EasyID: Design and Functional Specification

Project Requirements and Architecture

9 December 2022
Written by Taryn Chovan, Jesus Cantu, Joseph Lombardi

Owners and List of Contacts

Name	Email	Phone	Role
Taryn Chovan	tchovan@luc.edu	(222) 333-4444	Developer
Jesus Cantu	jcantu2@luc.edu	(333) 444-5555	Developer
Joseph Lombardi	jlombardi2@luc.edu	(444) 555-6666	Developer

Signoffs

Phase	Name	Date	Signature
1	Taryn Chovan	12/09/2022	Taryn Chovan
1	Jesus Cantu	12/09/2022	Jesus Cantu
1	Joseph Lombardi	12/09/2022	Joseph Lombardi

Revision History

Date	Reason for change(s)	Author(s)
12/09/2022	Initial publication (version 1)	Taryn Chovan Jesus Cantu Joseph Lombardi

Table of Contents

About this Document

Purpose and Contents

What this specification does

This specification explains the design, capabilities, and use of the EasyID software. It provides contact information for the authors to direct any further questions or concerns about the software not addressed in this specification.

What this specification does not do

This is not a project plan. It is a guide for system architecture and development, not for phasing, timelines or deliverables.

A 'Living Document'

Finally, this specification will change, continuously, as the project proceeds. We will add details and edit existing information as the structure and functionality evolve in the course of the project.

Assumptions

This specification assumes the user has properly installed the software requirements and can successfully run the program.

Questions and Comments

If you have questions or comments regarding this document, contact Taryn Chovan at tchovan@luc.edu, Jesus Cantu at jcantu2@luc.edu, or Joseph Lombardi at jlombardi2@luc.edu.

Software Overview

Product description

EasyID is a tool that launches a local web server. The server hosts a search bar that accepts user input and attempts to identify what a piece of data is. It accepts either a singular data string, or a file full of potential data strings as input. If there is a match, it outputs to the screen what it has identified on, and any extra details about the data. Example: 630-111-2222 matches on phone number and extrapolates an Illinois area code.

Product functional capabilities

EasyID is currently able to match a string against the following data types:

- Bitcoin Wallet Address
- Credit Card Number
- Date
- Email Address
- FedEx Tracking Number
- IPv4 Address
- IPv6 Address
- MAC Address
- Social Security Number
- United States Phone Number
- United States State Abbreviation
- United States Zip Code
- UPS Tracking Number
- URL
- USPS Tracking Number
- Windows GUID
- Windows UUID
- YouTube Link

User characteristics

EasyID is currently not hosted on a web server. The project will be open source and available for everyone to download from a GitHub repository. Users need to be acquainted with programming languages, as well as downloading/executing code to be able to use it. Some knowledge of C#, Visual Studio, and .NET is required and would be especially useful when making their own versions of the project.

User operations and practices

Users should only input one piece of data or file at a time. The box labeled ‘Identifier’ is used to submit data strings and or file names. Files need to be located inside the project for

EasyID to find/read them, and can be placed inside the ‘wwwroot’ folder. The box labeled ‘Type’ is used to identify whether the data given is a string or a file. Once both boxes have been filled, the user should click the ‘Submit’ button.

Submission of multiple data strings will result in invalid matches or errors. Submission of multiple data files is currently not an option. When possible, the user should copy and paste the data string into the submission box to avoid adding/removing characters to the string by error and thus affecting the resulting match. Matches are currently not case/delimiter sensitive. For EasyID to work the user does not have to change the format of any particular data string. Results for each submission are concatenated in the output screen until the user clicks the ‘Clear’ button. If there is a match, it outputs to the screen what it has identified on, and any extra details about the data. If there is no match, the output will read “No match!”.

General constraints and Limitations

The ability of EasyID to identify a piece of data is limited by the modules we created to test/validate different types of information. It is not all inclusive of the data that you may find on the web or receive as spam. In order to be as comprehensive as possible, more modules will have to be added in the future that span more domains than those originally included.

Some of our modules include validation functions based on regex (regular expressions). This limits the verification of the data, within each specific domain (e.g., email address, URL, etc.) to what is included in the regex. We used regex that were general and included a variety of different formats for each specific case, for example a phone number could be written by the user with delimiters such as dashes (XXX-XXX-XXXX), spaces (XXX XXX XXXX), and/or periods (XXX.XXX.XXXX). Our regex includes a variety of possible formats but does not include near possible cases, like if only the first few digits of a phone number is written, or formats that are not as common. Formats, particularly for URLs, can change over time and thus the regex expressions will have to continuously be re-evaluated to fit modern standards/regulations.

Our module selection algorithm could be optimized. The current approach is to test a data string on each of the different modules until a match is obtained, a machine learning approach (e.g., neural networks) could make the process faster and more robust. Our current algorithm also needs a better way to handle matching on multiple modules (i.e., a singular nine-digit numeric string will match on both UPS Tracking Number and Social Security Number with no way of differentiating between the two).

Lastly, some but not all modules can identify specific data but cannot confirm if that information is fake or real. If it follows the general format of a given data type it will label them as so. More API’s need to be implemented in some existing modules to test out if the piece of data (e.g., URL or tracking number) is real and currently active.

Assumptions

Once EasyID identifies a specific piece of data as X or Y, it is assumed that it is a 100% match or not. Near possible matches are not currently considered by the software. It also does not consider cases where a piece of data might be more than one thing. Future work should include a ranking system (e.g., percent likelihood) and give the user a list of possible matches if the situation arises.

Other software

A search engine, we recommend Google Chrome and the latest version of Visual Studio (with the following packages installed: Newtonsoft.Json and HtmlAgilityPack). .

System Architecture

Description

The architecture is divided into three logical layers to better meet the requirements of reusability, ease of code maintenance, and scalability.

Logical View

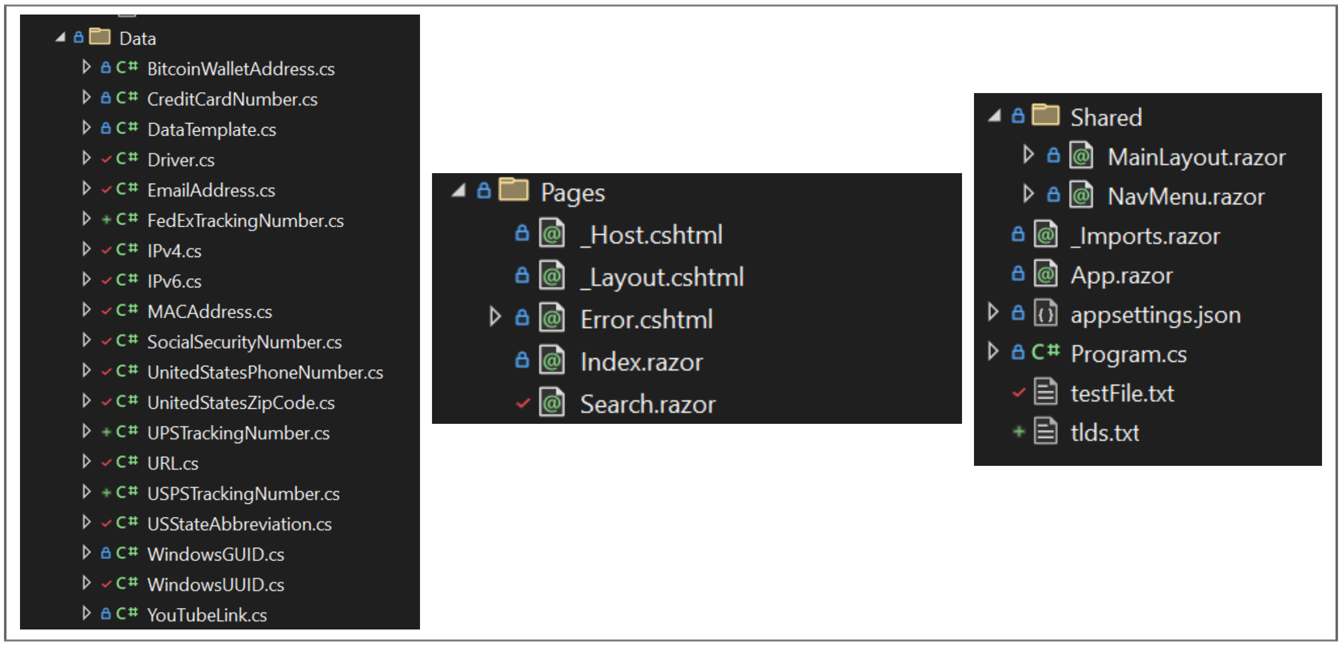


Figure 1: High Level Logical Architecture

1. **Web Layer/Presentation (Shared Objects)**
   The Shared folder consists of files and directories that need to be shared by the entire program. This includes App.razor for initializing the whole web app, and files used for module details like a text file of top level domains.

2. **Business Logic Layer (Page Layouts)**
   The page layouts folder consists of template code for pages on the user interface. The driving logic occurs in the Search page which instantiates Driver objects to compare against data modules.

3. **Data Access Layer (Data Modules)**
   The data folder consists of various modules that get tested against for identifying user input. It also holds the Driver module that drives the main logic for the program.

Physical View

The software uses Microsoft Blazor pages to instantiate a web server running on a high port of your local machine. All logic is handled locally, other than some API calls to various services for providing additional details about the identified input.

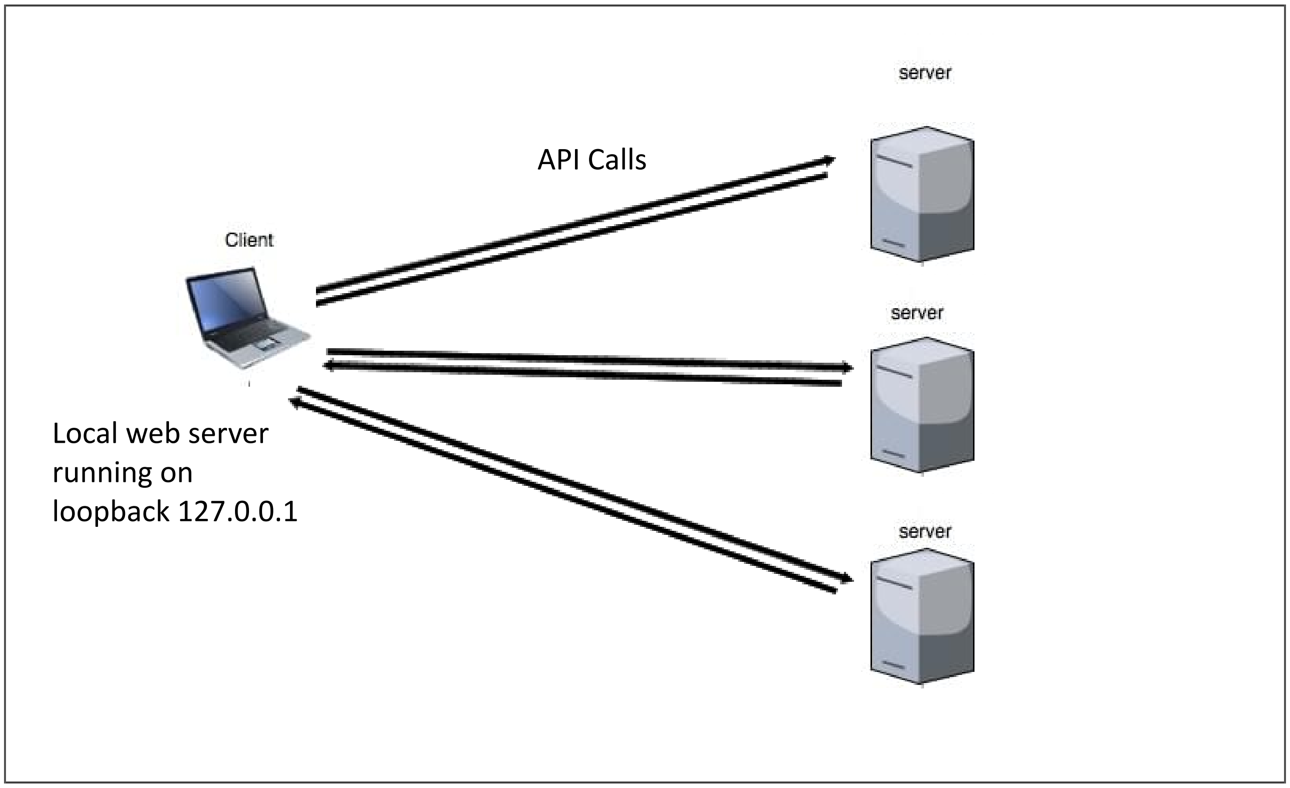


Figure 2 – Physical Deployment Diagram

Class Diagram

- Driver objects are created from the Search module
- All data modules inherit from the abstract DataTemplate module
- The Search module initializes data modules

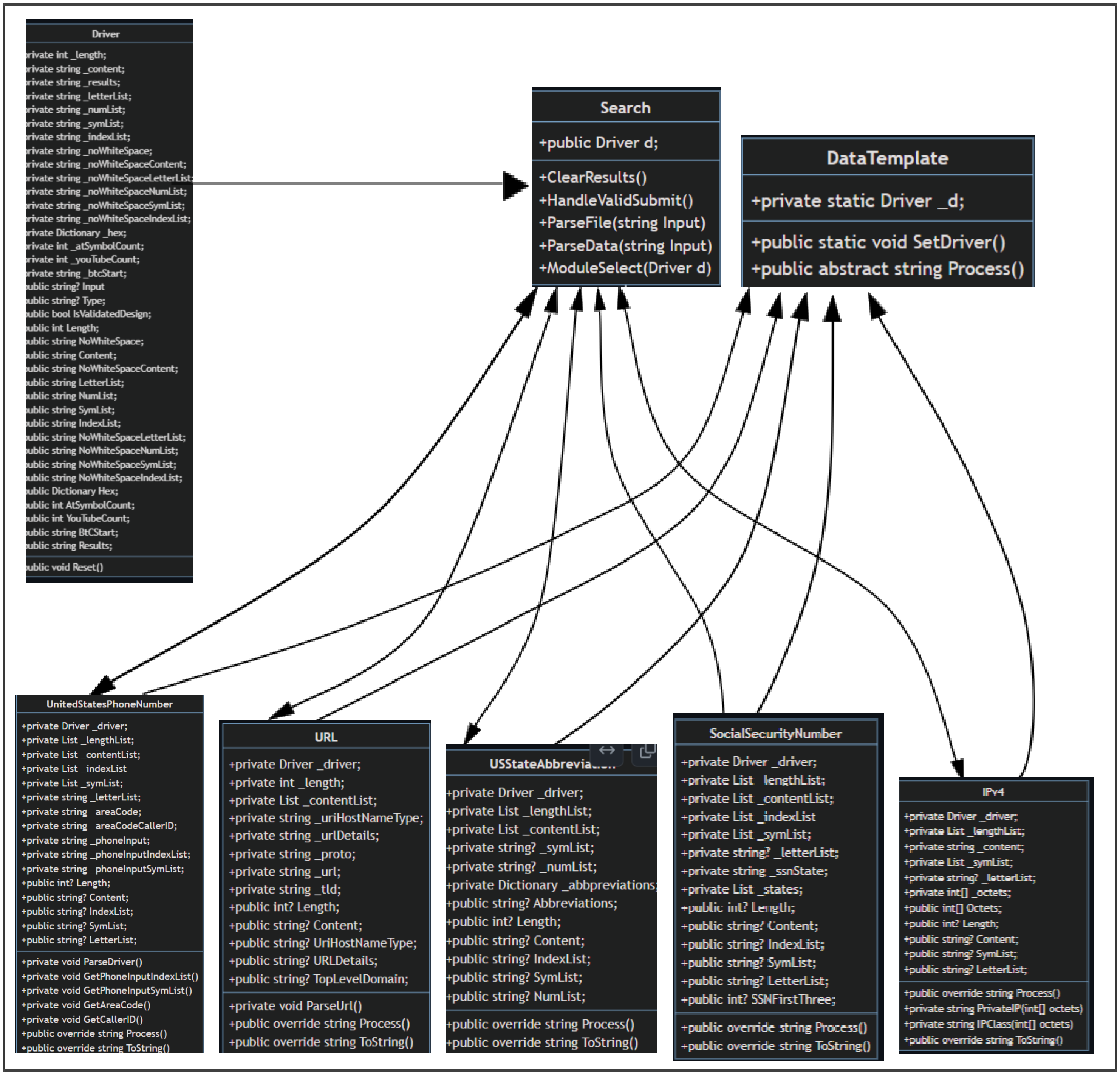

Driver
rivate int _length;
rivate string _content;
rivate string _results;
rivate string _letterList;
rivate string _numList;
rivate string _symList;
rivate string _indexList;
rivate string _noWhiteSpace;
rivate string _noWhiteSpaceContent;
rivate string _noWhiteSpaceLetterList;
rivate string _noWhiteSpaceNumList;
rivate string _noWhiteSpaceSymList;
rivate string _noWhiteSpaceIndexList;
rivate Dictionary _hex;
rivate int _atSymbolCount;
rivate int _youTubeCount;
rivate string _btcStart;
ublic string? Input
ublic string? Type;
ublic bool IsValidatedDesign;
ublic int Length;
ublic string NoWhiteSpace;
ublic string Content;
ublic string NoWhiteSpaceContent;
ublic string LetterList;
ublic string NumList;
ublic string SymList;
ublic string IndexList;
ublic string NoWhiteSpaceLetterList;
ublic string NoWhiteSpaceNumList;
ublic string NoWhiteSpaceSymList;
ublic string NoWhiteSpaceIndexList;
ublic Dictionary Hex;
ublic int AtSymbolCount;
ublic int YouTubeCount;
ublic string BtCStart;
ublic string Results;
ublic void Reset()
Search
+public Driver d;
+ClearResults()
+HandleValidSubmit()
+ParseFile(string Input)
+ParseData(string Input)
+ModuleSelect(Driver d)
DataTemplate
+private static Driver _d;
+public static void SetDriver()
+public abstract string Process()
UnitedStatesPhoneNumber
+private Driver _driver;
+private List _lengthList;
+private List _contentList;
+private List _indexList
+private List _symList;
+private string _letterList;
+private string _areaCode;
+private string _areaCodeCallerID;
+private string _phoneInput;
+private string _phoneInputIndexList;
+private string _phoneInputSymList;
+public int? Length;
+public string? Content;
+public string? IndexList;
+public string? SymList;
+public string? LetterList;
+private void ParseDriver()
+private void GetPhoneInputIndexList()
+private void GetPhoneInputSymList()
+private void GetAreaCode()
+private void GetCallerID()
+public override string Process()
+public override string ToString()
URL
+private Driver _driver;
+private int _length;
+private List _contentList;
+private string _uriHostNameType;
+private string _urlDetails;
+private string _proto;
+private string _url;
+private string _tld;
+public int? Length;
+public string? Content;
+public string? UriHostNameType;
+public string? URLDetails;
+public string? TopLevelDomain;
+private void ParseUrl()
+public override string Process()
+public override string ToString()
USStateAbbrevia...
+private Driver _driver;
+private List _lengthList;
+private List _contentList;
+private string? _symList;
+private string? _numList;
+private Dictionary _abbpreviations;
+public string? Abbreviations;
+public int? Length;
+public string? Content;
+public string? IndexList;
+public string? SymList;
+public string? NumList;
+public override string Process()
+public override string ToString()
SocialSecurityNumber
+private Driver _driver;
+private List _lengthList;
+private List _contentList;
+private List _indexList
+private List _symList;
+private string? _letterList;
+private string _ssnState;
+private List _states;
+public int? Length;
+public string? Content;
+public string? IndexList;
+public string? SymList;
+public string? LetterList;
+public int? SSNFirstThree;
+public override string Process()
+public override string ToString()
IPv4
+private Driver _driver;
+private List _lengthList;
+private string _content;
+private List _symList;
+private string? _letterList;
+private int[] _octets;
+public int[] Octets;
+public int? Length;
+public string? Content;
+public string? SymList;
+public string? LetterList;
+public override string Process()
+private string PrivateIP(int[] octets)
+private string IPClass(int[] octets)
+public override string ToString()

Figure 3 – Class Diagram

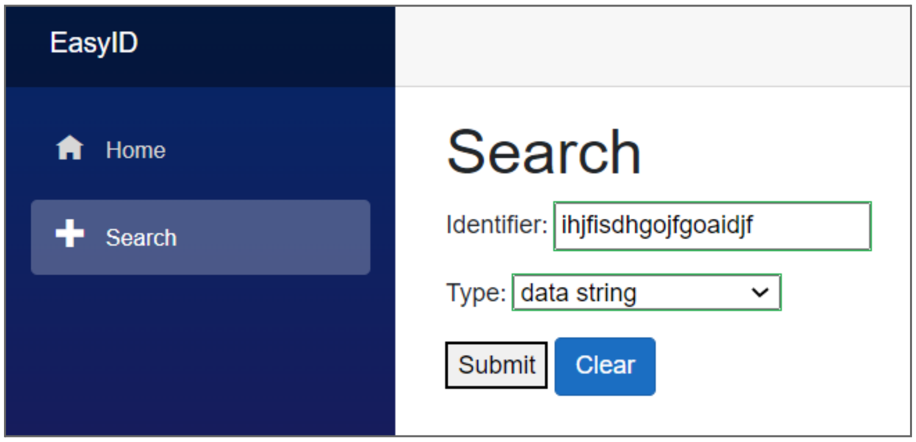
Specific Function Description

Product

User Interface
The user interface currently consists of a Home page and a Search page. The home page introduces the user to EasyID and the search page is where the user can attempt to identify their data.

Inputs

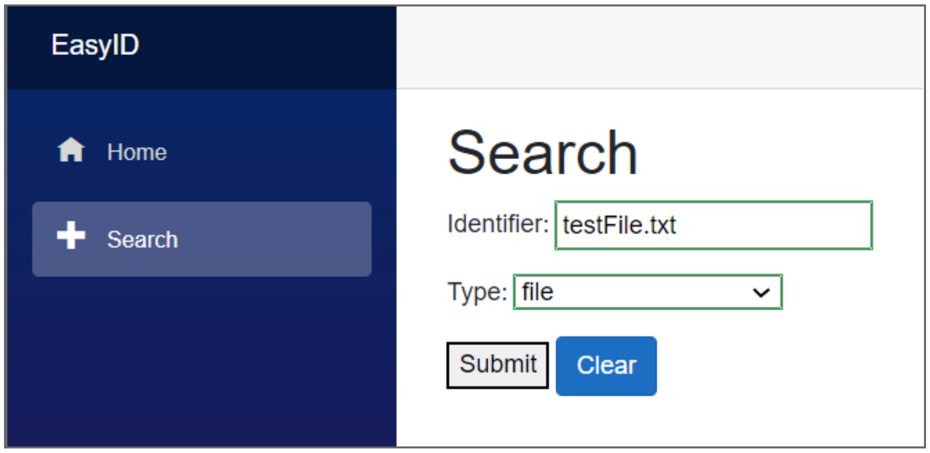
1. A singular data string



Identifier: ihjfisdhgojfgoaidjf

Type: data string

2. The name of a file, without the specified path. The file must exist in the current working directory.



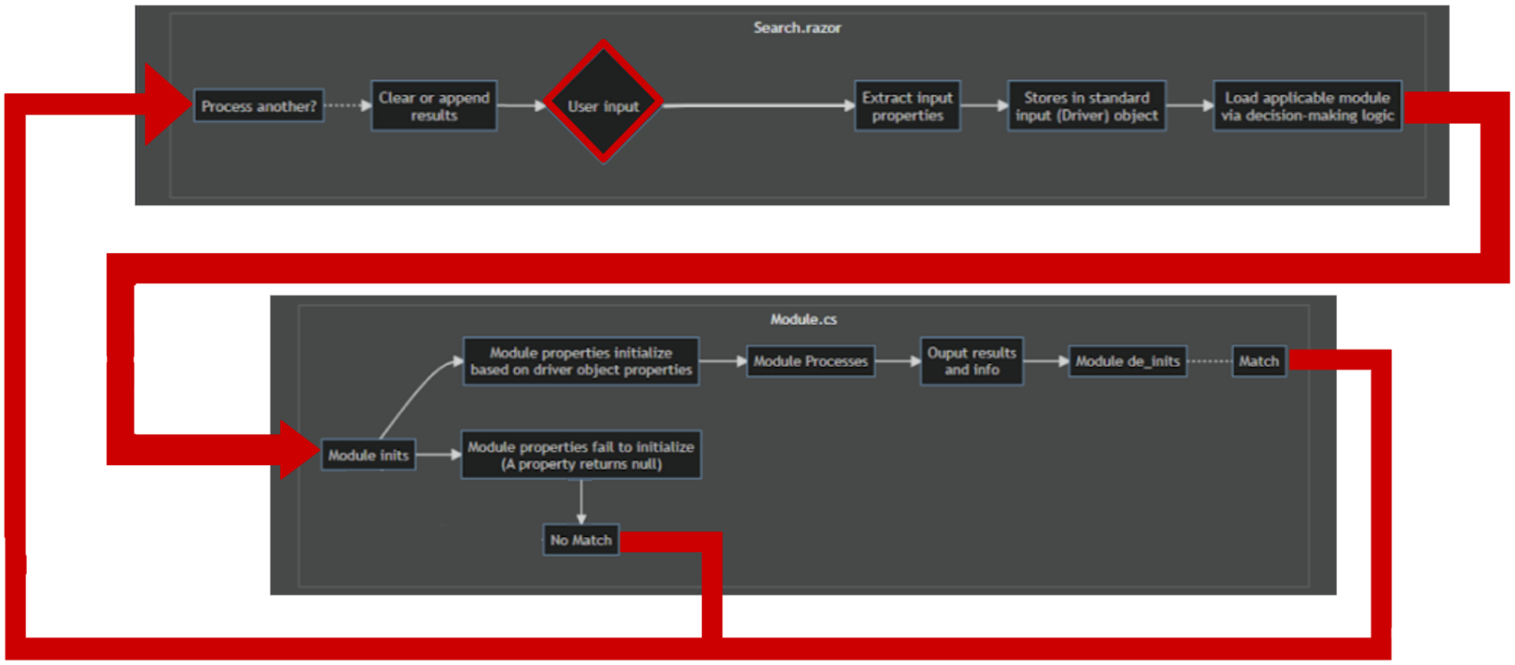
Identifier: testFile.txt

Type: file

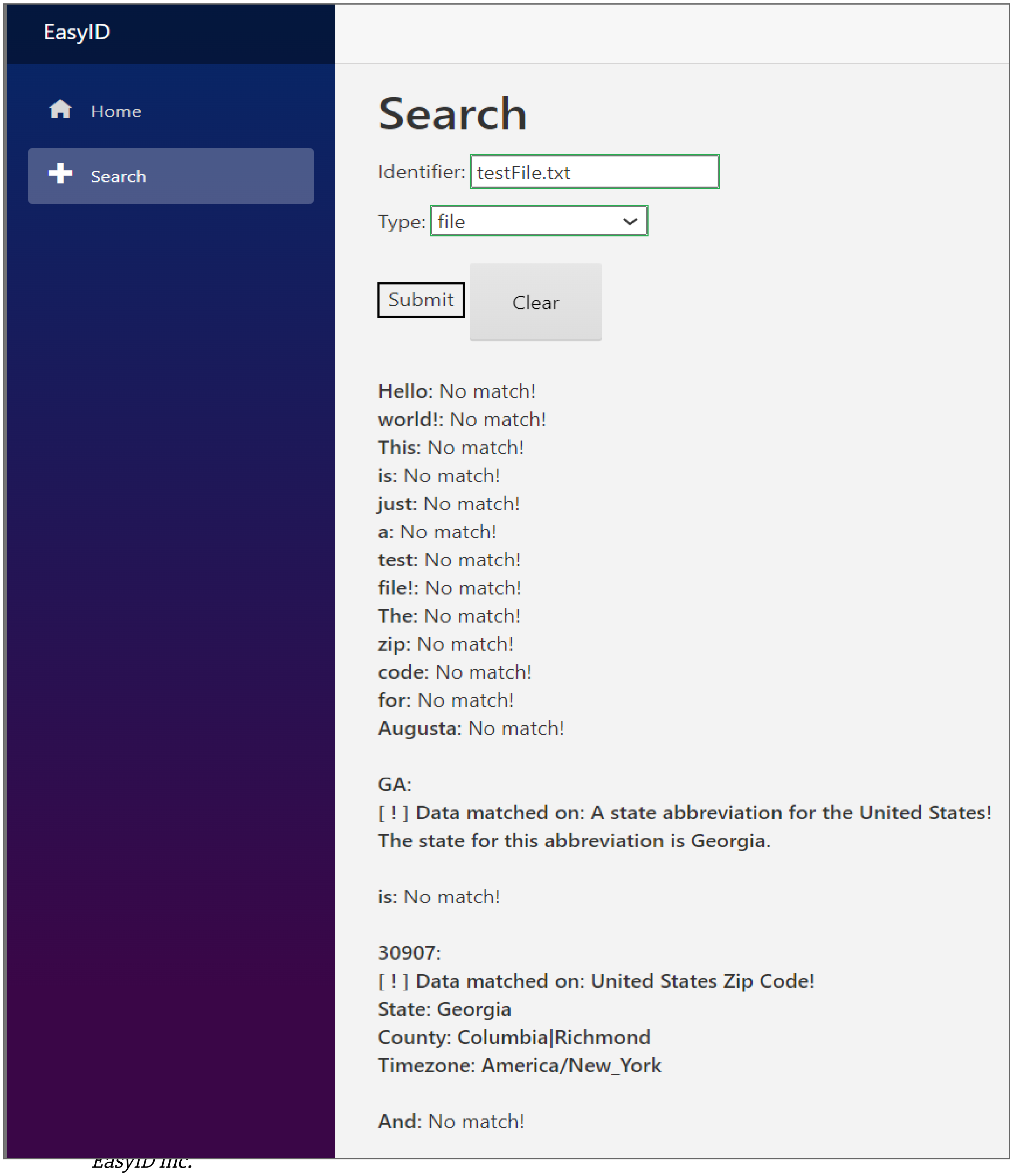
Processing

The input is processed in the following manner:

1. User inputs a file or data string
2. Data string properties are extracted and stored in a Driver object.
3. A module attempts to load, based on Driver's properties.
4. If the tested module returns null properties, no match is determined.
5. If the tested module return a value for all properties, the input matches on that module.
6. The results are outputted to the screen and the program is ready to either clear the results or process more input.


Search.razor
Process another?
Clear or append results
User input
Extract input properties
Stores in standard input (Driver) object
Load applicable module via decision-making logic
Module.cs
Module properties initialize based on driver object properties
Module Processes
Ouput results and info
Module de_inits
Match
Module inits
Module properties fail to initialize (A property returns null)
No Match

Output Screen:



Search

Identifier: testFile.txt

Type: file

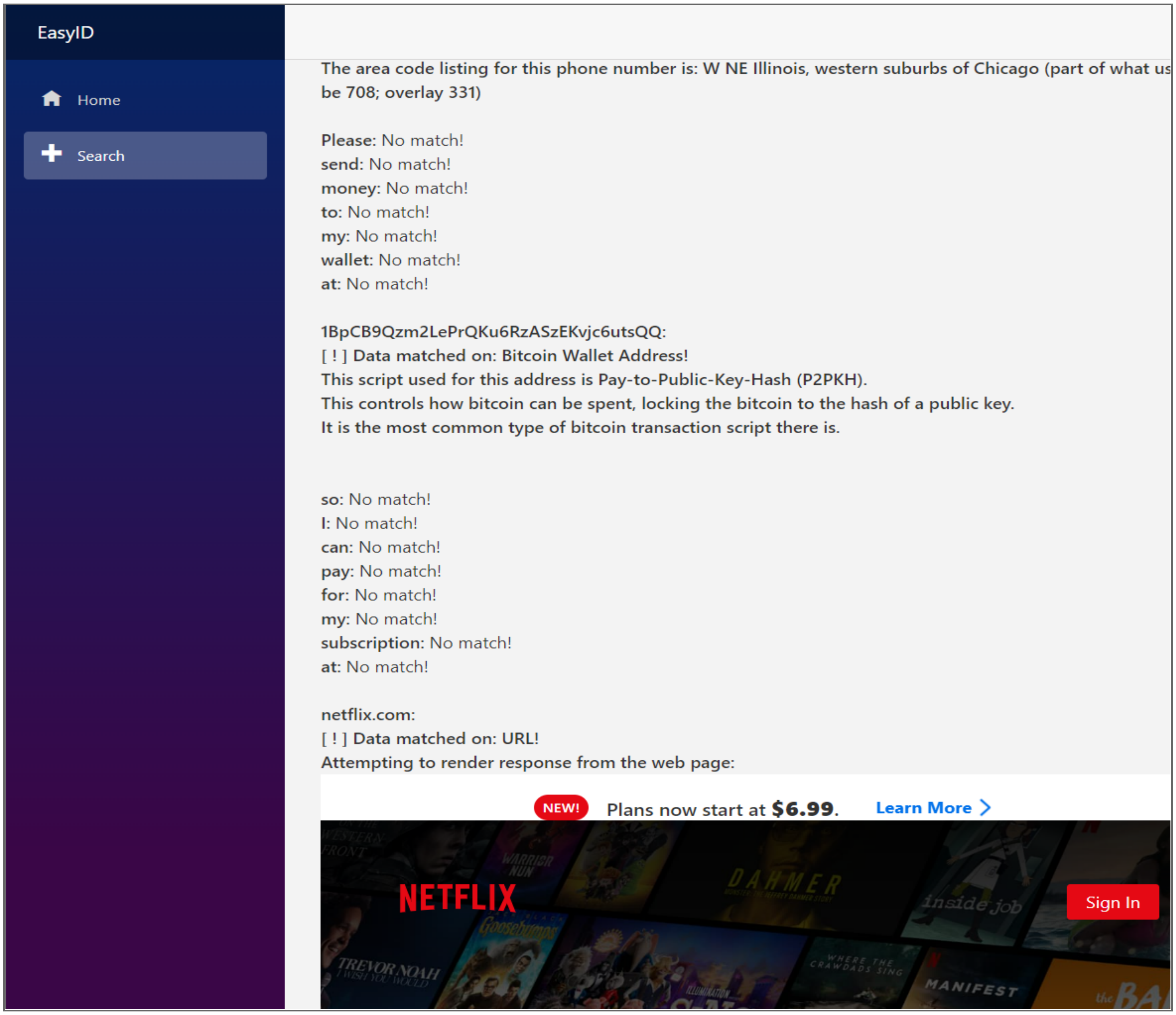
```
Hello: No match!
world!: No match!
This: No match!
is: No match!
just: No match!
a: No match!
test: No match!
file!: No match!
The: No match!
zip: No match!
code: No match!
for: No match!
Augusta: No match!

GA:
[ ! ] Data matched on: A state abbreviation for the United States!
The state for this abbreviation is Georgia.

is: No match!

30907:
[ ! ] Data matched on: United States Zip Code!
State: Georgia
County: Columbia|Richmond
Timezone: America/New_York

And: No match!
```



The area code listing for this phone number is: W NE Illinois, western suburbs of Chicago (part of what us
be 708; overlay 331)

Please: No match!
send: No match!
money: No match!
to: No match!
my: No match!
wallet: No match!
at: No match!

1BpCB9Qzm2LePrQKu6RzASzEKvjc6utsQQ:
[!] Data matched on: Bitcoin Wallet Address!
This script used for this address is Pay-to-Public-Key-Hash (P2PKH).
This controls how bitcoin can be spent, locking the bitcoin to the hash of a public key.
It is the most common type of bitcoin transaction script there is.

so: No match!
I: No match!
can: No match!
pay: No match!
for: No match!
my: No match!
subscription: No match!
at: No match!

netflix.com:
[!] Data matched on: URL!
Attempting to render response from the web page:

Security

The security of this program is dependent upon any traffic traversing the Internet. Some modules make API calls or web requests to servers that cannot be assumed to be secure. All other modules that do not make API calls keep all traffic contained to your machine and local web server. At the time of this writing we’ve only made API calls to servers running HTTPS and ensured other security features. It is up to the user to make sure they are running the most up-to-date version of the software to ensure that they are not making an API call that has since become unsecured.