Requirements Document for Logic Gate Simulator

October 26th, 2018

Aby Prasad, Jenna Westfall, Bright Sun, Robert Rose, Yael Weiss

Table of Contents:

1. Introduction

1.1 Purpose

This document describes the purpose and intended usage of a browser based circuit simulation application. This application allows users to build and test circuits inside a web browser, mitigating the need to install specialized software for the same purpose. It will also allow for easy sharing of circuit diagrams via URLs.

1.2 System overview

Logic Gate Editor (working title) will be a web version of the logic gate learning tool Logisim. Like Logisim, the app will simulate basic circuitry and logic gates. The end goal is to make a simulation environment that will be easily accessible and user-friendly to students. A top priority will be to improve upon Logisim's design by making a logic gate simulator that will be more user-friendly and more accessible (easy to access without download/setup, and be able to share work with a web link).

1.3 Document overview

This document goes over the software requirements for the Logic Gate Editor. There are no known security or privacy risks known from viewing this document. Section 2 will describe the project in more detail. Section 3 will cover user and system requirements. Section 4 will give ways to qualify the requirements stated in section 3. Section 5 provides a traceability table to view what project goal each of the requirements corresponds to.

1.4 Definitions

Throughout this documents the following terms are used:

- **Logic Gates:** Refers to an elementary building block of a digital circuit. Logic gates have inputs and one output. The inputs and outputs are binary values that indicated by 1 and 0. Different types of logic gates output different results.
  Ex: AND gate, OR gate, NOT gate, XOR gate, NAND gate, NOR gate, XNOR

- **Circuits:** A collective of logic gates that use one and others as inputs. They produce a single binary output.
- **Diagram:** The entire display consisting all a logic gates and circuits constructed.
- **GUI:** Stands Graphical User Interface
- **Logisim:** A downloadable program used to simulate logic gates and circuits.
- **Wires:** Used to connect the output of one logic gate to the input of another gate thereby creating a circuit.
- **Simulate:** To have the circuits constructed display its output.
- **Save:** Stores the constructed circuit as a file
- **Load:** Retrieves a constructed circuit as a file
- **Share:** Saves the constructed circuit to a public database that allows it to accessed by URL.

2. General Description

2.1 Product Functions

The software should support a computerized simulation program. Our program will allow the building and editing of diagrams, allow simulation of constructed gates, will communicate with the database to save and share constructed gate diagrams. The system requires appropriate saving and reloading features. The program also handles various exceptions outlined in 3.3. The program must run on a Windows server with a version of 8.1 or later and will work on any contemporary network.

2.2 Overview

To give a short overview of the functionality that the Logic Gate app will provide, here are some basic user scenarios (note: this section will reference the Figure 3.2 from section 3.2 when referring to sections (1), (2), and (3).

Creating a new diagram after opening the program:

1. A user will access the web app and be prompted with the main interface (figure 3.2). The edit space (figure 3.2 - (3)) will be blank. This blank diagram will serve as the starting point of a new project.

2. The user will be able to drag any gates they would like from the drawer section, located on the left side of the screen (figure 3.2 - (2)) to the middle of the edit area.
3. The user will be able to connect the gates chosen in step 3 with wires, by clicking anywhere on the edit area and then dragging the mouse to where the desired connection is.
4. The user will be able to test the gate setup by hitting a “run simulation” button located on the drawer.

Creating a new diagram while there is a pre-existing diagram open:

1. The user will hit “new” from the toolbar (figure 3.2 - (1))
2. The program will prompt the user, asking if they wish to save their work. The user will select either “yes” or “no”.
3. If the user chooses to save, they will be walked through the saving process.
4. The program will load a blank canvas for the user to work with.

Share:

0. Precondition: the user has a diagram open in the main interface. If this condition is not satisfied, the exception will be handled according to the process outlined in section 3.3.
1. The user will select a button from the toolbar (figure 3.2 - (1)) that reads “share.”
2. The program will then generate a unique URL and provide it in a pop-up GUI in the drawer section of the interface (figure 3.2 - (2)).
3. The user can distribute the URL to other users.
4. A secondary user can load the diagram by selecting the “load diagram by URL” button from the toolbar. The button will load a pop-up GUI in the drawer that will allow the user to paste a URL.
5. The user pastes the URL into the pop-up GUI and the program will load the diagram from the database.

Load from desktop :

0. Precondition: user has a file of the required type saved to their computer.
   If the user does not satisfy this requirement, the exception will be handled according to the process outlined in section 3.3
1. The user loads the logic gate app in their browser. The user clicks “load from desktop” from the toolbar (figure 3.2 - (1)).
2. The program will open small GUI in the drawer (figure 3.2 - (2)) that will allow the user to browse their computer for an compatible file.
3. Once a file is selected the intermediary GUI will disappear and the file’s data will be loaded to the screen in the edit area (figure 3.2 - (3)).
4. The user will be able to edit the loaded gates.

Run simulation:

0. Precondition: the user has at least one gate object in the edit area (figure 3.2 - (3)).
   If this condition is not satisfied, the program will not do anything.

1. The user hits the “run simulation” button located on the toolbar (figure 3.2 - (1)).
2. The gates in the edit area will respond according to the way the user set up the diagram.

2.3 Users

There will be no differentiation between user type -- all possible users will have the same accessibility.

3. Requirements

3.1 Required states and modes

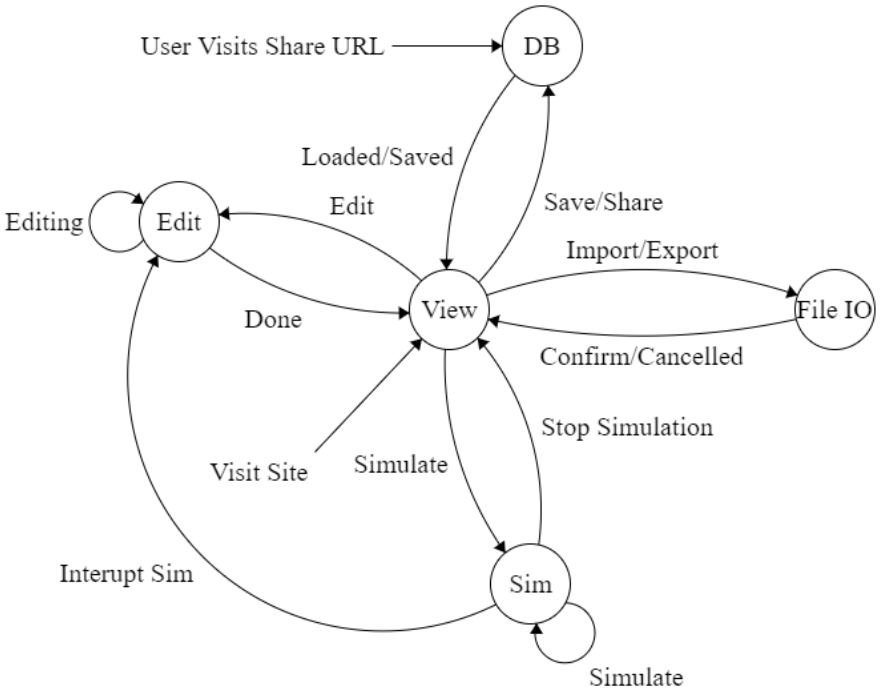


Figure 3.1

- **View:** The default, idle state of the program where no editing or simulation is taking place. When a user visits the site homepage, this is the entry state.
- **DB:** The state where server calls are made to save or load a circuit diagram from the database. This is the entry state when a user visits the site via another user’s saved url.
- **Edit:** The state where a user is editing the circuit diagram. This requires changing the diagram in memory and therefore is a separate state.

- **Sim**: When a user interacts with buttons or switches in the circuit diagram, the simulation state engages until interrupted by user edits or stopped by users.
- **File IO:** When a user wishes to import or export a circuit diagram, this state engages to display a GUI where the user may save the circuit as a JSON.

3.2 CSCI capability requirements

Functional Requirement 1 (R1) :

The user shall be provided with a simple extensible way of defining gates.

Functional Requirement 2 (R2) :

The user shall be able to access the program through all web browsers.

Functional Requirement 3 (R3) :

The user shall be able to share gate projects using a unique URL.

Functional Requirement 4 (R4) :

The user shall be able to save gate files.

Functional Requirement 5 (R5) :

The user shall be able to load existing gate files.

Functional Requirement 6 (R6) :

Allow the user to manipulate, connect, and simulate gates (including, but not limited to: AND gate, OR gate, NOT gate, XOR gate, NAND gate, NOR gate, XNOR gate, and wires)

Functional Requirement 7 (R7) :

When a user hits the “save” button, the program will open a GUI that will allow exporting of data to a local desktop.

(Optional) Functional Requirement 8 (R8) :

The user shall be able to define custom logic for gates in a customizable way.

3.3 Exception Handling

Exception handling Requirements (R9):

The program will check for and throw exceptions and/or warnings for the following events:

- The user tries to share a blank diagram

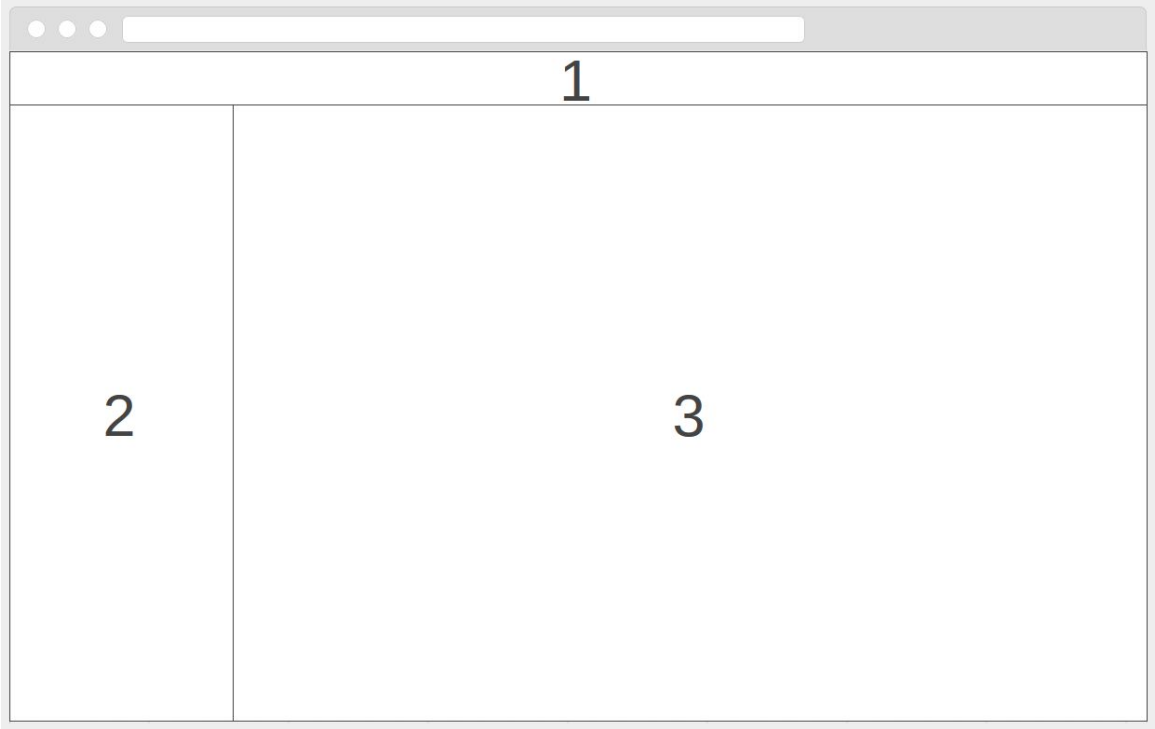
  The program will issue a warning, but not stop the user from sharing a blank diagram as long as it does not violate any file type restrictions

- The user tries to load a file of the wrong type
  The program will only allow compatible files to be selected in the pop-up GUI
- Browser Crash
  Current data will be lost, but a cached copy will be retrievable. It is recommended that the user saves his/her data on a regular interval, but an automatic cache save will occur on a regular interval on the server end. This cache will only last 24 hours, and will be lost if data has not been retrieved by then.
- PC shutdown
  See browser crash
- Server End Crash
  Upon the rare case of a server end malfunction, it is highly likely that the cached data will be lost. There is really nothing that can be implement in-code to solve this, so it is recommended for users to save their work.

3.4 CSCI external interface requirements

Interface Requirements (R10) :

The program shall use a javascript-based framework for the user interface. The main interface will preferably have a drag-and-drop feature for creating the logic gates.



Example diagram of potential user interface. (Figure 3.2)

Key:

1. The navigation bar with options for users to open.

2. The drawer with potential actions a user can take and gates a user can utilize when drawing.
3. The gate display area. Acts as an editor, allowing a user to draw and interact with gates.

3.5 External Hardware Interfaces

There are no external hardware interfaces.

3.6 CSCI internal interface requirements

The user commands available to the user are listed below in the following (table 3.6):

Case #	User Commands	Description
1	Open Interface	Displays the interface when the program is opened. Creates a GUI for the user to interact with.
2	Add Gate	Takes in a Gate class variable and displays it on the screen
3	Remove Gate	Removes a gate from the screen and displays and disconnects it from the circuit
4	Edit Gate	Changes the gates type.
5	Abstract Gate	Allows users to create their own personal gates.
6	Share	Save the circuit to the database and allows it to be accessed by a URL.
7	Save Gates	Save the circuit to a file.
8	Load Gates	Load the circuit to the display from a file.
9	Clear	Remove all circuits from the display.
10	Draw Wire	Connects gates to create a circuit.
11	Copy Gate	Stores gates gates to be pasted.
12	Paste Gate	Places the stored gates on the display.
13	Run Simulation	Runs the simulation of the circuit and displays the output.
14	Zoom In	Moves the display closer on to the circuits.
15	Zoom out	Moves the display further away from the circuits.

Table 3.6

3.7 CSCI internal data requirements

Internal Data Requirement (R11)

The program will utilize a database for storing the components of a user's gate.

3.8 Security and privacy requirements

Security Requirements (R12) :

The database for storing projects will be obscured from the user, making direct edits to it will be impossible.

3.9 CSCI environment requirements

Environment Requirement (R13) :

The computer must be at least Windows 8.1/10 to guarantee functionality.

3.10 Computer resource requirements

3.10.1 Computer software requirements

Software Requirements (R14):

Since the program will be a web application, the program will require a network connection to be used. The program will be able to run on any contemporary, updated browser (i.e: Firefox 62.0 and up, Chrome 70.0 and up).

3.10.2 Computer hardware requirements

Hardware Requirements (R15) : The program will be able to run on a computer that can satisfy the software requirements outlined in section 3.10.1.

3.10.3 Computer communications requirements

Communications Requirement (R16) : Computers will communicate through shared access to a database.

3.11 Software quality factors

Software Quality Requirements (R17):

The software shall be held up to the following quality factors:

- The software at minimum be able to simulate any 4 gates in any combination of circuits.
- The software shall be able to simulate any circuit in 3 seconds.
- The load times for a circuit file will be under 5 seconds.
- Downloads times shall scale to the users internet connection speed.

3.12 Design and implementation constraints

Constraints Requirement (R18) :

The development team will be limited to using development software (IDEs, databases, servers, etc) that is obtainable through UMBC.

3.13 Training-related requirements

Training-related Requirement (R19) :

There shall be a help screen explaining the functionality available to the users.

3.14 Precedence and criticality of requirements

The following requirements are listed in order from the highest priority to the lowest priority. The requirements marked with an asterisk (*) symbol are absolutely needed, and therefore, are of equal weight:

*.User interface
*.Functioning logic gates
1. Ability to share gate diagrams
2.Import/Export files from Logism
3.Save/Load Files
4.User-defined logic gates

4. Qualification provisions

The following (table 4.1) will display each of the requirements listed in section 3 of this document. It will provide a qualification method for each of the requirements to see how each of them will be tested. A description will follow suit explaining in further detail how each specific requirement will be tested.

Requirements	Qualification Method	Description
R1	Demonstration	The user will be able to test if the gates are simulating if the software runs and displays the gate to the screen with their corresponding output.
R2	Analysis	The website will be tested on multiple web browsers to assess the programs functionality.
R3	Demonstration	The website will need to display the a usable URL otherwise this requirement has not been met.
R4	Demonstration	A file will need to be downloaded to meet this requirement.
R5	Demonstration	A fully functioning circuit will need to be loaded onto the screen to fulfill this requirement.
R6	Test	All simple gates will be tested to see if they are fully functioning
R7	Demonstration	A pop up window will need to be displayed that allows the functions available for this requirement to be met.
R8	Test	Users and beta testing will root out potential configurations that prove to cause trouble for the software
R9	Test	The software will be stressed tested to expose these errors during development.
R10	Demonstration/ Inspection	The software will demonstrate drag and drop functionality to approve this requirement. The code will also be inspected to be complicit with a javascript-based framework
R11	Demonstration	The software will be tested to return stored values and will pass this requirement if succeeds in doing so.
R12	Test	Through users beta testing this requirement will be met if no user can access the database.
R13	Test	The program must be tested on all window servers after version 8.1 to ensure this functionality.
R14	Test	The program must be tested on all possible networks servers.

R15	Test	The program will be able to run on software highlighted in section 3.3.2. This is the platform that the program will be tested on.
R16	Demonstrations	The program will need to test the ability for one process to share a logic gate circuit to the database and for another process to load it from the database.
R17	Test	The program has to work for a specified number of gates, connections….etc. The program will be tested for every possible combination that is less than or equal to the number specified.
R18	Inspection	The program is restricted to and all inclusive of all UMBC tools. (ie - databases, servers…) The program will be tested with all UMBC tools.
R19	Demonstration	The user will be able to visually view the functionalities that the program has to offer.

Table 4.1

5. Requirements traceability

The traceability factors will refer to requirements based on the particular requirement's unique identifier (for example: Functional Requirement 1 will be referred to as R1 in this section).

- R1 will stem from the customer who will benefit from this requirement. This requirement will not have a tremendous effect on the data, technology or test cases.
- R2 is a very important functional requirement from the customer. The technology needs to support all browsers and data should always be loadable and viewable. This requirement needs to be tested thoroughly.
- R3 is a requirement from the customer. This technology needs to be in place in order to make this requirement sharable and really usable by multiple parties. This needs to work for all data and should be tested in depth.
- R4 relates to data. It is a requirement from the customer that needs to be able to save all data that was created. Besides for saving data, the user needs to be able to reload the data.
- R6 is a functional requirement that will be used by the customer. The data and technology needs to be complex enough to handle this requirement and needs to be tested in detail.
- R7 allows the customer to use this program in a very practical way. It should work for all data and needs to contain the technology to be this user-friendly. The customer would like this requirement to have the technology and data to make this program customizable and user friendly.

The following table describes the goal that each requirement fulfills by implementation:

	User-friendliness	Shareability	Functionality
R1	X		X
R2	X		X
R3	X	X	X
R4		X	X
R5			X
R6			X
R7	X	X	X
R8			X

Table 5.1