Aditya Khanna

10 November 2019

Software Architecture – Business Blueprint (BB)

Business Blueprint Topic: Climate Change Data Aggregation Web Application

Table of Contents:

<u>Part 1: Prioritize Stakeholder Qualities/Constraints and Associated Quality Categories</u>

Table 1. Stakeholder Qualities/Constraints – Climate Change Data Web Application Domain

Quality/Constraint	Description
Flexibility	The climate app is used by a wide variety of people with different goals. It is necessary for it to be flexible enough for different people to use. It is critical for the climate app to attract a large number of users because the service is free, so all funds come from advertising. It should include many features and functionalities that allow the different target markets (environmentalists and real-estate investors) to see environmental conditions and reach their end goals.
Reliability	Software reliability is the probability of failure free software operations for a specific time in an environment. Failures are due to inadequate testing, human error, and unexpected use. In the case of this application, reliability also refers to how reliable and accurate the data presented by the application is. Because of the potential for use in a scientific context, it is important for the data presented by the application to come from reliable sources. The information must be reliable and come from a verified source so that the target audiences (environmentalists and real-estate investors) can create safe as well as substantiated projects.
Availability	This website is used to show raw data in a more filtered way. Any down time of this website would not cause critical concern as this data is public information. In other words, availability is not one of the main concerns for this application. If a web page were to crash or become unavailable, there may be an issue with the server provider as this website will be hosted with a cloud provider.
Usability	The ease at which an individual is able to learn and use an application or product is known as usability. Our application will be structured in such a way to encourage user learning and ameliorate the application's usability. It will have an easily accessible home page that does not require the user to create an account or store information. It will also have an accessible list of tabs that enumerate the application's main functionalities (such as environmental information for an area and a recycling

	database) and organizes each function in an efficient way. Although these tabs should be able to link to information and access all content, they should be separate entities that allow the user to transition between them easily and return to the home page if necessary.
Maintainability	It is important for the sanity of the development team to keep the code understandable and easy to maintain. Therefore, we will require our programmers to maintain high standards and use best coding practices. The climate app will use servers provided by a 3rd party, like Amazon Web Services, so server maintenance will be an indirect concern since it is contractually placed with a third-party.
Scalability	Our application should be able to scale as users and data sources increase in size and complexity. As the number of users increase, our application should be able to handle increased traffic and latency. We should be able to ensure that the user experience is preserved as the number of users grow. Additionally, we should implement scalable modules and components in our application that can be restructured to be more complex as the sources that we draw information from increase in number. As our application grows over time, we hope to include more environmental information from a growing number of sources.
Compatibility	Compatibility is the property that the website will be available regardless of entry point or technology. The website will be accessible from any device and any browser. This will require more testing to ensure the website works for all browsers and platforms. Additionally, a mobile version of the website will need to be designed that could have a significantly different front-end compared to the main web application.
Time Delivery	The website must be deployed and live by the expected time and date.
Cost	As the website generates traffic, there will be a cost to maintain and run the website on an accessible infrastructure such as the Amazon Web Services (AWS) Platform. In order to address this constraint, advertisements will generate passive income; this income will then be reinvested into the website for maintenance and scalability.
Extensibility	This is the ability to add new features with ease. This website should allow for new features and modules without an extensive amount of effort. Bad site design can ruin this feature. This

	requires an accessible, flexible, and effective framework for both the back-end and front-end. This impacts the application developers more than any other stakeholder.

Table 2. Prioritized Stakeholder Needs

Priority	Stakeholder Need/Quality	Classification	Priority Justification
1	The stakeholders of our climate change data aggregation web application require that the environmental information provided is reliable and accurate. The information that we present within the application must be accurate and must come from verified sources so that the decisions, products, and studies created by our primary stakeholders are reliable and safe.	Reliability	Reliability is our first priority because the information from our application will be used in important decisions, products, and studies. Since our primary users include environmentalists (activists as well as scientists) and real-estate investors (construction companies as well as homeowners), the information that we present within the application must be accurate and come from verified sources. Since this data has the potential to be used in a scientific context, it is important that the data is accurate and reliable. This ensures that the studies produced from it are accurate, not misleading, and are safe. Additionally, studies pulled from accurate and reliable data only serve to encourage climate change activism. Additionally, construction companies that use our application must construct safe and reliable structures, which means that the information

			from our site must be reliable and verified. This ensures that design decisions by construction companies will be safe since they are based on reliable information. Finally, homeowners must use the information on this application in order to make informed purchase and relocation decisions. They require that the environmental information provided by our application is reliable so that they can live in safe areas and avoid potentially environmentally dangerous locations.
2	Our application should be accessible to any stakeholder concerned with environmental conditions. This means that our application should be able to scale as the data sources and users increase in size and complexity. As the number of users increases, our application should be able to handle increased traffic yet keep latency minimized (in order to ensure that the quality of the user experience is preserved). Additionally, as the number of environmentalists and real-estate investors that use our application increases, the quality of our application must be maintained or increase. This means that as	Scalability	Scalability is the second most prioritized quality because the goal of our application is to make environmental information more accessible. That means that our application (both front-end and back-end) should be able to scale as the number of users increases. To ensure this, we must implement scalable modules and components in our application that can be restructured to be more complex as the sources that we draw information from increase in number. This means that our website will be able to achieve its goal of providing reliable environmental information to its users even if the complexity of the accesses and accounts

	our application grows over time, we should aim to include more environmental information from a growing number of sources.		increases. Constructing a scalable application is imperative, for we will be able to achieve our goals even as we scale up.
3	Our application will be structured in such a way to encourage user learning and ameliorate the application’s usability. Our target audiences (environmental activists and scientists as well as real-estate investors) should be able to navigate through our application with ease and be able to access all of the functionalities easily. Our website should improve their experience and allow the users to efficiently access environmental information.	Usability	Usability is our third priority because it is important to the vision of our application. We aim to create an application that makes environmental information accessible to everyone. By making our application functionally usable and increasing its accessibility, we will be able to share this information with anyone interested in environmental data. This also ensures that environmentalists and real-estate investors are able to create the most effective studies, products, and decisions because they are able to access the information easily.
4	Stakeholders should be able to access our application through all technological entry points. Our application should be able to be compatible with different browsers and even different devices. Compatibility is also similar to usability in that it makes our application and its environmental information more accessible. Stakeholders should be able to use our application from any platform and still be able	Compatibility	This increased compatibility will increase the accessibility of environmental information and data – which is the primary goal of our application. We aim to make verified environmental information and the functionalities of our website accessible to anyone interested (specifically environmentalists and real-estate investors). By creating modules and components that are compatible with any

	to conduct their work efficiently.		technological platform, we are able to encourage the compositions of more effective studies, products, and decisions.
5	The stakeholders of our application require a robust set of functionalities that are flexible. Since our target demographic includes both environmentalists as well as real-estate investors, the functionalities of our application should be flexible enough to include both of their needs. The functions should be able to provide information for scientific studies, activist agendas, design decisions for construction, real-estate investment, and more.	Flexibility	Flexibility is our fifth priority because our application should be able to meet the needs of many different users. Since the vision of our application is to make environmental data more accessible, we need to create modules and components with functionalities that address all of the issues of our different users. This means that we must create flexible modules and components that can be accessed by users to provide different sets of information – which is a necessity of the system in terms of flexibility.
6	The stakeholders of our application require the application to be released on time so that they are able to organize their activities around it. The stakeholders need to know when our application is released so that they can plan on when to start implementing the use of our application into their schedule and plan how they are going to use the application’s functionalities to ameliorate their activities.	Time Delivery	Time delivery is our sixth priority because the release date of our application directly impacts our stakeholders. Our primary audiences (environmentalists and real-estate investors) will benefit from knowing when our application is released because they can start planning how and when to iterate its functionalities into their schedule. Time delivery is at a lower priority than the other stakeholder needs, however, because the developers and the stakeholders are able to plan

			around a modified or delayed deployment schedule.
7	The stakeholders of our application want to a minimized cost of use for the application. The budget for the application will collected from advertisement revenue. This money will then be reinvested into the web application for maintenance and scalability.	Cost	Cost is our seventh priority. The cost of use for our website is important to users. It also directly affects the accessibility to environmental information. The funding for the website, however, can be changed to a different model and modified if we the application were to be sponsored or invested in. That is why cost, although important, is placed lower on the list of priorities.
8	The stakeholders of our application would want an application that is up to date and maintained. This ensures that the information that they use for scientific studies, investment, construction, and important design decisions is current and accurate.	Maintainability	Maintainability is our eight priority because of our planned implementation for the application. Although the accuracy and currency of the information in our application is important, this information will be drawn from independent third-party sources. The third-party sources that we pull environmental data from will be maintained and kept up-to-date on their end. In a structural sense, this means that maintainability is ensured by the external sources and can be placed lower on our list of priorities.
9	Stakeholder want our application to be available at all times so that they can structure and organize their activities around it. Target	Availability	Availability is our ninth priority. Although our application consolidates all important environmental information into one easily

	audiences such as environmentalists and real-estate investors want this application and its environmental information to by reachable and perpetually hosted.		accessible location, the information itself is still always available. That means that while availability s important for our application, it is prioritized lower because our users are able to access this information elsewhere.
10	Stakeholders want our application to grow and include more features over time. Stakeholders such as environmentalists and real-estate investors want our application to add functionalities when they become more pertinent over time.	Extensibility	Extensibility is our last priority. Although it is important to add more features and functionalities over time, it is the last priority for our application because of its structure. Since our application pulls verified environmental data, our functionalities mostly consist of comparisons and projections. That means that any new features that we add will be based on analyzing public information. This can be completed by other applications and users provided they have the required amount of time. Additionally, the addition of new features does not affect the client as heavily as the developers. Since extensibility for our application is primarily server-side rather than client-side, it is our last priority.

Part 2: Derive a Business Blueprint (BB)

The following UML diagram represents the relationships between application modules enumerated within the Business Blueprint. It specifies components (as well as their respective functions and data) as well as connectors (which are I/O dependencies between components).

Figure 1. UML Diagram of Business Blueprint

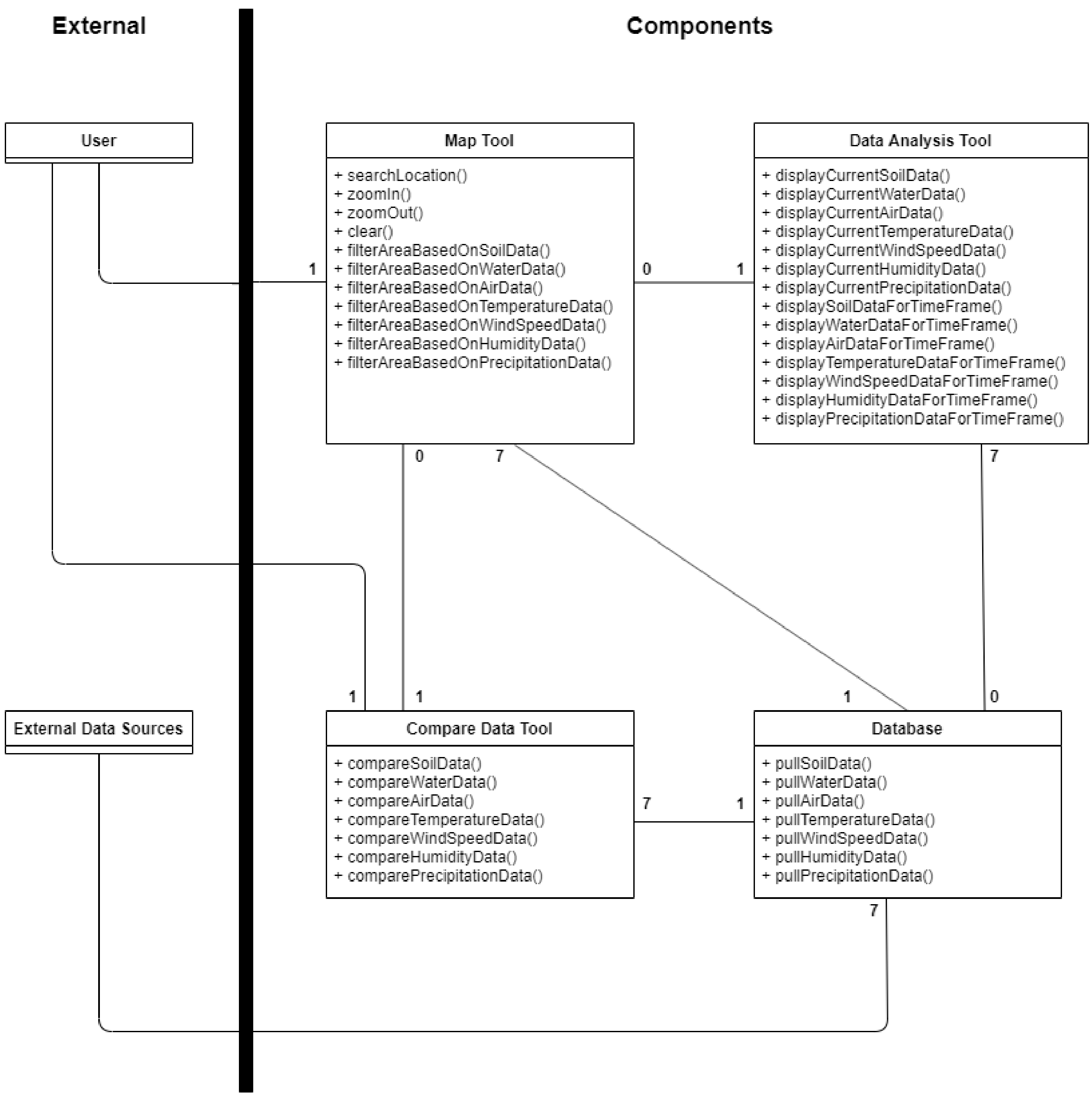

Table 3. Allocation of Functions and Data to Components

Component	Functions and Data
Map Tool	Functions: • searchLocation() • zoomIn() • zoomOut() • clear() • filterAreaBasedOnSoilData() • filterAreaBasedOnWaterData() • filterAreaBasedOnAirData() • filterAreaBasedOnTemperatureData() • filterAreaBasedOnWindSpeedData() • filterAreaBasedOnHumidityData() • filterAreaBasedOnPrecipitationData() Data: • Primary Location Data • Filtered Area Data – Soil • Filtered Area Data – Water • Filtered Area Data – Air • Filtered Area Data – Temperature • Filtered Area Data – Wind • Filtered Area Data – Humidity • Filtered Area Data – Precipitation
Data Analysis Tool	Functions: • displayCurrentSoilData() • displayCurrentWaterData() • displayCurrentAirData() • displayCurrentTemperatureData() • displayCurrentWindSpeedData() • displayCurrentHumidityData() • displayCurrentPrecipitationData() • displaySoilDataForTimeFrame() • displayWaterDataForTimeFrame() • displayAirDataForTimeFrame() • displayTemperatureDataForTimeFrame() • displayWindSpeedDataForTimeFrame() • displayHumidityDataForTimeFrame()

	• displayPrecipitationDataForTimeFrame() Data: • Specified Time Frame
Compare Data Tool	Functions: • compareSoilData() • compareWaterData() • compareAirData() • compareTemperatureData() • compareWindSpeedData() • compareHumidityData() • comparePrecipitationData() Data: • Secondary Location Data • Comparison Data – Soil • Comparison Data – Water • Comparison Data – Air • Comparison Data – Temperature • Comparison Data – Wind Speed • Comparison Data – Humidity • Comparison Data – Precipitation
Database	Functions: • pullSoilData() • pullWaterData() • pullAirData() • pullTemperatureData() • pullWindSpeedData() • pullHumidityData() • pullPrecipitationData() Data: • Soil Data • Water Data • Air Data • Temperature Data • Wind Speed Data • Humidity Data • Precipitation Data

Table 4. Component-to-Component I/O Dependencies

Component	Functions and Data
From: Map Tool To: Database	• *pullSoilData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullWaterData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullAirData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullTemperatureData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullWindSpeedData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullHumidityData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component • *pullPrecipitationData()* requires *Primary Location Data*, which was allocated to the *Map Tool* component
From: Compare Data Tool To: Database	• *pullSoilData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullWaterData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullAirData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullTemperatureData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component

	• *pullWindSpeedData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullHumidityData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component • *pullPrecipitationData()* requires *Secondary Location Data*, which was allocated to the *Compare Data Tool* component
From: Database To: Map Tool	• *filterAreaBasedOnSoilData()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component • *filterAreaBasedOnWaterData()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *filterAreaBasedOnAirData()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component • *filterAreaBasedOnTemperatureData()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *filterAreaBasedOnWindSpeedData()* requires *Wind Speed Data* from *pullWind SpeedData()*, which was allocated in the *Database* component • *filterAreaBasedOnHumidityData()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component • *filterAreaBasedOnPrecipitationData()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component
From: Map Tool To: Data Analysis Tool	• *displayCurrentSoilData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentWaterData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component

	• *displayCurrentAirData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentTemperatureData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentWindSpeedData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentHumidityData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayCurrentPrecipitationData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displaySoilDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayWaterDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayAirDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayTemperatureDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayWindSpeedDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayHumidityDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *displayPrecipitationDataForTimeFrame()* requires *Primary Location Data*, which was allocated in the *Map Tool* component
From: Database To: Data Analysis Tool	• *displayCurrentSoilData()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component • *displayCurrentWaterData()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *displayCurrentAirData()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component

	• *displayCurrentTemperatureData()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *displayCurrentWindSpeedData()* requires *Wind Speed Data* from *pullWindSpeedData()*, which was allocated in the *Database* component • *displayCurrentHumidityData()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component • *displayCurrentPrecipitationData()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component • *displaySoilDataForTimeFrame()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component • *displayWaterDataForTimeFrame()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *displayAirDataForTimeFrame()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component • *displayTemperatureDataForTimeFrame()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *displayWindSpeedDataForTimeFrame()* requires *Wind Speed Data* from *pullWindSpeedData()*, which was allocated in the *Database* component • *displayHumidityDataForTimeFrame()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component • *displayPrecipitationDataForTimeFrame()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component
From: Map Tool To: Compare Data Tool	• *compareSoilData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component

	• *compareWaterData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareAirData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareTemperatureData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareWindSpeedData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *compareHumidityData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component • *comparePrecipitationData()* requires *Primary Location Data*, which was allocated in the *Map Tool* component
From: Database To: Compare Data Tool	• *compareSoilData()* requires *Soil Data* from *pullSoilData()*, which was allocated in the *Database* component • *compareWaterData()* requires *Water Data* from *pullWaterData()*, which was allocated in the *Database* component • *cmopareAirData()* requires *Air Data* from *pullAirData()*, which was allocated in the *Database* component • *compareTemperatureData()* requires *Temperature Data* from *pullTemperatureData()*, which was allocated in the *Database* component • *compareWindSpeedData()* requires *Wind Speed Data* from *pullWindSpeedData()*, which was allocated in the *Database* component • *compareHumidityData()* requires *Humidity Data* from *pullHumidityData()*, which was allocated in the *Database* component • *comparePrecipitationData()* requires *Precipitation Data* from *pullPrecipitationData()*, which was allocated in the *Database* component

Table 5. External-to-Component I/O Dependencies

Component	Functions and Data
From: User To: Map Tool	• The *User* allocates the *Primary Location Data* to the *Map Tool*
From: User To: Compare Data Tool	• The *User* allocates the *Secondary Location Data* to the *Compare Data Tool*
From: External Data Sources To: Database	• The *External Data Sources* allocate the *Soil Data*, *Water Data*, *Air Data*, *Temperature Data*, *Wind Speed Data*, *Humidity Data*, and *Precipitation Data* to the *Database* • *pullSoilData()* acquires *Soil Data* from the *External Data Sources* • *pullWaterData()* acquires *Water Data* from the *External Data Sources* • *pullAirData()* acquires *Air Data* from the *External Data Sources* • *pullTemperatureData()* acquires *Temperature Data* from the *External Data Sources* • *pullWindSpeedData()* acquires *Wind Sped Data* from the *External Data Sources* • *pullHumidityData()* acquires *Humidity Data* from the *External Data Sources* • *pullPrecipitationData()* acquires *Precipitation Data* from the *External Data Sources* • *pullAirData()* acquires *Air Data* from the *External Data Sources*

Table 6. Derivation Plan

1	**Goal: Reliability**	
1.1	*BB Heuristic*: Isolate Risk - Data	• *Why*: Isolating risk from data elements within the components of our web application increases reliability. We

		isolate risk by maintaining pull functions in our Database component that acquire accurate data from verified third-party sources. This ensures that the data we display is always accurate. Additionally, this decreases risk, for the information is always maintained to a certain standard by a third-party source. Decreasing risk results in an increase in reliability, which ensures that our users able to produce safe and reliable studies, investments, and design decisions. • *Priority Justification*: This is the most important heuristic for our top priority (reliability) because it isolates and decreases risk for our web application. This means that we ensure that our application is reliable (because as risks decrease, reliability increases). It is also imperative that we provide accurate and reliable environmental data, since our primary audience (environmentalists and real-estate investors) rely on such information to produce reliable and safe scientific studies, activities, construction decisions, and investments. This is the most important quality of our application, which is why this heuristic is the highest priority.
1.2	*BB Heuristic*: Group Based on Task Similarity	• *Why*: By grouping components based on similar tasks, we create a system that is

		able to be modified and changed easily. This allows us to maintain a reliable web application. If any component of our web application was unreliable or inaccurate, we could swiftly and efficiently modify it. This is only possible because our system components are grouped by task similarity. Thus, this heuristic ensures reliability. • *Priority Justification*: Although this heuristic is important for the maintenance of reliability for our system, it is not as important as reducing risks associated with data. Reducing risks ensure that our application will have an increase reliability. By reducing risks, we are able to prevent problems (which increase reliability from the beginning). On the other hand, when we group components by task, we are able to address existing problems (which increases reliability down the line). With easily accessible components that can be modified efficiently, we are able to address problems and bugs as soon as they are discovered and increase the reliability of our system.
2	**Goal: Scalability**	
2.1	*BB Heuristic*: Group Based on Task Similarity	• *Why*: By grouping system components by task similarity, we are able to create scalable components within our system

		that are easily accessible. Additionally, these components are able to be efficiently modified due to their groupings – which is a direct influencer of scalability. By adhering to this heuristic, we will be able to create a solution with components that are scalable according to task similarity. • *Priority Justification*: This is the most important heuristic for scalability because it directly influences the grouping and structure of our application's components. By organizing our system with components that are scalable according to their groupings for similar tasks, we are able to create a development environment in which modifying and updating components is efficient and accessible. This grouping is imperative for the composition of a scalable system.
2.2	*BB Heuristic*: Group Based on Resource Demand	• *Why*: By grouping the components within our system by resource demand, we are able to create a scalable application. By placing functions that have similar resource requirements and constraints, we are able to decrease coupling. This means that whenever we need to modify a component or change the source for these methods, we would only need to

		modify a specific component rather than the entire system. • *Priority Justification*: This heuristic is at a lower priority than the previous heuristic (grouping components based on task similarity) because this is a subset of that for our application in the context of scalability. By grouping components based on similar tasks, we are able to ensure the composition of a scalable system that is easily accessible and able to be efficiently modified. Since the components are grouped by task, they will inherently be grouped by resource demand.
3	**Goal: Usability**	
3.1	*BB Heuristic*: Reduce Class Complexity – Size	• *Why*: By reducing class complexity, we are able to create more simple technologies and interfaces. That means that we will be able to ameliorate the user experience by interfacing simple technologies that are easily accessible, easy to use, and easy to learn from. This is important for achieving our goal regarding usability. • *Priority Justification*: This heuristic is the most important for this goal because it ensures that components will divided and made more simple. This ensures that our user interface and associated

		technologies are not overly complex and are easily understood.
4	**Goal: Compatibility**	
4.1	*BB Heuristic*: Group Based on Implementation Reality	• *Why*: By grouping functions according to the functionalities of existing technologies, we are able ensure that existing technologies will match component boundaries. This means that we are able to increase the compatibility within of our system if we group according to implementation reality. • *Priority Justification*: This is the most important heuristic for compatibility because it encourages the composition of our system to take existing technologies and requirements into account. This will increase the compatibility of our system and ensure that is able to interface with major technologies.
5	**Goal: Flexibility**	
5.1	*BB Heuristic*: Reduce Data/Event Dependency	• *Why*: By reducing coupling and increasing cohesion, we are able to make a flexible system that can adapt to change. We can achieve this by reducing data and event dependency – which allows our components to be independent of one another and be easily accessible or modification. • *Priority Justification*: This is the most important heuristic for flexibility, for a

		reduction in data and event dependency directly correlates with reduced coupling. Reduced coupling is also associated with increased cohesion. This means that our system is able to become more flexibly by adhering to this heuristic.
6	**Goal: Time Delivery**	
6.1	*BB Heuristic*: Specify Overlapping Capabilities	• *Why*: By specifying overlapping capabilities and shaping our system around this heuristic, we are able to ensure that our system is integrated and deployed in a timely manner. By using the inheritance concepts of this heuristic, we are able to reduce development time through abstraction and information sharing (that is associated with class relationships). • *Priority Justification*: This is the most important heuristic for an appropriate time delivery, for it reduces development time and allows for deployment organization. By employing the class relationships, inheritance concepts, and information sharing of OOP, we are able to reduce delivery time.
7	**Goal: Cost**	
7.1	*BB Heuristic*: Specify Overlapping Capabilities	• *Why*: We are able to reduce the cost of developing, integrating, and deploying our application by specifying overlapping capabilities. By employing the inheritance and class distinction

		concepts within OOP as stated by this heuristic, we are able to reduce the development phase and reduce the time for delivery – which ultimately reduces the cost of the project (since fewer resources are used over a shorter period of time). • *Priority Justification*: This is the most important heuristic for reducing costs for the project because it reduces delivery time and decreases the number of resources required for development. By organizing the system to OOP concepts under this heuristic, we are able to create robust components that in a very short amount of time. This ultimately reduces costs.
8	**Goal: Maintainability**	
8.1	*BB Heuristic*: Reduce Class Complexity – Weights	• *Why*: By reducing the weighted complexity of components and classes, we are able to construct a more maintainable system. By reducing the complexity of components and creating more easily understood classes (that are easily digestible), we are able to increase the maintainability of our system through the perspective of a developer. • *Priority Justification*: This is the most important heuristic for maintainability, for it ensures that our system is composed of simplified components that

		ameliorate the development environment. By creating simplified classes and following OOP concepts, we are able to create a maintainable application that developers can modify and update quite easily.
9	**Goal: Availability**	
9.1	*BB Heuristic*: Isolate Risk - Functions	• *Why*: By isolating the risks associated with functions, we are able to create a system that combats change (which increases system availability). Isolating the risks of functions and placing them in independent components, we are able to create a scalable solution that can resist errors associated with high-volume and risky behaviors. This ensures that our application is always available. • *Priority Justification*: This is the most important heuristic for availability, for it ensures that functions that contain risks are isolated within their own components. That means that if the risks were to ever occur, only the module with that component would be affected, and the entire system will still keep running. This increases the availability of our system.
10	**Goal: Extensibility**	
10.1	*BB Heuristic*: Group Based on Task Similarity	• *Why*: By grouping system components by task similarity, we are able to create an extensible system with components that

		are easily accessible and easily modifiable. This means that if we ever needed to add a new feature to the application, we would only need to modify the component that dealt with the targeted tasks. • *Priority Justification*: This is the most important heuristic for extensibility, for it allows for easy and efficient modification of components within the system. By grouping components according to task similarity, we are able to add features (as developers) more quickly and efficiently by modifying only the required components rather than the whole system.
10.2	*BB Heuristic*: Group Based on Resource Demand	• *Why*: By grouping the components within our system by resource demand, we are able to create an extensible system. That means that we would only need to modify and update components that have similar resource requirements when adding a new feature to the application. • *Priority Justification*: This heuristic is at a lower priority than the previous heuristic, for it is a product of the previous heuristic's implementation. By creating components that are grouped by task similarity, we will inevitable organize these components by resource similarity and requirement similarity.

Table 7. Potential Conflicts and Impact on Derivation Plan

Potential Conflict	**Possible Resolution**
The heuristic in section 1.2 and 2.1 of Table 6 (*Group Based on Task Similarity*) has the potential to conflict with the heuristic in section 4.1 (*Group Based on Implementation Reality*) since one would influence component groupings based on similar functions and tasks while the other would influence component groupings based on existing technologies. This could increase coupling and decrease cohesion. The first heuristic would impact our system by allocating resources in independent components based on their objectives. The second, heuristic, however, could negatively impact these groupings by creating dependencies between functions that can already be interfaced with technological endpoints – which would create a messy system structure.	Since the heuristic in section 1.2 and 2.1 is used to achieve our goal of reliability (which is our highest priority goal), we should resolve this potential conflict by prioritizing it. We should create a majority of the system according to this heuristic and group components by task similarity. That means that when we bootstrap the blueprint, we implement our solution and define our components according the *Group Based on Task Similarity* heuristic. Then, afterwards, we refine and modify the system to make it more efficient with the other heuristics that are enumerated in our derivation plan for goals that have lower priorities.
The *Reduce Data/Event Dependency* heuristic in section 5.1 could possible conflict with the heuristics that isolate risks in section 1.1 and 9.1 (*Isolate Risk – Data* and *Isolate Risk – Functions*). By isolating the risks, we create dependencies between components and external sources that we rely on to minimize and handle potential risks. These dependencies are contradictory to the heuristic for *Reduce Data/Event Dependency*.	Our solution to this would be to adhere to the heuristics *Isolate Risk – Data* and *Isolate Risk – Functions* over the heuristic *Reduce Data/Event Dependency*. Since the former heuristics are used to achieve our highest priority goal (reliability), we will implement them and overlook the increase in coupling that results in creating dependencies. The benefits from minimizing and addressing risks for our application far outweigh the negative effects that arise from an increase in dependency as well as coupling. We will first prioritize the heuristic for risk reduction (*Isolate Risk – Data* and *Isolate Risk – Functions*) and then we will implement the

	heuristic *Reduce Data/Event Dependency* for what is left.

Bootstrap Rationale: I implemented the Usage Cases Bootstrap because I wanted to create an efficient structure for my system that grouped components according to their respective uses, their task similarities, and their resource requirements. This bootstrap was a reasonable starting point for the derivation of the Business Blueprint (BB) because it reduced the complexity of the system and allowed us to create a simple yet effective architecture that consisted of four main components and two external influencers. Finally, this bootstrap aligned with the prioritized needs of the stakeholders; the Usage Cases Bootstrap allowed us to create a system structure that grouped components with similar functions. For example, we grouped functions that acquired environmental data and had certain risks associated with them. This allowed us to adhere to the *Isolate Risk – Data* heuristic and increase the reliability of our application – which was our top priority.

Part 3: Evaluate Business Blueprint Structure

Table 8. Coupling and Cohesion Metrics

Component	Number of Inputs (Data and Events)	Number of Outputs (Data and Events)	Number of Dependencies Between Components	Degree of Cohesion
Map Tool	7	3	3	4/11 = 0.3636 **36.36%**
Data Analysis Tool	8	0	2	0/14 = 0 **0%**
Compare Data Tool	8	1	2	0/7 = 0 **0%**
Database	2	21	3	0/7 = 0 **0%**

Note: Degree of Cohesion = (Number of Functions Without Dependencies / Total Number of Functions)

Table 9. Size and Complexity Metrics

Component	Number of Functions	Number of Data Elements	Number of Inputs and Outputs	Component Complexity
Map Tool	11	8	10	29
Data Analysis Tool	14	1	8	23
Compare Data Tool	7	8	9	24
Database	7	7	23	37

Note: Component Complexity = (Number of Functions + Number of Data Elements + Number of Inputs + Number of Outputs)

Support for Applied Heuristics: The heuristic from section 3.1 of Table 6 (*Reduce Class Complexity – Size*) can be demonstrated by the size and complexity metrics from Table 9. Instead of having 1 large and complex component for the system, my architecture uses four components (the Map Tools component, the Data Analysis component, the Compare Data component, and the Database component) that are grouped by function and task similarity. By adhering to the *Reduce Class Complexity – Size* heuristic, I am able to create a simplified yet effective system that distributes the complexity of the overall system across these four components. Instead of creating a system with a complexity value of 113, I have created four components that have component complexities of 29, 23, 24, and 37. This reduces the class complexity of my system by size and allows the application to be maintained and scaled as the number of users increases. Thus, I have used the size and complexity metrics from Table 9 to demonstrate the heuristic from section 3.1 of Table 6 (*Reduce Class Complexity – Size*).