HostsPro

Gage Price

CST-452 Capstone Project Final Architecture & Design

Grand Canyon University

Instructor: Professor Mark Reha

Revision: 2.0

Date: 3/12/2025

**ABSTRACT**

This project is an application to organize the Microsoft host’s file. The hosts file in the file explorer is a normal text file which contains host names and their corresponding IP addresses. With this file you can override DNS for defined domains on a computer. This file has little structure, the constraints for entries is limited to one line per entry, spaces sperate the domain name and IP address, and comments are created using the pound symbol. The task for this project is to create an application that provides a simple and clean user interface for the user. The user will be able to enter their desired name server and their specified host name. My program will run a background search for the IP address of the name server provided. The program will then use that IP address and note where it points to by storing a comment of the name sever entered. They can activate or deactivate an entry with a click of a button.

For the average person this file is never used, but for some developers this file is used a lot in testing. Others use this file to block websites and set restrictions on what website a device can access. This file editor interface will make their work a whole lot easier, simpler, cleaner, and more organized.

|  |
| --- |
| History and Signoff Sheet |

**Change Record**

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Revision Notes** |
| 11/17/2024 | Gage Price | Initial draft for review/discussion |
| 3/12/2025 | Gage Price | Updates from development changes |
|  |  |  |

|  |
| --- |
| **Overall Instructor Feedback/Comments** |

|  |
| --- |
| **Overall Instructor Feedback/Comments** |

**Integrated Instructor Feedback into Project Documentation**

Yes  No

**TABLE OF CONTENTS**

Design Overview 4

Detailed High-Level Solution Design 5

Detailed Technical Design 6

Appendix A – Technical Issue and Risk Log 7

Appendix B – References 8

Appendix C – External Resources 9

Design Introduction -112.5pts

1. Below is an introduction to the high-level design of the proposed solution for the HostsPro application. Please review the attached folder “Design Diagram” to view the files “Wireframe UI Diagram” and “UML Class Diagrams” for references.

The proposed solution for the HostsPro application is to create an easy to use and clean interface for editing and managing the hosts file. The front-end design highlights a clean and simple list structure to view and edit entries for the file. The user will be presented with a list of entry forms for both IP entries as well as comment entries, all of which are openly editable. There are also buttons to add entries, delete entries, and save the changes. The proposed problem this application is trying to solve is that the system hosts file is hard to work with and messy. This interface design aims to fix these problems by providing easy data manipulation. To support the front-end, a C# application will work in the back end. The application will use Model-View-View Model (MVVM) architecture to create databinding between the back end and the front. This class facilitates all needed background processes are needed to accomplish a user’s requests. The view model will keep track of all changes made by the user and validate the user is entering valid data. It also is used to call the business layer processes for all button click events from the view. When retrieving data from the hosts file, the data access layer will access the file and collect its contents. It then passes this data to the business layer where the data will be parsed into a model that can be used in the view. When saving to the file, the business layer translates the model data back into data that the file can read. The data then travels down into the data access layer where the data is written to the file. Another requirement for this project is to do automated IP address searches for the user. To accomplish this, another business layer class will be used to do a search, and return the result back to the view. Together the front end and back end will work together to provide a seamless and efficient application to accomplish the proposed solution.

1. This table below keeps track of external Design specifications. There are currently no external design resources.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Deliverable Acceptance Log | | | | | |
| ID | Deliverable Description | Comments | Evaluator (internal or external as applicable) | Status | Date of Decision |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |

Detailed High-Level Solution Design

1. Below is a detailed overview of how the proposed design fits into the overall solution.
   1. Below is a quick overview of the solution configuration changes that will be required to develop and implement the proposed solution.

There are not many solution configuration changes that must be made to successfully develop this project, except having permission to write to the hosts file. The host file on a windows computer has privileges set though that it requires admin permissions. Because of this the developer will have to open Visual Studio in admin mode to write to the file. For development purposes, a test hosts file will be made so that we can ensure the application runs while outside of admin mode. Also to develop this application, a developer would need Visual Studio 2022working with a target framework of .NET 8.0.

* 1. Below is the approach to assure that all non-functional requirements will be met.

The non-functional requirement for this project is to get a routing location from the user and return either a corresponding IP Address or a message notifying no IP Address was found in under eight seconds. To accomplish this, there is going to be an event handler for when the user clicks off the “DNS” cell (see UI Diagram below for reference). When the event is triggered, a system command will be called, and the application will try to extract the IP Address. At the same time, a timer will start and if the timer reaches eight seconds, then the user will get the warning message that there is no correlating IP address. This way, if there is a problem with the lookup, then the user can continue with their work.

1. Below is a table outlining the technologies that were chosen to be used in the building process of this project. Along with it is another table showing proof of concepts that were used to ensure the that the technologies selected were the best fit for this application.

|  |  |  |
| --- | --- | --- |
| Proof of Concepts | |  |
| **Description** | **Rationale** | **Results** |
| 1- POC of View Model class working with a view | The purpose of this proof of concept was to prove that the MVVM architecture is the best way to create a dynamic UI. | The result showed that the view model class successfully communicated with the view to handle changes and events that were triggered by the user. |
| 2 – POC of System Command Tool being used withing Visual Studio | The application requires it to lookup an IP address when giving a location name of where traffic can be routed. Because of this the command line tool would have to be used to perform this search, and return a validating response. | The result of this proved that the Command Line Interface could be access, and a result could be returned. The result would have to be parsed based on the results structure and syntax. |

|  |
| --- |
| Hardware and Software Technologies |
| 1 – Windows Presentation Foundation |
| 2 - .NET 8.0 Framework |
| 3 – Visual Studio 2022 |
| 4 – System Command Tool |

**Logical Solution Design:**

See the attached folder “Design Diagrams” and select the file “Logical Solution Diagram” to see the high-level solution diagram for the HostsPro application. This diagram displays how the software passes data though each logical layer. The presentation layer of the application is going to be the main window .xaml view, which is going to display a collection of file entries. That collection is going to be tied to a view model class, which will implement interface methods to display all user changes dynamically. The business layer will handle all of the data manipulation for both the model data and the view model data. The data access layer is used as an entry point for all external data. HostsPro’s purpose is to modify a systems “hosts” file, so the data store for this application is going to be that file. The data access layer will have service classes to retrieve data from the hosts file. The IpLookupManager class in the business layer will be used to retrieve IP information for the user through CLI command. These commands are done using built in C# functions to get information for the user.

**Physical Solution Design:**

See the attached folder “Design Diagrams” and select the file “Physical Solution Diagram” to see the high-level physical solution diagram for the HostsPro applciation. Bcause this is a desktop application that only communicates with one system file and the command line interface, there are not many pieces of hardware necessary to host this applciation. Deploying this application for a user to download is currenlty out of scope, but this solition was designed to respect a possible deployment. In the case that the application is downloaded onto a user’s windows computer, there are no configuration changes that a user would need to run this applciaiton. When the applciation is running, it talks to the systems hosts file to read and write data. It also take advatage of the systems command line interface to retrieve IP information. All windows computers come with these resources already. It should be noted that the applciaiton would need to be running in admin mode to make changes to the file.

Detailed Technical Design

**General Technical Approach:**

In the initial meeting with the client, certain requirements were outlined for the project. The client expressed that they wanted a desktop application that should provide an easier way to modify the hosts file. They also requested an automated process to lookup the IP address of a given DNS location to speed up their work. The team then conducted research on the best ways to provide a solution for these requirements. Because this desktop application will have to be responsive to adding and deleting entries, we decided the C# WPF framework combined with an MVVM architecture would be the best choice. With these choices comes the benefit of databinding, which provides the synchronization between the user interface and the view model that is required. This databinding also creates a great user experience because the back-end changes are immediately shown in the front-end.

**Key Technical Design Decisions:**

The goal of this application was to create a clean UI for a desktop application where a user can easily read and update their hosts file. Because this is a desktop application that would need data to be presented to the view dynamically, the decision was made to use Visual Studio as the IDE because it provides a lot of features to create desktop applications. The decision was made to use WPF as the framework because after some research we found that it supports complex and quick databinding.

**Database Store:**

See the attached folder “Design Diagrams” and select the file “Example File Structure” to see an example of the test format that will be used to store the application data. The system recognizes the pound symbol (#) as the start of a comment. The only information the systems needs to acknowledge an entry is an Ip Address and a DNS location that would be overwritten. Because of this, I can take advantage of separating data the system will read and other stored information by using multiple pound symbols. At the end of each entry, there will by one inline pound symbol to mark an inline comment and the “DNS” location. If the entry is marked active, then nothing will be added to the line in the file. If it is marked inactive, there will be a single leading pound symbol. For comment block sections, the start and end of a comment block will be defined by three pound symbols, and the content inside of those comment blocks will be defined with two leading pound symbols.

**Flow Charts/Process Flows:**

See the attached folder “Design Diagrams” and select the files “IP Lookup Flowchart”, “Read From File Flowchart”, and “Write To File Flowchart” to see the flow processes for this application. The read and write flowcharts outline the logic that will be needed to parse data between raw text and the models of the application. Because the hosts file has specific rules for how data is read, custom logic is required to accommodate this. Reference the attached file “Example File Structure” to see how the data will look in the file. The IP Lookup flowchart outline at a high level how the application will do an automated search for an IP address. Because this application uses MVVM architecture, the View Model will keep track of changes made in the view. When the DNS input loses focus, business layer methods are called to execute the search. The timer is to ensure the NFR (reference the NFR section below for more detail) is met. These methods will either return valid data or invalid data. The timer ending before a result will also cause the method to return invalid data. If the data is valid then the corresponding entry’s IP address input will be auto populated through the view model. If the data is invalid, then the appropriate error message will be shown. All other events done in the view such as adding or deleting entries as well as editing entries will just call the property changed event in the view model.

**User Interface Diagrams:**

See the attached folder “Design Diagrams” and select the file “Wireframe UI Diagram” to view the user interface diagram of what will be displayed to the user. This application will only have one page. All round cells will be editable, and on save those changes will be applied to the file. The red error popup in the top right will be displayed if there is a problem saving to the file. In the last line you can see red text, which is how the user will know if their “DNS” location is invalid. The user will be able to add both Comment and IP entries (see models and example hosts file for more information), as well as delete them.

**UML Diagrams:**

See the attached folder “Design Diagrams” and select the file “UML Class Diagrams” to see the class diagrams for this project. This diagram map outlines all of the classes that will be used in this project along with their properties and methods. There are arrowed lines showing association between the classes. The direction of the arrows goes from the class that is using another class to the class that is being used. The filled in diamond between the model classes represents composition. The classes are also organized by layers from top to bottom. See the next to the right of the diagram to see what layer each class is in.

**NFR’s (Security Design, etc.):**

The non-functional requirement for this application I to get a result from the IP Address search in under eight seconds, whether it displays the Ip address or a validation error (see project requirement’s excel sheet for more detail). To support this, when a user clicks off the “DNS” input cell on the page an event will be triggered to execute the lookup. At the same time, a countdown timer will be started. If the method doesn’t return a valid IP address before the timer is done then an error message will be displayed saying that the search took too long. This message will also instruct the user to do a manual search themselves. If the search does return either a valid or invalid IP address then the user will see those result in the view. For a valid IP, the IP input box will auto populate. If an invalid IP, there will be a message presented for an invalid IP.

**Operational Support Design:**

Because this is an isolated desktop application, logging will be done to a file on the computer where the application is downloaded. Try, catch blocks will be used to ensure an error-resistant application. If an error does occur, then the file will be written to. It will record information about the exception so that someone can look back at the file and identify where the problem is.

**Other Documentation:**

There is no further documentation that provides a direct benefit to the solution or design; however, it can be noted that weekly status reports were kept to monitor the state of the project.

Appendix A – Technical Issue and Risk Log

1. Below is a tale to log any issues and risks that currently exist in the project. All issues about technologies have been resolved, so there are currently no issues or risks.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Issues and Risk Log | | | | | | | | |
| **Issue or Risk** | **Description** | **Project Impact** | **Action Plan/Resolution** | **Owner** | **Importance** | **Date Entered** | **Date to Review** | **Date Resolved** |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

Appendix B – References

*N/A*

Appendix C – External Resources

|  |  |
| --- | --- |
| **GIT URL:** | *N/A* |
| **Hosting URL:** | *N/A* |