Software Requirements Specification (SRS)

X-Browser Editor

Team: 2

Authors: Tarpan Patel, Joshua Michaud, Ben Viscosi, Marcello Barbieri,

Bill Figliolini

Customer: Front-End Developers

Instructor: Dr. James Daly

# Introduction

X-Browser Editor is a web development tool created to streamline web development testing. This initial section will describe this document’s purpose and scope, define the definitions used throughout the rest of the document, and provide an outline of the following sections.

## Purpose

This document is the definitive set of requirements for the X-Browser Editor and will help clarify these requirements between the developers, the customer, and the instructor. This will serve to clarify the decisions and assumptions made during development, as well as the instructed set of features. The intended audience is the development team of the editor, the customers, as well as any users of the Editor who want to understand why the current design was chosen.

## Scope

X-Browser Editor will attempt to streamline the process of website development. Currently with the differing HTML, CSS, and JavaScript implementations between browsers, testing if one’s current website implementation functions and is consistent across platforms can be time consuming. By directly connecting the browsers to the editor, X-Browser Editor seeks to simplify this process and provide real-time feedback to the developer.

## Definitions, Acronyms, and Abbreviations

Editor: The X-Browser Editor

User: The website developer using the Editor

HTML: HyperText Markup Language

CSS: Cascading Style Sheets

JS: JavaScript

Website: the interpreted result of HTML, CSS, and JavaScript code

Browser: The Web browser which the Website will be run on to test its compatibility

Files: Any set of HTML, CSS, or JS files

## Organization

The remainder of the SRS is organized into the following 6 sections:

**Section 2:** An overview of the Editor as well as an overview of the Editor’s functions, user expectations, constraints, and dependencies

**Section 3**: A formal list of the software’s requirements

**Section 4**: Provides the expected use cases of the Editor, its classes and how they relate, and finally the Editor’s sequence diagrams

**Section 5**: This will provide an instruction manual for the Editor’s prototype, as well as a link to the prototype

**Section 6**: A list of all referenced works for the document

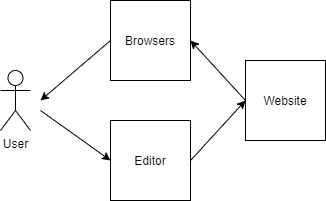
**Section 7**: Contact information for the instructor for questions regarding the Editor

# Overall Description

This section is intended to cover all the assumptions surrounding the user and the underlying design of the Editor. It will cover the perspective the developers have taken for the development of the Editor, its intended functions, and the expected abilities of the user. It then moves on to discuss the Editor itself, describing its constraints, the assumptions and dependencies made during development, and the features outside the Editor’s scope.

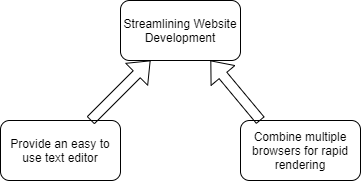
# Product Perspective

The Editor serves as an independent development environment for HTML, CSS, and JS. By integrating browsers directly into the development environment, it streamlines testing and allow the user to quickly see the effects of their changes and if they are consistent across the range of browsers which the user has selected for testing. This allows the user to rapidly correct any code which results in inconsistencies. This will save time over sending the website to a third party to verify if it correctly functions on multiple browsers. To maintain easy usability of the code, we have opted to design this editor as a web app, allowing it to be portable across multiple devices.



# Product Functions

The Editor will help streamline website development by allowing the user to see the effects of their changes to their website on multiple browsers in real-time. This will reduce overhead in development, and reduce time being wasted manually verifying if a new HTML, CSS, or JS feature has been adopted across all bowsers.



# User Characteristics

The user is expected to be a web developer with experience in HTML, CSS, or JS. They will be expected to have some knowledge of how to navigate a common IDE, as well as a browser.

# Constraints

The user will be accessing the Editor through a browser, so usage of new and possibly unsupported HTML, CSS, or JS features need to be avoided to maintain general usability. The user must be using a computer that meets the minimum specifications of at least one of the browsers supported at the final release.

# Assumptions and Dependencies

* The Editor will be based on a Web App, requiring an active internet connection and a working browser.
* The Editor will require permissions to save the created files to the user’s machine.
* The Editor will require the user to upload any already existing files for the Editor to display.
* The user is expected to understand at least one of HTML, CSS, or JS for the purposes of using the Editor.

# Apportioning of Requirements

The initial release of the Editor will be limited to displaying the resulting code, leaving the actual validation to the user. In a future update, the Editor may attempt to include automated testing of this, which will further streamline development.

# Specific Requirements

1. Code Editor
   1. The editor will accept user inputted HTML, CSS, and JS Code to make the Website
   2. The editor will prepare the code for rendering by the browsers
   3. The editor will allow the user to save any files created in the Editor.
      1. These files can then be downloaded to the user’s computer.
   4. The editor will allow the user to upload any Files that already exist.
2. Virtualized Browsers
   1. The editor will accept a list of browsers to test
   2. The editor will have a means of displaying each browser
   3. The editor will be able to connect and render multiple browsers simultaneously
   4. The editor will have real-time rendering of HTML file and associated CSS and JS files

# Modeling Requirements

## 4.1 Use Case Diagram

The XBrowser use case diagram is single user application which focuses on meeting user’s expectations as they interact on the website. The use case diagram starts by having the user uploading files and adding browsers to test their work. As the user saves their work, many features are executed to render the layout on the selected browsers corresponding to the contents on the files. Please see below for use case diagram and definitions below:

Diagram

Description automatically generated

|  |  |
| --- | --- |
| Use Case Name: | Upload Code |
| Actors: | User |
| Description: | By clicking on the upload button, the user can upload files. |
| Type: | Primary |
| Includes: | None |
| Extends: | None |
| Cross-refs: | Requirement 1.1, 2.4 |
| Uses cases: | At any point in XBrowser, the user can upload existing files. |

|  |  |
| --- | --- |
| Use Case Name: | Open Virtual Browsers |
| Actors: | User |
| Description: | The user can choose the browsers given browser (i.e. Chrome etc.) and its version to test their code |
| Type: | Primary |
| Includes: | Web App Layout |
| Extends: | None |
| Cross-refs: | Requirement 2 |
| Uses cases: | The user can add button by clicking on ‘+’ button and provide required information to add browser |

|  |  |
| --- | --- |
| Use Case Name: | Edit Code |
| Actors: | User |
| Description: | The user can edit their code, once they upload files or create new file. |
| Type: | Primary |
| Includes: | None |
| Extends: | Save Code |
| Cross-refs: | Requirement 1.1 |
| Uses cases: | The code files must be html, CSS, or JS. |

|  |  |
| --- | --- |
| Use Case Name: | Save Code |
| Actors: | User |
| Description: | The user can save files from last save point while editing |
| Type: | Secondary |
| Includes: | Update Virtual Browsers, Styling, PostCSS, Linting |
| Extends: | None |
| Cross-refs: | Requirement 1.3 |
| Uses cases: | The files must be saved to update browsers and other features to execute |

|  |  |
| --- | --- |
| Use Case Name: | Update Virtual Browsers |
| Actors: | XBrowser |
| Description: | This function renders the layout in each testing virtual browsers. |
| Type: | Secondary |
| Includes: | Web App Layout |
| Extends: | None |
| Cross-refs: | Requirement 2.3, 2.4 |
| Uses cases: | Testing Browser(s) must be added and the files must be saved for the layout for the virtual browsers to be updated |

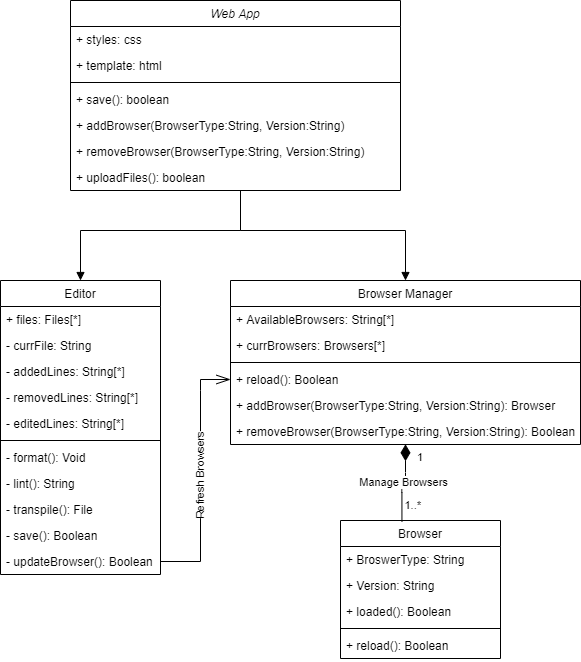
|  |  |
| --- | --- |
| Use Case Name: | PostCSS |
| Actors: | XBrowser |
| Description: | The postCSS function focuses on styling browsers as it translates latest CSS syntax for older browses and points out errors. |
| Type: | Secondary |
| Includes: | None |
| Extends: | None |
| Cross-refs: | Requirement 1.1 |
| Uses cases: | The user must save the edited file(s) after for PostCSS to apply CSS syntax to different browsers. |

|  |  |
| --- | --- |
| Use Case Name: | Linting |
| Actors: | XBrowser |
| Description: | Linting function assists users on debugging the code before the virtual browsers are updated |
| Type: | Secondary |
| Includes: | None |
| Extends: | None |
| Cross-refs: | Requirement 1.1 |
| Uses cases: | The user must save the edited file(s). |

|  |  |
| --- | --- |
| Use Case Name: | Styling |
| Actors: | XBrowser |
| Description: | Styling function applies small different properties (i.e. color text) to code for readability |
| Type: | Secondary |
| Includes: | None |
| Extends: | None |
| Cross-refs: | Requirement 1.1 |
| Uses cases: | The user must save the edited file(s). |

## 4.2 Class Diagram

The class diagram below displays the necessary classes and its operations that XBrowser website is built from. Please see the diagram and its data dictionary below.



|  |  |  |
| --- | --- | --- |
| **Element Name** | | **Description** |
| Web App | | The Web App class provides basic operations to set up an XBrowser working environment. |
| **Attributes** |  |  |
|  | Styles: CSS | CSS file required for the styling the website |
|  | Template: html | Html file required for the layout of the website |
| **Operations** |  |  |
|  | uploadFiles(): Boolean | Member of class Web App executes when the user selects the option to upload existing files. |
|  | addBrowser(BrowserType:String, Version: String) | This function interacts with Browser Manager class, allowing the user to add a browser given type (i.e., Chrome) and its version. |
|  | removeBrowser( BrowserType:String, Version: String) | Member of class Web App, interacts with Browser Manager class, allowing to remove browser from the view by specifying its type and version. |
|  | save() | The function allows users to save their edited files. |
| **Relationships** | The Web App class has associated Editor and Browser Manager classes and which creates the instances of the two classes. | |

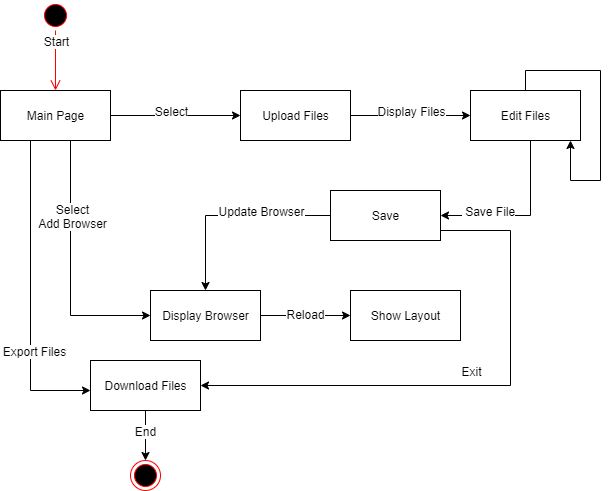
|  |  |  |  |
| --- | --- | --- | --- |
| **Element Name** | | | **Description** |
| Editor | | | The Editor class assists Users on editing files and keeping files error free. |
| **Attributes** |  |  | |
|  | files: Files[\*] | A container to hold file(s) that are uploaded | |
|  | currFile: String | Contains the name of the file that is currently being edited | |
|  | addedLines: String[\*] | Contains new lines of code added to a file(s) | |
|  | removeLines : String[\*] | Store lines of code that are removed from file(s) for reference | |
|  | editedLines : String[\*] | Contains edited lines of codes from file(s) | |
| **Operations** |  |  | |
|  | format(): void | Structures lines of code for readability, understanding and debugging. | |
|  | lint(): String | Assists users debug any errors in files | |
|  | transpile(): File | Generates equivalent source to source code in other languages | |
|  | save(): Boolean | Allows users to save edited files | |
|  | updateBrowser(): Boolean | Interacts with Browser Manager class to have the contents updated in the browsers. | |
| **Relationships** | Editor provides the user with different features such as transfile or linting to make the users code more understandable, readable and error free | | |

|  |  |  |
| --- | --- | --- |
| **Element Name** | | **Description** |
| Browser Manager | | The Browser Manager class fulfils users requests on getting browsers for testing |
| **Attributes** |  |  |
|  | AvailableBrowsers: String[\*] | Provides user all available testing virtual browsers for them to test |
|  | currBrowsers: Browsers[\*] | Contains the virtual browsers the user is currently testing the content on |
| **Operations** |  |  |
|  | reload(): Boolean | The operation that refreshes the layout on the browser(s) corresponding to edited files. |
|  | addBrowser(BrowserType:String, Version: String): Browser | Interacting with Browser class, this method adds a virtual browser for testing, given type (i.e., Firefox) and its version |
|  | removeBrowser( BrowserType:String, Version: String): Boolean | Interacting with Browser class, the method removes selected browser from currBrowsers array given its version and type (i.e., Chrome). |
| **Relationships** | The Browser has an association relationship with Web App class and has composition relationship with Browser class. The Browser Manager updates the browser per Users interaction in Editor class and can manage 1 to many objects of Browser class | |

|  |  |  |
| --- | --- | --- |
| **Element Name** | | **Description** |
| Browser | | The Browser class is managed by the Browser manager class |
| **Attributes** |  |  |
|  | browser: String | Contains the name of browser type (i.e. Safari) the user is currently using |
|  | version: String | Contains the version of the browser the user is currently using |
|  | Loaded: Boolean | Confirms the requests of browser manager has being fulfilled |
| **Operations** |  |  |
|  | reload(): Boolean | Updates the layout on all the virtual browsers that user has chosen to test their content on |
| **Relationships** | The Browser class has a composition relationship of the Browser Manager class. This class assist the Browser manager. | |

## 4.3 State Diagram

The state diagram demonstrates each stage for Users on how to navigate through XBrowser to set their working environment.



## 4.4 Sequence Diagram

The sequence diagram below shows the operations that occur when the user interacts with XBrowser. The normal flow of XBrowser operation starts off with the user selecting to uploading files. When the files are uploaded, the next step is to add virtual browsers for testing. When those steps are complete, the user can start editing files. The key component of this operation is Save. The save function executes internal features, followed by reloading the layout each virtual browser and results are displayed. The user can export files and save files once they are done editing.

Diagram

Description automatically generated

# Prototype

The present prototype demonstrates the basic functionality of the editor to give the user a clear view of its intended use. At present, it allows the user to upload HTML files, navigate the editor’s UI, and write or edit HTML files. It’s ability to support multiple browsers has yet to be implemented, but the user can add and remove mock virtualized browsers to see where they would reside on the interface and how they would be arranged. In addition, JS and CSS files are not presently supported, but the editor can handle inline JS and CSS.

# How to Run Prototype

The prototype requires a GitHub account and Yarn installed.

Prototype is accessible online at https://dazzling-tesla-17ff26.netlify.app/

To run your own instance:

1. Install node.js version 10.0 or greater

2. Install Yarn tool

3. Clone XBrowser repository from: https://github.com/marcebdev/XBrowser

4. Open a terminal in the repo directory

5. yarn install

6. yarn generate

7. yarn start, this will be you to the webpage

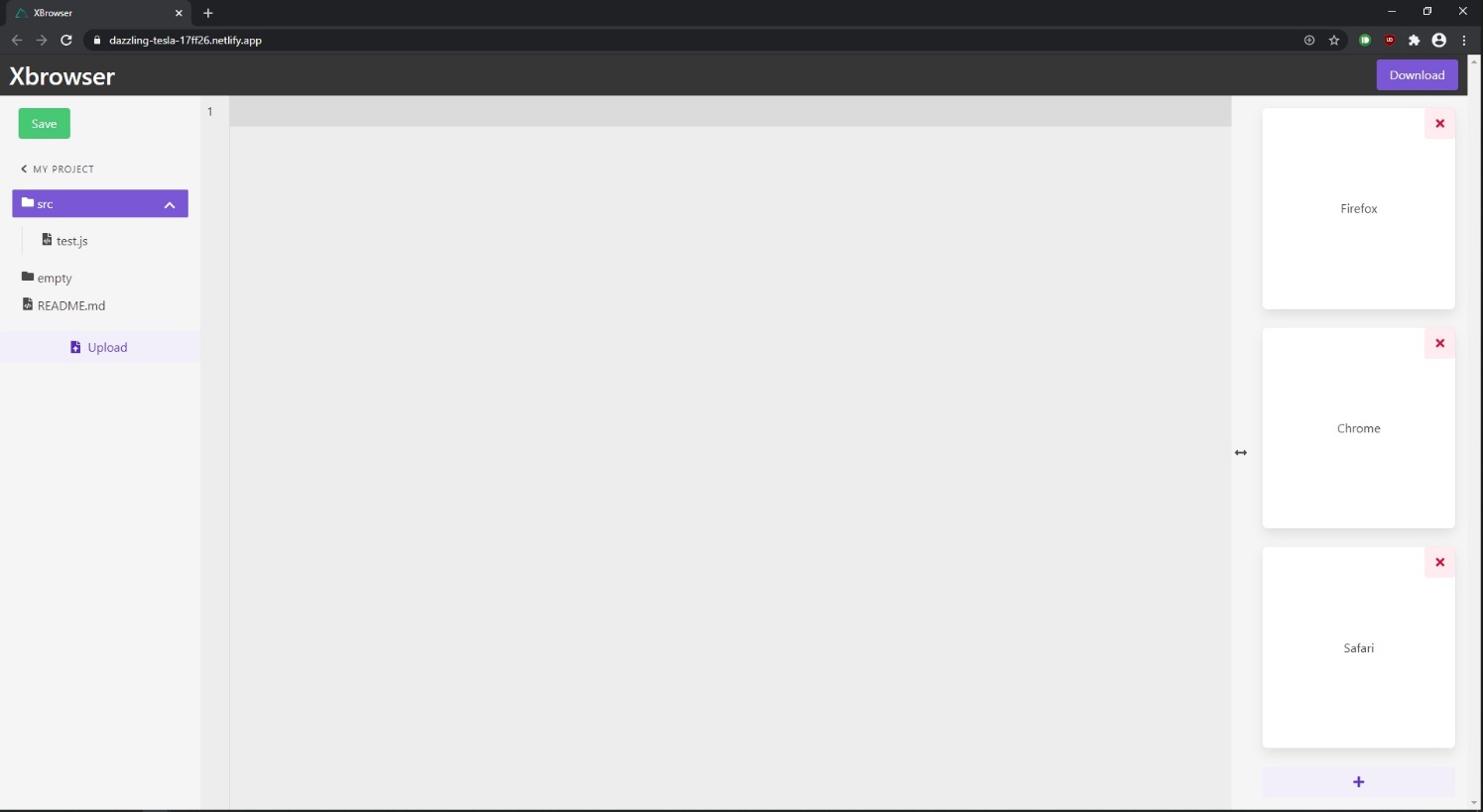
8. Go to: <http://localhost:3000/>

Helpful Link installing Yarn: [https://classic.yarnpkg.com/en/docs/install/#debian-stable](https://classic.yarnpkg.com/en/docs/install/" \l "debian-stable)

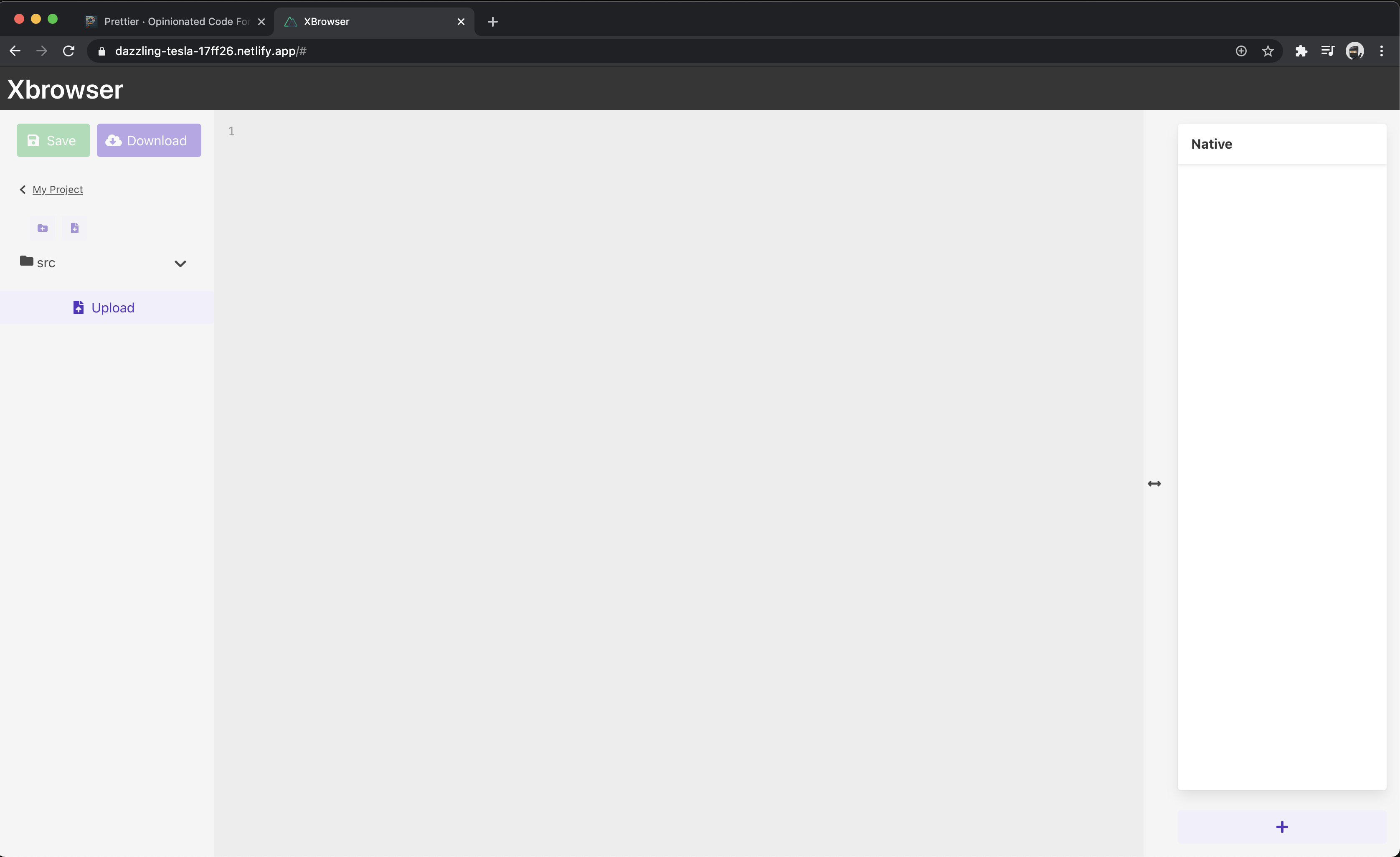
# Sample Scenarios

XBrowser is not a complicated application to use. The user starts on the main page, where they upload files. Once the files are uploaded, the user can select what browsers to test their websites on by providing version and browser type (i.e., Firefox, Chrome etc.). The user can now start editing their code. To view the updated layout on the virtual browsers, the user will have to save the edited files and content in virtual browsers will be refreshed. The user has the option to download files when they are done editing and then exit or start editing other files.

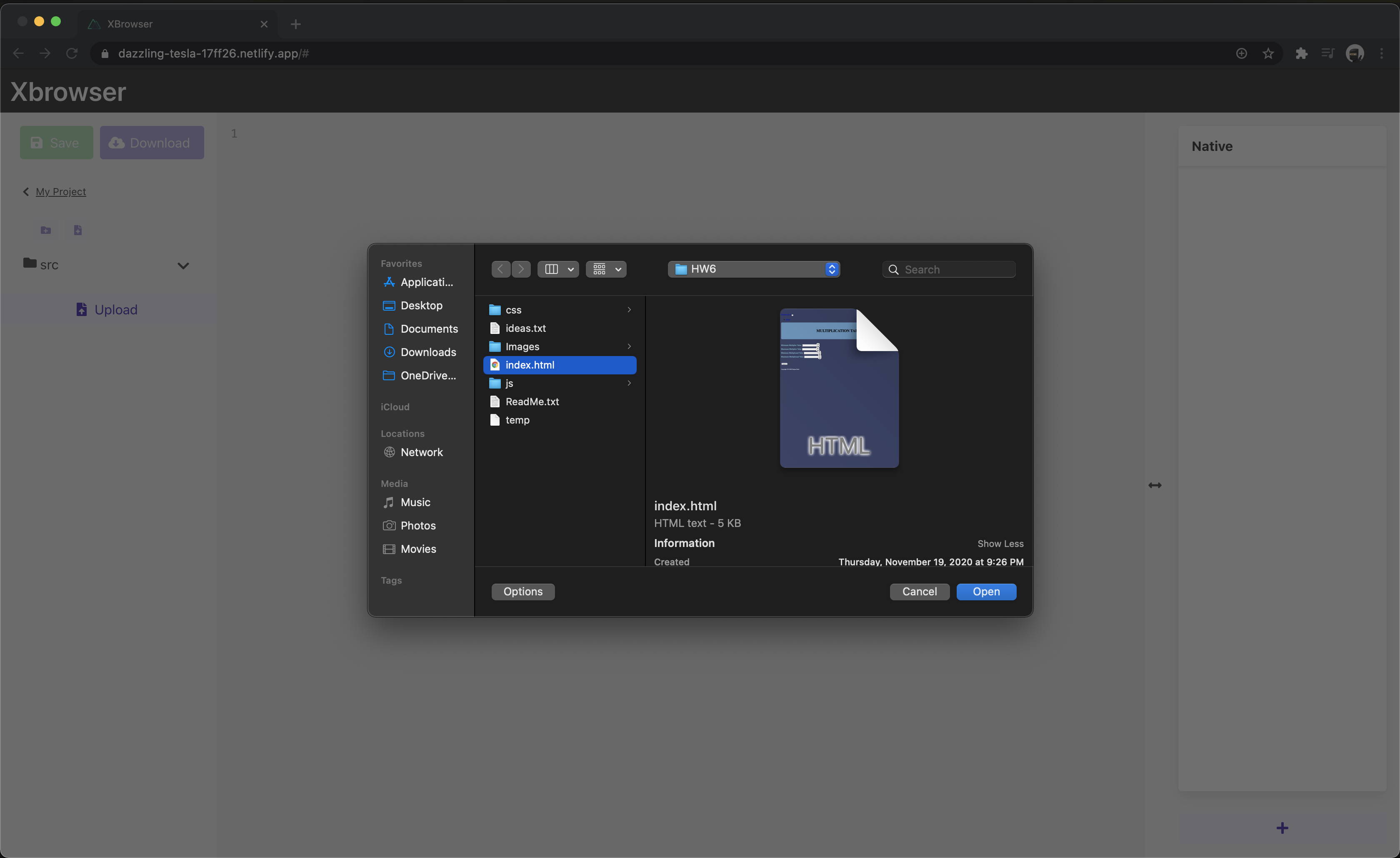
**Prototype V1: Basic Layout**



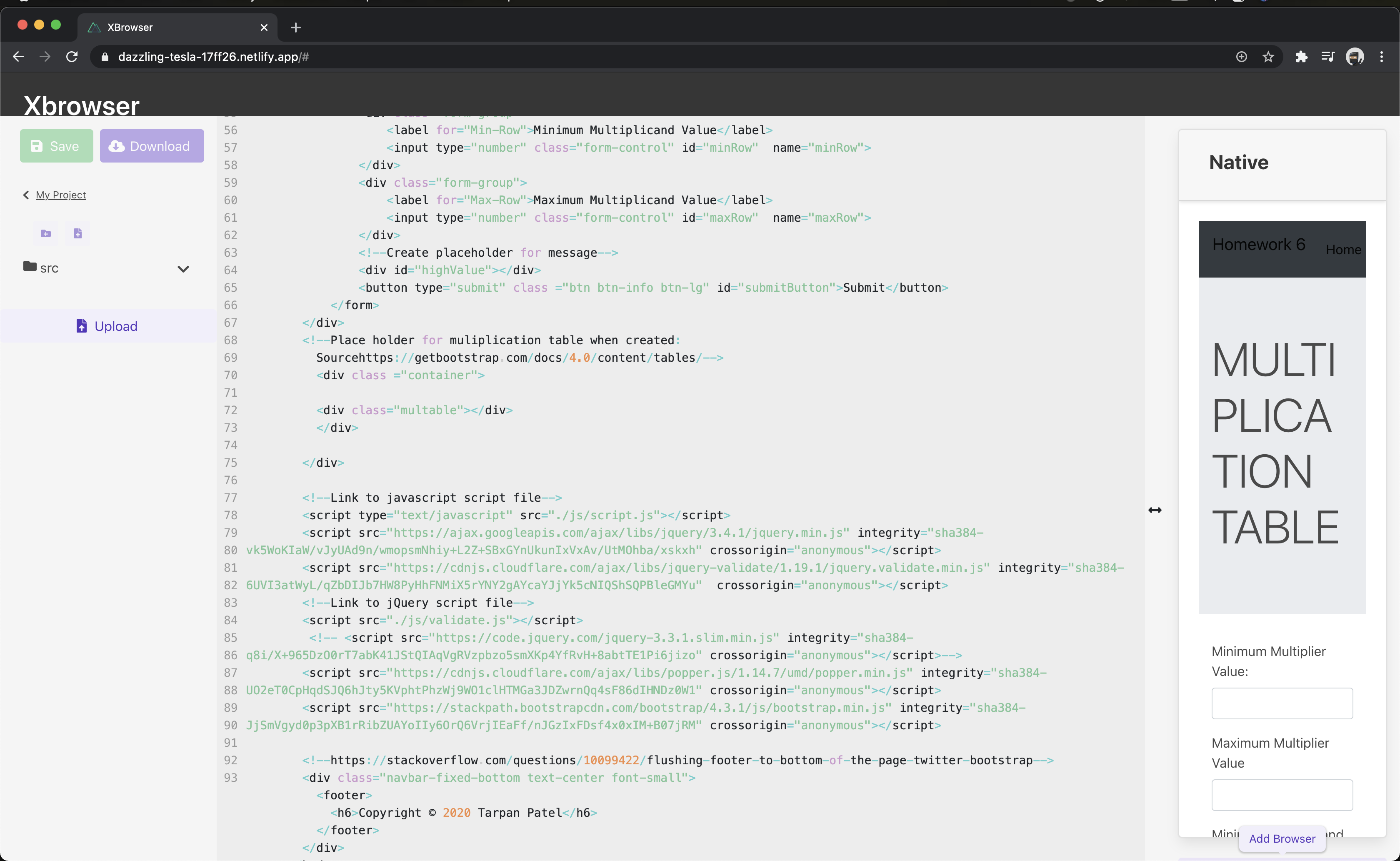
**Prototype V2: Updated Layout**



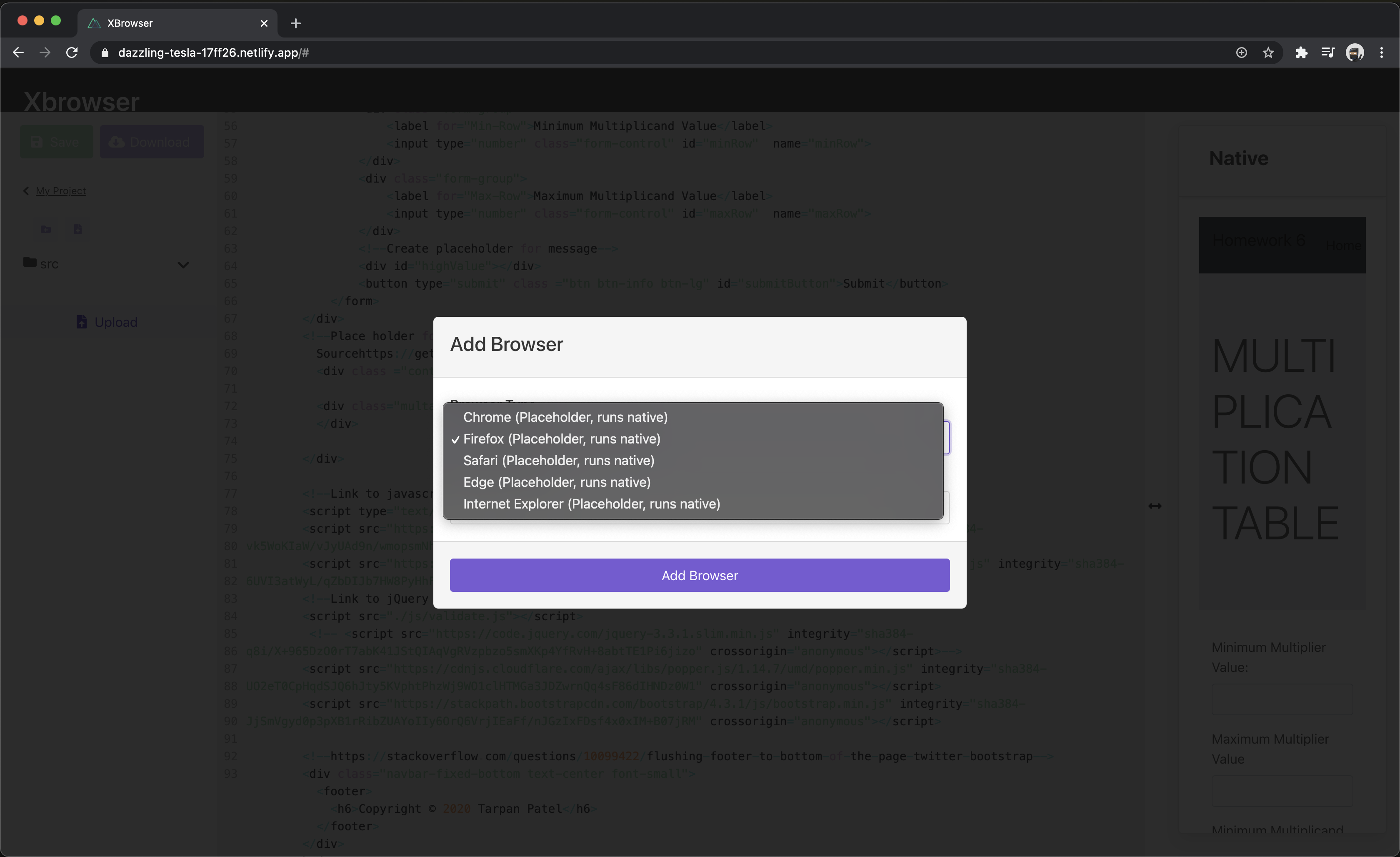
This shows the user uploading an html file into the Editor



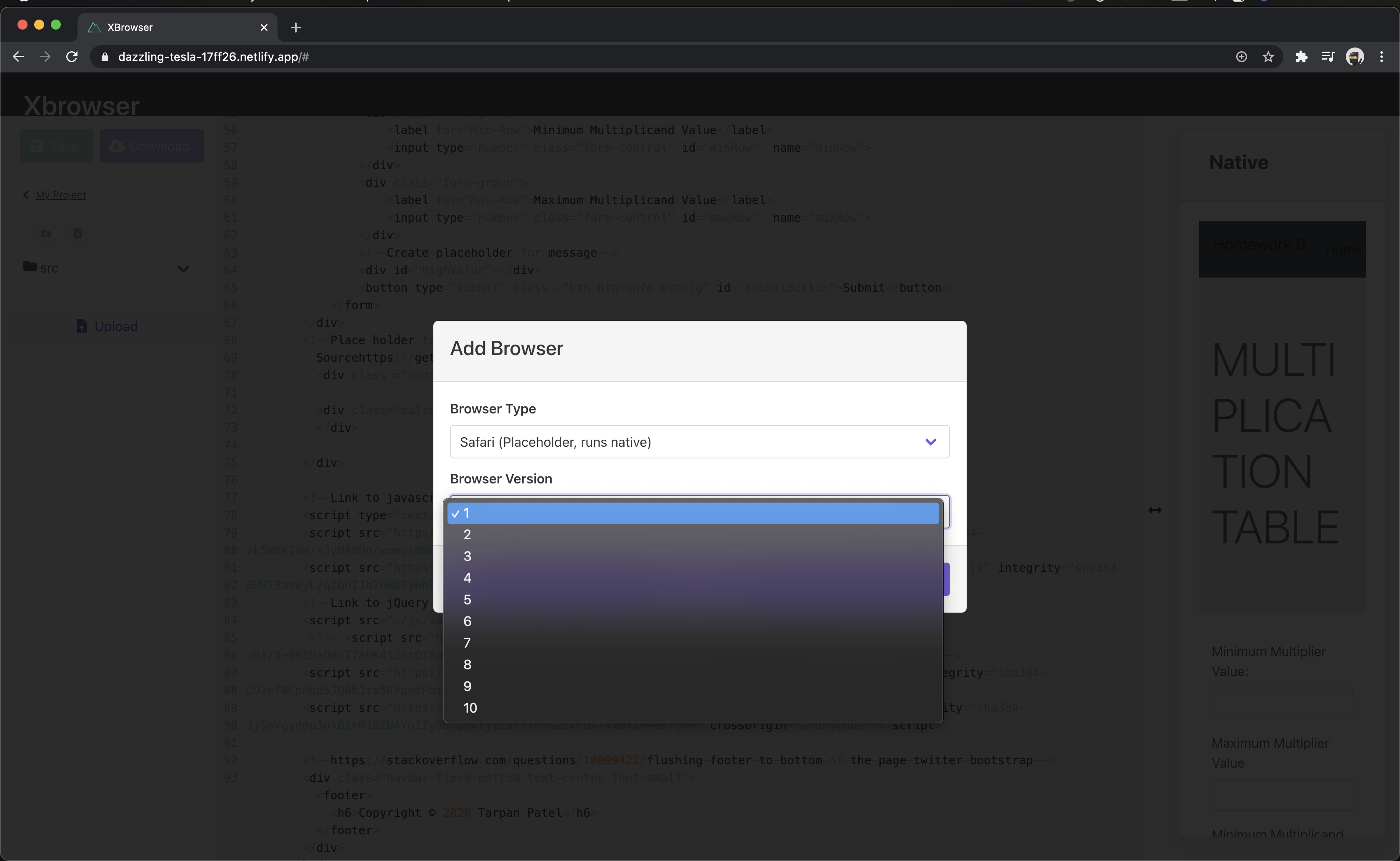
This one shows the file opened and displayed in the Editor (Note: the add browser button is below the Native browser)



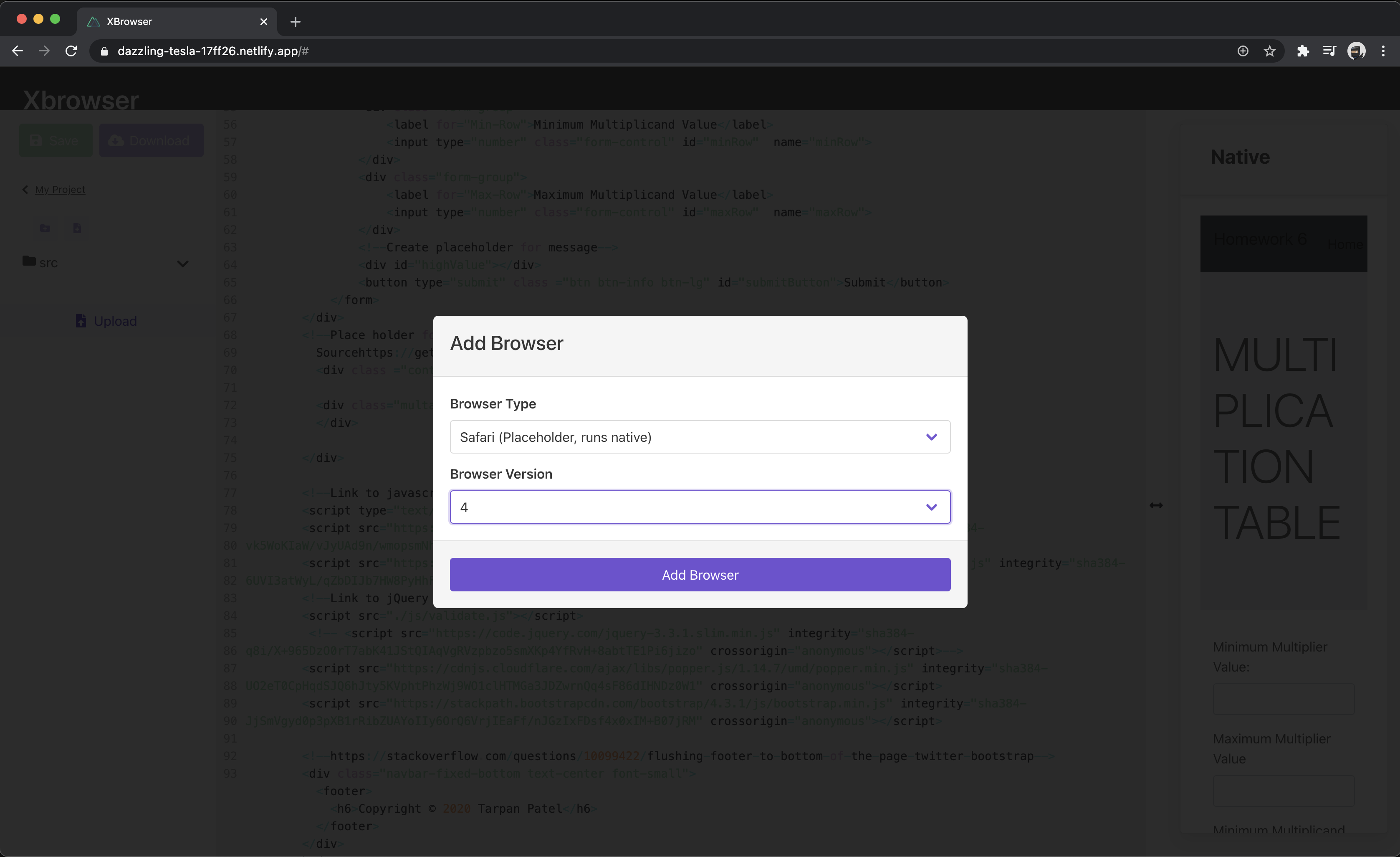
This one shows the user selecting a new browser to create

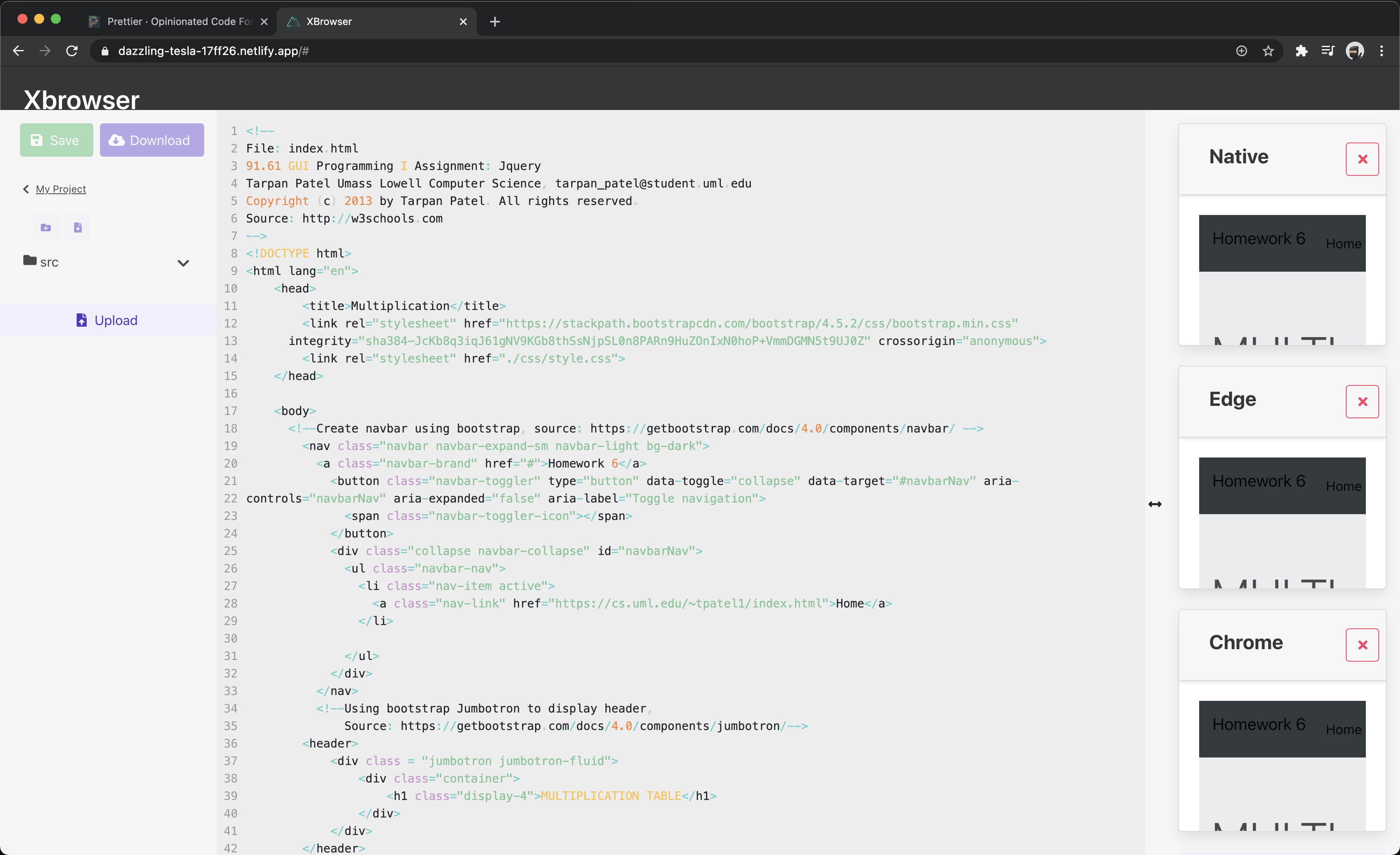


This shows the user selecting a version type for the selected browser



This is the basic layout of when the user presses the add browser button



This one displays the users uploaded html code on multiple browsers

# References

[1] Project Website: <https://bviscosi.github.io/XBrowser/>

[2] XBrowser GitHub Repository: <https://github.com/marcebdev/XBrowser>

[3] Nuxt JS Framework: <https://nuxtjs.org/docs/2.x/features/rendering-modes>

[4] Vue JS Framework: <https://vuejs.org/v2/guide/>

# Point of Contact

For further information regarding this document and project, please contact **Prof. Daly** at University of Massachusetts Lowell (james\_daly at uml.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.