Project Report: Web Annotator

Garvit Khurana
Department of Electrical Engineering
22115059

Project Overview

Web Annotator is a chrome extension which allows users to do annotations over different Web-pages. The core functionalities provided are pen, text-highlighter, customize color palette, save and undo feature. One of the key feature associated with highlighter is contextual note to each highlighted content taken from user itself. Also with the help of save feature the annotations will persist on web-pages whenever user revisits. The undo feature helps user to overcome the mistakes while doing annotations.
The report clearly explain both textually and visually through images about how the front-end of the extension looks as well as the explanation about the implementation of back-end. The challenges faced and possible updates or features which can be added are discussed at the end of the report. Such type of chrome extension can be used in different domains such as education, research, corporate, journalism, personal use, software development, designing etc.

Contents

1 General requirements to build Chrome-extension

- **Manifest.json File** :Manifest is a JSON file which stores an abstract information of our Web-extension. It's basically an identity of our extension storing items like name, version, description, permissions, icon and scripts of our extension. For any type of extension this file is the first one to be created.

- **Popup Files** :There are mainly three types of Popup files required which includes **popup.html, popup.css & popup.js.** Each of them servers their own purpose and are interlinked. The HTML file is the skeleton of our extension, in web-annotator we added buttons for each feature and an input color palette. Obviously the no one likes the default look so to adjust margin and add some designing we made CSS file which is linked to our HTML file. The main execution takes place from our javascript file which contains the logic of those html elements like buttons, inputs etc. This file is further linked to other scripts as well.

- **Content.js File** :This file runs in context with our web-page. The moment you open any web-page and assuming extension is installed and necessary URL permissions are given then this file will be loaded first. It stores the logic behind task to be performed. Like in our case it needs to store the drawing logic. Any type of command which is allowed to user, this file stores it's implementation in the form of different functions. You can think it as a brain of our extension. In our web-annotator it mainly stores the code of DOM manipulation.

- **Background.js File** :Background script serves the purpose of maintaining the current state of our extension. It includes implementation of actions such as saving annotations in our chrome local storage, loading annotations on reloading etc. Basically it's responsible of API calls. Background scripts can facilitate communication between different parts of your extension, such as the popup, content scripts, and other extension pages. This is done through message passing.

  These were some general files needed for any type of web-extension. You may need other scripts if the feature if complex. To understand it visually flowchart and diagram are also added to this report.

2 Flowcharts & Diagrams

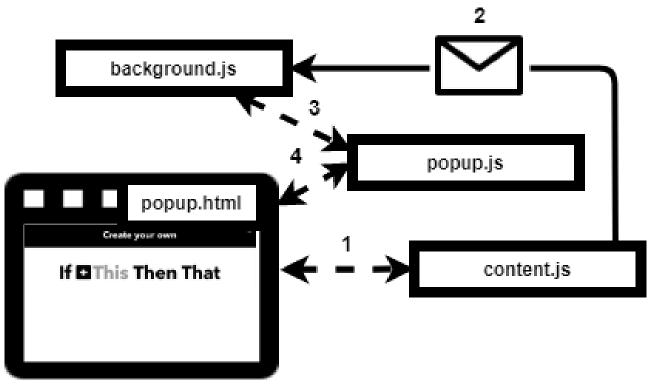


Figure 1: Architecture Diagram

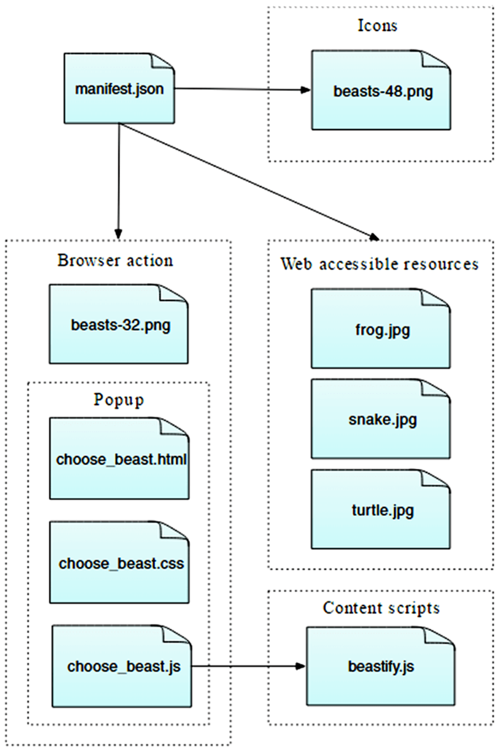


Figure 2: Flowchart

3 Front-end Behaviour & Implementation

- Web-Annotator's front-end is written in HTML,CSS and Javascript. First let's see how will it look if you load the extension and open it.

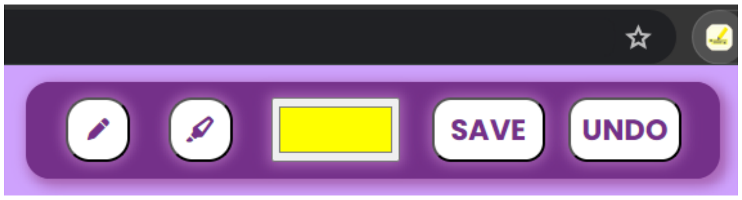


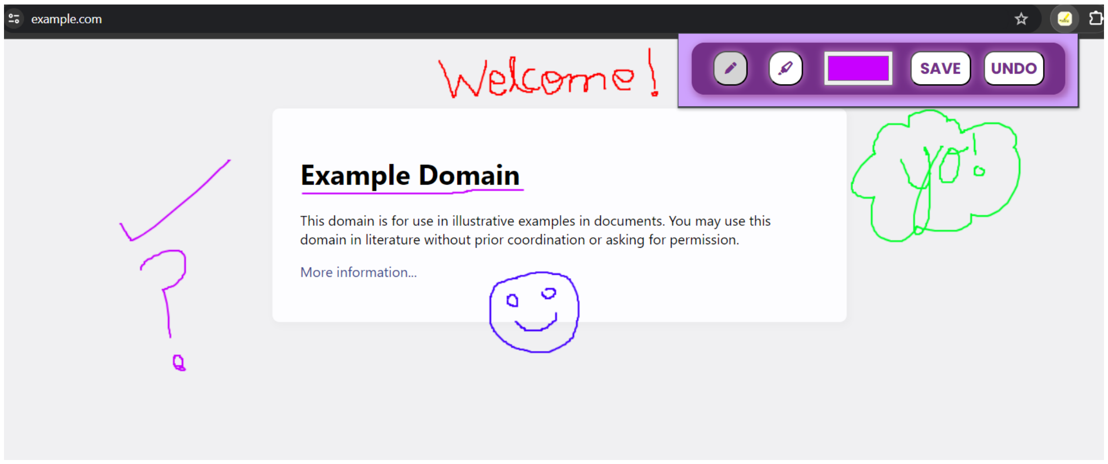
Figure 3: Extension's Front-end

- There are total of 5 feature which includes drawing pen, text-highlighter, color coded, save and undo option. Let's go through each one of them to see how they works and what's logic behind it.

- **Drawing Pen** : For implementing a pen I created the canvas which will allow me to draw. Everytime web-page is opened a canvas will be created therefore it's written in content script. There are two major things which are :-

  - Canvas : It is a transparent 2D graphic HTML element used to draw graphs, animations etc. I set canvas dimensions same as that of windows to make it suitable for every monitor size. On this canvas three Event listeners were added to detect the pointer's movements.
  - Mouse Down allows us to detect if the mouse button is pressed within the range of element. The call back function for this is listener is **HandleMouseDown** where if the pen status is active/current tool is pen we will call **StartDrawing** function.
  - Mouse Move allows us to detect if the mouse if moved over the element. The call back function is **HandleMouseMove** from where we will call **draw** function where main drawing logic is implemented.
  - Mouse Up allows us to detect when we stopped moving after mouse down and move. The call back function is **HandleMouseUp** from where we called **StopDrawing** function. In this Stop drawing function we are storing the annotation path in a stack. This will help us later on in saving and undoing the annotations.

- Drawing Functions : If current tool is pen and canvas detects mouse down then our flag ”isDrawing” will be true and we get the coordinates using **e.clientX, e.clientY**. This will be starting of our path and maintain this path as form of **(X,Y)**.
- After this as the mouse moves due to listeners our draw function will be called again and again and we keep on tracking the coordinates and storing them. Basic functions like begin path, move to, line to and stroke are used. Used 2D context render to keep stoke style equal to current color and stroke width remain constant.
- The moment it stops we will push them in **annotations** stack for further use of saving and undoing.



Example Domain

This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

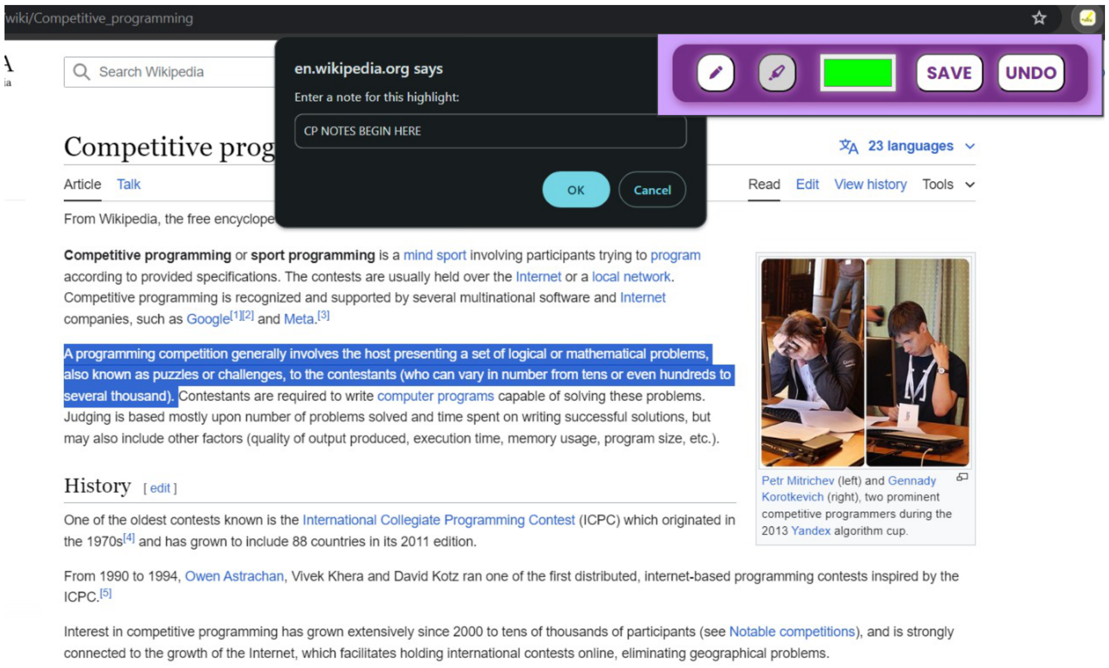
More information...

Figure 3: Pen Tutorial

- **Text-Highlighter With Notes** : Highlighter has nothing to do with canvas because implementation of it involves D0M manipulation. Here we are changing the background color of selected text. So the event listeners are added to the document object.
  - Document Event Listeners : There are 4 listeners added to document if current tool is a highlighter. The three are same as that of pen that is **HandleMouseDown, HandleMouseMove, HandleMouseUp** with some modifications. The moment mouse up event occurs a **prompt appears which takes the input of note associated to corresponding highlight from user**. The 4th document event listener is of **click** which gives an alert showing the note attached to the highlight.

  - <u>DOM Manipulation</u> : This is done via wrapping of selected text with span element. Using **window.getSelection()** to get the range and creating a span element and changing it's background color and simply wrapping the range using **range.surroundContents()**. One thing to note is giving a specific id to span as this will help us to track the information of the element when we will redraw the highlights on reloading the web-page. Lastly pushed this information in highlights stack to keep record and can be used while saving/undoing.

- **Let's see how it works!**



en.wikipedia.org says

Enter a note for this highlight:

CP NOTES BEGIN HERE

Competitive prog

From Wikipedia, the free encyclope

Competitive programming or **sport programming** is a mind sport involving participants trying to program according to provided specifications. The contests are usually held over the Internet or a local network. Competitive programming is recognized and supported by several multinational software and Internet companies, such as Google[1][2] and Meta.[3]

A programming competition generally involves the host presenting a set of logical or mathematical problems, also known as puzzles or challenges, to the contestants (who can vary in number from tens or even hundreds to several thousand). Contestants are required to write computer programs capable of solving these problems. Judging is based mostly upon number of problems solved and time spent on writing successful solutions, but may also include other factors (quality of output produced, execution time, memory usage, program size, etc.).

History [edit]

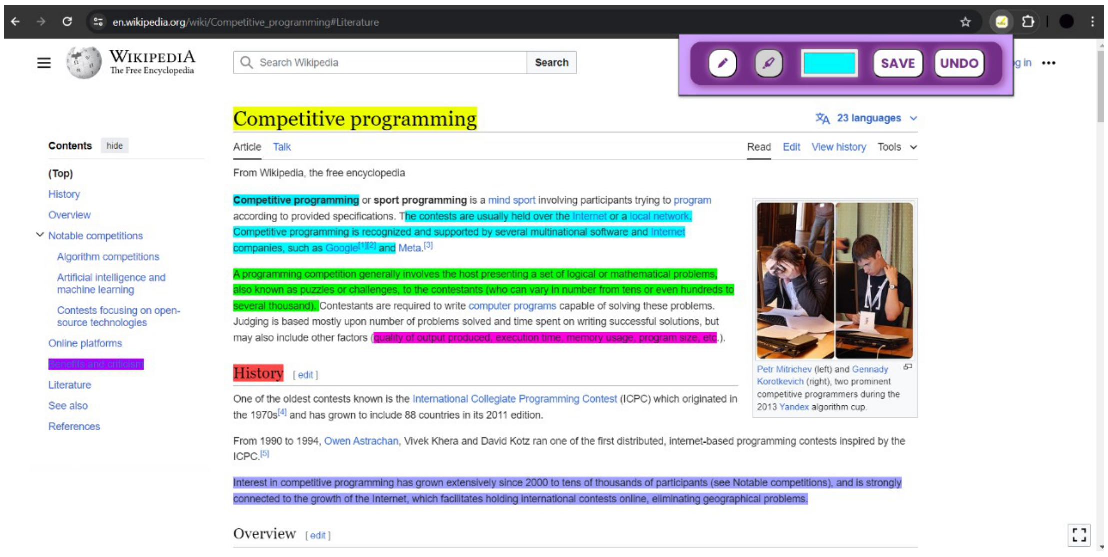
One of the oldest contests known is the International Collegiate Programming Contest (ICPC) which originated in the 1970s[4] and has grown to include 88 countries in its 2011 edition.

From 1990 to 1994, Owen Astrachan, Vivek Khera and David Kotz ran one of the first distributed, internet-based programming contests inspired by the ICPC.[5]

Interest in competitive programming has grown extensively since 2000 to tens of thousands of participants (see Notable competitions), and is strongly connected to the growth of the Internet, which facilitates holding international contests online, eliminating geographical problems.

Petr Mitrichev (left) and Gennady Korotkevich (right), two prominent competitive programmers during the 2013 Yandex algorithm cup.

Figure 4: First Select Then Add Notes



Competitive programming

From Wikipedia, the free encyclopedia

Competitive programming or **sport programming** is a mind sport involving participants trying to program according to provided specifications. The contests are usually held over the Internet or a local network. Competitive programming is recognized and supported by several multinational software and Internet companies, such as Google[1][2] and Meta.[3]

A programming competition generally involves the host presenting a set of logical or mathematical problems, also known as puzzles or challenges, to the contestants (who can vary in number from tens or even hundreds to several thousand). Contestants are required to write computer programs capable of solving these problems. Judging is based mostly upon number of problems solved and time spent on writing successful solutions, but may also include other factors (quality of output produced, execution time, memory usage, program size, etc.).

Petr Mitrichev (left) and Gennady Korotkevich (right), two prominent competitive programmers during the 2013 Yandex algorithm cup.

History [edit]

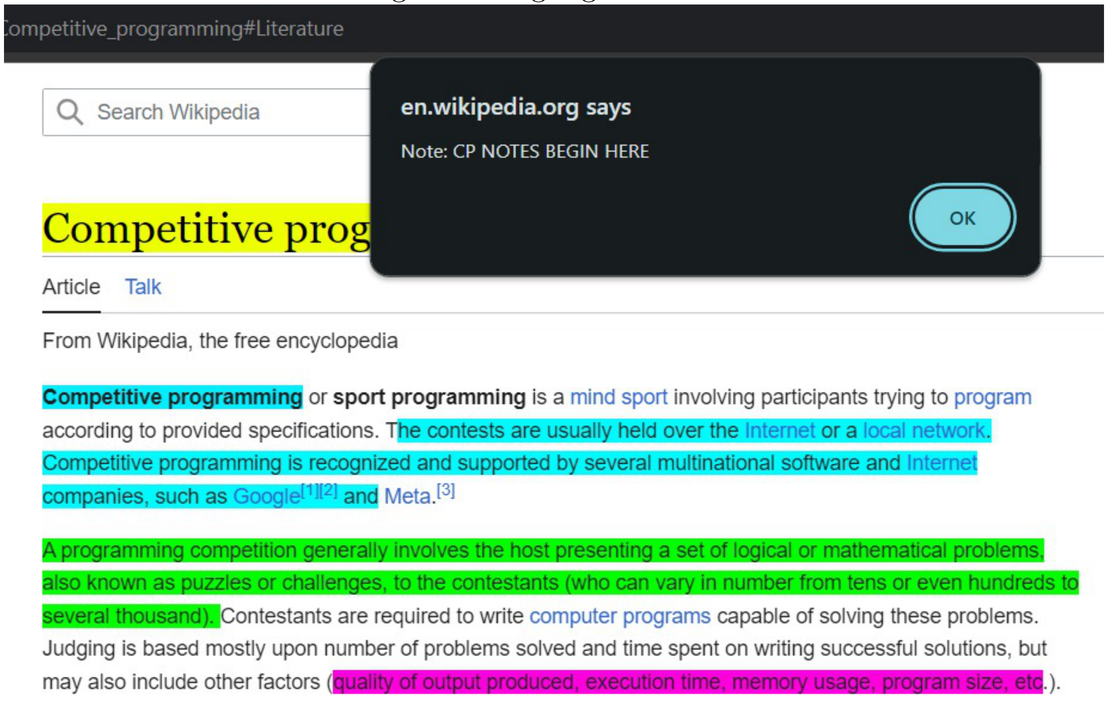
One of the oldest contests known is the International Collegiate Programming Contest (ICPC) which originated in the 1970s[4] and has grown to include 88 countries in its 2011 edition.

From 1990 to 1994, Owen Astrachan, Vivek Khera and David Kotz ran one of the first distributed, internet-based programming contests inspired by the ICPC.[5]

Interest in competitive programming has grown extensively since 2000 to tens of thousands of participants (see Notable competitions), and is strongly connected to the growth of the Internet, which facilitates holding international contests online, eliminating geographical problems.

Overview [edit]

Figure 5: Highlight Tutorial



en.wikipedia.org says

Note: CP NOTES BEGIN HERE

Competitive prog

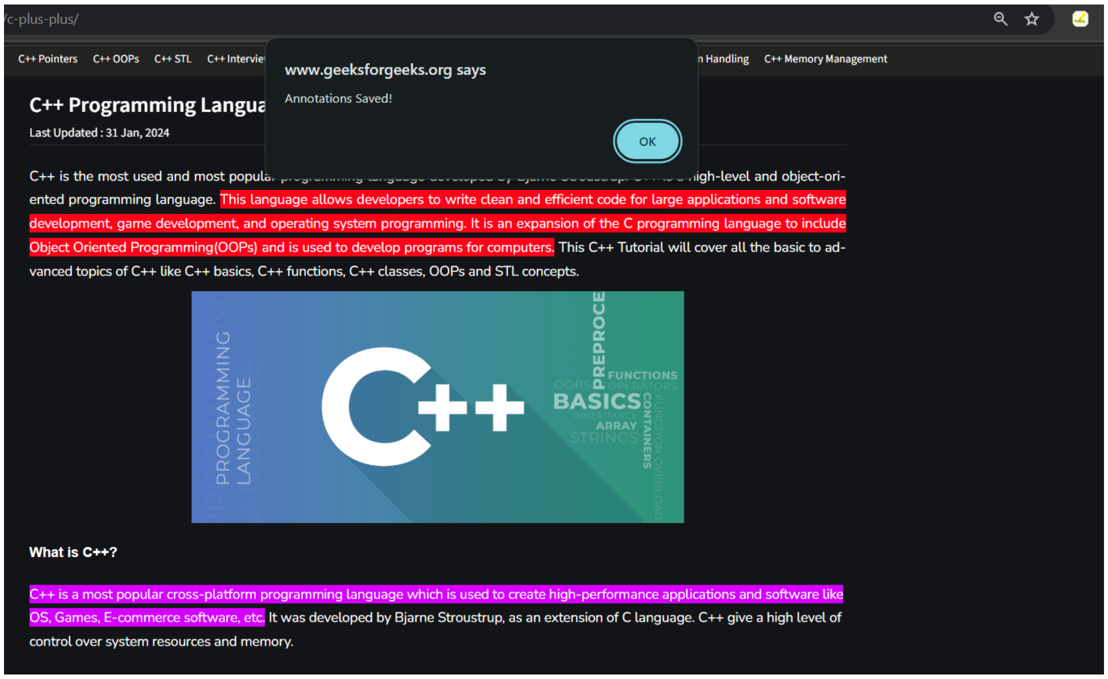
From Wikipedia, the free encyclopedia

Competitive programming or **sport programming** is a mind sport involving participants trying to program according to provided specifications. The contests are usually held over the Internet or a local network. Competitive programming is recognized and supported by several multinational software and Internet companies, such as Google[1][2] and Meta.[3]

A programming competition generally involves the host presenting a set of logical or mathematical problems, also known as puzzles or challenges, to the contestants (who can vary in number from tens or even hundreds to several thousand). Contestants are required to write computer programs capable of solving these problems. Judging is based mostly upon number of problems solved and time spent on writing successful solutions, but may also include other factors (quality of output produced, execution time, memory usage, program size, etc.).

Figure 6: Pop Up Showing Notes After Clicking On Highlighted Text

- **<u>Customize Color</u>** : This feature is implemented using simple HTML input element. Listeners were added in popup javascript files which store the value of selected color as a string and pass it to content script. By default the color will be yellow. Any RGB value of color can be given too.
- **<u>Saving & Undoing</u>** : As told in pen and highlighter functionalities that a stack is implemented to store the last stroke of pen and highlighter in two different stacks named **annotations & highlights** respectively. These two stacks were passed to background script to make Chrome API calls and save it in local storage. We will look about background script in next section.
  For Undo one more stack was implemented to have seamless interaction which is **Actions**. This actually stores the value as 1 **OR** 2. 1 represents that last action was of pen else if it's 2 then a highlighter was used. Depending on the it's top value we can know out of other 2 stacks of annotations which has to be pop. A simple redraw function is called out which has the similar logic the way we did pen and highlighter.



www.geeksforgeeks.org says

Annotations Saved!

C++ Programming Langua

Last Updated : 31 Jan, 2024

C++ is the most used and most popula … igh-level and object-oriented programming language. This language allows developers to write clean and efficient code for large applications and software development, game development, and operating system programming. It is an expansion of the C programming language to include Object Oriented Programming(OOPs) and is used to develop programs for computers. This C++ Tutorial will cover all the basic to advanced topics of C++ like C++ basics, C++ functions, C++ classes, OOPs and STL concepts.

What is C++?

C++ is a most popular cross-platform programming language which is used to create high-performance applications and software like OS, Games, E-commerce software, etc. It was developed by Bjarne Stroustrup, as an extension of C language. C++ give a high level of control over system resources and memory.

Figure 7: Pop-up Showing Annotations Saved

4 Back-end API Calls

- To build this extension Chrome API was used multiple times to pass messages from popup files to content script, content script to background script or to get responses back from files. Also to access local storage to save and retrieve our annotations and current status of our pen, highlighter and stroke-color.
- Below are some calls and queries made are listed as:-
  - Set & Get Calls : **chrome.storage.local.get() OR chrome.storage.local.set()** are used to retrieve and save the values from local storage. A callback function was made once we get our result while using get function. Everything is accessed using a key value pair. We pass key to get or set the new values. Around 10MB is provided while using local storage which gets cleared on removing extension. We will discuss about changes/updates in last section.
  - Tabs query & Message : Tabs query is made as **chrome.tabs.query({...},())**. The response help us in interactive with browser's tabs. This way we get tabs ID and can be passed with other parameters using **chrome.tabs.sendMessage()**. These calls were written in popup.js to communicate with tabs and content scripts.
  - Runtime Messages & Listeners : **chrome.runtime.sendMessage() and chrome.runtime.onMessage().addListener** were used in pair to send annotations while saving or loading annotations while reloading the web-page.

5 Challenges faced

- I personally faced a lot of difficulty in undoing annotations. Later on using debug statements I realised that I was filling up my 2 stacks of annotations and highlights manytime even if cursor moves it was getting pushed in stack due to which I needed to press multiple times my undo button. To fill this bug I came up with an idea of making another stack and just pushing values like 1/2 to distinguish and pushing it only when it was needed.
- Pointer events actually was a big task to resolve. The error was related to switching between pen and highlighter. Since we are drawing pen on canvas so it required pointer events as 'auto' and for highlighter else if current tool is null then I made it 'none'. This wasn't a big thing to do but main problem then i faced was with mouse up,down and move listeners. The solution was to add these listeners to both canvas and document and a single function to handle both listenings.

6 Possible upcoming updates & features

Here are some updates or features which can be added to this project :-

- **Brush Feature** : This feature can be added similarly the way we added a pen. Just we need to play with canvas stroke style and thickness to get appropriate output.
- **Customize Thickness** : allowing user to scale their thickness within a integral range can be useful as compared to having a fixed one.
- **Eraser** : Along with undo I personally felt that there should be a feature of brush along with sizes to erase on canvas. This is only for pen/brushes not for highlighter.
- **Shapes Tool Box** : This can help in a way if user wants to outline the text. Using a pen will not bring an accurate result as it free drawn so basic shapes like rectangle, square, circle etc. should be added.

With this I would like to end my project's report. The extensive research and testing during the project helped to achieve it's objectives. I would like to share my sincere gratitude to Tinkering Lab for giving me a chance to build this amazing project.