7-2019

The Hyperwall: A Geospatial Education Exhibit for the Science Museum of Virginia

Alexander Walton

University of Redlands

The Hyperwall: A Geospatial Education Exhibit for the Science Museum of Virginia

A Major Individual Project submitted in partial satisfaction of the requirements for the degree of Master of Science in Geographic Information Systems

by

Alexander Walton

July 2019

The Hyperwall: A Geospatial Education Exhibit for the Science Museum of Virginia

The report of Alexander Walton is approved.

[illegible]

Douglas Flewelling, Ph.D.

[illegible]

Ruijin Ma, Ph.D., Committee Chair

July 2019

Acknowledgements

I would like to acknowledge the M.S. GIS faculty and staff for sharing their experience and knowledge with me and aiding me throughout this year, especially my advisor Dr. Ruijin Ma for guiding my project and keeping me on track.

Thank you to Thad Tilton with Esri’s ArcGIS Runtime SDK for .NET team for answering my many .NET questions.

To my mom, thanks for your unconditional support in everything I’ve ever done. Even when it didn’t make sense to you.

Amber, without your encouragement I would never have applied and without your patient support I wouldn’t have succeeded.

Abstract

The Hyperwall: A Geospatial Education Exhibit for the Science Museum of Virginia

by

Alexander Walton

Modern museums rely on technologically advanced platforms to attract visitors and convey information. The Science Museum of Virginia (SMV) proposed a new interactive geospatial education exhibit called the EarthLab Data Hyperwall to support their mission statement of "inspiring Virginians to enhance their lives through science." A hyperwall/video wall is a visualization tool composed of a large screen array used for education and collaborative work. The SMV's Hyperwall Exhibit consists of a high-resolution video wall controlled by a connected, visitor-accessible touchscreen. The exhibit educates visitors on several environmental and climatic topics using interactive maps to increase comprehension. This project produced a .NET Framework application that accesses the datasets and maps for each topic by implementing Esri's ArcGIS Runtime SDK for .NET. The Museum needed the Hyperwall platform to display the underlying data through an interactive and accessible User Interface. The system's design considered Museum visitor's varied demographics and recent educational theory relating to cognition and spatial literacy. The educational goals of the exhibit are: teach visitors about the Museum's research into heat illness risks related to urban heat islands, improve visitor's wayfinding and overall spatial literacy, and educate visitors on environmental conditions and the changing climate locally and the world over.

Table of Contents

Table of Figures

List of Tables

List of Acronyms and Definitions

API	Application Programming Interface
ESRI	Environmental Systems Research Institute
GIS	Geographic Information System
GUI	Graphical User Interface
IDE	Integrated Development Environment
MVVM	Model-View-ViewModel
RAD	Rapid Application Development
SDK	Software Development Kit
SDLC	Software Development Life Cycle
SMV	Science Museum of Virginia
VR/AR	Virtual Reality / Augmented Reality
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language

Chapter 1 – Introduction

The Science Museum of Virginia (SMV) planned a new educational exhibit called the EarthLab Data Hyperwall. The SMV needed to share environmental research with Museum visitors through an interactive map-based platform. They approached the University of Redlands for help developing a Windows-based application to manage the Hyperwall's data and hardware. The application provided a graphical user interface (GUI) that could engage the Museum's visitors visually and kinesthetically. Further, the application needed to incorporate a geographic information system (GIS) for accessing and processing the Hyperwall's spatial data.

The first section of this chapter discusses the project client. The second section describes the problem that needed solving. The third section summarizes the proposed solution including project goals, objectives, scope, and methodology. The final two sections address the intended audience for this report and describes the structure of the rest of the report.

1.1 Client

This project's client was the Science Museum of Virginia. The SMV is a science and technology center located in Richmond, Virginia. Its mission is to educate visitors about scientific topics. Dr. Jeremy Hoffman acted as the primary point of contact and liaison for the Museum. In this role, Dr. Hoffman provided guidance throughout the project design and development process. He aided in planning project goals/objectives, identifying key elements of the system, and provided specifications about the exhibit's hardware.

1.2 Problem Statement

The Museum needed a system that managed the Hyperwall's data, hardware, and presented interactive, educational maps to visitors. Currently, the SMV engages hundreds of thousands of visitors every year with their spatially-linked education exhibits. Two exhibits, the Dome and Science on a Sphere, engage visitors in learning about interplanetary and global phenomena. However, the design of the two exhibits precludes the ability to display state, county, city, or other local data, nor does either exhibit contain a system for supporting direct visitor interaction with the displayed data. They are only available during museum staff-led presentations. As such, the SMV needed a more versatile exhibit that encouraged visitor engagement and presented larger-scale data. The overall goal for this exhibit is summed up by quoting the cognitive psychologist Jerome Bruner, who said, "learners are encouraged to discover facts and relationships for themselves" (Bruner, 1960).

The Museum needed the new exhibit to share research with visitors as easily understood map products, to display large scale data, to allow individualized interaction through a user-friendly interface, and to accommodate future datasets. Dr. Jeremy Hoffman recognized the possibility of complimenting current museum installations with a platform containing these features but needed a system to implement such features.

1.3 Proposed Solution

As designed, the Hyperwall Exhibit hardware consisted of a wall-mounted LCD screen array controlled via a visitor-accessible touchscreen (Figure 1-1). This project proposed to develop a Windows desktop application and a GIS for managing the system's hardware and spatial data. The project's GIS stored the Hyperwall's data in both a web-based platform (Esri’s Portal for ArcGIS and ArcGIS Online) and a local Esri File Geodatabase. The web-based data consisted of publicly available Feature Services accessed via REST endpoints. The exhibit incorporated standard Esri map products (including imagery and Mobile Map Packages) generated from locally stored datasets. The web-based and local data were brought into the exhibit using Esri’s ArcGIS Runtime Software Developer Kit (SDK) for .NET and implemented into a Model-View-ViewModel (MVVM) architecture using Visual Studio’s Windows Presentation Foundation (WPF). The MVVM pattern allows discrete handling of an application's business and presentation logic. The application's GUI design process prioritized delivering an enjoyable visitor experience. Visitors switch between topics presented in the Hyperwall and explore maps/imagery using preset bookmarks or independently through standard map controls (zoom in/out, rotate, pan).

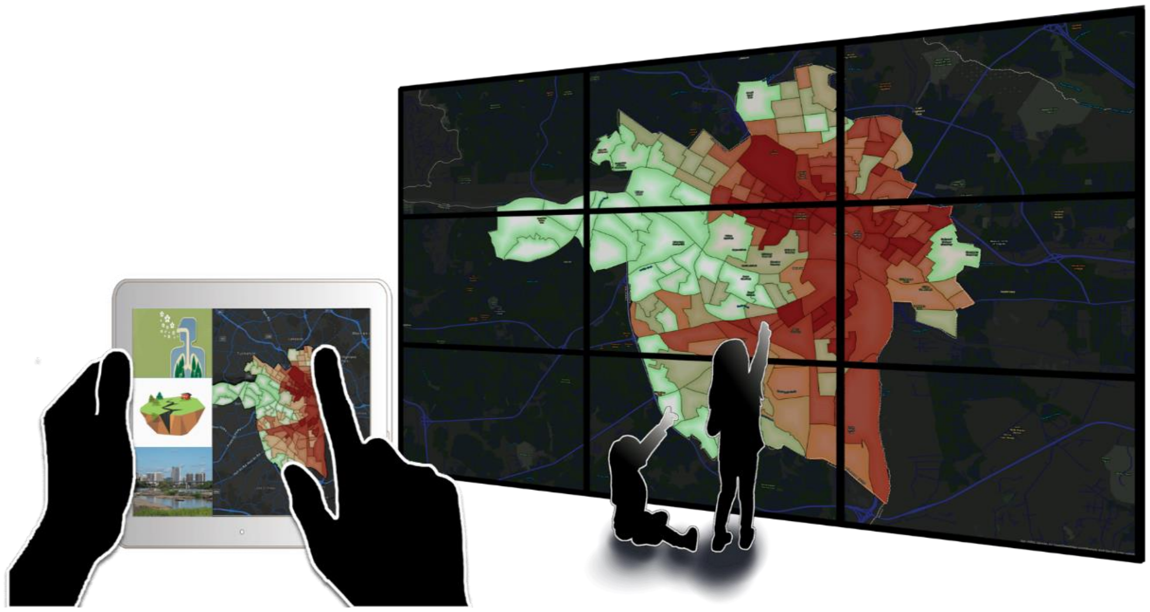

Figure 1-1: Hyperwall Exhibit Design

1.3.1 Goals and Objectives

The goal of this project was to use a modern approach to present some of the Museum's research and other environmental subjects through an educational and interactive system. The system educates visitors on environmental factors affecting their own neighborhood. After leaving the Museum, visitors gain a better understanding of their environment by applying this knowledge.

The Hyperwall Exhibit fulfilled the client's goal by ensuring the system incorporated a GUI with high-quality visualization and intuitive end-user interaction, and a GIS that supported all current and future datasets. This project converted spatial data into comprehensive map products with clear visual hierarchy, symbology, and intent to maximize visitor comprehension and accessibility. The system supplemented symbolized data with content like custom bookmarks emphasizing important areas and other features. It also incorporated basic GIS analyses such as interpolated surfaces and widgets that enabled comparison of layers. The system's GUI used gesture controls like a smartphone or tablet-based map applications to support intuitive map navigation for the end-user. The maps incorporated high-resolution satellite imagery, vector feature data, and live-feed web content to prove the system's compatibility with any future data formats.

1.3.2 Scope

This project produced a Windows .NET Framework application for managing the Hyperwall's runtime environment, a hybrid web and local GIS, and a set of instructions for maintaining the system. The three topics presented by the application contained maps exported from ArcGIS Pro or accessed through ArcGIS Online. The map products presented in the Hyperwall Application consisted of REST Feature Services, a Mobile Map Package, and a set of georeferenced raster imagery.

The client was responsible for providing tabular data from their Heat Vulnerability research about heat-related illnesses in the City of Richmond. The client was also responsible for determining and building the hardware for the Hyperwall as well as creating a space to house the exhibit.

The Hyperwall Application was developed using Windows Presentation Foundation (WPF) v4.5, ESRI's Runtime SDK for .NET v100.5, the C# programming language, and Microsoft's Extensible Application Mark-Up Language (XAML). Visual Studio was the Integrated Development Environment (IDE) used to implement the programming syntax, toolkit class references, and MVVM pattern. The project produced a hybrid GIS for storing and accessing the required data from web/local sources and creating exhibit maps.

Specifically, the GIS accessed exhibit data through ArcGIS Online REST Endpoints and a local File Geodatabase. The completed system needed to reliably display the maps interactively for users and enable implemented features/widgets on the GUI. The client received the data and application upon project completion. The team also developed a set of instructions for maintaining and updating the system.

1.3.3 Methods

The project's time constraint for creating the exhibit application and supporting GIS required selection of methodologies oriented towards reducing time spent on each stage of the software development life cycle (SDLC). The project utilized Rapid Application Development (RAD) workflows to manage the SDLC. RAD is an Agile project management system used for quickly developing software products. RAD workflows encourage methodology that organizes software development into basic categories for achieving tasks and objectives. RAD defined tasks operate under the umbrella of the Planning, Design, and Development/Testing (Martin, 1991). For this project, each phase was expected to require approximately 2-3 months.

The Planning phase was constrained to three objectives that supported the following phases. The first phase of the planning phase was conducting a requirements analysis with the client. Based on these requirements, the second phase involved determining the development environment for the project, including programming languages used, code editors, necessary Software Development Toolkits (SDKs), and additional code libraries required for the project. In the final planning stage, the datasets supporting each of the three subjects in the Hyperwall Application were collected and collated into formats from which two and three-dimensional map products could be made. The three datasets were the City of Richmond Heat Vulnerability dataset, an air quality index produced by the Environmental Protection Agency (EPA) and hosted on Esri's ArcGIS Online, and a satellite imagery dataset showing the effects of natural disasters throughout the world.

Following the completion of the planning objectives, the Design phase began. The objective for the design phase was to determine all graphical elements, features, and functions for the controlling WPF application and hybrid GIS. During design, the system's conceptual model was created and revised based on client input. The conceptual model outlined the broad relationships between data storage structures, analyses, map production, IDE structures, hardware, and other system entities. The logical model expanded on the conceptual model's entities according to the features that were best suited to facilitate system processes. Simplistic structures were favored to support future maintenance and modification of the Hyperwall Exhibit.

The application was built, tested, demonstrated, and refined during the Development/Testing phase according to RAD workflows and best practices for Object-Oriented Programming. The objective of this phase was to convert the entities and relations in the logical model into a functional system that fulfilled the project requirements. This included building a method of data storage appropriate for the system's data types, implementing the hybrid GIS to manage both web and locally stored data, and building an application that worked with the entire system and fulfilled end-user requirements. The map, Feature Services, and imagery used in the Hyperwall exhibit were established programmatically using ArcGIS Runtime SDK for .NET. Finally, the completed application was tested to ensure smooth operation within the entire system. The development and testing process followed the Software Development Life Cycle wherein the application was tested as each new feature was added.

1.4 Audience

The intended audience for this report is the client, University of Redlands faculty, GIS developers who have a basic understanding of .NET development, and anyone wishing to learn about the development of spatial education tools, GIS software, or interactive museum exhibits. Most of these individuals possess at least a high-level understanding of geographic information systems, application development, and geospatial education.

1.5 Overview of the Rest of this Report

Chapter 2 is an overview of project-related background material including interactive museum exhibits, geospatial education strategies, and an explanation of Windows 10 desktop application development. Chapter 3 lists the project requirements, project plan, and system design. Chapter 4 discusses the conceptual and logical models required for the project and how project data were prepared and stored. Chapter 5 discusses project implementation including development strategies and system architecture. Chapter 6 shows how RAD strategies were used to test and improve system features, functionality, and maintainability. Chapter 7 summarizes the project and discusses possible future work.

Chapter 2 – Background and Literature Review

This chapter introduces how geospatial education improves spatial thinking, spatial literacy, wayfinding, and understanding of abstract data. Interactive applications and museum exhibits are useful platforms for teaching map comprehension and exposing members of the public to new research about climate change and environmental factors. Section 2.1 introduces the concept of hyperwalls/video walls as visualization, learning, and collaboration tools. Section 2.2 compares approaches to spatial literacy. Section 2.3 discusses examples of museums using interactive exhibits to expose visitors to new places and concepts. Section 2.4 discusses .NET Framework development environments.

2.1 Hyperwalls

It is worth noting here that hyperwall systems differ from video walls or screen banks. Hyperwalls are purpose-built, incorporate more processing power and more complex architecture. The NASA Ames Research Center created the first hyperwall display system in the early 2000s. NASA used its hyperwall design for collaboration between professionals and researchers. Users interacted with the system’s visual data via a flat panel LCD screen array (or video wall) powered by several high-end processors (Sandstrom et al., 2003). Since then, NASA has built additional versions of its system to pace technological advancement (Sellers, 2011). Private companies have since built their own hyperwalls and accompanying software. A company called “Hiperwall” has successfully marketed the format to private and governmental organizations for use as operations dashboards, corporate AV, education, and entertainment (Hiperwall, 2018). However, the hyperwall designs created by NASA and Hiperwall were large-scale installations, which required a significant initial investment in hardware and software to run successfully.

2.2 Spatial Literacy in the Modern Era

Before discussing work dealing specifically with spatial literacy, it is important to briefly discuss the epistemological underpinnings of educational theory and the learning process. Jean Piaget focused on child cognitive development throughout the first half of the 20th century. He theorized that as children grow, they adopt new schemas (information/models) and adapt elements of their prior learning to new domains (Piaget, 1936). This is a recursive method wherein an individual learns something new and applies it repeatedly to different situations. Piaget's model is embedded in many pedagogical topics. Spatial literacy is a relatively new field derived from the need for better geographic information systems that humans can operate intuitively (Mark & Egenhofer, 1995), the need for more empirical data to inform geographic education (Downs 1994), and the need for cohesive lexicon when teaching geospatial concepts to various age groups (Marsh et al., 2007).

Spatial literacy is as necessary for modern life as reading, writing, and arithmetic (Goodchild, 2006). Benchmarks for critical spatial thinking and spatial problem-solving skills is a preeminent problem in GIS (Bednarz & Kemp, 2011). It is also difficult to measure the success of geospatial educational endeavors even when using purpose-built spatial learning materials (Ang & Wang, 2006; Bartoschek et al., 2013; Downs & DeSouza, 2006; Marsh et al., 2007). Last, it is challenging to engage non-professionals in spatial thinking. There are examples where researchers successfully circumvented this problem by taking advantage of naturally developed schema while avoiding jargon-laden explanations and providing adequate support systems through software and other educational tools (Egenhofer & Mark, 1995; Goodchild, 2011; Piaget, 1936).

Creators of spatial education tools have struggled to measure how successful a given tool is at achieving its goals. Goal achievement is difficult to gauge, and researchers have used several approaches to qualitatively or quantitatively measure a tool's success. Researchers in Cyprus implemented multiple augmented reality exhibits, TouchTable exhibits, and virtual tour installations. Users were then asked to rate their personal experiences using each installation on a qualitative scale of personal enjoyment (Michael et al, 2010). Other researchers gauged the success of their wayfinding app game (called "Ori-Gami") by measuring user ability to achieve in-game objectives. They took statistics on how well users completed the orienteering tasks within the app (path efficiency, time to complete objectives) and created heat-maps showing where users were pressing the tablet's screen (Bartoschek et al., 2013). Other approaches have stuck to more traditional survey-type questionnaires based on ontology and lexicological ability (Kim & Bednarz, 2013; DeSouza & Downs, 2006).

2.3 GIS-Based Museum Exhibits

Modern museums are centers for research, education, and public outreach. Museum GIS's can be used to model spatial data for visual analysis in an immersive manner using augmented or virtual reality systems (Hirose, 2006; Lepouras & Vassilakis, 2004; Michael et al, 2010). Alburshaid (2012) from the University of Redlands designed a multi-touch table exhibit for the San Bernardino County Museum to display important geological features of San Bernardino County. Other forms of interactive museum exhibits include using virtual reality for tours of faraway places, using an interactive, large curved screen to navigate the Mayan ruins of Copan (Hirose, 2006).

There is enormous potential for museums in newer technologies such as interactive, virtual exhibits that are educational and entertaining. However, the technological cost and design requirements for virtual exhibits are high. The requirements for virtual reality and augmented reality (VR/AR) exhibits need to be carefully considered (Lepouras & Vassilakis, 2004). Augmented reality allows museum guests to do things like taking an active role in the exhibit through game technology (Sylaiou et al., 2015). Alternatively, augmented reality can be used to create accessible 'virtual museums' for physically disabled guests (Liarokapis, 2004). Thinking about the audiences' preferences and abilities is necessary when designing an VR/AR or other form of exhibits. Additionally, the design needs to ensure the technology (VR/AR, or otherwise) supports the educational/entertainment goals of the exhibit.

2.4 Development Environment and the .NET Framework

Software development often requires a customized software suite for writing and compiling code called the Integrated Development Environment (IDE). Visual Studio is a widely used code editor that is regularly maintained and updated and will be supported by Microsoft indefinitely. Visual Studio supports many languages and development formats, making it an essential part of many IDE's. Windows Presentation Foundation (WPF) with C# and XAML (Extensible Application Markup Language) is an established sub system of Visual Studio used for creating .NET Framework applications. WPF is especially useful for creating user interfaces that mesh well with C# business logic. C# was introduced in 2002 and continues to be one of the most heavily used object-oriented languages. This language allows developers to exert some of the fine-grain control of C++ while avoiding its complexity (Sørensen & Mihailesc, 2010). WPF uses C# and XAML in tandem to manage an application’s business logic (C#) and its appearance (XAML) on the .NET Framework (Nathan, 2013).

The Model-View-ViewModel (MVVM) structure (introduced by Microsoft in 2005) allows software developers to write readable and easily maintained code (Sørensen & Mihailesc, 2010). MVVM abstracts the application’s graphical user interface (GUI) away from the business logic dictating its behavior. This results in a cleaner and easier to maintain code. Figure 2-1 is an illustration of how the MVVM design pattern operates in a WPF application (adapted from Hakeem, 2017).

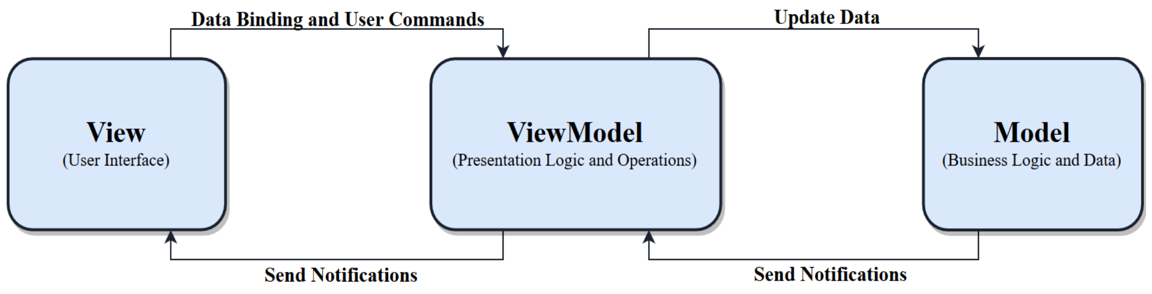
Data Binding and User Commands
Update Data
View
(User Interface)
ViewModel
(Presentation Logic and Operations)
Model
(Business Logic and Data)
Send Notifications
Send Notifications

Figure 2-1: Model-View-ViewModel Pattern

2.5 Summary

This chapter briefly discusses examples of hyperwall hardware architecture, the significance of spatial literacy for modern people, example implementations of spatial education tools, and museum installations that rely on spatial data. The SMV’s Hyperwall project incorporates elements of previous hyperwalls while maintaining a simple, modifiable design. In consideration of recent and traditional sources of educational theory, the user interface and operability of the Hyperwall Application needed to engage laypersons using the exhibit while still conveying the intended information.

Chapter 3 – Systems Analysis and Design

This chapter examines the design requirements for the EarthLab Data Hyperwall Exhibit. The system is a geographic information system (GIS) built using the Windows .NET Framework. Designing the project required taking the client's vision for the Hyperwall Exhibit and then translating it into practical software and GIS development plans. Section 3.1 reiterates the problem statement. Section 3.2 describes the project's functional and non-functional requirements. Section 3.3 examines the design of the system used in the project. Section 3.4 outlines the project plan and discusses the changes that occurred during the project.

3.1 Problem Statement

The Science Museum of Virginia (SMV) needed a new exhibit capable of presenting spatial data over various geographic regions and sharing environmental research conducted by the SMV and others with visitors. The exhibit needed to be a comprehensive platform for sharing information about environmental subjects by displaying related spatial data at large and small scales. The exhibit's learning material needed to be delivered to museum visitors interactively and be modified to accommodate new data. The solution that fulfilled these four basic requirements was the proposed EarthLab Data Hyperwall Exhibit. This project specifically dealt with designing and building the GIS application used for controlling and managing the Hyperwall Exhibit.

3.2 Requirements Analysis

The undertaking of this project necessitated breaking its intended purpose into clear statements describing the project's technical needs, behaviors, and goals. The client wanted the platform to educate visitors about environmental topics such as the Museum's own research. The client selected the hardware for this project based on previous hyperwall installations. Additionally, the client knew they wanted the project's maps to incorporate three datasets: heat illness vulnerability data collected by the Museum in the city of Richmond, VA; an air quality index for the United States; and a series of before/after satellite images of areas affected by natural disasters and other events. The client delivered the Heat Vulnerability data as a spreadsheet. The other two datasets needed developing. The three datasets needed conversion into map products that the Hyperwall's application could load and display.

A requirements analysis occurred after consulting with the client about long-term goals for the project and discussing the reasons for the project's inception. The purpose of the requirements analysis was to convert the client's abstract desires for the project into objectives geared towards accomplishing the client's needs. The needs were documented using precise and specific language outlining the capabilities, functions, and features of the system. This process required consideration of the project's deadline. Requirements analysis included capturing both functional and non-functional requirements (Tomlinson, 2007). Functional requirements refer to what the system should accomplish, and non-functional requirements detail how the system achieves those goals.

3.2.1 Functional Project Requirements

Project stakeholders produced a list containing the Hyperwall Exhibit's core capabilities from cycles of discussing and refining project needs. The Museum needed interactive two- and three-dimensional maps to support the Hyperwall's educational topics. The Hyperwall needed a touchscreen user interface that visitors could use to interact with the maps. The touchscreen needed to interact with the rest of the system and update the display screen array with visitor input. Achieving the functional requirements supported the system's overall goal of allowing visitors to explore dramatically different datasets about their immediate environment and the global climate. Ideally, visitors would also improve their map comprehension abilities. Table 1 contains the functional requirements for the system.

Table 1. Functional Requirements

Requirement	**Description**
Presentation of data in application	The system should present the three chosen spatial datasets as interactive maps using ArcGIS Runtime SDK for .NET.
Natural Disaster Scene data management	The system should allow users to compare before/after event imagery in the Natural Disaster module with gesture controls.
Air Quality Index Map data features	The system should provide a search widget for users to locate areas of interest on the Air Quality Index Map module.
Heat Vulnerability Map Management	The system should present the SMV's research into Heat Vulnerability in the city of Richmond, VA as a Map module in the exhibit's application.

3.2.2 Non-Functional Project Requirements

The non-functional requirements defined how the system achieved the goals and objectives of the project. The non-functional requirements determined software and hardware related concerns, determinations related to the integrated development environment (IDE), and interaction with web-based data. The development environment included Visual Studio 2017 (the IDE), Windows Presentation Foundation (the API), ArcGIS Runtime Software Develop Kit for .NET (the SDK) and was built using the C# programming language and the XAML declarative language. This environment developed based on the technical and operational requirements outlined in Table 2.

Table 2. Non-Functional Requirements

Requirement	Description
Application Reliability	The WPF application wrapper for the project should not have any critical errors during runtime.
.NET Framework use	The system must be compatible with Windows 10.
Project wrap-up documentation	Documentation for system operation, maintenance, and adding data shall be delivered with the software.
Extending Natural Disaster Map module	The application should enable system administrators to easily add additional imagery to the Natural Disaster Map module.
Accessing ArcGIS Online/Enterprise	The system should interact with the client's ArcGIS Enterprise or Online account to access exhibit maps.
Use of Esri Software Developer Kit	The system should use Runtime SDK for .NET to manage data presentation and visual features (compass, legend, etc.).
Visitor Accessibility	The system should be easy to use for young children and adults with no prior GIS knowledge.

The core requirements for the Hyperwall Exhibit's application included the need for accommodating current and future datasets/features, stable loading of the chosen maps, and smooth runtime operation. The application's framework and system architecture were carefully developed based on a few key factors to meet these requirements.

3.3 System Design

The system design phase consisted of taking the system's core functionalities and developing them into a series of technical solutions. The final system needed to successfully read spatial data from the web and local sources and engage visitors through a well-designed user interface. The methodology for creating the system covered several areas: overall system architecture, hardware architecture, development environment, and web GIS.

3.3.1 System Architecture Design

The system architecture determined where each component of the system belonged, including hardware, software, and web-based system components. It also loosely defined the interactions and relationships between each part of the system. Figure 3-1 depicts the system architecture for the entire Hyperwall Exhibit.

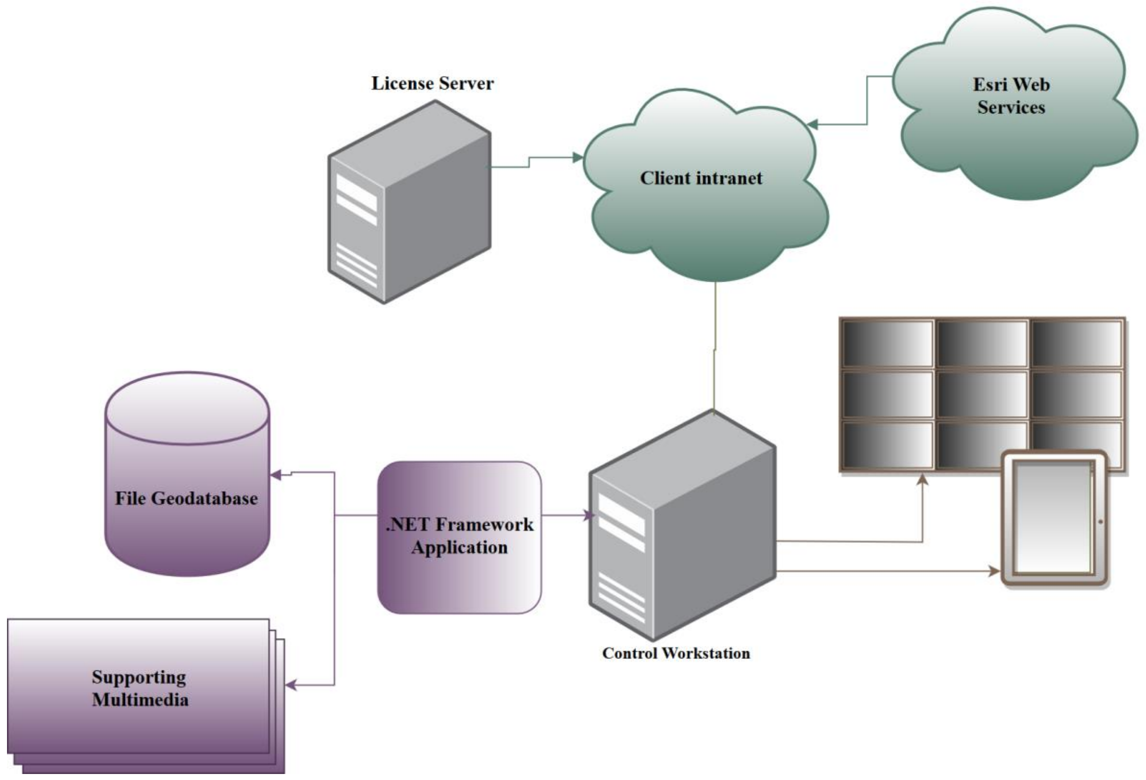

Figure 3-1: Hyperwall System Architecture

The system included both local and web-based GIS data and supporting educational media. The system should access the online data through Esri’s Web Services using the Museum's Esri license, while local data resided on the control computer's hard drive in a file Geodatabase or local directories. The system's design includes both online and offline data options because the Museum wanted to further develop the system's maps, data, and supporting materials. Ideally, the client could create content on their ArcGIS Portal or locally using ArcGIS Pro, and the system would use Esri’s Runtime SDK for .NET to bring the content into the exhibit.

3.3.2 Data and Map Product Design

Three maps were produced for the system: Heat Vulnerability in Richmond, Live Air Quality for the Contiguous United States, and Natural Disasters of the World. The

Museum produced the data for the Heat Vulnerability Map and stored the data in a table. The table was spatially linked to Census Block Groups to create vector Feature Layers. The Heat Vulnerability map teaches visitors (many of which are Richmond and Virginian locals) about factors that contribute to dangerous Urban Heat Islands within the city.

The Live Air Quality Map contains two layers: an air quality sensor location feature layer, and an interpolated surface feature layer derived from the sensor locations. The Air Quality Index (AQI) displayed in the map is displayed ordinally by the interpolated surface and numerically at each sensor location. This map shows visitors a little about the process of air quality monitoring and contextualizes it within their own communities.

The final map consists of before and after imagery of natural disasters and events (volcano eruptions, hurricanes, flood, etc.). This Natural Events Map needed georeferenced imagery (satellite or aerial), which could be overlaid and compared. The imagery was sourced from NASA's Earth Observatory and placed onto a 3D virtual globe that visitors can explore through bookmarked locations or through gesture controls.

Each of these maps required different data formats such as feature layers derived from table data, live data from federal sources, and high-res satellite imagery (georeferenced .TIFs). The air quality surface for the Live Air Quality Map is hosted on Esri's ArcGIS Online and produced by the EPA. Because of the high computational and storage cost of caching imagery through ArcGIS Online, the imagery for the Natural Disasters of the World map needed to be stored locally on the control computer.

3.3.3 Hardware Design

The client determined the hardware needs for the Hyperwall Exhibit. Figure 3-2 depicts the proposed hardware architecture. The exhibit's hardware includes the video wall, touchscreen, and control computer. Visitors to the Museum interact with the video wall through a graphical user interface (GUI) hosted on the touchscreen. A Windows 10 station hosts the software for controlling each display and manages transactions/communication between the displays. The control computer is outfitted with a multiplexer for managing the video wall. A multiplexer is a device used for splitting a computer's display across multiple LCD screens.

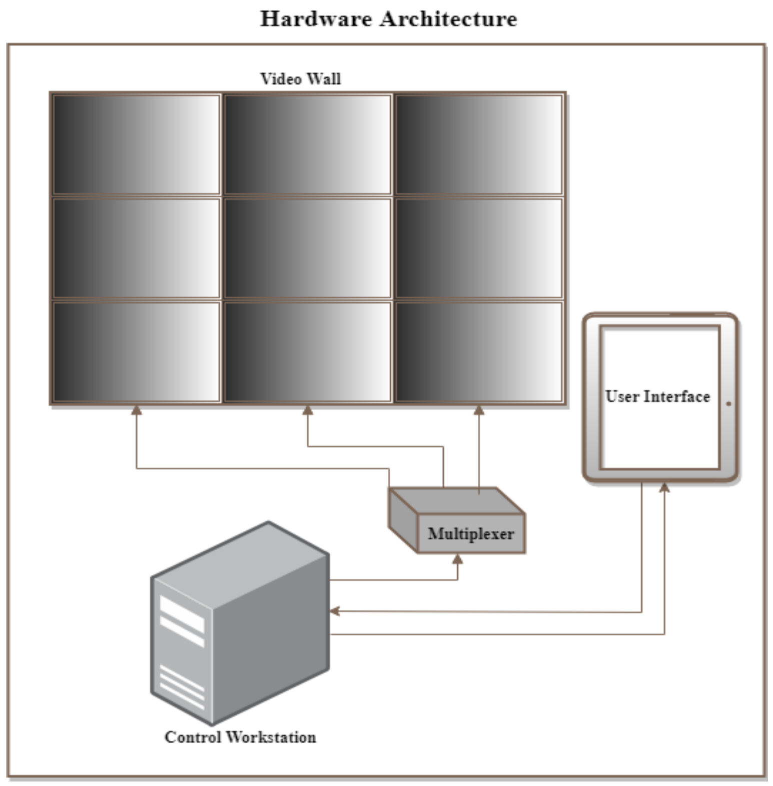

Figure 3-2: Hardware Architecture

The number of individual screens composing the video wall is still being discussed internally by the client, who will build and implement the final hardware components for the exhibit.

3.3.4 Software Design / Development Environment

The Hyperwall Application consists of a custom-built Windows Presentation Foundation 4.5 (WPF) application that accesses the Hyperwall Exhibit’s three maps from web-based and local sources with Esri’s ArcGIS Runtime Software Developer Kit (SDK) for .NET v100.5. The application was developed in Microsoft’s versatile Integrated Development Environment (IDE), Visual Studio 2017. The Hyperwall Exhibit Application used Esri’s Runtime SDK for .NET, which managed the GIS aspects of the application such as bringing in map data, defining feature functionality, and accessing Feature Services through REST endpoints.

Based on the requirements, the final project needed to be a Windows 10 .NET Framework application that implemented best practices for the MVVM design pattern and incorporated Esri’s Runtime SDK for .NET to manage the three currently selected maps. The software was designed to be able to accept future map products and supporting information.

3.3.5 Web GIS

The system incorporated web-based Feature Services through Esri’s online server. ArcGIS Runtime SDK for .NET can access Feature Services through the URLs of their REST Endpoint. The Runtime SDK creates C# objects from the URLs. The URLs are then used to create map layers in the application. By building a model in the application that accomplished this task, the client can implement maps or feature layers through ArcGIS Online or through their Portal for ArcGIS Enterprise.

3.4 Project Plan

The project plan came from a combination of Rapid Application Development (RAD) workflows and project deadlines laid out by the GIS department. The basic structure of any RAD project is divided into four basic phases: Requirements Planning, Prototyping, Testing, and Implementation. These four stages applied to the development process of the Hyperwall Application. Adjustments were made to the project plan throughout the project, based on reevaluation of system architecture and the need for additional time to developing proficiency with Object-Oriented Programming and C# language syntax.

3.4.1 Original Plan

The original plan was scheduled based on program deadlines and topics taught in project management courses. Concepts from those courses determined the format, structure, and timeline for the project plan. The client provided the data during project initiation, and a requirements analysis occurred early in the planning process. However, project stakeholders only understood the basics of the Software Development Life Cycle (SDLC), which caused problems later in the project. The plan split into major phases (a combination of program milestones and RAD/SDLC stages), which were then divided according to tasks signaling phase objective completion (Table 3).

Table 3. Original Project Timeline

Phase	Task	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug
Project Initiation	Proposal	X	X									
	Establish client contact		X	X								
Self-Training	Workshops, online training		X	X	X	X	X	X				
Planning	Scoping	X										
	Requirements Capture		X									
	Literature Review			X								
	Build Development Environment			X	X							
	Finalize Plan					X						
Data Preparation	table data to vector format				X							
	Collect raster imagery					X						
System Design	Create conceptual model					X	X					
	Determine C# toolkits						X					
Prototyping	Write executable code						X	X				
Testing	Run and debug code						X	X	X	X		
Deployment	Create application installer										X	
Finalization	Prepare system for transfer										X	
	Deliver to client											X
User Conference	Present									X		
Defense												X

3.4.2 Plan Revisions

The original plan needed revision when it became clear that the system's application would need additional time for development and testing. Initially, the plan allotted two months to prototyping the application. However, synchronizing the system's GUI with the video wall proved more difficult programmatically than initially thought. Scope reduction occurred to accommodate the extra time needed for redesigning the system architecture. As a result, two planned application features were removed from the scope.

3.5 Summary

The project design combined several related areas crucial to the project's success. The three distinct elements of the project were the requirements analysis, system design, and project plan. The client's requirements were identified through a process of consultation and documentation based on the stated problem, goals, and client needs. The requirements constrained the system design to specific deliverables and structures. The perceived labor required for developing each part of the system determined the time allotted to a given phase or objective. Problems encountered during the project were contextualized within the system and addressed in relation to the corresponding components. Through careful planning and conceptual design of the entire system, problems related to data, features, or functionality did not affect the overall project's success.

Chapter 4 – Database Design

The methods of data storage and access determine many structural components of a system incorporating different formats of spatial data from various sources. Well-designed data storage ensures that data is stored safely, permanently, and is free of redundancy. Best practices for data storage design assume the domain closure axiom; stating that only elements contained within the system are relevant to the system. The closed world assumption supposes that all elements within the system are considered true in the system design. Any data absent from the system is false. Following these principles while examining the data and data storage needs for the Hyperwall Application ensured that all data types had a place in the system, no unused data or datatypes existed in the system, and that every dataset had one storage location and one access point.

The one caveat to these guiding principles was the core project requirement allowing the client to add datasets for future use to the EarthLab Data Hyperwall Exhibit. Some assumptions about future datasets: only vector or raster data readable by Esri's ArcGIS Pro, ArcGIS Online, or ArcGIS Runtime SDK for .NET belongs in the system; all data processed by the system display as Feature Layers or Raster Layers; and all datasets render as standard map products.

4.1 Conceptual Data Model

The conceptual model abstractly considered the interaction of all entities involved in the system. This included physical entities, such as the Museum and its visitors, and conceptual entities more directly linked to data processing and control. Organization of the model aided understanding of the relations between system operators, data, and hardware/software components. By exploring these relationships, a model of how they operate in the real world developed. Concrete behaviors governing the system could then be inferred and used in system implementation.

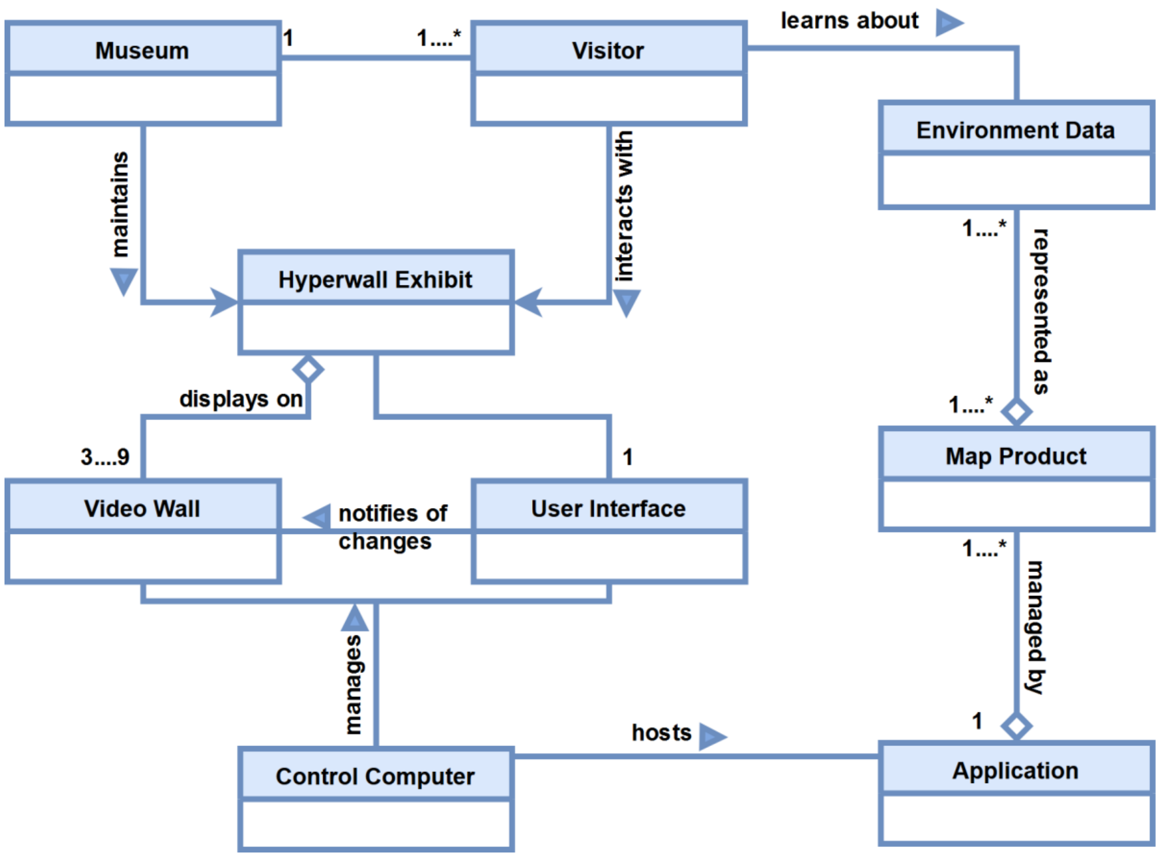

Figure 4-1: Conceptual Data Model

The structure of the conceptual model diagram involved all real-world entities interacting with the system. The system includes human elements, spatial data, data outputs, software entities, and hardware entities. It was made using Unified Modeling Language as the format for visualizing entities within the system.

The conceptual model diagram (Figure 4-1) shows the relationship between each part of the system. Environmental Data represents the most basic element in the system and the client’s inspiration for the exhibit. The system takes raw environmental data inputs and shares it with the system’s users, represented by the Visitor class. The Museum maintains the exhibit and supplies the Visitor with consumable information through the aggregated Hyperwall Exhibit platform.

The Museum plans to use up to nine LCD screens for the Video Wall portion of the Exhibit. The Video Wall will rely on the User Interface for its display and change when Visitors interact with the User Interface. This tripartite relationship is the most crucial functional part of the system. Further, several of the project requirements dealt with Visitor interaction with the exhibit, such as needing a mild learning curve and providing overall enjoyable user experience.

4.2 Logical Data Model

The conceptual model abstractly rendered the entities involved in the project's data system. The conceptual model aided in identifying the relationships between the entities and understanding the organization of components within the system. The next step was building a logical data model. The logical data model outlined a data structure for the Hyperwall Application that was derived from real-world requirements. The logical model identified datasets with specific operational elements and represented each element in tabular structures.

The Environmental Data entity in the conceptual data model represented all datasets accessed and displayed by the Hyperwall Application. It also represented the core information the Museum visitors explore in the Hyperwall Exhibit. There are three datasets comprising the Environmental Data entity, each containing data about different aspects of the environment. The organization and preparation of the Heat Vulnerability dataset, Natural Disaster dataset, and Air Quality dataset determined several aspects of the system’s usability. First, how consistently and easily the data is converted to map/information products that can be read by the Hyperwall Application. Second, the process needed to be repeatable so the client could augment the data in the future. The design of each data entity accounted for these two considerations.

After reviewing the available data management options, it was decided that a file Geodatabase associated with an ArcGIS Pro Project would be the best data storage platform. Use of a file Geodatabase coincided with to goal of easing the client's handling of the data. The client was familiar with Esri's ArcGIS Pro/ArcGIS Online and had access to the software through the SMV's Enterprise account. Figure 4-2 shows the implemented data design, including the Geodatabase and web GIS storage structure. Also shown are the information products (as read by the Hyperwall Application) in the system. The datasets were output to ArcGIS Online Feature Services, a Mobile Map Package, and two folders containing imagery and pyramid files. ArcGIS Runtime SDK for .NET can access and display these formats.

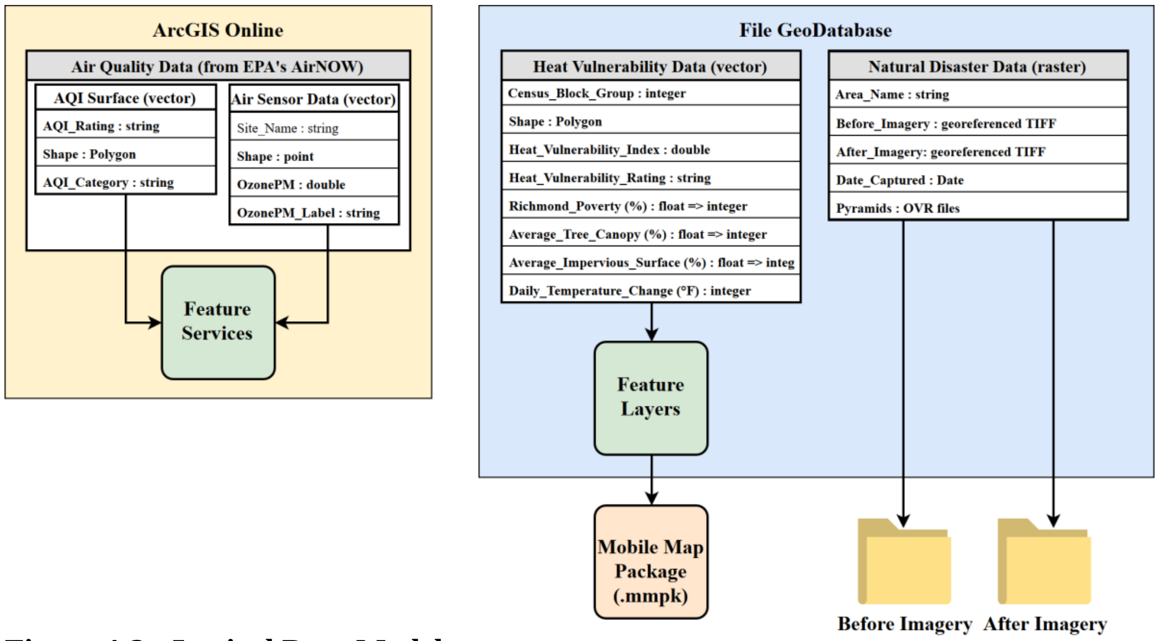
ArcGIS Online
Air Quality Data (from EPA's AirNOW)
AQI Surface (vector)
AQI_Rating : string
Shape : Polygon
AQI_Category : string
Air Sensor Data (vector)
Site_Name : string
Shape : point
OzonePM : double
OzonePM_Label : string
Feature Services
File GeoDatabase
Heat Vulnerability Data (vector)
Census_Block_Group : integer
Shape : Polygon
Heat_Vulnerability_Index : double
Heat_Vulnerability_Rating : string
Richmond_Poverty (%) : float => integer
Average_Tree_Canopy (%) : float => integer
Average_Impervious_Surface (%) : float => integ
Daily_Temperature_Change (°F) : integer
Natural Disaster Data (raster)
Area_Name : string
Before_Imagery : georeferenced TIFF
After_Imagery: georeferenced TIFF
Date_Captured : Date
Pyramids : OVR files
Feature Layers
Mobile Map Package (.mmpk)
Before Imagery
After Imagery

Figure 4-2 : Logical Data Model

4.2.1 Heat Vulnerability Dataset

The Science Museum of Virginia (SMV) produced the Heat Vulnerability Dataset, which contained information about heat illness risk in the City of Richmond. A Heat Vulnerability Index demonstrated the heat-illness risk using a nominative scale. The index derived from several climatic measurements divided geographically based on Richmond's Census Tract numerical designations.

The data provided by the SMV was in a Microsoft Excel table. The table needed to have excess leading and trailing spaces stripped from the fields and needed associating with polygon features for use in ArcGIS Pro. A Python script was created and ran to remove the extra spaces. The second issue was remedied by splitting the GeoID field on the table (see Figure 4-3) and joining the newly created Tract and Block Group fields to a shapefile containing the corresponding boundaries for Richmond.

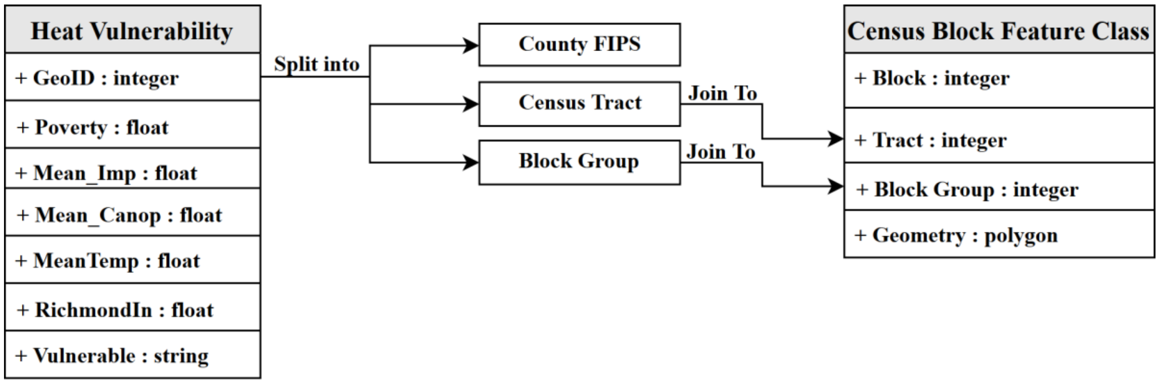

Figure 4-3 : Heat Vulnerability Dataset

Each row in the feature classes corresponded to a Census Block Group number that was spatially joined to a polygon layer using Block Group as the joining field. After generating a feature class in ArcGIS Pro from the Heat Vulnerability table data, the data was split into five separate feature classes corresponding to fields in the table: Average Temperature Change (MeanTemp), Average Tree Canopy Coverage (Mean_Canop), Average Impervious Surfaces (Mean_Imp), Richmond Poverty Level (Poverty), and a

Heat Vulnerability Index (Richmondln) derived from the first three measurements. The Museum ranked the Heat Vulnerability Index using a qualitative scale ranging from "Lowest" to "Highest" in the Vulnerable field.

These five classes became each of the five layers for the final Mobile Map Package (.mmpk) read into the Hyperwall Application. The rationale for generating a .mmpk instead of reading the feature layers individually from the geodatabase was so the feature symbology would be retained in the application. The alternative would be to designate the symbology using the ArcGIS Runtime SDK for .NET. This would have made it difficult for the client to modify the dataset without potentially impacting the application's runtime reliability.

4.2.2 Natural Disaster Dataset

The Natural Disaster dataset came from gathering satellite imagery from NASA's Earth Observatory, a massive repository of imagery and other materials provided free for public and commercial use (Levy & Przyborski, 2019). The imagery used in the Hyperwall Application was collected by instruments on a variety of satellites including Earth Observing-1 ALI, Terra MODIS, Landsat 8 OLI, and Landsat 7 ETM+. The system stored all images in a geo-referenced TIFF format. After downloading the imagery, it was opened in ArcGIS Pro to build pyramids for faster loading. Two folders within the geodatabase (Before Imagery and After Imagery) contain all the imagery. The application accessed the imagery through these two folders, and any additional imagery is accessed in the same manner (see Chapter 5 for a complete discussion of the workflow).

4.2.3 Air Quality Dataset

The third dataset the SMV wanted for the Hyperwall Exhibit was an Air Quality Index (AQI) represented as vector geometries. This dataset needed to derive from regularly sampled air quality sensors. Developing an AQI surface from raw data was outside of the scope of this project but will likely be completed for use in the Hyperwall Exhibit in the future. However, to prove the system's ability to access and display similar "live" air quality data, a substitute was found.

The US Environmental Protection Agency (EPA) maintains an extensive network of sensors, which detect airborne ozone and particulate matter. The EPA regularly samples sensors and disseminates the data through an ArcGIS Online Web Map. The web map is named AirNow Interactive Map of Air Quality, US EPA, OAR, OAQPS. The AirNow web map contains point geometry layers of sensor locations, and interpolated vector surfaces generated based on sensor readings. The interpolated surfaces show current, past, and forecasted air quality information for the contiguous United States. This web map is publicly available and has no use restrictions (Smith, 2019).

The web map consists of dozens of layers containing past, present, and forecasted air quality measurements. To ensure accessibility for visitors of varying technical ability, the application only used two layers of the AirNow web map. The first was a point collection of air sensor locations (labeled based on city of residence). The second consisted of an interpolated surface updated every hour. The surface feature layer displays an AQI derived from ozone and particulate matter measurements. The layers are stored as feature services hosted by the same account as the EPA's AirNow ArcGIS Online account. As a result, no additional preparation was needed for the Air Quality data prior to setting up the web GIS portion of the application.

4.3 Summary

The data design phase of the project ensured that there was a comprehensive plan for creating, storing, and managing the data required for the project. Additionally, this phase ensured the data used in the project supported the purpose of the project and had clearly defined relationships with other entities involved in the project. All dependencies between the data, .NET application, and the real world were identified. Following this, a logical model detailed the process of building, cleaning, and preparing the three datasets used in the project. The design of the datasets accounted for modification after project completion.

This project was unusual because of its use of three discrete datasets as input, with relatively little analyses or output datasets. The outcome of the data design was instead to ensure the client would be able to manage the datasets in the context of the Hyperwall Exhibit as a whole, according to the client's needs. Consideration of the entire system has several implications for the datasets: all data needed publicly available sources with no restrictions on re-use or display, data needed to be permanently available, and the data design needed to operate reliably within the client's existing GIS infrastructure.

The data design was revised as needed throughout the project to account for ArcGIS Runtime SDK for .NET data access requirements. A new version of the SDK was released during the project and the data storage needed revaluating to ensure reliable access. The datasets were modified over time to interact better with the SDK and be more maintainable for the client. Time was spent improving file naming conventions and associated file directory structure. The map/information product outputs were modified to improve visual hierarchy and overall comprehensibility, which required occasional modification of the root data throughout the project.

Chapter 5 – Implementation

Development of the EarthLab Data Hyperwall's control application required the implementation of GIS and Windows 10 software to manage the exhibit's hardware, Geodatabase, and Web GIS. This chapter discusses the process of creating a development environment that incorporated Esri and non-Esri software to build the application, which managed the Hyperwall Exhibit's hardware architecture, GIS data, and graphical user interface (GUI). Figure 5-1 illustrates the overall system architecture indicating the flow of data from storage formats to the Hyperwall application. The application contained logic dictating a set of rules and behavior for managing the transfer of information between the end user and the project's data. Additional code defined processes managing user input and response.

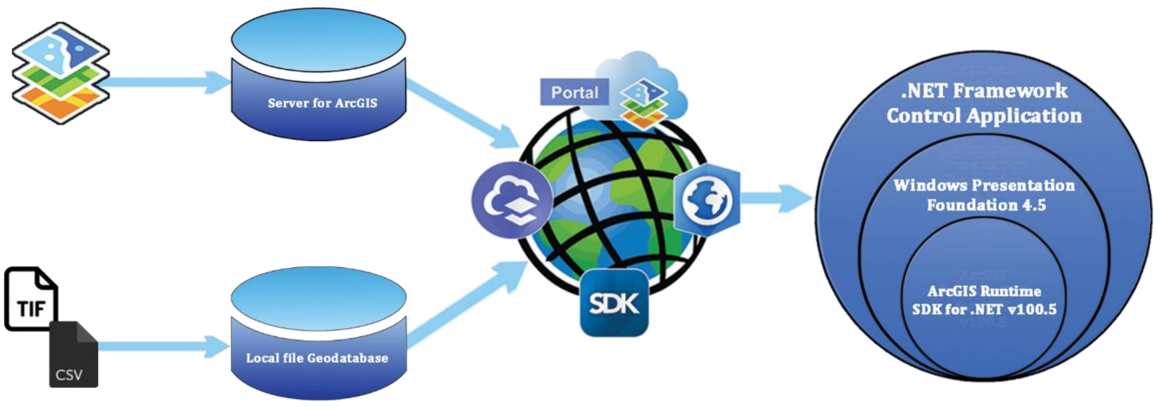
Server for ArcGIS
Portal
SDK
TIF
CSV
Local file Geodatabase
.NET Framework Control Application
Windows Presentation Foundation 4.5
ArcGIS Runtime SDK for .NET v100.5

Figure 5-1: System Architecture

5.1 Development Environment

The first step in the construction of the Hyperwall Application was building the development environment. The software packages used were selected based on the client's project requirements. The client planned for the exhibit's control computer to run the Windows 10 operating system and access data through both a file Geodatabase and the client’s Enterprise account. The final application needed to synchronize the Hyperwall’s video wall display with the touchscreen in response to user input. Meeting the client’s needs required the implementation of a suite of GIS and non-GIS software. Table 4 summarizes the software used for the development of the Hyperwall Application.

Table 4. System Software

Software Usage	**Name**	**Version**
Operating System	Windows 10	-
Framework	Windows .NET	4.7.2
Integrated Development Environment	Visual Studio Community Edition 2017	15.8.7
APIs	Windows Presentation Foundation	4.5
GIS Software	ArcGIS Pro ArcGIS Online	2.3.2 -
SDKs	ArcGIS Runtime ArcGIS Runtime Toolkit	100.5 100.4

5.1.1 Non-GIS Software

Visual Studio 2017 Community Edition was selected as the Integrated Development Environment (IDE) for this project. An IDE is different from a simple code editor in that it incorporates options for multiple programming languages, Application Programming Interfaces (APIs), and Software Development Kits (SDKs). The Visual Studio IDE was created by Microsoft primarily for use with Windows Operating Systems. Visual Studio is well-established and widely used with reliable support for Windows 10. Another important benefit to using Visual Studio was the ease of adding third-party packages to an in-progress solution, necessary for managing the different data types and operations present in the Hyperwall Application. The application’s system architecture changed as the project progressed, which required frequent modification to the development environment; a process that Visual Studio simplifies.

Visual Studio supports several APIs for creating .NET Framework applications. Windows Presentation Foundation (WPF) 4.5 was chosen as the API for this project. Windows Presentation Foundation was designed for use with Window’s .NET Framework, which ensured compatibility with the other software or code toolkits used in this system. The API incorporates the C# programming language for application business logic and Extensible Application Markup Language (XAML) for GUI design logic.

Additionally, the Model-View-ViewModel (MVVM) pattern used to structure the business and design logic for the Hyperwall Application was created specifically for use on WPF. This pattern decouples the presentation logic (GUI appearance) from the business logic (operations) of an application (See Chapter 2.4 for a full discussion of MVVM).

The application's Models and ViewModels were written using C# syntax, and the Views were written using XAML. XAML largely consists of definitions for front-end GUI controls (Buttons, TextBoxes, etc.). Many XAML controls respond to explicitly defined user behaviors, such as a user pressing or clicking a button on the GUI. A property in the GUI control detects the change and activates a C# event in the View-Model. Events are blocks of C# code that execute when activated by their corresponding XAML control. An important part of the MVVM pattern is handling these events and ensuring that user input notifies the correct program elements and updates the rest of the program. Figure 5-2 illustrates an example implementation in the Hyperwall Application.

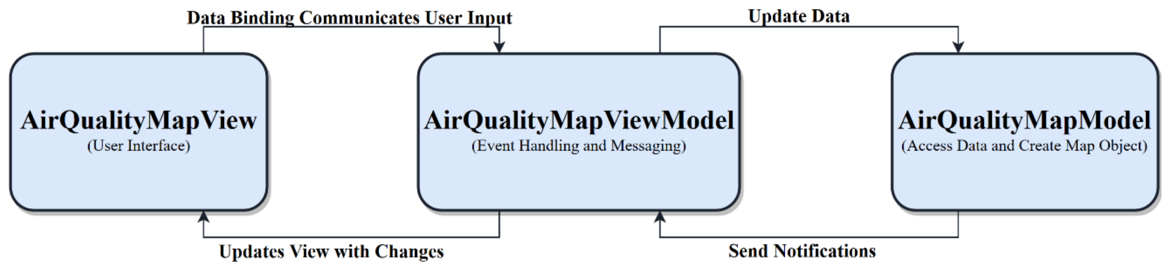
Data Binding Communicates User Input
Update Data
AirQualityMapView
(User Interface)
AirQualityMapViewModel
(Event Handling and Messaging)
AirQualityMapModel
(Access Data and Create Map Object)
Updates View with Changes
Send Notifications

Figure 5-2: MVVM Pattern Example in the Hyperwall Application

In Object-Oriented Programming, the code is divided into objects (also called instances), which are class definitions loaded into the memory during runtime that inherit behavior from referenced class libraries. The application primarily inherited classes from .NET Framework class libraries to reduce the complexity of its infrastructure and internal behaviors. The development environment needed supplemental GIS class libraries for handling spatial data and managing the application's maps.

5.1.2 GIS Software

The development environment described so far provided a platform for building the Hyperwall Application, but there was no built-in spatial data management. By supplementing the developer environment with Esri's ArcGIS Runtime SDK for .NET, the application could be built to accept spatial inputs such as maps, scenes, and spatial data formats. The Runtime SDK exposes capabilities of the ArcGIS platform's programming and contains many of the same rendering abilities as ArcGIS Pro, such as displaying vector and raster data on maps, applying geolocation services, and geoprocessing tools. The advantage of the Runtime SDK is it provides finer GUI control compared to ArcGIS Pro. The Runtime SDK connected to the IDE through Visual Studio's built-in NuGet Package Manager. At this point, the development environment was ready to accept the prepared spatial datasets, build models to display the accessed data as symbolized maps with basemaps and incorporate operational logic for handling events throughout the application.

5.2 Application Architecture

A significant challenge for the Hyperwall Application's architecture was managing the system's proposed display hardware. The system design included a video wall comprised of an LCD screen array (treated as a single display within the Hyperwall Application) and touchscreen. The application needed to detect changes to the touch-screen GUI and update the video wall with those changes. The application's back-end architecture was designed to manage these two requirements, regardless of what map models were implemented.

Designing the system architecture required adherence to the principles of Object-Oriented Programming. For the application to show the same map data, with the same symbology (different extents and GUI controls), on two separate screens required creation of multiple instances of map classes that relied on the system architecture to communicate changes and remain synchronized.

Following the MVVM-based design pattern, the application is divided into Views, ViewModels, and Models. The application's Views correspond to XAML documents that contain only the GUI control definitions. The ViewModels contain the logic that dictates the behavior of the GUI controls (such as Buttons) and exposes properties of the Models. The Models contain logic for accessing the application's input datasets and implementing the *INotifyOfPropertyChanged* interface. The *INotifyOfPropertyChanged* interface is a set of commands that ensure all code objects are updated with changes happening either at the data-level or user interface-level of the application.

The Hyperwall Application's structure did not perfectly follow the MVVM pattern because some elements of the ViewModel classes directly referenced GUI controls in the Views, but it is a convenient way to organize the discussion of different classes in the Hyperwall Application.

The ViewModel corresponding to the GUI managed creation and placement of the two displays. If there are two screens available, it detects which is the primary screen, and which is the secondary screen. This operation queries the display settings of the computer for the settings of the primary and secondary screens. The GUI window is assigned to the primary screen/display, and the video wall window is assigned to the secondary screen/display. If only one screen is available, the application implements error handling to ensure that only the user interface window is created and prevent runtime error. Figure 5-3 displays the logic of this process.

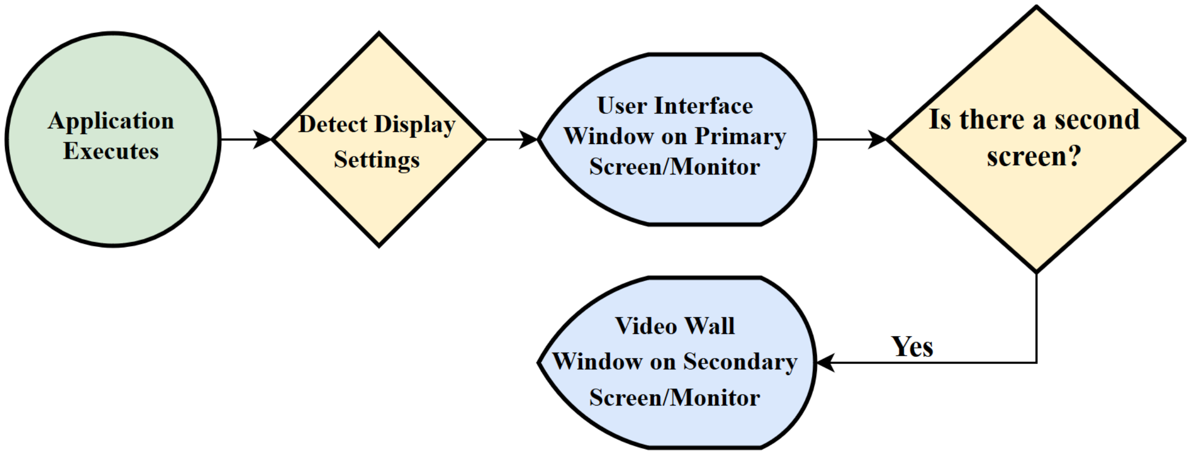
Application Executes
Detect Display Settings
User Interface Window on Primary Screen/Monitor
Is there a second screen?
Yes
Video Wall Window on Secondary Screen/Monitor

Figure 5-3: Window Object Creation

After creating the User Interface Window and Video Wall Window on the appropriate displays, the Hyperwall Application needed to synchronize the two windows. When visitors enact change on the GUI, such as changing a map extent through touch controls or pressing a button, the map-object being held in the video wall window must also show the change. Synchronizing the displays is conducted through programmatic event handling and messaging (Figure 5-4).

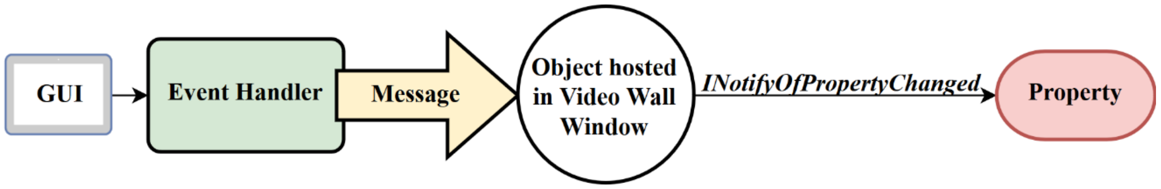

Figure 5-4: Event Handling and Messaging

The visitor enacts some changes on the GUI, such as dragging the open map to a new extent or pressing a button. The control manipulated notifies the associated object of a change through databinding of a control property to an event or through the *INotifyOfPropertyChanged* interface. This fires an event handler, which executes some code. For a map extent change, this code would get the new map extent from the GUI and update the map hosted in the video wall. When the receiving object (in this case the map hosted on the video wall display) receives that information, it updates the corresponding data-level property with this new information. Fortunately, messaging between the sending object and the listening property is handled through the *INotifyOfPropertyChanged* interface and did not require explicit definition.

In summary, the Hyperwall Application system architecture detects the hosting computers display configuration and relies on a system of change detection for user-driven GUI changes and passes that information to an event handling and messaging service to notify data-level properties of the update.

5.3 Preparing Map Products

The client wanted the Hyperwall to have three topics for their visitors to explore and learn about. These topics each have an associated spatial dataset presented in the application as 2-D or 3-D maps alongside a set of user interface controls. The three educational topics were Heat Vulnerability, Air Quality, and Natural Disasters. Developing the application required the deployment of several Runtime SDK library classes because each subject's associated dataset was of a different spatial data format and had a different educational intent.

5.3.1 Building the Heat Vulnerability Map and Model

The Museum expected the final platform to disseminate the Museum's research to visitors. Dr. Jeremy Hoffman, client representative and climate scientist with the Museum conducted research exploring the Urban Heat Island effect in the City of Richmond. He and his team with the Museum collected data for each Census block group in Richmond and created a measurement of Heat Vulnerability. Three separate metrics comprised the Heat Vulnerability attribute: mean daily temperature change, average impervious surface coverage (pavement, asphalt, concrete buildings, etc.), and average tree canopy coverage. The Museum created the Heat Vulnerability risk range from these layers. The values ranged from 0.00 – 4.00. The values were then sorted into five nominative categories ranging from "Lowest" to "Highest" risk to ease comprehension of the index. This dataset was the basis for creating the Heat Vulnerability subject for the Hyperwall Application. The dataset also contained a Poverty in Richmond layer that does not contribute to the Heat Vulnerability Index but was instead included so visitors using the Hyperwall could draw their own conclusions about the connection between impoverished areas and heat illness vulnerability. Figure 5-5 illustrates the relative relationships between these layers.

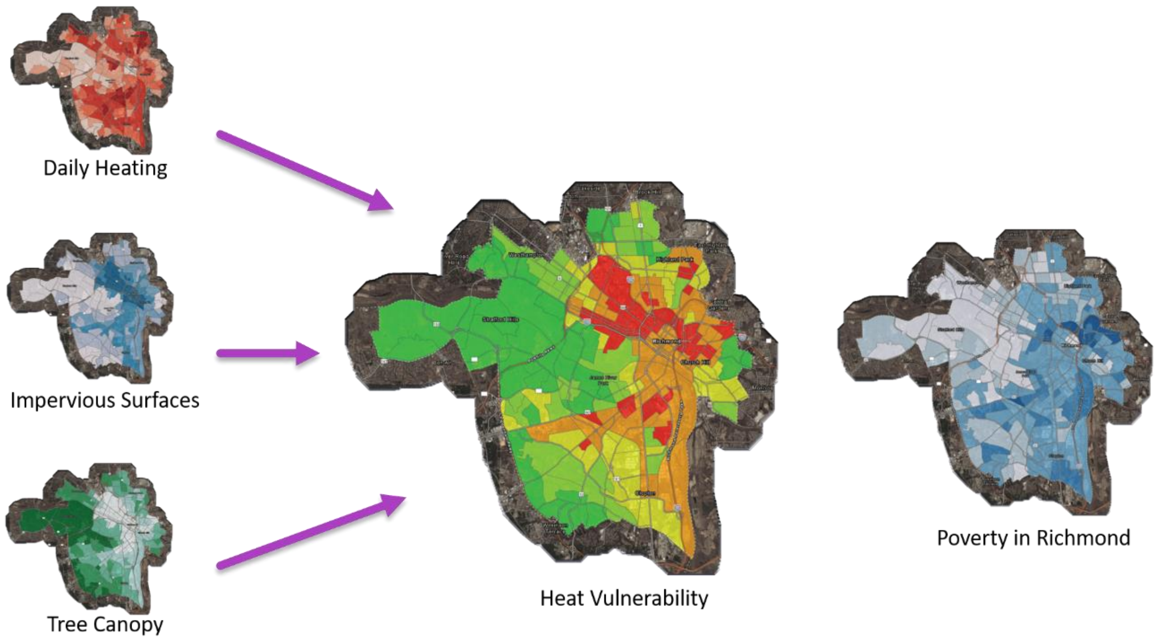

Figure 5-5: Heat Vulnerability Dataset

The primary educational goal for the Heat Vulnerability section of the application was to teach visitors about the danger of heat illness in the City of Richmond (the home for many of the Museum's visitors). The secondary educational goal was not to spread public health data, but instead to improve visitor's spatial literacy through map interaction. These two goals guided the process of turning the Heat Vulnerability tabular data into a fully interactive portion of the application.

As discussed in Chapter 4, the raw Heat Vulnerability data were spatially joined to a polygon Census block group feature layer through the shared block group identification field. The Heat Vulnerability was stored in the project file Geodatabase as feature layers representing each field in the table. Originally, the plan was to have the layers load directly into the application from the File Geodatabase. However, ArcGIS Runtime SDK for .NET does not support direct loading from a File Geodatabase. Instead, the layers were exported from an ArcGIS Pro map project as a Mobile Map Package (.mmpk) with custom symbology. The .mmpk format is intended for offline map use, which is appropriate for the Heat Vulnerability data. After export, the .mmpk was placed into the MobileMapPackages folder within the Visual Studio project's file directory. The application used the Runtime SDK's MobileMapPackage Class to open the HeatVulnerabilityMap.mmpk from the MobileMapPackages folder. After reading the .mmpk file, the Heat Vulnerability Map Model created a map object and generated a basemap on the fly through Esri's online basemap service. At this point, the Model inserted the map into a XAML control via the corresponding ViewModel. Figure 5-6 depicts this workflow.

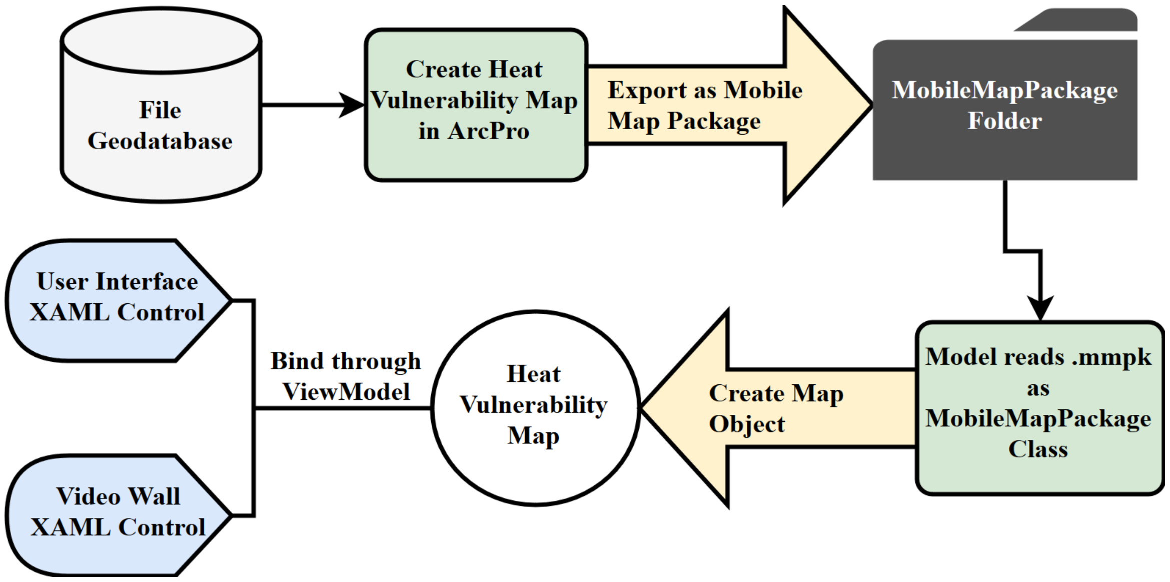
File Geodatabase
Create Heat Vulnerability Map in ArcPro
Export as Mobile Map Package
MobileMapPackage Folder
Model reads .mmpk as MobileMapPackage Class
Create Map Object
Heat Vulnerability Map
Bind through ViewModel
User Interface XAML Control
Video Wall XAML Control

Figure 5-6: Heat Vulnerability Map Model

5.3.2 Incorporating the Air Quality Feature Service and Model

The second topic in the application is Air Quality. The original plan called for deriving an air quality index from a JavaScript Object Notation (JSON) format document from a company called PurpleAir that posts their sensor data daily and makes it freely available to researchers and educators. However, due to the time constraints of the project, creating the logic for reading the JSON, generating an AQI from its attributes, and symbolizing it as a map format that the ArcGIS Runtime SDK could read was not possible. Fortunately, the Environmental Protection Agency (EPA) produces a Web Map called AirNow, which is publicly available through ArcGIS Online and contains hourly updated air quality information for the contiguous United States. The feature service for the AirNow Web Map contains air quality readings tied to point geometry sensor locations and interpolated air quality surfaces derived from these points.

The AirNow Web Map contains over a dozen layers representing different types of air quality measurements over different temporal ranges and with different geometry. The layers measure Particulate Matter (PM) of various sizes, ozone, and some layers display AQI values. The map layers include point geometry of approximate sensor locations and interpolated surfaces generated from the sensor readings. For the purposes of this project,

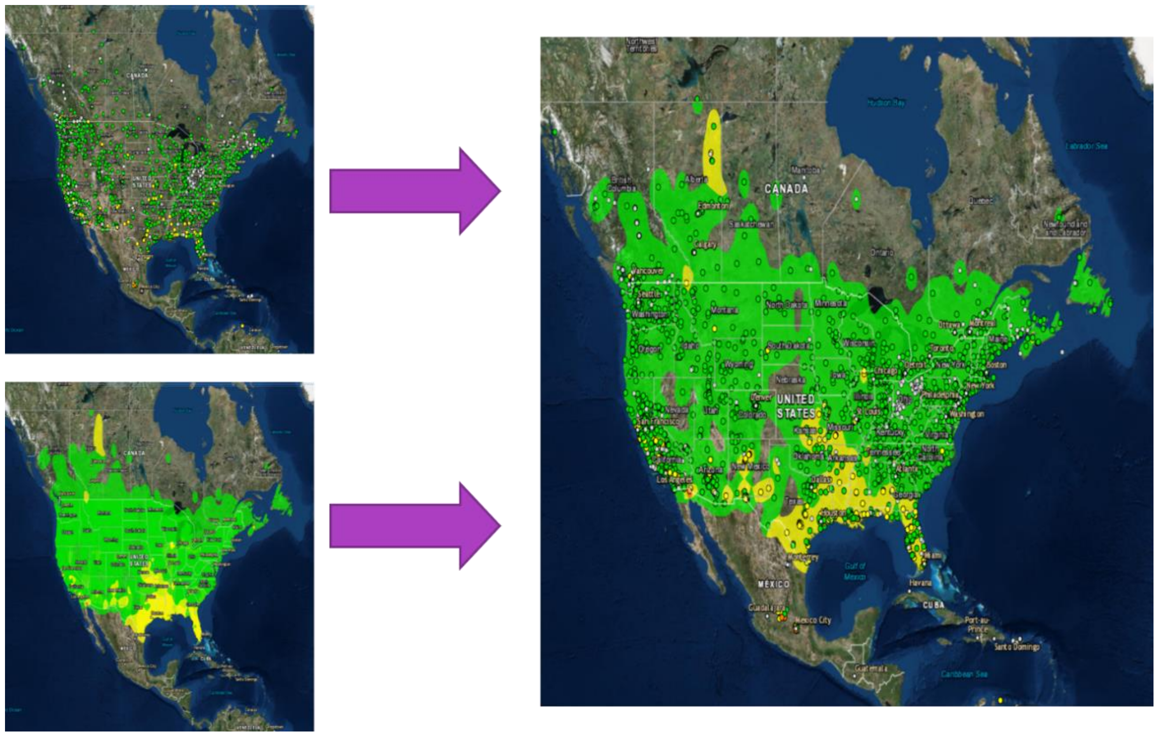

Figure 5-7: Air Quality Sensor Locations (upper left), Surface (lower left), and Overlay (right)

pulling many different layers and attempting to organize them would be detrimental to both the visitor's comprehension and the Hyperwall's ability to hold their interest. Figure 5-7 shows the two layers selected for use in the Air Quality section of the application.

For the Museum's visitors to learn about what air quality indexes represent, the Air Quality Map pulled two feature layers from the AirNow Web Map via ArcGIS Feature Service REST endpoints (Figure 5-8). The first is a vector point collection that represents sensors in major U.S. cities. Each point is marked with the current Air Quality Index for that city. The second layer is an interpolated vector geometry surface with a nominative scale ranging from "Good" to "Hazardous". The symbology of the two layers follows the same color ramp to ease visitor comprehension. The Air Quality Map uses the two layers to illustrate the difference between data points and created surfaces. Also, museum visitors can explore the map to find current air quality indices in areas that personally interest them.

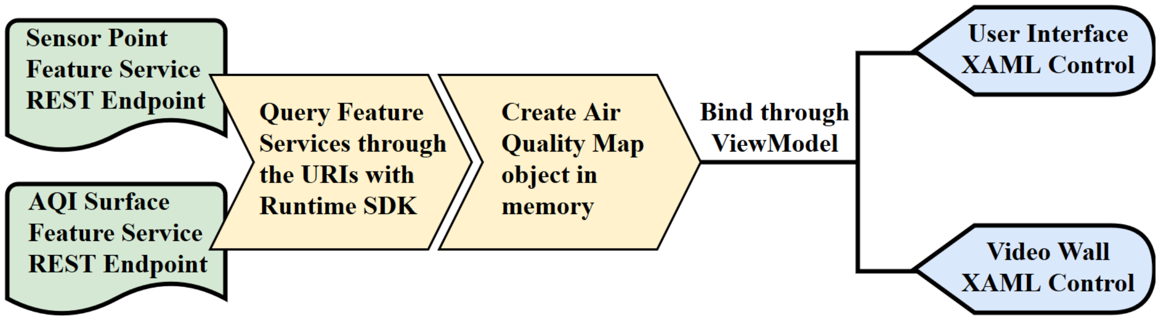
Sensor Point Feature Service REST Endpoint
AQI Surface Feature Service REST Endpoint
Query Feature Services through the URIs with Runtime SDK
Create Air Quality Map object in memory
Bind through ViewModel
User Interface XAML Control
Video Wall XAML Control

Figure 5-8: Air Quality Map Model

5.3.3 Incorporating the Natural Disaster Imagery and Model

The intent for the Natural Disaster section of the application is to allow visitors to explore the Earth's changing surface through the imagery of natural disasters and other events including volcanic eruptions, country-sized power outages, major floods and more. The client wanted the imagery placed onto an interactive map for this section of the

Hyperwall Application. Visitors using the application would compare paired imagery from before and after large scale disasters or events. The map was designed based on this need.

The application stored the imagery for the Natural Disaster dataset in the working/local directory. Two folders, appropriately called "BeforeImagery" and "AfterImagery", contained the imagery. When the application launches, the Natural Disaster Model creates two Scenes containing only basemaps from Esri’s online services. These two Scenes are the virtual globes for displaying the imagery. After creating the Scenes, the model loads the imagery into the application from the folders.

The client planned on expanding the available imagery in the future, so the method for loading the imagery from the folders into the Scenes needed to be flexible to minimize the amount of work required for this process. To add images to the Natural Disaster section, the client places a georeferenced TIFF image or another raster into the appropriate folder and the application manages the process of building the raw imagery into a format applicable to an ArcScene.

The Natural Disaster Model loads the georeferenced TIFF files by first querying the relative file directory path to each folder. Then the model iterates through each item in the folders. For each image the model encounters, it loads the image as an instance of the ArcGIS Runtime SDK Raster Class. Then the model creates a Raster object in the memory and applies it to an ArcScene displayed by the application. The Model identifies the geographic location and extent for each image and places a bookmark at that location that has a scale of 1.5 times the images extent (to show more of the surrounding area). The model finally names each bookmark according to the file name for the image after the file extension is removed. Figure 5-9 demonstrates the workflow for accessing the imagery from storage, adding it to the application's Scenes, and creating bookmarks.

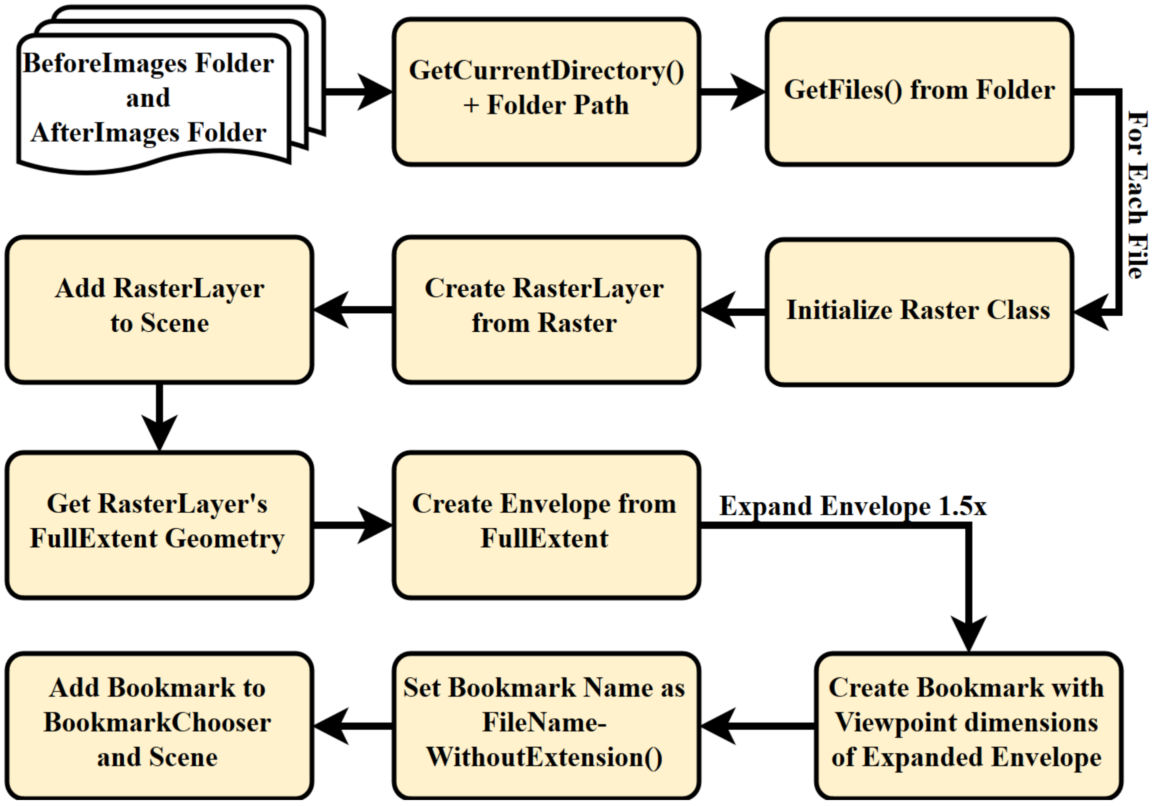
BeforeImages Folder and AfterImages Folder
GetCurrentDirectory() + Folder Path
GetFiles() from Folder
For Each File
Initialize Raster Class
Create RasterLayer from Raster
Add RasterLayer to Scene
Get RasterLayer's FullExtent Geometry
Create Envelope from FullExtent
Expand Envelope 1.5x
Create Bookmark with Viewpoint dimensions of Expanded Envelope
Set Bookmark Name as FileName-WithoutExtension()
Add Bookmark to BookmarkChooser and Scene

Figure 5-9: Natural Disaster Scene Model

5.4 Creating Hyperwall Application Interaction Logic

At this point in the project implementation, the data models built for accessing the Heat Vulnerability Mobile Map Package, Air Quality Feature Services, and Natural Disaster .TIFF imagery was completed. Each model produced a Map or Scene object that displays the data on top of a basemap generated through Esri's Online Services. In the next part of the project implementation, the ViewModels and Views needed defining. The ViewModels contained the logic for meshing the Map/Scene objects with the system architecture. The Views contained the mark-up definitions for the GUI appearance and implemented the data binding of specific visual elements to the relevant event handlers and methods in the ViewModels.

WPF has three levels of ‘wrappers’ for hosting visual elements of applications: Windows, Pages, and UserControls. The Hyperwall Application used Windows and UserControls. Both a Window and a UserControl contain a XAML-based View with information like the color, size, shape, animation or other visual aspects. They also contain a class document similar in structure to a ViewModel, but specifically contains logic dictating the behavior of user-facing controls and ensures that the correct messages are sent based on user input.

The User Interface and Video Wall are WPF Windows, each with their own View and ViewModel. The Heat Vulnerability Map, Air Quality Map, and Natural Disaster Scene are WPF UserControls, each with corresponding Views and ViewModels. The User Interface’s View contained a XAML control called a ContentControl, used for hosting a UserControl. The Video Wall also hosted a ContentControl in its View. To ensure that both the User Interface and Video Wall were displaying the same UserControl at any given time, the User Interface’s ViewModel implemented the system’s event handling and messaging (see section 5.2) to create instances of the UserControls and notify the Video Wall of which UserControl was currently active in the GUI and what it was doing. The User Interface and Video Wall each host one instance of the active UserControl in their respective Views. The system architecture syncs these two instances based on user input.

5.5 Summary

The implementation of this project consisted of designing a hybrid GIS for managing web and locally stored data, processing the data into formats compatible with the ArcGIS Runtime SDK for .NET, and using the Runtime SDK to incorporate them into a .NET

Framework application. The application went through many cycles of planning, developing, and testing of system elements/features. The system architecture went through many iterations and changes to determine the best pattern for managing the Hyperwall's proposed hardware. The final application implemented a system that meets the client's basic hardware for the proposed exhibit. The system generates separate map objects that interact via a series of event handlers, which in turn notify different properties of changes as they occur. The next chapter explores the results of the development process, framed around the user experience.

Chapter 6 – Results and Analysis

The project successfully produced a GIS-based application that demonstrated the potential usefulness of the proposed EarthLab Data Hyperwall to the Science Museum of Virginia (SMV). The application incorporates data prepared by ArcGIS Online and ArcGIS Pro, which are platforms the client can access through its ArcGIS Enterprise account. This is beneficial because the system needs some additional development and feature implementation before deployment as part of the Hyperwall.

6.1 User Experience

The Hyperwall Application's system architecture was designed to satisfy the client's hardware requirements, but the layout of the GUI and the application's methods were designed based on the desired visitor experience. Although the Museum's target demographic is school-aged children, it attracts visitors of all ages. To accommodate the varied audience, the Hyperwall Exhibit needed as broad an appeal as possible. To this end, the Hyperwall Application was designed to maximize accessibility and comprehension.

The layout for the GUI implemented straightforward controls and contained maps with clear intent and symbology. This project's limited scope precluded structured testing of the application through museum visitors or similar focus groups. However, the principles of accessibility of material and simplicity of interaction drove the development process. The following paragraphs contain a discussion of the user experience framed like a use-case scenario or user story. Figure 6-1 shows a conceptual design for the

Hyperwall, which incorporates the current layout of the application and how the exhibit may appear to the operating visitor.

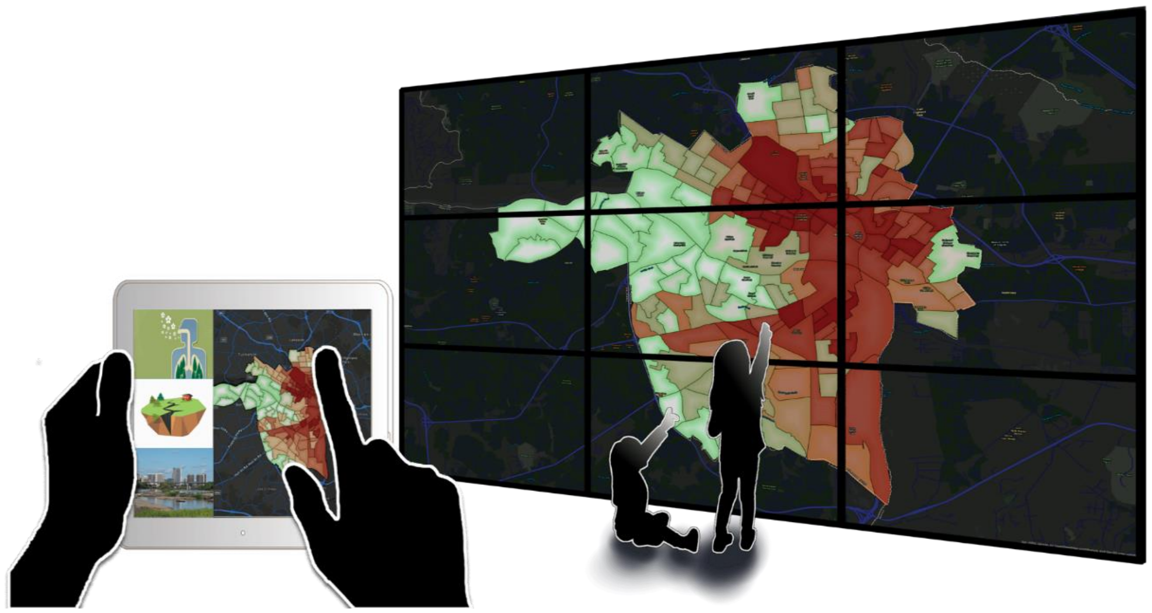

Figure 6-1: Hyperwall Exhibit from Visitors Perspective

The visitor operating the Hyperwall's touchscreen kiosk first sees the starting page of the GUI, which contains just three buttons. Each button is an image representing the subject matter of the map or Scene. The goal in this case, and throughout the application is to reduce reliance on text-based directions.

After pressing one of the buttons, a map opens on the Video Wall and a smaller version of the same map opens on the GUI. Visitors control the map with the same touch controls they would use for a smartphone navigation app. This assumes that many of the visitors are familiar with operating a smartphone. The three environmental topics contained in the application contain drastically different datasets, but similar modes of interaction. Museum visitors have the option of exploring the Heat Vulnerability Map, Air Quality Map, or Natural Disaster Scene.

6.1.1 Heat Vulnerability Map Interaction

Visitors compare layers on the Heat Vulnerability Map by turning the five available layers on and off. The visitor operating the exhibit can zoom in to specific areas of Richmond and visually compare the layers. Through comparison of the Heat Vulnerability Index to the three layers contributing to it (Daily Heating, Impervious Surfaces, and Tree Canopy) visitors develop an understanding of why parts of Richmond have a higher vulnerability to heat and heat-related illness than other areas. As many of the visitors are local to Richmond, perhaps this public health dataset will help them in their daily life navigating the city.

6.1.2 Air Quality Map Interaction

After selecting the Air Quality button in the GUI, visitors see a map of the contiguous United States that loads the two live air quality layers. Visitors can turn the layers on and off independently using buttons on the GUI and use touch controls to explore the map and see current AQI readings for the United States.

6.1.3 Natural Disaster Scene Interaction

In the Natural Disaster Scene, visitors see an ArcScene, or virtual globe, which has high-resolution imagery from NASA's Earth Observatory draped over its surface in spatially accurate positions. Visitors can explore the globe using touch controls or use a drop-down menu to select specific images. Upon selection, the globe will automatically go to the bookmarked image. Visitors use the custom swipe widget to compare the 'before' and 'after' of each pictured event. Through this, they can instantly see the effects of glacial melt, volcanic eruptions, flooding, and other examples of the Earth's ever-changing surface.

6.2 Application Features

Programmatic methods and events define the implemented features and connect feature behavior to the system architecture. When a visitor interacts with a feature on the GUI, a method or event activates and sends change notifications throughout the application to ensure that the video wall reflects the change. Project time constraints restricted the number of implemented features, but the following is a discussion of the currently implemented features exposed in the GUI.

6.2.1 Touch Control for Maps

Visitors interact with the Hyperwall via touch controls like smartphone-based mapping applications. When the visitor using the exhibit selects one of the three subjects, the corresponding map, and its controls load onto the screen. In addition to touch-responsive buttons exposed on the GUI, standard gesture map controls are applied. Visitors control the video wall portion of the exhibit by manipulating the map visible on the GUI with their fingers. Any changes made to the GUI's view are immediately replicated on the video wall. Visitors can pan, rotate, zoom in, and zoom out on the map using dragging, pinching, and rotational movements with one or two fingers. Additionally, when exploring the Natural Disaster subject, visitors can adjust the pitch of the camera to look across the surface of the virtual globe by dragging two fingers up or down the face of the tablet. Table 5 provides instructions for navigating the application's maps.

Table 5. Map Navigation in Hyperwall Application

Gesture/Hand Motion	Map Response	Description	
One-Finger Drag	Pan	Drag one finger across the map to move it in that direction.	
Pinch	Zoom In/Out	Place two fingers on the map and spread them apart or bring them together to change the map scale.	
Two-Finger Drag	Pitch	Place two fingers on the map and move them up or down to change the angle of the camera's pitch. Note: This only works on the 3-D Natural Disaster Scene.	
Two-Finger Rotation	Rotate	Place two fingers on the map and rotate them around a central point to change the orientation of the map.	

6.2.2 Custom Swipe Widget

The two Natural Disaster Scenes each display within an ArcGIS Runtime SDK SceneView. A SceneView is like a WPF ContentControl, but specifically for hosting Maps and Scenes in the GUI. The two SceneViews overlap on the GUI with the Before Scene on top of the After Scene. Museum visitors control the visibility of the After Scene through a custom swipe tool that peels back the Before Scene to reveal the After Scene. Figure 6-2 illustrates how this feature works.

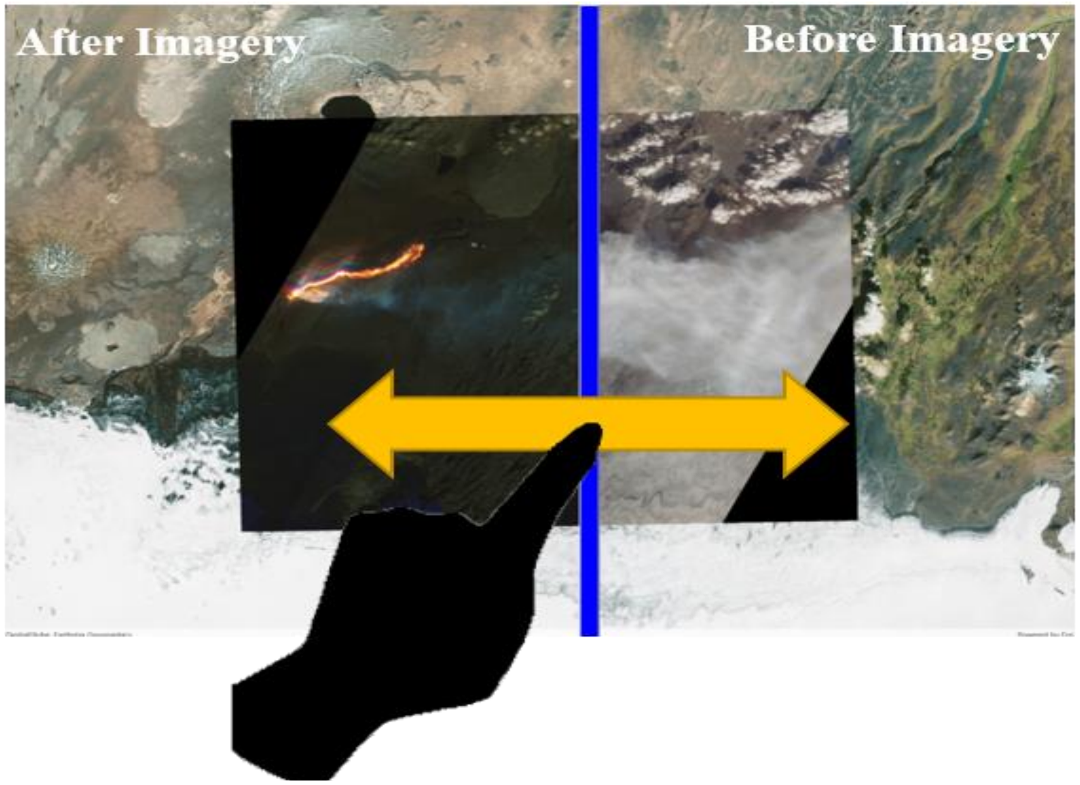

Figure 6-2: Swipe Widget Operation

6.2.3 Imagery Selector

The Natural Disaster View/ViewModel incorporated a drop-down menu allowing visitors to select from bookmarked image locations as an alternative to exploring the imagery with touch controls. At startup, The ViewModel executes a method for populating the View's drop-down menu. This entire workflow executes within the loop that iterates through the imagery folder and loads each image. After loading, the application converts the image into an ArcGIS Runtime Raster, adds it to either the Before Scene or After Scene, detects the raster's geometry, creates a bookmark 1.5 times the extent, assigns a name to the bookmark and adds the name to the BookmarksChooser drop-down menu on the View.

6.3 Data Complications

The Natural Disaster and Air Quality datasets were developed during the project. Originally, a File Geodatabase would house all three datasets. Each subject's dataset as locally stored feature layers and imagery directly from the File Geodatabase. After some testing and discussion with the ArcGIS Runtime SDK for .NET development team, it was discovered that the Runtime SDK is not compatible with file Geodatabases. The Runtime SDK is compatible with Mobile Geodatabases. However, the Mobile Geodatabase is an inconvenient and inflexible structure for data storage and access.

The Hyperwall Application loads each dataset from either the project's File Geodatabase or through ArcGIS Online. The Heat Vulnerability map layers export from ArcGIS Pro as a Mobile Map Package (.mmpk). After exporting, the .mmpk went inside the application's directory for accessibility during runtime. However, this workflow could not be replicated for the Air Quality data. Although the ArcGIS Online feature service containing the two layers could be brought into an ArcGIS Pro map project and referenced in the project's Geodatabase, the map layers did not stay updated with the corresponding ArcGIS Online Feature Services. To ensure data integrity for the Air Quality section of the application, the feature layers were instead accessed directly by the application through their REST Endpoints. Although this potentially creates extra steps for the client if they wish to add to or modify one of the two datasets, the chosen map product formats meet project needs better. Also, using a Mobile Map Package and Feature Service URIs ensures compatibility with the ArcGIS Runtime SDK.

6.4 Discussion of Results and Client’s Needs

This project provided the client with a system that supports further development and deployment of the EarthLab Data Hyperwall Exhibit. The system consists of a GIS and .NET Framework application. The system incorporated Esri map products including satellite imagery, online feature data services, and ArcGIS Pro Mobile Map Packages. The .NET Framework application's interaction and presentation logic were decoupled through the implementation of the Model-View-ViewModel pattern. The MVVM pattern simplifies the process of maintaining and expanding the application. If the client wanted to change the layout or design of the GUI, they could modify the XAML directly through Blend for Visual Studio without changing the application's code.

6.5 Summary

The goal of the Hyperwall Exhibit is to complement the currently existing spatially linked exhibits through providing an interactive platform for museum visitors interested in learning about both local and global environmental topics potentially affecting them personally. The .NET Framework application and GIS produced by this project fill the Science Museum of Virginia’s need for a platform that presents local data and encourages visitors to engage with environmental topics.

Chapter 7 – Conclusions and Future Work

This project set out to provide the Science Museum of Virginia with a system for managing the presentation of the data for their planned EarthLab Data Hyperwall exhibit. The goal for the exhibit is to teach the Museum's visitors about environmental factors affecting their everyday lives and the Earth through interactive 2-d and 3-d maps. The system brings both web-based and local spatial data into a custom .NET Framework application, which creates the maps and handles visitor inputs.

The exhibit's hardware consists of a multi-screen video wall and a touchscreen user interface. Museum visitors use the touchscreen's GUI to navigate the exhibit's three maps, which sync to the video wall. The maps derive from three different datasets containing live web-based air quality data, museum research into heat vulnerability for census block groups in the city of Richmond, and georeferenced imagery. The .NET Framework application implements Esri's ArcGIS Runtime SDK for .NET to access the data from REST endpoints and a File Geodatabase on the exhibit controlling personal computer. In addition to the system, the project produced a written guide containing workflows for adding imagery to the Natural Disaster Scene, changing map data and map visualization.

7.1 Summary

The system is a novel approach to help the Science Museum of Virginia share educational environmental and climatic data with its visitors by utilizing GIS visualization. Additionally, it supports the creation of additional maps/scenes, and implementation of additional or updated data through some pre-planned workflows requiring little, if any, modification of the Hyperwall Application's source code.

By implementing a system that accepts web-based and offline data, the Science Museum of Virginia has maximum flexibility for improving this exhibit over time and ensuring that visitors immerse themselves in the interactive spatial learning experience. Through this system, the Museum can successfully implement the EarthLab Data Hyperwall exhibit to augment its other educational exhibits and continue the Museum's mission of "inspiring Virginians to enrich their lives through Science."

7.2 Future Work

The current system architecture supports visitor learning through interaction and encourages the Museum's visitors to contextualize environmental data in their own lives and allows visitors to improve their map comprehension and spatial literacy. There are additional features that could be implemented to improve the visitor experience and the exhibit's functionality.

The maps in the application require accompanying text or other multimedia. Before system deployment, it needs this feature so visitors could understand the data presented in the exhibit. Different dialogs would appear on the video wall depending on the map, layers, or imagery currently displayed.

Implementing a sub-system for collecting usage telemetry could greatly augment the Museum's ability to improve the Hyperwall exhibit over time. There are two ways to collect user data for analysis: generating a heat map showing where on the user interface visitors are touching, or outputting the application's journal to a table.

The Natural Disaster Scene included a small set of imagery demonstrating how the Scene operated within the system. Additional imagery could be added for visitors to enjoy and explore. The new imagery could be downloaded from NASA’s Earth Observatory or another data source. Any georeferenced raster file formats compatible with the Raster class in ArcGIS Runtime SDK for .NET could be used.

Migrating the locally stored datasets to the client’s Enterprise Portal would allow the client to manage the data without needing to change the code. The application can already open data from REST endpoints, so if all the data were exposed in that format, the client could make desired changes to the maps or data through their Enterprise deployment and maintain the same REST endpoints. This would simplify the workflows for maintaining and updating the exhibit as a whole and provide more flexibility for the application’s data and presentation.

Works Cited

Alburshaid, Y. (2012). *A Multi-touch GIS-Based Tour for Museum Exhibits.* Redlands: InSPIRE@Redlands. Retrieved from https://inspire.redlands.edu/gis_gradproj/16

Ang, K. H., & Wang, Q. (2006). A Case Study of Engaging Primary School Students in Learning Science by Using Active Worlds. *Proceedings of The First International LAMS Conference 2006: Designing the Future of Learning* (pp. 5-14). Sydney: LAMS Foundation. Retrieved April 10, 2019, from http://lamsfoundation.org/lams2006/papers.htm

Bartoschek, T., Li, R., Schwering, A., & Munzer, S. (2013). Ori-Gami - An App fostering spatial competence development and spatial learning of children. *16th AGILE Conference on Geographic Information Science.* Castellon: ResearchGate. Retrieved October 17, 2018, from https://www.researchgate.net/publication/236341079_Ori-Gami_-_An_App_fostering_spatial_competency_development_and_spatial_learning_of_children

Bednarz, S. W., & Kemp, K. (2011). Understanding and Nurturing Spatial Literacy. In Y. Asami (Ed.), *International Conference: Spatial Thinking and Geographic Information Sciences* (pp. 18-23). Tokyo: Elsevier Ltd. Retrieved October 17, 2018, from https://www.sciencedirect.com/science/article/pii/S1877042811013279

Bruner, J. S. (1960). *The Process of Education.* New York: Harvard.

Downs, R. M. (1994). The Need for Research in Geography Education: It Would be Nice to Have Some Data. *Journal of Geography, 93*(1), 57-60.

Downs, R. M., & de Souza, A. R. (2006). *Learning to Think Spatially*. Washington D.C.: National Academies Press. doi:10.17226/11019

Egenhofer, M. J., & Mark, D. M. (1995). *Naive Geography*. National Center for Geographic Information and Analysis. UC Santa Barbara. Retrieved from https://escholarship.org/uc/item/3r80v86f

Goodchild, M. F. (2006, Fall). The Fourth R? Rethinking GIS Education. *ArcNews Online*. Retrieved October 27, 2018, from https://www.esri.com/news/arcnews/fall06articles/the-fourth-r.html

Goodchild, M. F. (2011). Spatial Thinking and the GIS User Interface. In Y. Asami (Ed.), *International Conference: Spatial Thinking and Geographic Information Sciences* (pp. 3-9). Tokyo: Elsevier Ltd. Retrieved October 17, 2018, from https://www.sciencedirect.com/science/article/pii/S1877042811013255

Hakeem, H. (2017, September 7). Android by example: MVVM + Data Binding -> Introduction (Part 1). San Francisco, California, United States of America. Retrieved from https://medium.com/@husayn.hakeem/android-by-example-mvvm-data-binding-introduction-part-1-6a7a5f388bf7

Hiperwall Headquarters. (2018, March 15). *Hiperwall: See the Big Picture*. Retrieved April 25, 2018, from Hiperwall: https://www.hiperwall.com/

Hirose, M. (2006). Virtual Reality Technology and Museum Exhibit. *The International Journal of Virtual Reality, 5*(2), 31-36.

Lepouras, G., & Vassilakis, C. (2004, December 15). Virtual museums for all: employing game technology. *Virtual Reality, 8*, 96-106. doi:10.1007/s10055-004-0141-1

Levy Robert, P. P. (2019, June). *Image Use Policy*. Retrieved June 20, 2019, from Nasa Earth Observatory: https://earthobservatory.nasa.gov/image-use-policy

Liarokapis, F., Sylaiou, S., Basu, A., Mourkoussis, N., White, M., & Lister, P. (2004). An Interactive Visualization Interface for Virtual Museums. *VAST 2004: The 5th International Symposium on Virtual Reality* (pp. 1-10). Oudenaarde: The Eurographics Association. doi:10.2312/VAST/VAST04/047-056

Marsh, M., Golledge, R., & Battersby, S. E. (2007, December). Geospatial Concept Understanding and Recognition in G6-College Students: A Preliminary Argument for Minimal GIS. *Annals of the Association of American Geographers, 97*(4), pp. 696-712. Retrieved 10 30, 2018, from https://www.jstor.org/stable/4620307

Martin, J. (1991). *Rapid Application Development.* Indianapolis, IN, USA: Macmillian Publishing Co.

Michael, D., Pelekanos, N., Chrysanthou, I., Zaharias, P., Hadjigavriel, L., & Chrysanthou, Y. (2010). *Comparative Study of Interactive Systems in a Museum.* Cyprus: Springer-Verlag Berlin Heidelberg.

Nathan, A. (2013). *WPF 4.5 - Unleashed.* Indianapolis, Indiana: SAMS. Retrieved January 20, 2019, from https://dl.acm.org/citation.cfm?id=2564751

Piaget, J. (1936). *Origins of intelligence in the child.* London: Routledge & Kegan Paul.

Sandstrom, T. A., Henze, C., & Levit, C. (2003). *The Hyperwall.* NASA, Ames Research Center. Exploratory Computing Environments Group. Retrieved October 17, 2018, from https://www.researchgate.net/publication/4024207_The_hyperwall

SCN Staff. (2013, December). Look Around You. *AV Network - Systems Contractor News*, pp. 64-65. Retrieved May 28, 2019, from https://www.avnetwork.com/avnetwork/eiki-international-promotes-steve-rubery-to-national-sales-manager

Sellers, P. (2011). *NASA's Hyperwall Revealing the Big Picture.* NASA Goddard Space Flight Center, Greenbelt, MD. Retrieved from https://ntrs.nasa.gov/search.jsp?R=20120002556

Smith, J. (2019, May 10). *AirNow Interactive Map of Air Quality, US EPA, OAR, OAQPS.* Retrieved June 20, 2019, from ArcGIS Online: http://univredlands.maps.arcgis.com/home/item.html?id=92e772c4f65a4848a29bcc24c8f61bab

Sørensen, E., & Mihailesc, M. I. (2010). *Model-View-ViewModel (MVVM) Design Pattern using Windows.* University of Southern Denmark, Computer Engineering, Odense. Retrieved May 5, 2019, from https://www.researchgate.net/publication/283571344_Model-View-ViewModel_MVVM_Design_Pattern_using_Windows_Presentation_Foundation_WPF_Technology

Sylaiou, S., Mania, K., Liarokapis, F., White, M., Walczak, K., Wojciechowsk, R., . . . Patias, P. (2015, November). Evaluation of a Cultural Heritage Augmented Reality Game. In *Cartographies of Mind, Soul, and Knowledge* (pp. 153-185). Thessaloniki: School of Rural and Surveying Engineers. doi:978-960-89320-7-4

Tomlinson, R. (2007). *Thinking About GIS: Geographic Information System Planning for Managers.* Redlands: Esri Press.

Appendix A. Hyperwall Application Structure

This appendix contains the class diagrams and code for each part of the Hyperwall Application. The class diagrams are in the first section, and each subsequent section contains the code for the Models and ViewModels, respectfully. The Views have been omitted because they are merely mark-up documents and do not contribute the application's functionality, only its appearance. It is worth noting here that for methods that have functionally identical logic, only one of the methods is shown completely, and the others have been collapsed to save space.

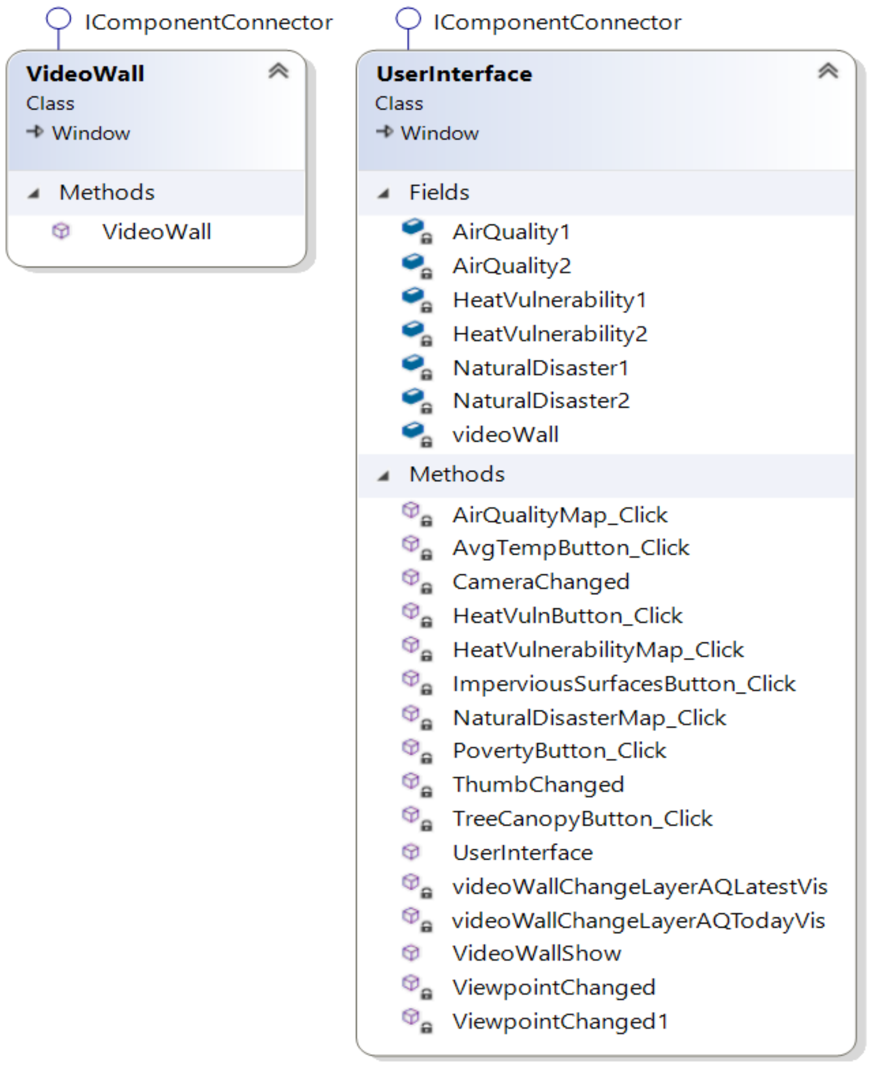

Figure A-1: WPF Windows for Exhibit LCD Screens

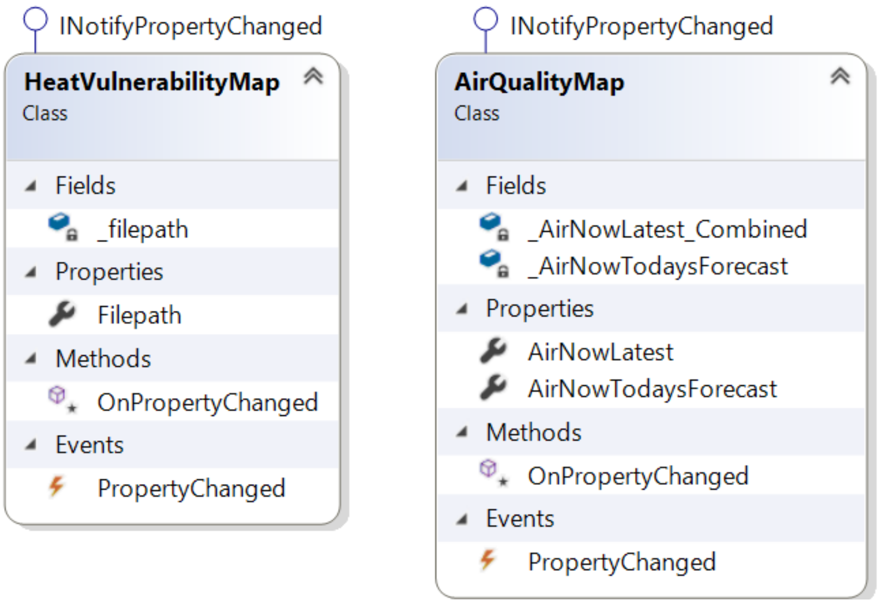

Figure A-2: Map Models

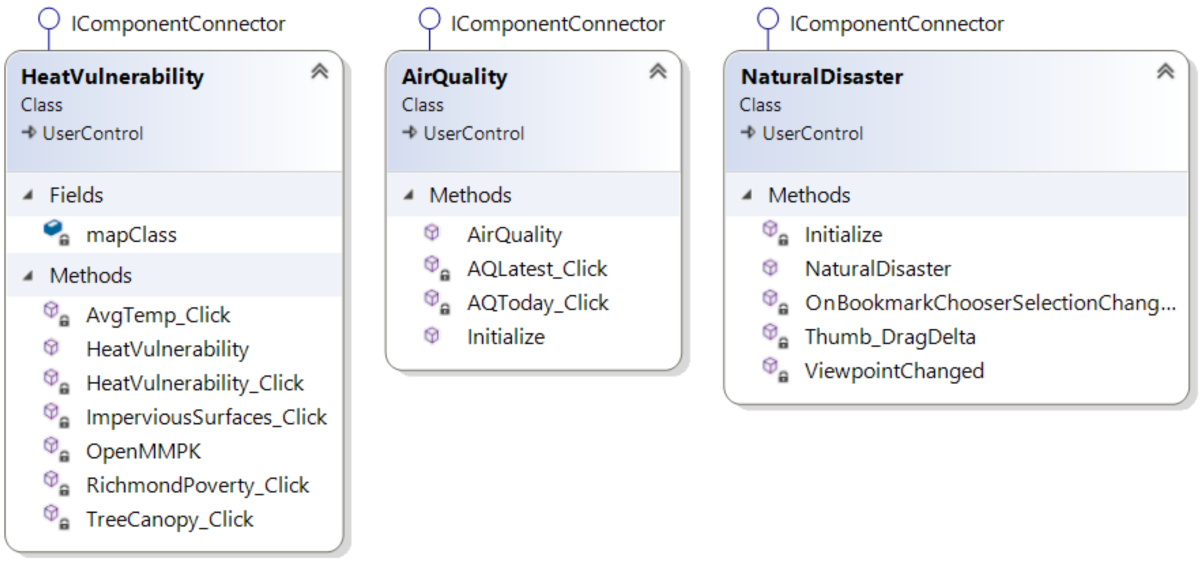

Figure A-3: Map ViewModels

NOTE: There is no separate Map Model for the Natural Disaster topic. This is because the accessing and loading of the imagery dataset is built into the NaturalDisaster ViewModel.

UserInterface Class

```
using Esri.ArcGISRuntime.Geometry;
using Esri.ArcGISRuntime.Mapping;
using System;
using System.Diagnostics;
using System.Linq;
using System.Windows;
using System.Windows.Controls.Primitives;
using System.Windows.Forms;
using System.Windows.Media;

namespace Hyperwall3
{
    /// <summary>
    /// The UserInterface acts as the controller for the entire application, it handles all object
    /// creation and event handling between the exhibit's touchscreen and video wall
    /// Hyperwall Application Created By: Alex Walton
    /// Affiliation: University of Redlands, Department of Geographic Information Sciences
    /// Last Updated: July 25, 2019
    /// </summary>
    public partial class UserInterface : Window
    {
        // Object creation for UserInterface Window (MapName1) And the VideoWall Window (MapName2)
        NaturalDisaster NaturalDisaster1 = new NaturalDisaster();
        NaturalDisaster NaturalDisaster2 = new NaturalDisaster();
        AirQuality AirQuality1 = new AirQuality();
        AirQuality AirQuality2 = new AirQuality();
        HeatVulnerability HeatVulnerability1 = new HeatVulnerability();
        HeatVulnerability HeatVulnerability2 = new HeatVulnerability();
        VideoWall videoWall = new VideoWall();

        /// <summary>
        /// Constructor initializes ViewModel creation and handles viewpoint changes from user.
        /// </summary>
        public UserInterface()
        {
            InitializeComponent();
            // Fires the ViewPointChanged events whenever UserInterface ViewPoint Changes
            if (AirQuality1.AirMap != null)
            {
                AirQuality1.AirMap.ViewpointChanged += ViewpointChanged;
            }
            if (HeatVulnerability1.HeatVuln != null)
            {
                HeatVulnerability1.HeatVuln.ViewpointChanged += ViewpointChanged1;
            }
            if (NaturalDisaster1.NaturalDisasterBefore.Camera != null)
            {
                NaturalDisaster1.NaturalDisasterBefore.ViewpointChanged += CameraChanged;
                NaturalDisaster1.thumb.DragDelta += ThumbChanged;
            }

            // Syncs Click Events between Windows
            AirQuality1.AQLatest.Click += videoWallChangeLayerAQLatestVis;
            AirQuality1.AQToday.Click += videoWallChangeLayerAQTodayVis;
            HeatVulnerability1.AvgTempButton.Click += AvgTempButton_Click;
            HeatVulnerability1.HeatVulnButton.Click += HeatVulnButton_Click;
            HeatVulnerability1.ImperviousSurfacesButton.Click += ImperviousSurfacesButton_Click;
            HeatVulnerability1.PovertyButton.Click += PovertyButton_Click;
            HeatVulnerability1.TreeCanopyButton.Click += TreeCanopyButton_Click;
            // Method for creating video wall display on secondary monitor
            // ADD OR REMOVE "//" TO FRONT OF "VideoWallShow()" BELOW TO TURN VIDEOWALL ON/OFF
            //VideoWallShow();
        }

        /// <summary>
        /// Button_Click events manage the turning on/off layers, according to user input.
        /// </summary>
        private void TreeCanopyButton_Click(object sender, RoutedEventArgs e)
        {
            if (HeatVulnerability1.HeatVuln.Map.OperationalLayers[2].IsVisible == false)
            {
                HeatVulnerability2.HeatVuln.Map.OperationalLayers[2].IsVisible = false;
            }
            else
            {
                HeatVulnerability2.HeatVuln.Map.OperationalLayers[0].IsVisible = false;
                HeatVulnerability2.HeatVuln.Map.OperationalLayers[1].IsVisible = false;
                HeatVulnerability2.HeatVuln.Map.OperationalLayers[2].IsVisible = true;
                HeatVulnerability2.HeatVuln.Map.OperationalLayers[3].IsVisible = false;
                HeatVulnerability2.HeatVuln.Map.OperationalLayers[4].IsVisible = false;
            }
        }
        private void PovertyButton_Click(object sender, RoutedEventArgs e)
        {
          …
        }
        private void ImperviousSurfacesButton_Click(object sender, RoutedEventArgs e)
        {
          …
        }
        private void HeatVulnButton_Click(object sender, RoutedEventArgs e)
        {
          …
        }
        private void AvgTempButton_Click(object sender, RoutedEventArgs e)
        {
          …
        }
        private void videoWallChangeLayerAQLatestVis(object sender, RoutedEventArgs e)
        {
            if (AirQuality1.AirMap.Map.OperationalLayers[0].IsVisible == false)
            {
                AirQuality2.AirMap.Map.OperationalLayers[0].IsVisible = false;
            }
            else
            {
                AirQuality2.AirMap.Map.OperationalLayers[0].IsVisible = true;
            }
        }
        private void videoWallChangeLayerAQTodayVis(object sender, RoutedEventArgs e)
        {
          …
        }

        /// <summary>
        /// VideoWallShow() method detects computer display settings and applies windows to
        /// correct screens
        /// </summary>
        public void VideoWallShow()
        {
            try
            {
                var primaryScreen = System.Windows.Forms.Screen.PrimaryScreen;
                var secondaryScreen = Screen.AllScreens.First(screen => screen != primaryScreen);
                videoWall.Left = secondaryScreen.Bounds.Left;
                videoWall.Top = secondaryScreen.Bounds.Top;
                videoWall.Width = secondaryScreen.Bounds.Width;
                videoWall.Height = secondaryScreen.Bounds.Height;
                videoWall.WindowState = WindowState.Normal;
                videoWall.Loaded += (_s, _e) => videoWall.WindowState = WindowState.Maximized;
                videoWall.Show();
            }
            catch (Exception ex)
            {
```

```
                Debug.WriteLine(ex.Message);
            }
        }

        // Opens the Air Quality Map and sends it to the video wall mapview
        private void AirQualityMap_Click(object sender, RoutedEventArgs e)
        {
            ActiveMap.Content = AirQuality1;
            videoWall.VideoWallDisplay.Content = AirQuality2;
            AirQuality2.AQLatest.Visibility = Visibility.Collapsed;
            AirQuality2.AQToday.Visibility = Visibility.Collapsed;
        }

        // Opens the Heat Vulnerability mmpk and gives it a basemap
        private void HeatVulnerabilityMap_Click(object sender, RoutedEventArgs e)
        {
            ActiveMap.Content = HeatVulnerability1;
            videoWall.VideoWallDisplay.Content = HeatVulnerability2;
            HeatVulnerability2.HeatVulnButton.Visibility = Visibility.Collapsed;
            HeatVulnerability2.ImperviousSurfacesButton.Visibility = Visibility.Collapsed;
            HeatVulnerability2.AvgTempButton.Visibility = Visibility.Collapsed;
            HeatVulnerability2.PovertyButton.Visibility = Visibility.Collapsed;
            HeatVulnerability2.TreeCanopyButton.Visibility = Visibility.Collapsed;

        }

        // Opens the Natural Disaster Scenes
        private void NaturalDisasterMap_Click(object sender, RoutedEventArgs e)
        {
            ActiveMap.Content = NaturalDisaster1;
            videoWall.VideoWallDisplay.Content = NaturalDisaster2;
            NaturalDisaster2.BookmarkChooser.Visibility = Visibility.Collapsed;
        }

        // syncs the UI's view and the Video Wall's viewpoints as needed
        private void ViewpointChanged(object sender, EventArgs e)
        {
            try
            {
                Viewpoint UIMapViewpoint = AirQuality1.AirMap.GetCurrentViewpoint(ViewpointType.Bo
undingGeometry);
                var UIMapViewGeometry = UIMapViewpoint.TargetGeometry.Extent;
                EnvelopeBuilder newEnvelope = new EnvelopeBuilder(UIMapViewGeometry);
                // The video walls map extent is 1.5x the UI's extent
                newEnvelope.Expand(1.5);
                AirQuality2.AirMap.SetViewpoint(new Viewpoint(newEnvelope.Extent));
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex.Message);
            }
        }

        private void ViewpointChanged1(object sender, EventArgs e)
        {
            try
            {
                Viewpoint UIMapViewpoint = HeatVulnerability1.HeatVuln.GetCurrentViewpoint(Viewpoi
ntType.BoundingGeometry);
                var UIMapViewGeometry = UIMapViewpoint.TargetGeometry.Extent;
                EnvelopeBuilder newEnvelope = new EnvelopeBuilder(UIMapViewGeometry);
                // The video walls map extent is 1.5x the UI's extent
                newEnvelope.Expand(1.5);
                HeatVulnerability2.HeatVuln.SetViewpoint(new Viewpoint(newEnvelope.Extent));
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex.Message);
            }
        }

        // syncs the UI's camera and the video wall's camera as needed
        private void CameraChanged(object sender, EventArgs e)
        {
            try
            {
                Camera cam1 = NaturalDisaster1.NaturalDisasterBefore.Camera;
                cam1.Elevate(100000);
                if (cam1 != null)
                {
                    NaturalDisaster2.NaturalDisasterBefore.SetViewpointCamera(cam1);
                }
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex.Message);
            }
        }

        // syncs the thumb drag from NaturalDisaster1 and 2
        private void ThumbChanged(object sender, DragDeltaEventArgs e)
        {
            var transform = (TranslateTransform)NaturalDisaster2.thumb.RenderTransform;
            transform.X = Math.Max(0, Math.Min(transform.X + e.HorizontalChange,
                NaturalDisaster2.ActualWidth - (NaturalDisaster2.thumb.ActualWidth - 8)));
            NaturalDisaster2.NaturalDisasterAfter.Clip = new RectangleGeometry()
            { Rect = new Rect(0, 0, transform.X, NaturalDisaster2.ActualHeight) };
        }
    }
}
```

VideoWall Class

```
using System.Windows;

namespace Hyperwall3
{
    /// <summary>
    /// allows creation of an empty window to host the Video Wall maps
    /// </summary>
    public partial class VideoWall : Window
    {
        public VideoWall()
        {
            InitializeComponent();
        }
    }
}
```

AirQualityMapModel Class

```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using Esri.ArcGISRuntime.Mapping;

namespace Hyperwall3.MapClasses
{
    /// <summary>
    ///  AirQualityMap Class exposes Feature Services from the EPA’s AirNow Web Map
    /// </summary>
    public class AirQualityMap : INotifyPropertyChanged
    {
        // Property for REST endpoint for the "Ozone and PM (PM2.5 and PM10) - Current" Contours
        private Layer _AirNowLatest_Combined = new FeatureLayer(new Uri(
                "https://services.arcgis.com/cJ9YHowT8TU7DUyn/arcgis/rest/services/AirNowLatestCon
toursCombined/FeatureServer/0"));

        // "Ozone and PM (PM2.5 and PM10) - Today's Forecast" Sensor Locations
        private Layer _AirNowTodaysForecast = new FeatureLayer(new Uri(
                "https://services.arcgis.com/cJ9YHowT8TU7DUyn/arcgis/rest/services/Air_Now_Current
_Monitors_Ozone_and_PM/FeatureServer/0"));
        public Layer AirNowTodaysForecast
        {
            get { return _AirNowTodaysForecast; }
            set { _AirNowTodaysForecast = value;
                OnPropertyChanged();}
        }

        protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
        public event PropertyChangedEventHandler PropertyChanged;
    }
}
```

HeatVulnerabilityMapModel Class

```
using System.ComponentModel;
using System.IO;
using System.Runtime.CompilerServices;

namespace Hyperwall3.MapClasses
{
    /// <summary>
    /// Opens the Mobile Map Package (.mmpk) for the Heat Vulnerability Map for Richmond
    /// </summary>
    public class HeatVulnerabilityMap : INotifyPropertyChanged
    {
        private string _filepath = Path.Combine(Directory.GetCurrentDirectory(), "MobileMapPackage
s\\HeatVulnerabilityMap.mmpk");
        public string Filepath
        {
            get { return _filepath; }
            set { _filepath = value;}
        }

        protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
        public event PropertyChangedEventHandler PropertyChanged;
    }
}
```

AirQuality ViewModel Class

```
using Esri.ArcGISRuntime.Geometry;
using Esri.ArcGISRuntime.Mapping;
using System.Windows;
using System.Windows.Controls;

namespace Hyperwall3
{
    /// <summary>
    /// This UserControl accesses the AirQualityMap class
    /// </summary>
    public partial class AirQuality : UserControl
    {
        /// <summary>
        /// Constructor initializes instance of class and creates AirQualityMap Instance/Layers
        /// </summary>
        public AirQuality()
        {
            InitializeComponent();
            Initialize();
        }

        /// <summary>
        /// Method creates instance of AirQualityMap for display and adds map layers
        /// </summary>
        public void Initialize()
        {
            var mapClass = new MapClasses.AirQualityMap();
            var airQualityContour = mapClass.AirNowLatest;
            var airQualityCities = mapClass.AirNowTodaysForecast;
            Map airMap = new Map(Basemap.CreateDarkGrayCanvasVector());
            airMap.OperationalLayers.Add(airQualityContour);
            airMap.OperationalLayers.Add(airQualityCities);
            AirMap.Map = airMap;
            AirMap.Map.InitialViewpoint = new Viewpoint(new Envelope(-134.44, 12.8577894, -
57.1276444, 57.91, new SpatialReference(4326)));
        }

        // Toggles Contours on/off
        private void AQLatest_Click(object sender, RoutedEventArgs e)
        {
            if (AirMap.Map.OperationalLayers[0].IsVisible == false)
            {
                AirMap.Map.OperationalLayers[0].IsVisible = true;
            }
            else
            {
                AirMap.Map.OperationalLayers[0].IsVisible = false;
            }
        }

        // Toggles Sensor Points on/off
        private void AQToday_Click(object sender, RoutedEventArgs e)
        {
            if (AirMap.Map.OperationalLayers[1].IsVisible == false)
            {
                AirMap.Map.OperationalLayers[1].IsVisible = true;
            }
            else
            {
                AirMap.Map.OperationalLayers[1].IsVisible = false;
            }
        }
    }
}
```

HeatVulnerability ViewModel Class

```
using Esri.ArcGISRuntime.Mapping;
using System;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using Hyperwall3.MapClasses;

namespace Hyperwall3
{
    /// <summary>
    /// Creates Heat Vulnerability Map and Button Events
    /// </summary>
    public partial class HeatVulnerability : UserControl
    {
        // Exposes instance of model class
        HeatVulnerabilityMap mapClass = new HeatVulnerabilityMap();

        /// <summary>
        /// Constructor initializes methods for opening the mobile map package and adding layers
        /// </summary>
        public HeatVulnerability()
        {
            InitializeComponent();
            OpenMMPK();
        }

        // Opens Heat Vulnerability Map .mmpk and adds layers to view
        private async void OpenMMPK()
        {
            var filepath = mapClass.Filepath;
            Basemap heatBase = Basemap.CreateImageryWithLabels();

            try
            {
                // Load directly or unpack then load as needed by the map package.
                if (await MobileMapPackage.IsDirectReadSupportedAsync(filepath))
                {
                    // Open the map package.
                    MobileMapPackage heatMap = await MobileMapPackage.OpenAsync(filepath);
                    Map HeatVulnerabilityMap = heatMap.Maps.First();

                    // Check for map in .mmpk and give to corresponding views
                    if (heatMap.Maps.Count > 0)
                    {
                        HeatVuln.Map = HeatVulnerabilityMap;
                        HeatVuln.Map.Basemap = heatBase;
                        HeatVuln.Map.OperationalLayers[0].IsVisible = false; // Avg Temp / Daily H
eating
                        HeatVuln.Map.OperationalLayers[1].IsVisible = false; // Impervous Surfaces
                        HeatVuln.Map.OperationalLayers[2].IsVisible = false; // Tree Canopy
                        HeatVuln.Map.OperationalLayers[3].IsVisible = true; // Heat Vulnerability
                        HeatVuln.Map.OperationalLayers[4].IsVisible = false; // Richmond Poverty
                    }
                }
                else
                {
                    // Create a path for the unpacked package.
                    string unpackedPath = filepath + "unpacked";

                    // Unpack the package.
                    await MobileMapPackage.UnpackAsync(filepath, unpackedPath);

                    // Open the package.
                    MobileMapPackage package = await MobileMapPackage.OpenAsync(unpackedPath);
                    // Load the package.
                    await package.LoadAsync();
                }
            }
            catch (Exception e)
            {
                System.Windows.MessageBox.Show(e.ToString(), "Error");
            }
        }

        // Layer Visibility Click Events
        private void AvgTemp_Click(object sender, RoutedEventArgs e)
        {
            if (HeatVuln.Map.OperationalLayers[0].IsVisible == false)
            {
                HeatVuln.Map.OperationalLayers[0].IsVisible = true;
                HeatVuln.Map.OperationalLayers[1].IsVisible = false;
                HeatVuln.Map.OperationalLayers[2].IsVisible = false;
                HeatVuln.Map.OperationalLayers[3].IsVisible = false;
                HeatVuln.Map.OperationalLayers[4].IsVisible = false;

            }
            else
            {
                HeatVuln.Map.OperationalLayers[0].IsVisible = false;
            }
        }
        private void ImperviousSurfaces_Click(object sender, RoutedEventArgs e)
        {
          …
        }
        private void TreeCanopy_Click(object sender, RoutedEventArgs e)
        {
          …
        }
        private void HeatVulnerability_Click(object sender, RoutedEventArgs e)
        {
          …
        }
        private void RichmondPoverty_Click(object sender, RoutedEventArgs e)
        {
          …
        }
    }
}
```

NaturalDisaster ViewModel Class

```
using Esri.ArcGISRuntime.Geometry;
using Esri.ArcGISRuntime.Mapping;
using Esri.ArcGISRuntime.Rasters;
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Media;

namespace Hyperwall3
{
    /// <summary>
    /// UserControl class for creating/managing Natural Disaster Before/After Scenes
    /// </summary>
    public partial class NaturalDisaster : UserControl
    {

        /// <summary>
        /// Constructor handles creation of scene objects and manages Swipe widget
        /// </summary>
        public NaturalDisaster()
        {
            InitializeComponent();
            Initialize();
            thumb.RenderTransform = new TranslateTransform() { X = 0, Y = 0 };
            NaturalDisasterAfter.Clip = new RectangleGeometry() { Rect = new Rect(0, 0, 0, 0) };
            NaturalDisasterBefore.ViewpointChanged += ViewpointChanged;
            InfoText.Opacity = 0.25;
        }

        // This event handler was originally written by Thad Tilton with the Runtime SDK for .NET
team at Esri and adapted to fit my system
        private void Thumb_DragDelta(object sender, DragDeltaEventArgs e)
        {
            try
            {
                var transform = (TranslateTransform)thumb.RenderTransform;
                transform.X = Math.Max(0, Math.Min(transform.X + e.HorizontalChange, this.ActualWi
dth - thumb.ActualWidth));
                NaturalDisasterAfter.Clip = new RectangleGeometry() { Rect = new Rect(0, 0, transf
orm.X, this.ActualHeight) };
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex.Message);
            }
        }

        // Syncs camera between Before and After Map
        private void ViewpointChanged(object sender, EventArgs e)
        {
            try
            {
                Camera beforeCamera = NaturalDisasterBefore.Camera;
                if (beforeCamera != null)
                    NaturalDisasterAfter.SetViewpointCamera(beforeCamera);
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex.Message);
            }
        }
        // Event handler corresponds to ComboBox on NaturalDisasterView.xaml
        private void OnBookmarkChooserSelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            // Get the selected bookmark and apply the view point to the map
            Bookmark selectedBookmark = (Bookmark)e.AddedItems[0];
            Camera cam1 = NaturalDisasterBefore.Camera;
            NaturalDisasterBefore.SetViewpoint(selectedBookmark.Viewpoint);
        }

        // Initialize creates the Map objects, assigns rasters, and creates bookmarks
        private async void Initialize()
        {
            // add imagery basemap to both scenes
            Scene BeforeMap = new Scene(Basemap.CreateImageryWithLabels());
            Scene AfterMap = new Scene(Basemap.CreateImageryWithLabels());

            // wait for scenes to load
            await BeforeMap.LoadAsync();
            await AfterMap.LoadAsync();

            // Assigns Scene objects to View
            NaturalDisasterBefore.Scene = BeforeMap;
            NaturalDisasterAfter.Scene = AfterMap;

            // List containing paths to each "before" raster
            string[] beforeimages;
            String fpath = Path.Combine(Directory.GetCurrentDirectory(), "BeforeImages\\");

            beforeimages = Directory.GetFiles(fpath, "*", SearchOption.AllDirectories).Select(x =>
Path.GetFileName(x)).ToArray();

            // List containing paths to each "after" raster
            string[] afterimages;
            String fpath2 = Path.Combine(Directory.GetCurrentDirectory(), "AfterImages\\");

            afterimages = Directory.GetFiles(fpath2, "*", SearchOption.AllDirectories).Select(x =>
Path.GetFileName(x)).ToArray();

            // Iterate through "before" raster list and add each one to the BeforeMap
            foreach (var item in beforeimages)
            {
                // specify filepath to raster location
                string filepath = Path.Combine(Directory.GetCurrentDirectory(), "BeforeImages\\" +
item);

                // Load the raster file
                Raster myRasterFile = new Raster(filepath);

                // Create the layer
                RasterLayer myRasterLayer = new RasterLayer(myRasterFile);

                // Add the layer and bookmark to the map
                BeforeMap.OperationalLayers.Add(myRasterLayer);

                // Wait for the layer to load
                await myRasterLayer.LoadAsync();

                //Creates an envelope for the current Raster
                var rasterGeometry = myRasterLayer.FullExtent;
                EnvelopeBuilder newEnvelope = new EnvelopeBuilder(rasterGeometry);
                newEnvelope.Expand(1.5);

                // Creates an envelope for comparison to bookmark location
                EnvelopeBuilder textEnvelope = new EnvelopeBuilder(rasterGeometry);
                textEnvelope.Expand(2.5);

                var xMax = newEnvelope.XMax;
                var yMax = newEnvelope.YMax;
```

```
                    var xMin = newEnvelope.XMin;
                    var yMin = newEnvelope.YMin;
                    var spatialreference = newEnvelope.SpatialReference;

                    // Converts newEnvelope to a geometry object that can be read as a Viewpoint
                    Envelope rasterEnvelope = new Envelope(xMin, yMin, xMax, yMax, spatialreference);

                    // Create Bookmark location and name for current raster
                    // Raster needs spatial reference to load
                    try
                    {
                        if (rasterEnvelope.SpatialReference != null)
                        {
                            Viewpoint viewpoint = new Viewpoint(rasterEnvelope);
                            Bookmark bookmark = new Bookmark
                            {
                                Name = Path.GetFileNameWithoutExtension(item),
                                Viewpoint = viewpoint
                            };
                            NaturalDisasterBefore.Scene.Bookmarks.Add(bookmark);
                            BookmarkChooser.Items.Add(bookmark);
                        }
                    }
                    catch (Exception ex)
                    {
                        Debug.WriteLine(ex.Message);
                    }
                }

                // Iterate through "after" raster list and add each one to the AfterMap
                // Same as the one for the BeforeMap
                foreach (var item in afterimages)
                {
                    // specify filepath to raster location
                    string filepath = Path.Combine(Directory.GetCurrentDirectory(), "AfterImages\\" +
item);

                    // Load the raster file
                    Raster myRasterFile = new Raster(filepath);

                    // Create the layer
                    RasterLayer myRasterLayer = new RasterLayer(myRasterFile);

                    // Add the layer and bookmark to the map
                    AfterMap.OperationalLayers.Add(myRasterLayer);

                    // Wait for the layer to load
                    await myRasterLayer.LoadAsync();

                }
            }
        }
}
```

Appendix B. Hyperwall How-To

The following manual describes the process for updating the maps in the Hyperwall Application.

Adding Imagery to the Natural Disaster Scene

Step 1: Get georeferenced Raster format (preferably TIFF) images from Before and After the natural disaster or event. Change the file name for each image to be the name that will show up on the bookmark selector within the application. Example: v0i31mv030g10202015.tif → Seti Valley Flooding.tif

Step 2: Add the images to an ArcGIS Pro project and build pyramids for the images.

Step 3: Save the before image (with .ovr pyramid file) to the BeforeImages folder. Find this folder by going to the file directory of the Hyperwall Application and open the following directory: C:\…\Hyperwall3\Hyperwall3\bin\Debug\BeforeImages.

Step 4: Save the after image (with .ovr pyramid fil) to the AfterImages folder**.** Find this folder by going to the file directory of the Hyperwall Application and open the following directory: C:\…\Hyperwall3\Hyperwall3\bin\Debug\AfterImages.

Step 5: The application handles the rest. Run the Hyperwall Application, open the Natural Disaster Scene and ensure the images load correctly in the application.

Updating the Heat Vulnerability Map

Step 1: Open the data for the Heat Vulnerability Map in ArcGIS Pro.

Step 2: Customize the layers to show the legend and symbology of choice.

Step 3: Package the map as a Mobile Map Package (under the “Share” tab).

Step 4: Navigate to the MobileMapPackages folder within the application’s file directory: C:\…Hyperwall3\bin\Debug\MobileMapPackages.

Step 5: Ensure the file name matches the Mobile Map Package currently in the MobileMapPackages folder. “HeatVulnerabilityMap.mmpk”.

Step 6: Run the Hyperwall Application then open the Heat Vulnerability Map and ensure the map was updated.

Updating the REST Endpoints for the Air Quality Index

Execute this workflow when updated air quality feature services are desired.

Step 1: Copy URLs of the REST Endpoints for the sensor locations and the air quality surface.

Step 2: Open the Hyperwall3.csproj file in Visual Studio. The file can be found in your file directory at: C:\...\Hyperwall3.

Step 3: Open the AirQualityMap.cs class file located in the MapClasses folder.

Step 4: Navigate to Line 15 of the class file and replace the URL there with the new URL of the selected REST Endpoint for the air quality index surface.

Step 5: Navigate to Line 25 of the class file and replace the URL listed with the new URL for the sensor locations.

Step 6: Run the Hyperwall Application and ensure the new air quality feature services load.