Loom: One Web Development Tool for Everyone

Kyle Cutler
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14623
kjc9861@cs.rit.edu

***Abstract*—Loom is a desktop application designed to bridge the gap between beginner and expert web development tools. Existing tools in the domain provide either easy-to-use interfaces intended for novice users, sacrificing expressiveness, or advanced interfaces which are intended for developers who already know web technologies, but are inaccessible by ordinary users. Loom is a first-of-its-kind tool that provides novice users the full power of web technologies such as HTML and CSS without the need to write code. In addition, Loom aims to impart the basics of these technologies to users as they work with the tool, teaching valuable concepts that can be carried over to other tools.**

I. Introduction

There are many existing web-development tools intended for a wide range of users. Some of these tools are targeted at novice users, and provide very easy-to-use interfaces with high-level features such as drag-and-drop and WYSIWYG editing. Other tools are intended for developers, and provide textual interfaces for working with code. Others still are intended for designers and may not support web technologies at all. The separation of these tools and the incompatibilities between them lead to several problems:

- Coordination between these groups is difficult. If a novice user using a tool such as Wix[1] or Squarespace[2] decides they need a developer to continue their work, existing work will likely be wasted as the developer chooses to use a different tool.
- The work done by designers in tools such as Figma[3], Sketch[4] or InVision[5] has to be mirrored by developers, wasting effort.
- Novice users are learning how to use a specific tool – not the underlying technologies. Experience with one tool often does not carry over to others.

Some tools try to target more than one type of user. For example, WordPress[6] has a code editor, enabling developers to directly edit HTML code. Blocs[7], a desktop application, provides designer-friendly tools, WYSIWYG editing and templates for beginners, and code editing for developers. But one problem with these tools still remains: novice users may never learn how to use ”advanced” features such as code editing without doing their own research into web development.

II. Approach

Loom takes a different approach from most existing tools. It provides one common interface which is closely related to the underlying web technologies. This way, every user has the potential to use the full power of these technologies. The remaining challenge is making the interface intuitive to beginners, and familiar to developers and designers. To accomplish this, Loom uses a few strategies:

- **Provide helpful, simple abstractions over existing technologies:** working directly with web technologies can be frustrating, even for developers. Loom aims to provide a handful of simple, but useful, abstractions over these technologies to make working with them easier.
- **Provide a user interface that ”scales” with the user:** The interface should be easy for novices to pick up, but provide the full power for those who want it. For example, WYSIWYG editing and graphical components such as color pickers should make styling easy for beginners, but developers should be able to enter textual values for these properties as well.

Loom has the following motivating goals:

A. Powerful

Loom strives to give users the full power of the underlying technologies. For example, the user has the ability to use arbitrary HTML tags and attributes, use the full range of CSS selectors, or other advanced features. These features may not all necessarily have beginner-friendly interfaces, but should be accessible and intuitive to developers.

B. Unified

Loom’s tagline is ”one web development tool for everyone”. But everyone has different needs. For example, designers are most interested in fast prototyping and easy styling tools, but developers will be concerned with the underlying workings of the site and modifying those in detail, and novice users will be focused on just learning the basics. In addition, a user may find themselves taking on multiple different roles – for example, a developer who is also learning web technologies – in which case, they should have total control over the page, but also guidance as to how to change every aspect of the content. So Loom must accommodate each user’s needs, while remaining intuitive to those with different needs. This is the motivation behind Loom’s ”scaling” user interface.

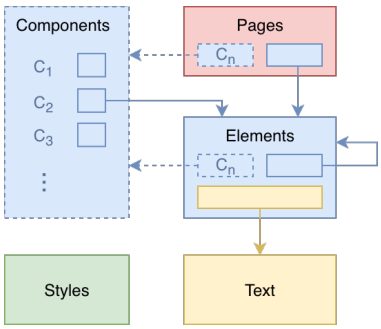

Fig. 1. Relationships between each kind of object within Loom.

C. Educational

For users who are not familiar with web technologies, Loom aims to be a tool they can use to learn these technologies. This includes both technical and non-technical users. For example, Loom should provide help with HTML and CSS whenever possible, and provide context clues and familiar interfaces to help the user make associations between the actions they perform and the resulting content changes.

D. Simple

Loom should be easy for anyone to learn to use. This includes novice users all the way to experienced developers. We do however assume a basic working knowledge of how to use a computer – for example, users with basic proficiency in using a web browser.

III. Design

To accomplish its goals, Loom must have a logical design that allows users to create and edit every aspect of a website. There are three top-level types of objects the user can modify with Loom: Pages, Components, and Styles. Figure 1 shows the relationships between the different kinds of objects.

A. Pages

Loom aims to give users the ability to create entire static websites. The ability to create multiple pages is essential to this goal and is therefore one of the fundamental features in Loom. The user has the ability to create and name many different pages within a project. Each page represents a separate HTML page within the site.

Following standard HTML practices, in Loom all pages contain two elements nested within a root `html` element: a `head` and a `body`. Loom gives users the ability to modify these elements and their children by setting attributes, and creating and deleting elements and text content.

Elements are standard HTML elements – consisting of a tag, zero or more attributes, and zero or more child elements or text. Therefore page contents form a tree-like structure just like standard HTML.

B. Components

In standard HTML, every Element belongs to just one parent. Thus there is no concept of ”reusable” Elements in pure HTML – if you want multiple identical elements, you have to repeat yourself. This quickly becomes a pain to manage, especially across many pages. Loom includes the ability to create reusable elements by the name of ”Components”. Components are essentially just named Elements which can be reused across pages.

One common use case for components is to implement a site header and footer. Most websites reuse a header and footer across all pages. These often include things like navigation, a site logo, and contact information. In plain HTML, there is no way to reuse these across several pages without duplicating their entire HTML contents. But with Loom, the user can create components for the ”header” and ”footer”, and then include them in several pages. Any changes to these components will then immediately be reflected across all pages using them. In a way, there are two ”halves” to components: the definition, which associates an element with a given name, and references, which look up the component of a given name and insert its element in their place.

In Loom, the user has the ability to create new Components, update their contents and properties (just like any Element), and reference them from another Page or Component.

C. Styles

In general, every HTML Page can ”link” to one or more CSS Stylesheets, whose styles will then be applied to that Page. However, this requires management of every page, ensuring that the correct Stylesheets are included for any elements on that page. In combination with components, it can become difficult to track which Stylesheets must be included on each page.

Loom simplifies this by using one shared Stylesheet across the entire project / website. This Stylesheet will be used by every page, and contains the styles for every Component. This greatly simplifies the management needed to ensure the correct styles get used on each page, but it also means the Stylesheet can become large and complex. The UI compensates for this by providing mechanisms to help the user easily find the style rules they are looking for, such as only showing styles applicable to a selected element. Note that this simplification can also be considered a drawback: splitting styles across multiple stylesheets can improve code organization and modularity, and would likely be a desirable feature for developers. Adding the ability to edit multiple stylesheets is an item for future work.

Loom gives the user the ability to create, delete, and update style rules in order to style the elements in the website. These rules use standard CSS syntax, and Loom provides simplified interfaces for working with many of the properties CSS provides.

IV. UI Design

For Loom to accomplish its goals, it must have a sensibly designed UI. The UI of Loom is divided into three main sec-

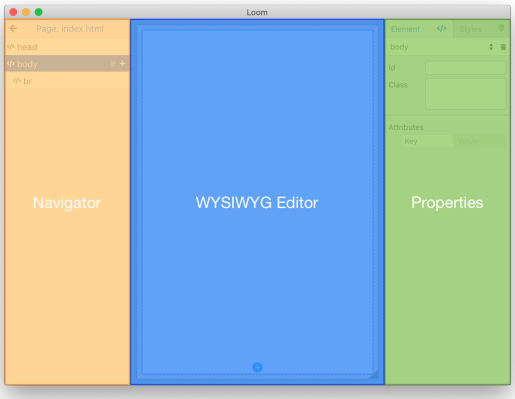

Fig. 2. Main Sections of Loom's UI

tions: the Navigator, the WYSIWYG Editor, and the Properties Editor.

A. Navigator

The Navigator allows the user to navigate between Pages and Components, as well as to select data such as elements or text within them. In this section, the term **definition** is used to refer to either a Page or Component, and the term **data** is used to refer to objects within a definition: either an Element, Component reference, or Text. The navigator has a two-step interface to allow the user to select both a definition item and a data item within. In the first step, the user first selects a definition to edit, and then is presented with a hierarchical layout showing the HTML contents of the selected definition. In the second step, the user has the ability to create, move, or delete any data within these content, as well as to select a data item for editing.

Both levels of selection (definition and data) are used by the other areas of the UI as the target for editing.

B. WYSIWYG Editor

The WYSIWYG (What-You-See-Is-What-You-Get) Editor gives the user a live preview of what the selected page or component will look like, and allows them to interact with the contents within. When a page is selected in the Navigator, the editor will display the entire contents of the page. If a component is selected, the editor will show the contents of just that component. If the user clicks on an element shown within the WYSIWYG Editor, that element will become editable and will also be selected in the Navigator. The user can edit the contents of the element simply by typing into the WYSIWYG Editor. Some keyboard shortcuts are also available to make the text bold, italic, etc. For example, Ctrl+B will make the selected text bold by wrapping it in a new element with the `b` tag. This connection between familiar shortcuts and creation of HTML elements should help teach novice users about how HTML works, and some of the basic tags used for tasks such as emphasizing text.

Fig. 3. The edit frame shown in the WYSIWYG editor, highlighting the selected element. Margins (outside of the element) are highlighted in yellow and padding (inside of the element) are highlighted in green.

In addition to the editability of selected content, a frame around the selected data will be shown in the editor, displaying an outline of the data along with a few basic floating controls for tasks such as creation and deletion of content (see Figure 3). This serves as a visual marker so the user can quickly see where the data they have selected resides on the page, including its visual dimensions and other visual properties such as padding and margins.

C. Properties Editor

The Properties Editor shows information about the currently selected definition / data, and allows the user to modify properties about these objects such as HTML attributes and styles.

The properties editor is organized into various tabs where the user can modify different aspects of the selected content. There are 4 potential tabs that may be shown depending on the current selection:

- **Element:** When an HTML element is selected, this tab displays properties about that element such as the tag name and attributes. The user can modify all of these (with the exception of the tags on `head` and `body` elements, which cannot be changed, though attributes on these elements are still modifiable).
- **Component:** When a component definition is selected (in the first step of the Navigator), this tab gives the user a way to rename the component. When a component reference (inside of a page or component definition) is selected, this tab allows the user to choose another component to reference instead.
- **Text:** When a piece of text is selected, this tab gives the user another way of editing the text content.
- **Style:** This tab is always shown, but shows slightly different things depending on the selection. For example, if an element is selected, only the style rules that apply to that element will be shown. If a page is selected, but no data within that page is selected, all style rules will be shown. Text selections will show the style rules for their parent element, and component selections will show the style rules for their root element.

The properties editor makes use of various user-friendly inputs in order to assist novice users. For example, the editor makes heavy use of a 'combo-box' style input. This allows the user to both type text freely, or to select from a pre-defined list of curated values. This provides power to more technical users, such as developers, who may wish to enter a value other than those which have been pre-defined, while providing an

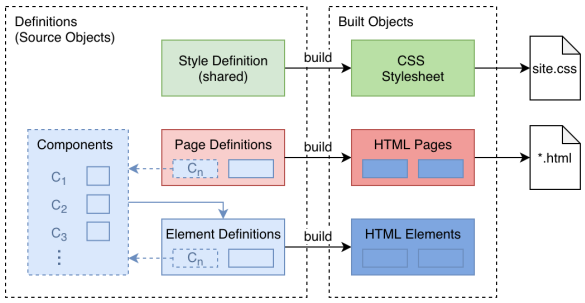

Fig. 4. Connections between Definitions, Built Objects, and outputs of Loom

easy-access option to novice users who may not otherwise know what values can be used.

V. Implementation

Loom is implemented using TypeScript[8] and the Electron[9] framework. Implementing Loom using web technologies enables a high level of code reuse, and means that Loom can take advantage of Web APIs in order to implement some features, simplifying the codebase. The Electron framework enables web applications to act as native applications by making native APIs available to scripts running in a pre-packaged web browser. Electron has built-in cross-platform support, meaning that Loom works across Mac, Windows, and Linux.

The implementation of Loom is divided into two primary packages: the UI, and the ”Core”.

A. Core

The Core portion of Loom contains the underlying data structures used to implement Loom’s features. These are conceptually divided into two halves: definitions, or source data, which is the data the user directly modifies with Loom, and ”built” data, which corresponds directly with web technologies such as HTML and CSS. The separation of the two halves enables abstractions. For example, components only exist within definitions, and get replaced with their corresponding elements (thereby duplicating the elements) within built objects.

The build process to transform Definitions into Built Objects is ”reactive”. That is, a set of Built Objects (collectively called Results) automatically updates along with the corresponding Definitions (collectively called Sources), so that if any value within the Sources change, the Results will update appropriately as well. To enable this reactive connection, Loom makes heavy use of event-driven programming, including a collection of event-driven data structures specifically created for use within Loom, creatively titled ”Loom Data”.

Figure 4 shows the relationship between each basic type of Definition and its corresponding Built Object. Page and Element Definitions can both make use of Component references, but their corresponding Built Objects cannot – the references must be replaced with concrete Elements.

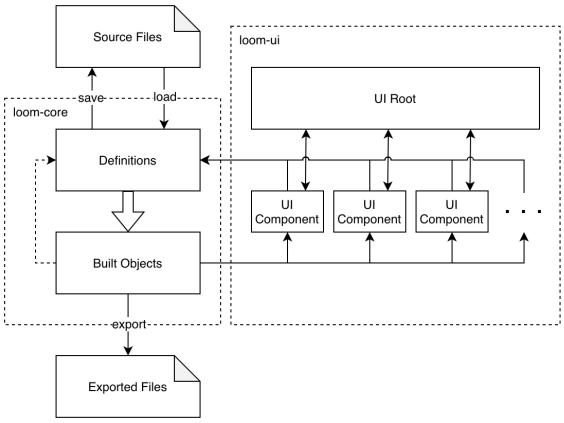

Fig. 5. Flow of data / events within Loom Core and UI.

B. UI

The UI implements all GUI aspects of Loom using the Electron framework. The primary function of the UI is to display the data from Built Objects and allow the user to modify the corresponding data within Definitions. The resulting data flow is similar to that of an MVC architecture (where Built Objects are the model and Definitions take the place of a Controller) and helps maintain a logical separation between the UI and Core. Figure 5 visualizes this data flow.

Due to the heavy use of event-driven programming and custom data structures within Loom Core, the UI portion of loom does not use any UI frameworks such as React[10] – instead using a custom solution that is designed to work well with the event-driven behavior and data structures present in Loom Core.

Every component within the UI inherits from a base class that enables components to emit events, manage their child components, and define lifecycle behaviors for creation and destruction. This allows components to easily watch for events from a data source, update their contents when events happen, and remove any event listeners when they get destroyed (a very common pattern).

The root component of the UI also stores global states such as the loaded project and selected data. Many components within the UI read and modify this data in order to implement their behavior – for example, the Navigator reads and modifies the selected data, and the Properties editor reads the selected data to determine what to display. Events are emitted whenever these values change, so that components can update themselves accordingly.

C. Serialization / Deserialization

Loom allows the user to save & load projects, and export projects to web-ready files. File formats and structures are specified for both of these functions.

A saved project is a directory consisting of:

- A ”pages” or ”src” directory, containing all of the pages in the project. These are XHTML format, except for component references, which use the XML namespace

`'loom:'` to differentiate themselves from HTML elements.
- A "components" directory, containing all of the components in the project. The name of each html file in this directory determines the name of the component, with the root element in each file being the root element for that component.
- A "loom.json" file, containing metadata about the project. At present this may specify custom paths for the pages and components directories.

When a project is exported, its contents are built and the built objects are serialized into output files. All pages, as well as the global stylesheet, are output into a shared root directory. Pages are in standard HTML format and the stylesheet is in standard CSS format.

VI. Evaluation

To evaluate how well Loom achieves its goals, we evaluate the ability to perform various target tasks from the point of view of each category of user, including novice users, developers, and designers.

A. Novice Users

Novice users include those who have little knowledge about web technologies, and potentially minimal experience with other web design tools. We do however assume that they are experienced web users in general. Loom aims to give these users the ability to accomplish simple tasks with little or no frustration. The evaluated goals for novice users are shown in Table I.

B. Developers

Developers may already be familiar with web technologies, or may just be learning them. In either case, Loom aims to give power to these users and enable rapid prototyping. The evaluated goals for developers are shown in Table II.

C. Designers

Designers are less interested in the underlying workings of a layout, and are instead looking for a way to rapidly create layouts and style components before they hand it off to a developer. The evaluated goals for designers are shown in Table III.

VII. Conclusion

Loom aims to fill a void in current web development applications by allowing users of various backgrounds to develop web sites using one unified tool. This would improve collaboration among users of different backgrounds, such as novices, developers, and designers, as well as acting as a learning tool for those who are unfamiliar with web technologies – teaching common concepts about web technologies that users can apply to other tools or even to understanding / writing web code.

Loom does not currently achieve all of its goals, even among those that have been evaluated, showing that there is room for improvement. Nonetheless, we have demonstrated the potential for Loom to be a useful tool for a wide variety of users with differing backgrounds and needs. The foundation of Loom is flexible enough to support additional features to further achieve these goals.

VIII. Future Work

There is much future work to be done on Loom. At present, the user interface is not as user-friendly as it could be. Several features could improve the experience for beginner users:

- Additional drag-and-drop features, such as the ability to create new elements by dragging from a list of pre-made elements
- Additional controls within the WYSIWYG editor, such as the ability to modify styles of a selected element or text
- Additional help text, such as to explain available HTML attributes, CSS selectors, and CSS properties
- Improved interface for creating and editing CSS selectors and rules in general
- Better UI for renaming pages and organizing site structure
- Other general bugfixes and UX improvements

Further evaluation of Loom such as user trials could also help identify additional areas for improvement.

In addition to usability improvements, there are several additional features the author has considered, but deferred for this initial prototype of Loom:

- Parameterized components, where uses of a component can pass "variables" such as text or child components for use within the component definition
- Support for media files, such as images or custom fonts
- Support for scripts and script editing within Loom
- Support for multiple stylesheets, and editing of inline / external stylesheets
- Automatic linking between pages – for example, if the location of a page is modified, other pages which link to that page will automatically have their references updated
- The ability to import an existing website into Loom
- Custom attributes, such as animations, which would have complex compilation rules to determine how the value of the attribute affects other values such as HTML attributes and styles.
- Post-processing of output content, such as automatically shortening class names, polyfilling CSS code for cross-browser support, or minifying / inlining CSS and scripts.
- Creation of a CLI tool to allow Loom projects to be built without the use of the UI.
- Integration of version control tools, enabling Loom to be used by many users collaborating together on one project.
- Integration of deployment services such as Netlify[11], enabling Loom to be used like a Content Management System.

TABLE I
EVALUATED GOALS FOR NOVICE USERS

Goal	Evaluation	Comments
Create and Delete Pages	**good**	”Add Page” button is shown when choosing a page. Deletion controls are available in right-click menu and Properties editor when a page is selected
Create and Delete Elements	**good**	Controls for creation are shown in the navigator and WYSIWYG editor. Controls for deletion are shown in all three parts of the interface
Bold / Italic / Underline Text	**fair**	Keyboard shortcuts are supported, but no interface controls that novice users would be familiar with
Create Images / Links	**poor**	Images are currently unsupported. Links are not very user-friendly
Set Basic Styles	**fair**	Search functionality and friendly style choosers for some styles, but not all – and CSS style names may not be obvious to novice users
Save Project as Static Pages	**good**	Export option within ”File” menu allows exporting sites in a few clicks
Associate Visual Content with HTML	**good**	Selected text / Elements are highlighted in the WYSIWYG editor and update immediately when modified
Associate Visual Content with CSS	**fair**	Changes made to styles can be seen immediately within the WYSIWYG editor, but this is currently the only way to associate styles on the page with the CSS rules applying them
Get Help with HTML	**poor**	There is currently no help text for HTML elements, attributes, etc.
Get Help with CSS	**fair**	CSS properties have a ”help text” icon that shows information about the property when clicked. However, this help is technical and may not be entirely useful for novice users

TABLE II
EVALUATED GOALS FOR DEVELOPERS

Goal	Evaluation	Comments
Set Custom HTML Attributes	**good**	A key-value entry interface within the Properties editor allows users to edit arbitrary HTML attributes
Set Custom CSS Property values	**fair**	Arbitrary CSS properties can be set within the Style tab of the Properties editor. However, some aspects of CSS such as media queries are currently unsupported
View Properties of an Element	**good**	A key-value interface within the Properties editor displays all set properties of the selected element
View Styles for an Element	**good**	All CSS styles can be viewed within the Propertied editor. In addition, when an emelemt is selected, only style rules which apply to that element are shown
Create Custom CSS Rules	**fair**	Creation of CSS rules with arbitrary selectors is supported. However, some aspects of CSS such as media queries are currently unsupported
Create and Edit Scripts	**poor**	Currently unsupported

TABLE III
EVALUATED GOALS FOR DESIGNERS

Goal	Evaluation	Comments
Create Elements Quickly	**good**	There are several ways to create new elements
See Immediate Output	**good**	Any time the content or style of any element is changed, the WYSIWYG editor immediately updates to display the change
Layout Content Quickly	**poor**	Organizing content on the page is currently a very ”manual” process – involving creating elements and tweaking their CSS layout properties. Elements also cannot be moved from within the WYSIWYG editor
Create Flows Across Pages	**poor**	Currently unsupported
Modify Font Styles	**fair**	CSS properties allow modification of font styles, however finding the correct properties to apply may be a frustrating process for designers
Set Padding and Margins	**fair**	CSS properties allow modification of padding and margins, however there is currently no ”friendly” interface for this and so this may be frustrating to designers
Copy and Tweak Existing Pages	**poor**	Currently unsupported
Palette of Standard Colors	**poor**	Currently unsupported

REFERENCES

[1] Wix. Wix.com Ltd. Accessed: Apr. 29, 2020. [Online]. Available: https://www.wix.com

[2] Squarespace. Squarespace, Inc. Accessed: Apr. 29, 2020. [Online]. Available: https://www.squarespace.com

[3] Figma. Figma, Inc. Accessed: Apr. 29, 2020. [Online]. Available: https://www.figma.com

[4] Sketch. Bohemian Coding. Accessed: Apr. 29, 2020. [Online]. Available: https://www.sketch.com

[5] Invision. InVision App Inc. Accessed: Apr. 29, 2020. [Online]. Available: https://www.invisionapp.com

[6] Wordpress.com. Automattic. Accessed: Apr. 29, 2020. [Online]. Available: https://wordpress.org

[7] Blocs. Cazoobi Limited. Accessed: Apr. 29, 2020. [Online]. Available: https://blocsapp.com

[8] Typescript. Microsoft Corporation. Accessed: Apr. 29, 2020. [Online]. Available: https://www.typescriptlang.org

[9] Electron. Accessed: Apr. 29, 2020. [Online]. Available: https://www.electronjs.org

[10] React. Facebook Inc. Accessed: Apr. 29, 2020. [Online]. Available: https://reactjs.org

[11] Netlify. Netlify. Accessed: Apr. 29, 2020. [Online]. Available: https://www.netlify.com