**Apollo - An Immersive Approach to Web Tutorials**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Connor Hamlet University of North Carolina at Chapel Hill cphamlet@cs.unc.edu |  |  |

**ABSTRACT**

We have developed a tutorial creation software which has the ability to create and view tutorials that are embedded into the task at hand, an improvement over alternative methods such as making a PDF or recording a video. Our software also permits tutorial creators to organize their tutorials with multiple paths, providing users with a choice when following a tutorial. Our implementation is an extension of the Chrome web browser, which makes it suited for web tutorials.

**1 INTRODUCTION**

As of 2017, there are an estimated 1 billion websites [1] which offer a variety of services. The inherit challenge with introducing many of these web based services is teaching foreign or new concepts. Given the widespread use and exponentially growing number of website services, this encourages the development of easy to follow, scalable tutorials.

Distributing tutorials and instruction has remained an inherently tedious task in the Internet age. Current methods involve either meticulously taking screenshots and collating an annotated pdf or the use of desktop recording software to share a video. These methods, while common standard, are time consuming for large tutorials and need to be remade in case a part of the tutorial changes. Most importantly, these methods divert the user's attention away from the task at hand through continuous context switching between task and tutorial.

The amount of structured data that is present on the web, specifically HTML led us to develop a Chrome extension to easily create, navigate, highlight, and annotate steps for the web. Named Apollo, our tool embeds tutorials into the task at hand.

In this paper we will walk through a case study for creating and replaying a tutorial, describe its implementation, and perform analysis between our software and current alternative methods for creating and viewing tutorials.

**CCS CONCEPTS**

• **Computer systems organization → Web Development**

**ADDITIONAL KEYWORDS AND PHRASES**

Web tutorials, scalable learning, distributed learning, web navigation

**2 CASE STUDY - TUTORIAL CREATION**

For clarity, we denote a subtasks of a tutorial as a *step* and the tutorial as a *recording.* Recordings are created by tutorial creators and are defined as a series of annotated steps over a series of webpages. Together, these steps make up the entire tutorial. Recordings do not capture mouse movements and are not timed, but instead capture instructor denoted sequences and can span multiple websites. The instructor is able to create a step by first selecting and then annotating an HTML element.

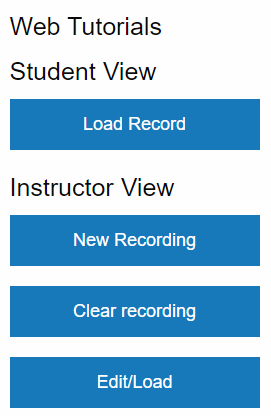
Next we will look at a case studies to demonstrate the utility of our tool, and show the UI. We explore the following case:

Jon is a faculty member looking to distribute an easy to follow and highly scalable tutorial for launching a virtual machine using the Atmosphere cloud platform for 300 students. In this case study we show how fitted our tool is for a complex task such as launching a virtual machine and how easy it is for students to use it.

Atmosphere is a service provided by Cyverse for launching virtual machines for data analysis and research purposes. In our case, Jon’s objective is to have students:

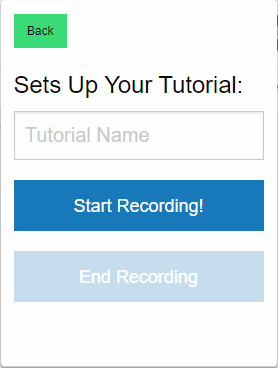
1. Register for an account on Cyverse
2. Have the students launch a minimal resource Ubuntu image

Jon first begins recording the tutorial by clicking on the chrome extension icon and selecting “New Recording”.



**Figure 1.** *The menu which is displayed upon clicking on the Chrome Extension icon.*

Upon clicking on “New Recording” Jon is prompted to enter a name for the tutorial and start recording.

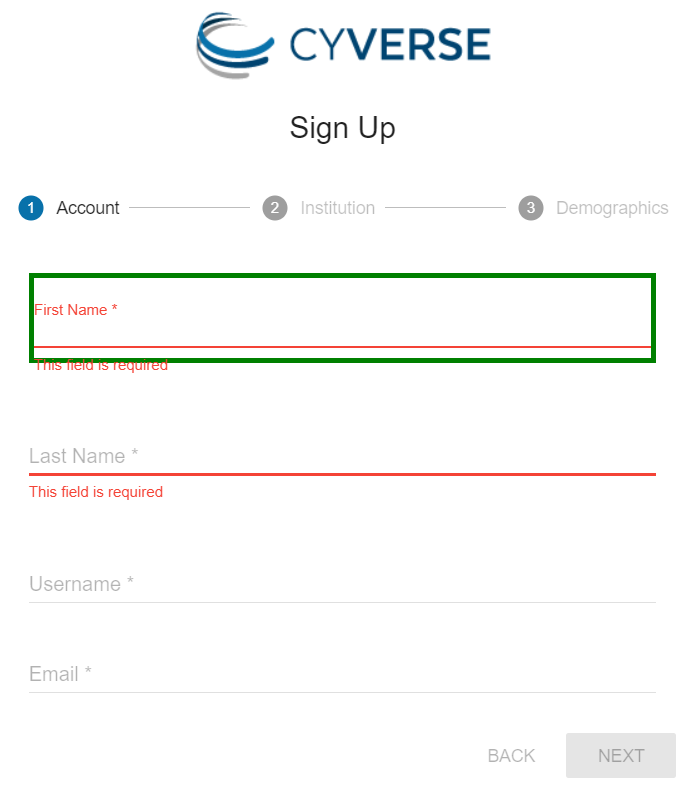


**Figure 2.** *Once Jon enters a tutorial name and clicks “Start Recording” he will be able to select HTML elements.*

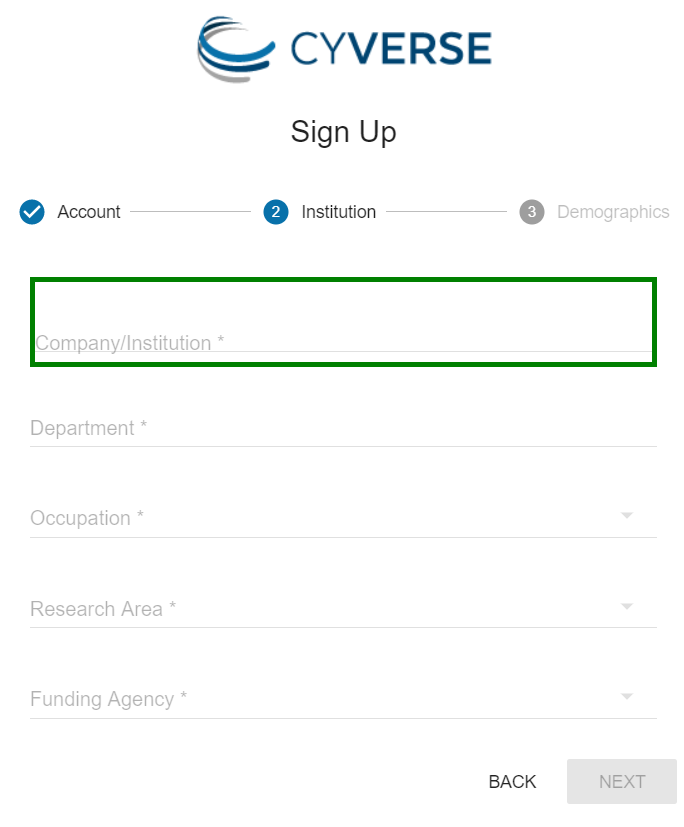
Once the tutorial has started, Jon should navigate to the first page he wants his students to see. In our case, this is the registration page for Cyverse. In order to create a step, he hovers his mouse over the sign-up button he would like to select and presses “Ctrl+Q” which highlights the HTML element.

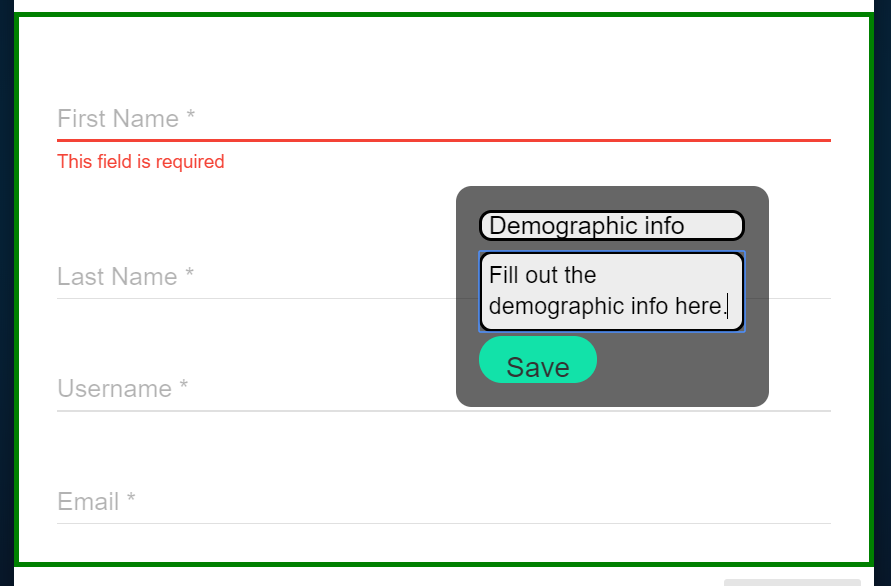
**Figure 3.** *The first step in Jon’s recording. Note that the link is bordered with a green color, indicating where the student should click.* Now that Jon has selected his first item, he annotates it by providing a text description. After he has completed the text description, he clicks the green “Save” button in order to save the step in the recording. This will save the step and allow him to select another element. When the tutorial is replayed, the HTML element Jon selected will be highlighted, and the text description he wrote will appear before the user. Jon proceeds through the steps, highlighting areas which require user interaction or areas of importance. In the meantime, our extension is monitoring for this activity and is storing it locally.

In the second and third steps, Jon highlights the demographic information for the student to fill out.

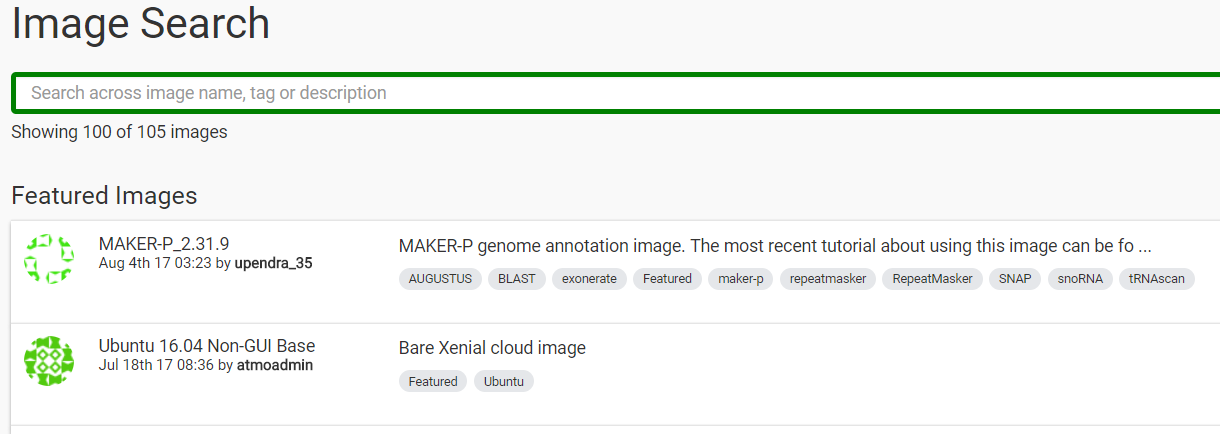


**Figure 4.** Step 2:*Jon selects the demographic area to indicate where the student should type.*

**Figure 5.** Step 3: *Any HTML element can be selected, from buttons, to textboxes, to entire webpages.*

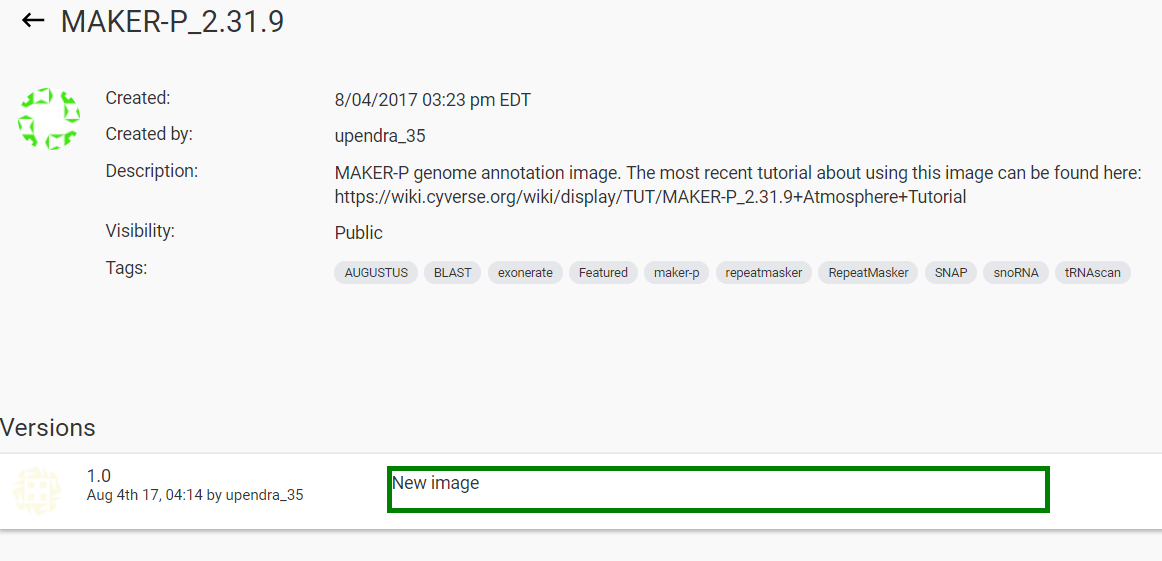
   
**Figure 6.** Step 4: *Jon continues to annotate all the steps in the tutorial. Note that Jon has to complete the task himself in order to create the tutorial.*

After registering an account, Jon follows the steps to launch a virtual machine by selecting the search bar for available images.



**Figure 7.** Step 5:*A highlighted search bar which Jon selected.*

Typing in an image type then presents the student with more detailed information, giving them an option to launch the machine.

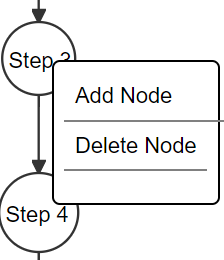


**Figure 8.** Step 6:*Jon selects the image to launch*  
  
  
After finishing the recording, Jon saves it by opening the extension menu and subsequently clicking “End Recording”. He then is presented with a visualization of his tutorial as a Directed Acyclic Graph (DAG).



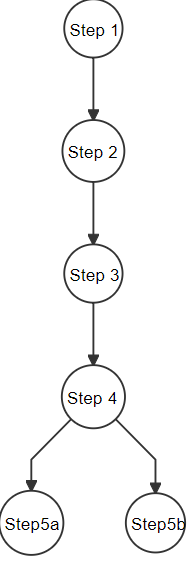
**Figure 9.** *A graph visualization of the Cyverse recording. The head node is colored green.*

With the graph visualization displayed, the instructor has the ability to delete specific nodes by righting clicking on the node in question.



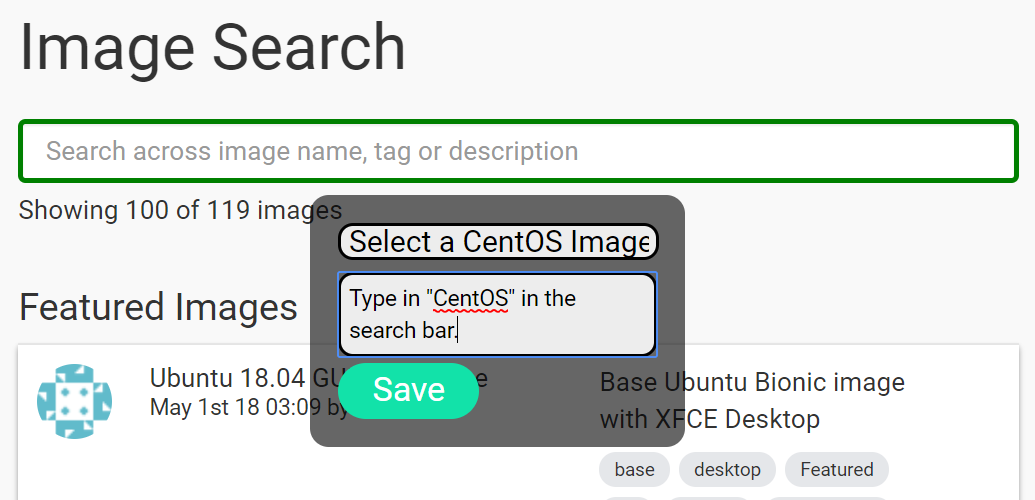
**Figure 10.** *The options displayed after right clicking a node.*

Traditionally all tutorials are linear, but since Apollo stores the tutorial as a DAG, instructors have the option to present students with multiple paths. For example, if Jon wishes to revise his recording by giving students the option to launch either an Ubuntu *or* a CentOS image, he can accomplish this by left clicking on the “Add node” button which creates a branch.



**Figure 11.** *Note a branch point has been created at step 4.*

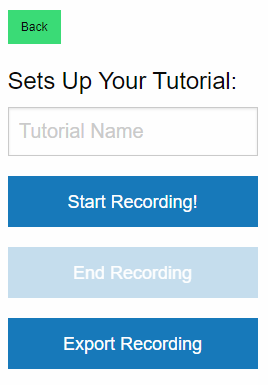
Once Jon creates a branch, Jon needs to go back to the Cyverse page and save that step in the recording, which finalizes the branch point. He proceeds to visit the webpage he wants the branch destination to be and he selects Ctrl+Q again to select the element and save it.



**Figure 12.** *Jon is creates a separate Step 5, to give the tutorial user an additional choice when they reach this point in the tutorial.*

This will present the student taking the tutorial with the option to determine what their destination will be.

Now that Jon is finished he can click “End Recording”. This will display an export button which, upon clicking, will copy the entire tutorial to his clipboard, and can be shared with students.

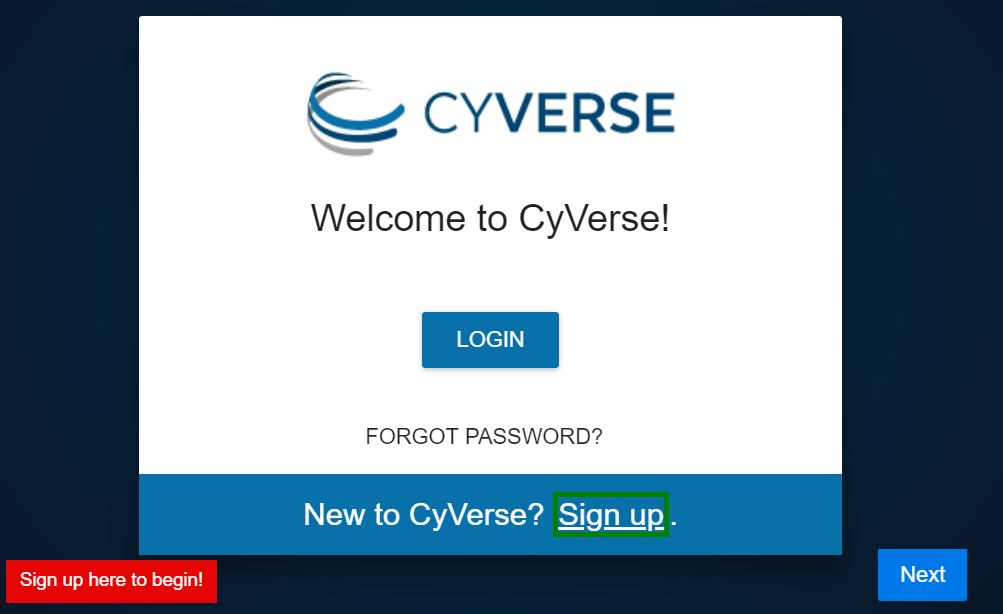
.

**Figure 13.** *Jon can export the tutorial after he completes the recording.*

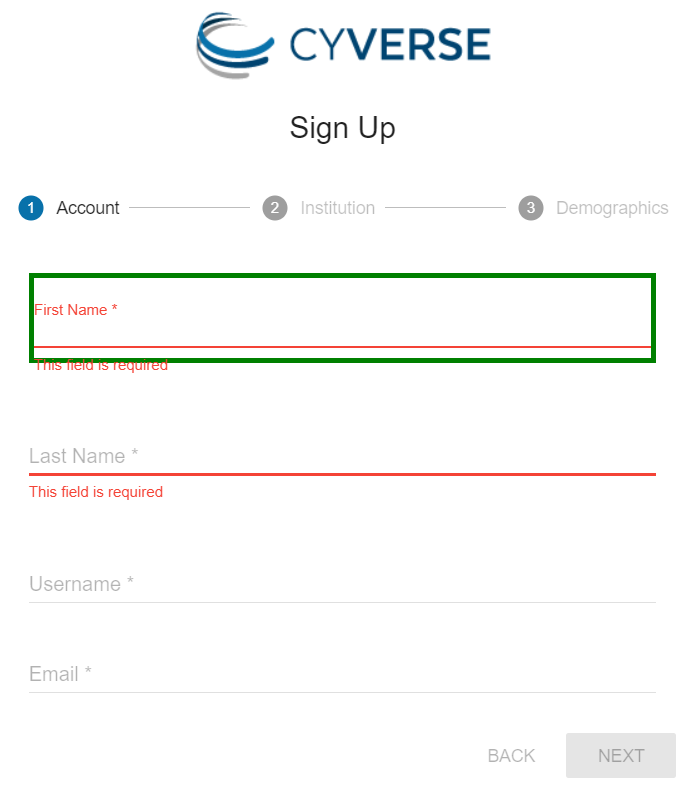
As we can see from the scenario, creating this tutorial does not involve taking a large amount screenshots, nor dealing with desktop recording software and its voice annotation. Moreover, Jon does not need to spend an excessive amount of time editing a video, or formatting a document. In our case, creating the tutorial requires completing the task at hand along with providing text descriptions. Next we will show how the tutorial Jon created will be replayed by a student.

**2 CASE STUDY - TUTORIAL PLAYBACK**

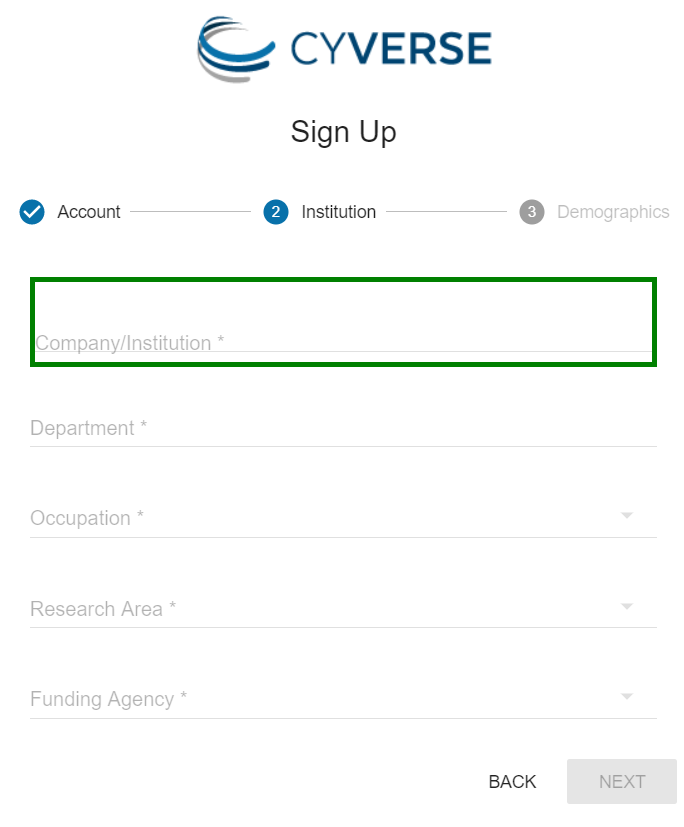
Once a student has obtained Jon’s tutorial, they can begin the recording by left clicking “Load Record” (Figure 1). This will cause the student to be automatically navigated to the first step in the recording. On the bottom right of the screen, a “Next” button is presented. Clicking on the next button will cause the student to be navigated to the next page automatically. On the bottom left of the screen, the text description that Jon typed in is presented in a red box.

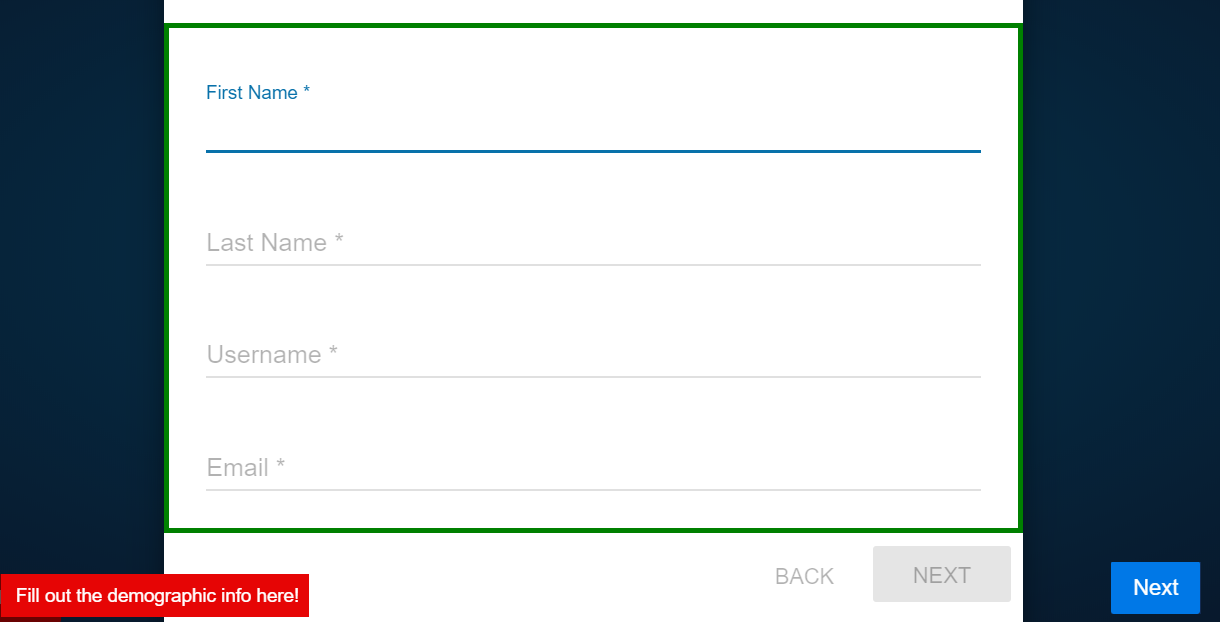
**Figure 14.***Step 1:**The sign up is highlighted and the instructor text is displayed on the bottom left. The next button is displayed on the bottom right.*

Once the student is ready to proceed to the next step, they can click the next button. In our case it will navigate the student to steps 2, 3, and 4 which are identical to Jon’s view. When the next button is clicked the user traverses to the next step and the instructor selected area is highlighted.

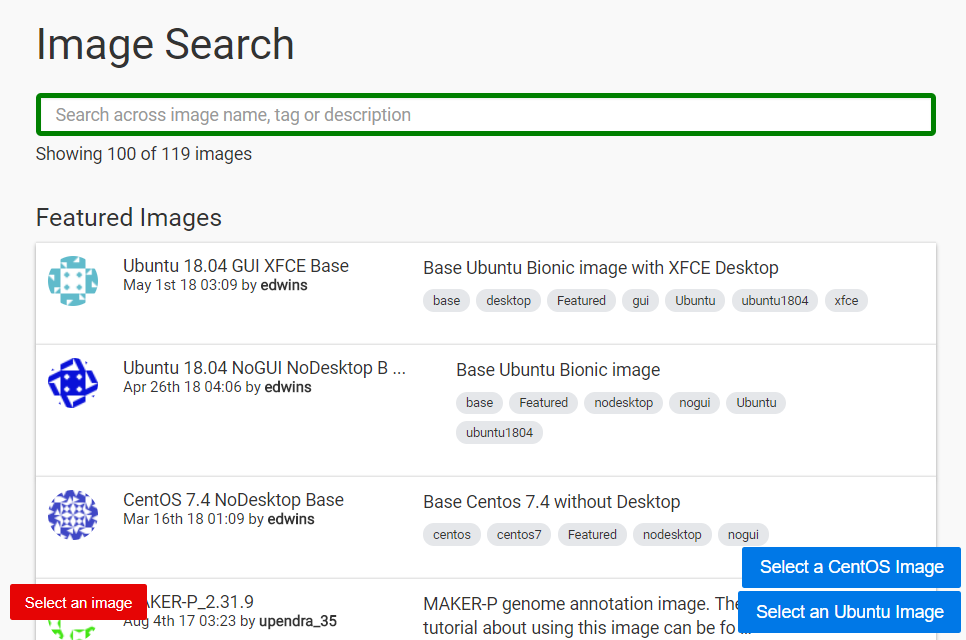


**Figure 15.** *Step 2 of the recording is displayed. Note the green border around the instructor selected element.*

**Figure 16.** *Clicking next: Step 3 of the recording is displayed*

  
**Figure 17.** *Step 4 of the recording Jon created. Note the red box on the bottom left of the screen which displays the instructors text description.*

When the student reaches a part of the tutorial where there are multiple choices, they will be displayed on the bottom right, replacing the next button (see Figure 17.)

****

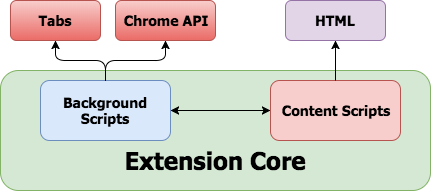
**Figure 18.** *Final step of the recording Jon created. Note that the student is presented with two options to choose from on the bottom right side of the screen.*

Once the student completes the last step of the tutorial, the extension exits. It is noteworthy to mention that because the tutorial is embedded into the task itself, students do not need to constantly switch between the tutorial and the task, saving a significant time in the process.

**5 TECHNICAL IMPLEMENTATION**

We made the decision to develop our software as a Chrome extension to leverage the structured format of HTML. Chrome extensions are segments of JavaScript designed to enhance the Chrome web browser. They add additional functionality such as adding and changing content on webpages, or offering a personalized experience to the end user. For example, the popular AdBlock™ extension actively targets and prevents the loading of web advertisements. Other ones such as Grammarly™ aim to provide corrections to a user’s grammar and spelling in text forms, preventing mistakes. All of the programming in these extensions use JavaScript in order to manipulate the page and communicate with remote servers.

Chrome Extensions consist of two main sections, content script and background scripts. Content scripts are JavaScript files have direct access to the HTML webpage. They have the ability to manipulate, save, and analyze any HTML. Their memory and process state is ephemeral to the specific page the user is currently viewing. Thus when a user changes the page the content script is started again and all variables are cleared. Secondly, there is one content script instance per tab that is open in the user’s browser. If you want data that is persistent across pages or tabs, you need what is known as a background script. Unlike content scripts, there is only one background script per browser instance. Google Chrome allows communicating between the background script and content scripts using the Message Passing API. This allows data in JSON format to be shared between both processes.



**Figure 19.** *Content scripts and background scripts communicating with each other. Content scripts manipulate the HTML and background script make calls to the browser API.*

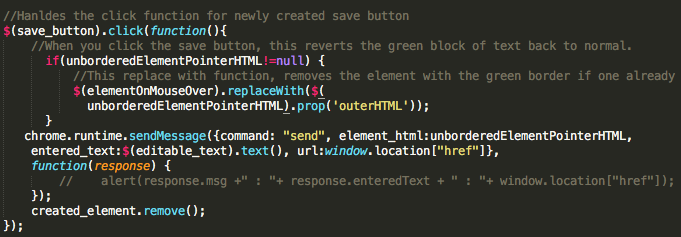
Background scripts also allows developers to access deeper and more technical Chrome API calls, such as utilizing and processing a user’s browser history. Lastly, there is the manifest.json file, which is responsible for listing out what permissions your app requires, how many content scripts you have, and all of the other required JavaScript files that are necessary to run the extension.

Our product works by implementing three separate scripts.

1. A background script which maintains the internal state of the recording and receives saved annotations from the content script.
2. A content script which manipulates the user’s webpage by adding highlighted annotations.
3. A popup script which is responsible for displaying the user authentication UI and the “Load Recording” and “Save Recording” buttons.

Earlier we discussed that when the user clicks on the “Next” button, the next step is loaded by navigating to the next URL in the list and highlighting the specified HTML element. In order to maintain this data persistently and to not have the page cleared on every load, this necessitated the use of a background script. We also incorporate plenty of HTML manipulation through the use of displaying annotated text. In order to save these recordings and maintain their persistence, this extension relies heavily on communication between the background script and content script.

For example, when an instructor selects an HTML object on a page, all of the logic for highlighting and searching occurs in the content script. After a particular node has been saved, our content script will then send this section of the recording to the background script through the chrome.runtime.sendMessage call in a JSON format.

  
**Figure 20.** *The send message call which sends the instructor selected HTML block and instructor message to the background script.*

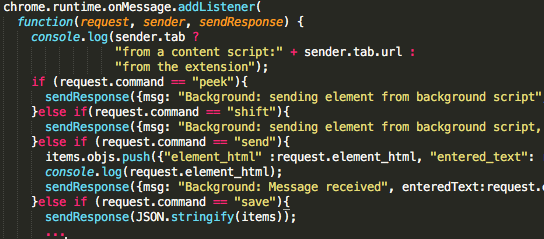
Simultaneously, the background script is running a chrome.runtime.onMessage listener, which is listening for a message from the content script in an event loop. When this message is received in JSON, our background script will dynamically add it to a queue that it maintains, which represents the entire recording from start to finish. This queue will maintain the URL of the page, HTML of the instructor selected element, the text description, and the order which the elements were placed.

One important item to note about tutorials is that not all tutorials are a single linear sequence of steps. For this reason it was important to us that we store the tutorial as a DAG which supports a number of features such as merging, and branching. With a DAG, a student has the option to *choose* what path they will take when they continue the tutorial. For example, the outcome of a tutorial might depend whether or not a user has a Windows or Linux computer. This is important for OS specific tutorial, such as downloading different versions of software.

Our background script provides the following API services:

1. Peek. This command will return the front of the queue which holds the next step.
2. Get Next. This command takes in as input what path a user would like to take outputs the DAG out in JSON form, sending it to the content script and sending the user to the previous step.
3. Get Prev. This command takes in no input, and reverts back a step, sending the user to the previous step.
4. Save. This command sends a copy of the DAG from the content script to the background script to save persistently.
5. Get Options: This command will output all of the outgoing edges from the current node. This is important for presenting to the user choices on which path to take.
6. Record Action: This command inputs a “node” and inserts it into the DAG. This occurs when an instructor saves a step. A step consists of the page url, the html they selected, a title, and the caption for the step.
7. Clear: This command will delete the DAG.

If our background script detects an unrecognized command, it will ignore it.



**Figure 21.** *Our background script listener implementation highlighting the commands it accepts.*

For storing the tutorial, we use a JSON formatted as follows:

{

DAG: { [graphlib object] },

current\_node\_id: …,

root\_node\_id: …,

tutorial\_name:

}

For storing the DAG we use *graphlib* [2], on open-source JavaScript Graph Library and DagreD3 [3] for graph visualization. The graph view gives instructors a visual way to make edits, as well as add and delete steps. Creating links between nodes are accomplished by clicking on a node and dragging it to another to connect them. This gives users multiple choices when stepping through the tutorial. *.*

**6 CHALLENGES FACED**

The biggest overall challenge on this project was persevering against the troubles imposed by learning the Chrome Extension platform. At first, it did not seem intuitive that content scripts and background scripts exist in isolated environments with only the message passing API as the available communication method. Before we moved to a background and content script combination, all recording functionality originally existed as a content script. This presented a few now obvious problems because when the next page was loaded, the recording kept automatically resetting and all ephemeral data was lost. Implementing a background script allowed data to be stored between pages and upon refreshes.

Another main issue which took a few weeks to solve was obtaining the instructor selected HTML from the recording and using it to parse the student view HTML page and find a match. This proved to be quite challenging because there is no intuitive built in method of string matching entire HTML blocks in either JavaScript or JQuery. In order to solve the string matching problem we were required to iterate over all blocks of HTML and manually attempt to match each one to the instructor selected element. Another issue involved how to edit the HTML page in a manner which works on a variety of different colored backgrounds.

**7 LIMITATIONS AND FUTURE WORK**  
It is worthy to consider the application of supporting embedded tutorial creation for the entire desktop environment. However this is a challenging problem to solve given the unstructured layout of user programs, unlike HTML. It would be interesting to further pursue an effective software program which allows stepping you through a series of steps across a multitude of applications, such as the web browser and command line.

Unlike web applications, such a product for desktop applications could not rely on data structured as HTML. Apollo for the desktop environment seems a likely probability given existing image recognition techniques such as OpenCV (Open Source Computer Vision Library)[4]. There are also a number of limitations that are present in the current iteration of this product. We address each of these in detail.

***An Improved Student View***

There are a few features which can be added which would improve the student view’s user experience. First, a DAG editor panel exists, which gives the user to form new links, but the UI is not as intuitive as we’d hoped. Secondly, there is no ability for the user of the tutorial to skip ahead several steps at a time - or at any particular point. For example, a user may want to skip ahead when they’ve closed and reopened the browser and want to resume a tutorial session. Lastly, if a user diverges from the path, there is no signal or reminder for the student to get back on course. Implementing these features would greatly improve the user’s experience.

***An Improved Instructor View***

For instructors, there is no current way to edit a recording midway through the recording process. This way, instructors can review and edit portions of their recording before making it available to students. This will prevent instructions form having to re-record entire tutorials if there is a simple error.

***A Sound Server Implementation***

Currently, there is no implemented server which is designed to handle user authentication or to host recordings. Future work will need to address these user functionality concerns, as well as implement the server which will act as a distribution hub for all of the tutorials. Once the distribution server has been implemented, instructors will be able to upload and edit tutorials as well as send out invitations to students for private access. As mentioned previously, a tutorial can be hosted on an instructor's website (the same is such for a video or PDF). However, a server implementation would create a central hub of tutorials, allowing students to explore different options on their own.

***Desktop Applications***

As a Chrome extension, Apollo lacks the ability to context switch to the desktop environment. Therefore, Apollo is not suitable for a hybrid based environment, such as if part of the tutorial is over the web and part of it involves a desktop program or an application outside of Chrome. This is not an implementation issue on behalf of the Chrome extension, but a reality of developing a tool designed for the Chrome environment.

**8 CONCLUSIONS**

Using Apollo is often faster than creating a detailed PDF tutorial and requires no editing or practicing, as is required for screen recording. Most importantly it is embedded in the student’s application, completely removing the barriers from switching back and forth from tutorial to target application which saves students’ valuable time.

|  |  |  |  |
| --- | --- | --- | --- |
|  | PDF | Video Recording | Apollo Extension |
| Effort Required | Requires screenshots + a significant amount of descriptive writing | Requires voice annotation + stepping through the task and editing | Requires installing the chrome extension + stepping through the task |
| Embedded? | No | No | Yes |
| File Type | PDF | Video | Text (JSON) |

**Table 1.** *A comparison between Apollo and alternative tutorial creation methods*

As noted in Table 1, Apollo meets the qualitative metric as an embedded tutorial over alternative methods. It also boasts a high data compression comparative over binary data types for videos and PDFs.

One could argue that developers can implement the same features that Apollo [5] provides directly into their application. This is true, however development of such features takes time, where Apollo has the advantage to create a tutorial in a shorter amount of time and will work for any website.

Apollo creates an easy to utilize overlay for tutorial creation and viewing in web applications. It leverages HTML’s structured property to make it an ideal tutorial creation option to be used for virtually any web based application.  
  
**A HEADINGS IN APPENDICES**

**A.0 Abstract**

**A.1 Introduction**

**A.2 Content Scripts vs Background Scripts**

**A.3 Scenarios**

**A.4 An In-Depth View - The Student View**

**A.5 An In-Depth View - The Instructor View  
A.6 Technical Implementation**

**A.7 Challenges Faces**

**A.8 Limitations and Future Work**

**A.9 Conclusions**

**A.10 References**

**REFERENCES**

|  |  |
| --- | --- |
| [1]  [2]  [3]  [4]   [5] | *Internet Live Stats - Internet Usage & Social Media Statistics*, www.internetlivestats.com/.  cpettitt, et al. “Dagrejs/Graphlib.” *GitHub*, DagreJS, 18 Aug. 2013, github.com/dagrejs/graphlib.  cpettitt, et al. “Dagrejs/dagreD3.” *GitHub*, DagreD3, 18 Aug. 2013, github.com/dagrejs/dagre-d3.  *Open Source Computer Vision Library - OpenCV* [www.opencv.org](http://www.opencv.org)  cphamlet et. al. “cphamlet/chrome-assistant.” *Github*, 18 Apr. 2018, github.com/cphamlet/chrome-assistant |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |