design changes and feedback |

[Introduction](#_heading=h.gjdgxs)

[Software Architecture](#_heading=h.30j0zll)

[Database Design (if applied)](#_heading=h.1fob9te)

[Security Design](#_heading=h.tyjcwt)

[Design Patterns](#_heading=h.3rdcrjn)

[Key Algorithms](#_heading=h.3dy6vkm)

[UI Design](#_heading=h.3znysh7)

[Classes and Methods](#_heading=h.26in1rg)

[References](#_heading=h.4d34og8)

[Glossary](#_heading=h.17dp8vu)

# Introduction

This document provides a high-level design architecture of the StockGetter app. The main design goals of the software system are:

* Intuitive UI, facilitating convenience and ease of use for the end-user
* A responsive system getting data and generating forecasts in the minimum amount of time needed, thus enhancing user experience
* A robust/modular/OOP-based design, facilitating maintainability, extensibility, and future scalability of the system

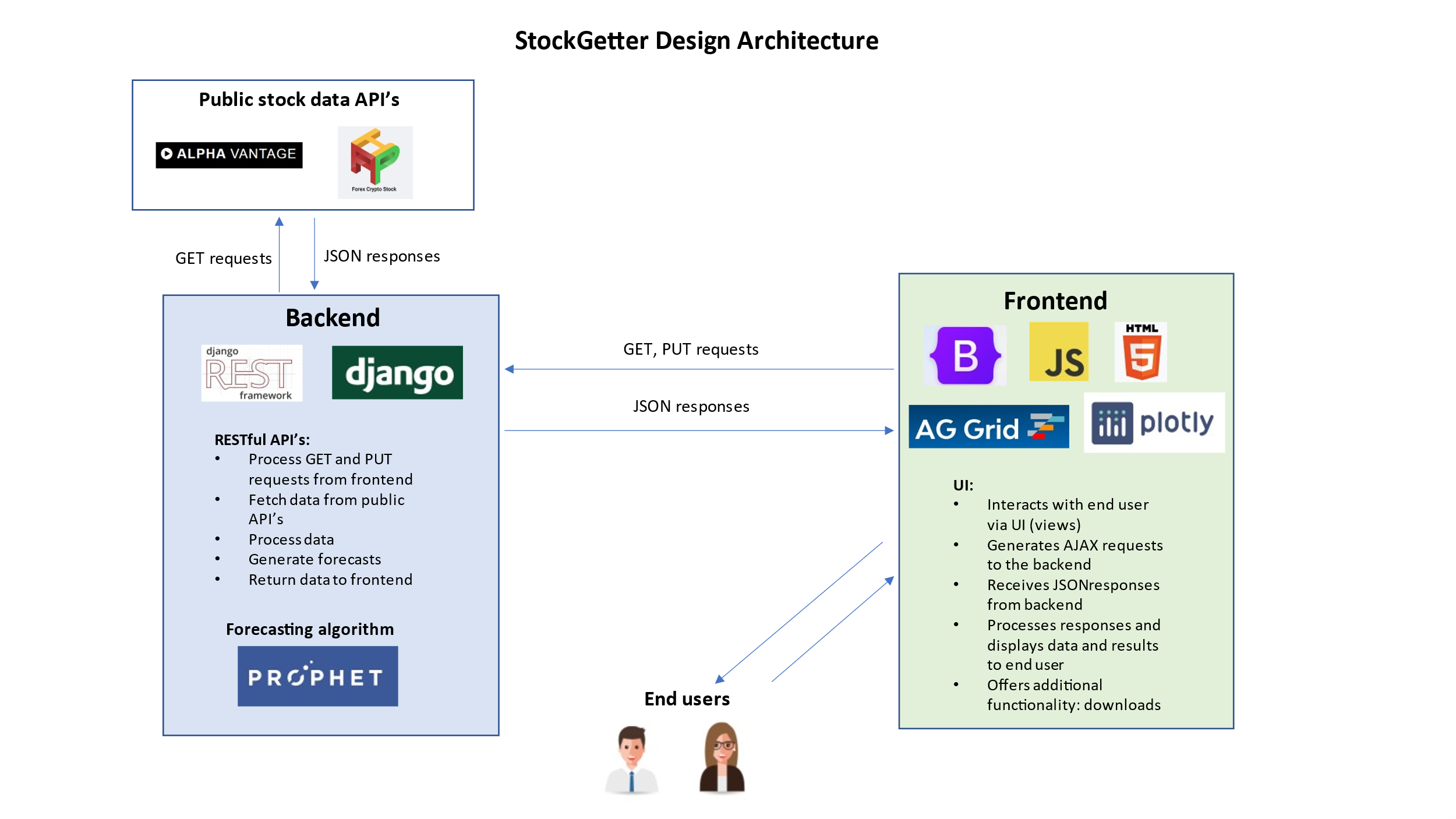
# Software Architecture

This section describes the software architecture of the system (see diagram). The overall architecture is based on the server-client principle, with the web based front end interfacing with a Django-based backend via a RESTful API implemented on the backend and utilizing GET and PUT requests.

The frontend will be developed using HTML, CSS, and JavaScript utilizing the Bootstrap 5 framework, and using an array of JavaScript libraries (AgGrid, Plotly, etc). The UI will be implemented in the frontend, and the frontend will interact with the backend via AJAX requests (implemented in JavaScript), with data passed between the frontend and backend using JSON string objects. The frontend will be responsible for the UI and related functionality, with all of the data-processing and forecasting work done in the backend.

The backend, implemented in Python, using the Django framework, will offer a number of services via a RESTful API. This will utilize the Django REST framework library. These services will in turn be used by the frontend to interact with the end user. Specifically, the services offered by the backend will be getting data from public stock API’s, processing data as needed, and generating forecasts. The backend will accept GET and POST requests from the frontend and will return data using JSON objects to the frontend.

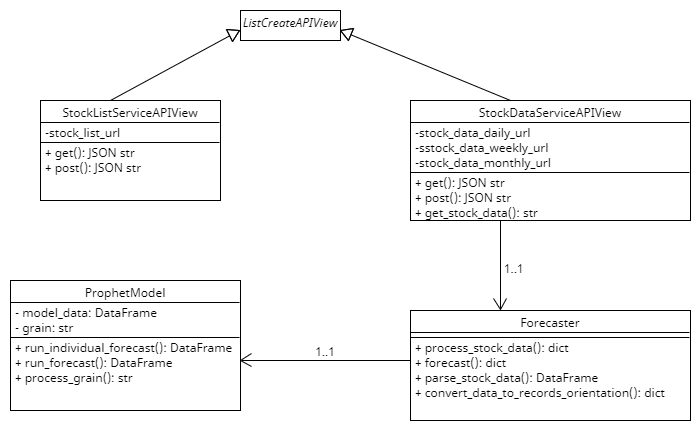
Being a Django framework, the backend will technically follow the MVC design philosophy, but we will be using Django views as end-points for the RESTful API’s, rather than for returning web pages. The diagram below describes the high-level overview of our architectural design.



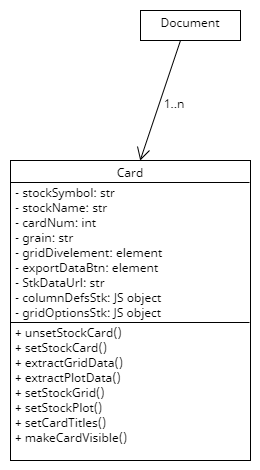
# Class Diagram

Below are the draft (subject to change per implementation) diagrams for the main components of the backend and frontend:

**Backend:**

****

**Frontend:**

****

# UI Design

In addition to the app landing page and the About page, the user will interact with a flexible/dynamic (single page web app) page, where the user will be able to select a stock symbol from the table, and get historic or forecasted data for the stock. The user will be able to get data for up to 5 stocks in this manner, with stock information displayed in the cards on the page. The user will be able to clear all cards and start anew. The user will also be able to download plots and underlying data for an individual stock or all stocks currently displayed in cards. The rough wireframe for the main interaction page is displayed below:



# Database Design

We will not be persisting any data for the POC of our software, hence we will not be implementing a database for our product.

# Security Design

The product will implement robust security using the built-in Django security settings and proper practices. Specifically, the product will have the following security settings:

* CORS cross-domain protections specified in the backend, allowing only the front-end to interact with the backend (and excluding all other domains)
* API keys and other data stored in configuration files and loaded at runtime for each developer or deployment. These will not be committed to the repo by including these in the .gitignore file.

# Business Logic and/or Key Algorithms

The key algorithm in our implementation is the Facebook Prophet ML algorithm for forecasting time series data: [Prophet | Forecasting at scale. (facebook.github.io)](https://facebook.github.io/prophet/). This will be implemented on the backend, along with all necessary data pre-processing algorithms needed to interact with the prophet forecasting algorithm.

# Any Additional Topics you would like to include.

# References

# Glossary