Summary

In this Analysis of Alternatives we will evaluate our development approach and its alternatives.

This document describes a high level overview of the options available to us (dev team) for use in designing the widget requested by the client. Recommendations will be made towards the end.

Based on communications from the client, we have to build a Covid-19 dashboard widget that tracks data/trends. Lower-level details about the widget will be discussed with the client as development continues. So our current goal with this analysis is to **make high level decisions** on the **basic infrastructure** of a dashboard app.

The Terms of reference section details the criteria for selection, we will be focusing on **low development cost** options that fits the **team's specialization** and results in a **high-quality user experience.**

In short, we recommend developing a web app, following the Model-View-Controller design architecture, and implementing features with the Python + Streamlit tech stack based on the criteria we identified.

Terms of reference

We want to choose a platform, an architecture, and a tech stack to support our development of an interactive Covid-19 dashboard app.

The criteria for platform choices are mainly influenced by usability and cost of development.

The criteria for architecture are mainly influenced by a need for high level design & clarity.

The criteria for tech stacks are mainly influenced by team knowledge.

<table>
<tr><th colspan="3">DECISION CRITERIA</th></tr>
<tr><td>Platform</td><td>Architecture</td><td>Tech stack</td></tr>
<tr><td>Cross compatibility</td><td>Maintainability</td><td>Team specializations</td></tr>
<tr><td>User experience</td><td>Flexibility</td><td>UI design capabilities</td></tr>
<tr><td rowspan="3">Scalability & Cost</td><td>Clarity & Readability</td><td>Support for data analytics</td></tr>
<tr><td rowspan="2">Compatibility with platform choice</td><td>Tech stack complexity</td></tr>
<tr><td>Refactoring costs</td></tr>
</table>

Platform

A dashboard requires real time data from a reliable source, and formats the raw data into a presentable format. An interactive dashboard would also need to allow user manipulation of said data. All these features can be implemented as a web app, mobile app, or desktop app; as such, we will compare between these platforms.

PLATFORM PROS & CONS			
Criteria	Web App	Mobile app	Desktop App
Cross-platform compatibility	Yes (anything that has a modern web browser works)	No, app must be designed for a specific OS.	
User experience	Easier user onboarding (just load the page to try) Latest version is loaded on page refresh	More difficult user onboarding (must download & set up app) Latest version must be downloaded and installed	
Scalability & Cost	Computation is centralized on server host Application logic must be processed by server, browser only displays result Higher load on server-side, increasing users requires better/more servers Web app is developed once, and available across any platform that supports modern web browsers	Computation can be decentralized Application logic can be local to user devices, only needs to retrieve updated data Lower load on server-side, increasing users have less impact on server load Very costly if we want to make app available across platforms.	

Architecture

MVC architecture.

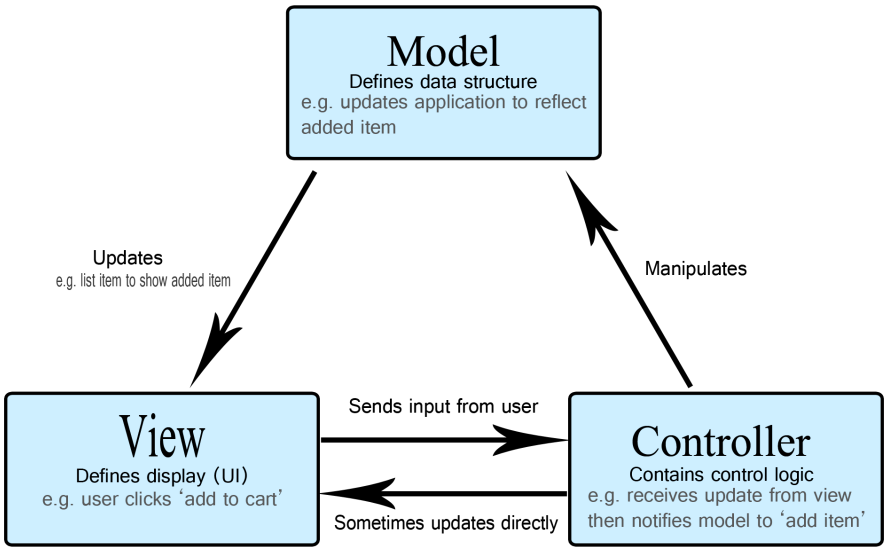

Image retrieved from MDN Web Docs

Some explanations:

Model

- Format COVID-19 data imported from an api
- Compute and store charts/graphs

View

- Display charts/graphs along with other pertinent info
- UI elements that users interact with

Controller

- Sends the charts/graphs from Model to View
- Decides what to 'do' when the user presses a button or input something, modifies the Model if necessary, and then sends the results to View

MVC. (2021). In MDN Web Docs. Retrieved August 23, 2021, from https://developer.mozilla.org/en-US/docs/Glossary/MVC

Why MVC Architecture?	
Development Costs	Clear separation of app components into Model, View, Controller allows delegation of implementation to several developers, increasing development efficiency. Compartmentalization of app components support best Object Oriented Programming practices. Simple architecture means no micromanagement of implementation details. Provides guidelines to the team on how to approach OOP in the context of this project.
Maintainability	Compartmentalization supports OOP best practices which allows developers to keep track of dependencies, making it easier to make updates to any module without breaking the entire codebase.
Flexibility	Highly flexible in implementation with any tech stacks or platform. Multiple ‘View’ components can be added without any changes to business logic in Model or Controller. So it is features-scalable. Different development tools (e.g languages) can be used in each of the MVC components, they just need a way to communicate with the others.
Clarity & Readability	Design clearly defines where each type of app logic is located which helps in identifying bugs and problems.
Compatibility	It's an industry convention to use MVC for web app development. The architectural framework fits well with the Web App platform. Even if we are developing desktop or mobile apps the design still applies well.

Tech stack

Our team has identified two tech stacks that can be used to make a dashboard web app, explanations here:

1) Javascript, HTML5/CSS and ReactJS

    HTML5 is a descriptive markup language that modern web browsers use to structure web pages.

    CSS is a language that provides styling options to web elements.

    Read this for full details on the difference between HTML5 and CSS.

    Javascript can be used to add dynamic elements to a web page, and also provide the programming logic of the web page. Its scripts can be directly embedded by HTML5 to add interactive functionalities to the web page. In comparison, Python scripts cannot be directly sourced by HTML5 in a presentable manner easily without the use of additional frameworks.

    ReactJS is a handy Javascript front-end framework that adds even more interactive elements to the web app.

2) Python and Streamlit

    Python has many libraries for data analysis/handling. However it is not as powerful as Javascript when it comes to adding interactive elements on the front-end (without front-end frameworks). So we need a front-end framework for Python to make a web app.

    Streamlit is a Python framework designed for data scientists to quickly build data apps (like dashboards). It provides Python methods to deal with HTML markups, so we won't have to deal with any other front-end tools. It has everything we need to build a simple dashboard app for fast presentation to clients and prototyping of features.

    With this tech stack, we can also tack on other tools when trying to make a better UI

PYTHON VS JAVASCRIPT		
Criteria	Python powered tech stack	Javascript powered tech stack
Team specializations	We are good with Python	We are new to Javascript
UI design capabilities	Limited range of front-end designs, UI is very basic and bare.	Very robust customization options for the front-end. Able to implement almost any front-end functionalities.
Support for data analytics	Many good libraries to handle/analyze data like pandas and matplotlib. Unlike Javascript, Python supports different types of numerical values like integers and floats.	There are good data *visualization* libraries (e.g InfoVis), but data cleaning and actual analysis options are not great. Javascript only has 1 type of numeric value, a float. This can cause bugs in calculations.
Tech stack complexity	Simple tech stack with Streamlit, only dealing with Python Need to setup external dependencies like Rust and MSVC, these files could be quite big	Unfamiliar tech stack with many interdependent tools Little to no dependencies outside the tech stack, easy to set up
Refactoring costs	Since we are only dealing with Python, it will be easier to make sure dependencies are at a minimum and make refactoring easy.	Due to unfamiliarity and the amount of interdependent tools, there is a higher risk of tightly coupled-code making refactoring hard and costly.

Recommendations

Platform: Web App

We think making a Web App is the best platform according to the criteria. It is true that Web App incurs an operating cost in the form of hosting services, but that can be justified with much lower development costs and a better user experience. Also, web hosting prices are very competitive and cost per user is very low.

Architecture: MVC

We talked about why we choose MVC above, here we share some thoughts on the most influential criteria:

1. Development Cost: We wanted an architecture that supports OOP practices to better maintain a cohesive codebase, MVC complements OOP so that's great
2. Compatibility: MVC is also the most widespread architecture for web applications so there are a lot of resources online that we can refer to on best practices
3. And overall, MVC is a very intuitive architecture that every team member understood

Tech stack: Python

While a Javascript tech stack is more powerful in making a better front-end experience, we still think Python is a better choice because it is much easier to get started and fits better in other criteria. We did not give that much weight to front-end development capabilities because our architectural design allows us to simply tack on the javascript tech stack onto View if needed.

The most influential criteria in making a recommendation on the tech stack is our team's expertise, which mainly lies with the Python tech stack.

Python has libraries that enable us to process data (panda), visualize data (matplotlib) and numbers handling (numpy). We need these capabilities in developing our dashboard app's business logic.

Python is capable of having 3 distinct types of numeric values: int, float, and complex. The data types allows us to statically type hints in our code which helps us better manage dependencies and pre/post conditions checking. Javascript only

has 1 type of numeric, the float, which puts it at a disadvantage in producing rigorous code.

The Python tech stack is also more ‘compact’, only needing the Streamlit framework and some libraries to make a web application. In comparison, Javascript tech stack is much more complex because of the different essential tools (Javascript + HTML5 + CSS) and we are just not that familiar with these.