AN INDUSTRY ORIENTED MINI PROJECT REPORT

(CS704PC)

On

DESKTOP NOTIFICATIONS FOR GOOGLE MEET

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

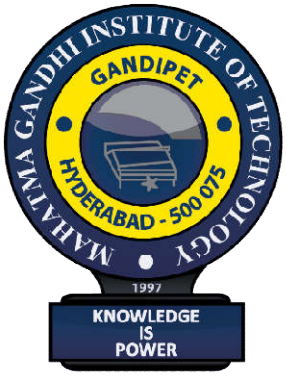
Submitted by

Mr. EATHAMUKKALA AKARSH REDDY (18261A0571)

Under the Guidance of

Ms. K. Sunitha

Assistant Professor, CSE.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

(Affiliated to JNTUH, Hyderabad; Six UG Programs Accredited by NBA 3 times; Accredited by NAAC with 'A' Grade)

Kokapet (vill), RajendraNagar (Mandal), Ranga Reddy (Dist.),

Chaitanya Bharathi P.O., Hyderabad-500 075.

2021-2022

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY

Kokapet(Village), Gandipet, Hyderabad, Telangana - 500075. www.mgit.ac.in

MOTIVATE INNOVATE EMPOWER 24 YEARS

CERTIFICATE

This is to certify that the industry oriented mini project work entitled **Desktop Notifications For Google Meet** submitted by **Mr.Eathamukkala Akarsh Reddy (18261A0571)** in partial fulfilment of requirements for the award of degree of Bachelor of Technology in Computer Science & Engineering as specialization is a record of the bonafide work carried out under the supervision of **Ms. K. Sunitha**, and this has not been submitted to any other university or institute for award of degree or diploma.

Supervisor

Ms. K. Sunitha

Assistant professor

Dept. of CSE

Head of the department

Dr. C. R. K. Reddy

Professor and HOD

Dept. of CSE

External Examiner

DECLARATION

I hereby declare that the industry oriented mini project work entitled **DESKTOP NOTIFICATIONS FOR GOOGLE MEET** is original and bonafide work carried out by me as a part of fulfilment for Bachelor of Technology in Computer Science & Engineering, Mahatma Gandhi Institute of Technology, Gandipet, Hyderabad, under the guidance of **Ms. K. Sunitha**, Assistant Professor, Department of CSE, MGIT.

No part of the project work is copied from books/journals/internet and wherever the partition is taken, the same has been duly referred in the text. The report is based on the project work done entirely by me and not copied from any source.

EATHAMUKKALA AKARSH REDDY

(18261A0571)

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the project.

I would like to express our sincere gratitude and indebtedness to our project guide **Ms. K. Sunitha,** Assistant Professor, Dept. of CSE**,** who has supported me throughout my project with immense patience and expertise.

I am also thankful to the honourable Principal of MGIT **Prof. K. Jaya Sankar** and **Dr. C. R. K. Reddy,** HOD and professor, Department of CSE, for providing excellent infrastructure and a conducive atmosphere for completing this project successfully.

I am also extremely thankful to our Project Coordinator **Dr. A. Nagesh,** Professor, Dept. of CSE, **Dr. V. Subba Ramaiah,** Sr. assistant professor, Dept. of CSE, **Mr. A. Ratna Raju,** Assistant professor, Dept. of CSE, for their valuable suggestions and interest throughout the course of this project.

I convey my heartfelt thanks to the lab staff for allowing me to use the required equipment whenever needed.

Finally, I would like to take this opportunity to thank my family for their support all through the work. I sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

EATHAMUKKALA AKARSH REDDY

(18261A0571)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Google Meet™ has become quite popular during the pandemic, especially among educational institutes. It comes with many broadcasting features to help teachers facilitate education over the internet. But it lacks the link for efficient interaction. One such is that the broadcaster is not notified whenever there is a new message, such as a query from a student in the meeting, requiring him to open the meeting window to look for new messages. This isn't very convenient for reciprocity.

The novel approach to enhance the interaction is to push notification containing the message to the desktop environment to allow readability to the broadcaster. This is accomplished by setting up 'listeners' where we can observe new messages, which then perform the necessary message to desktop notification task. To avoid overhead that may arise due to the use of special APIs, only standardized APIs of Chrome and JavaScript are utilized. It is incorporated as Google Chrome™ extension.

This innovative Google Chrome™ Extension ameliorates the experience by sending a desktop notification containing the participant's name and their message to the presenter. The presenter need not to navigate to the browser window to view the new message and is notified of the new message immediately. This extension approach assists many kinds of users, one such example, is allowing lecturers to be informed of any queries(messages) from the students without interrupting the flow of lecture and helping students by allowing their queries to be notified to lecturer.

1. Introduction

Google Meet™, a free virtual meeting service offered by Google, has been adopted as a platform of education by substantial number of institutions during the pandemic. It is embraced for being free, quick & easy to setup, multi-platform and for its domain privacy features. Once in the virtual meeting, the platform enables communication through voice and video, and allows any participant to share their screen to present to large number of audiences. Since its utilization proliferated, the platform has gone through various updates to make provisions for a more efficient virtual meeting environments. Although Google was able to bring in newer features over time, the system stumbles at efficient communication. Incorporating the desired features through utilization of Google Chrome™ extension is the focal point of this project.

1.1 Problem Definition

Google Meet™ has become a prominent platform for virtual meetings, although it implements adequate voice and video communication, but the potential use of text messages is dilapidated. The text messages are not effectively conveyed and often go unnoticed if the presenter is on a screen other than the meeting window. This could be a hassle to communicate through texts on this platform.

1.2 Existing System

In the existing system, the text messages are only readable in the Google Meet meeting window in the browser. Text messages can be observed in two regions depending on state of chat box. A new text message can be observed as a message bubble (requires the presenter to be in the Google Meet meeting window in the browser) if the chat box is closed, or it can be viewed in the chat box otherwise.

Drawbacks:

- The new message may not be viewed timely by the presenter, since it can only be viewed voluntarily by him.
- The message may go unnoticed if the presenter never checks their chat box for new messages.

- The presenter must access the meeting window in the browser to be able to read new messages.

1.3 Proposed System

The inefficient message delivery can be optimized by encapsulating messages as a desktop notification. Independent of the window the presenter is currently on, the new message is immediately notified to him by directing a notification containing the participant's name and message directly to the desktop environment of the presenter. The presenter need not to navigate to the browser window to view the new message and is notified of the new message immediately.

1.4 System Requirements

1.4.1 Software Requirements

Operating System: Windows 7, Windows 8, Windows 8.1, Windows 10 or later

Platform: Google Meet

Permissions: Notifications

Browser: Chromium-based browser (Google Chrome or Microsoft Edge)

1.4.2 Hardware Requirements

Processor: Intel Pentium 4 or AMD Athlon 64, AMD Phenom, and later processors

RAM: 4 GB

Hard Disk: 500 MB minimum available storage

2. Literature Survey – Comparative Study

	Google Meet	Microsoft Teams	Zoom	Webex by Cisco
Free Tier Availability	Yes	Yes	Yes	Yes
Meeting participants (Default)	100	250	100	200
Integrated Video	360p or 720p video meetings	1080p with up to 30fps	1080p with bandwidth of 3Mbps	1080p
Security	adheres to security standards such as DLTS, SRTP, IETF	Proprietary Microsoft Network Protocol version 24	AES 256-bit GCM encryption	TLS 1.2 protocol for streaming, protects all user data using SHA-2
End-to-end Encryption	No (Only encrypted in-transit)	Yes (Only one-on-one team calls)	Yes (With restrictions)	Yes (With restrictions)
Direct Web Access	Yes	Yes, but desktop app for full experience	Yes, but desktop app for full experience	Yes
Screen Sharing	Yes	Yes	Yes	Yes
Instant Messaging	Yes	Yes	Yes	Yes
Message Notification	No	Yes, but only on mobile and desktop apps	No	Yes, but only on desktop app
Supported Platforms	Android, iOS, Web	Windows, Android, Linux ,macOS, iOS	Windows, Android, Web, Android, iOS, macOs	Windows, Mac, Web , Android, iOS

Table 2: Literature Survey

3. Desktop Notifications For Google Meet

3.1 System Architecture

The following figure depicts and explains the system architecture of Desktop Notifications for Google Meet. The meeting participants are termed as 'actors'. The Google Chrome extension only activates on a Google Meet domain. When on a Google Meet page, it checks whether the user has entered the meeting with help of polling. When active, it listens for new messages through use of MutationObserver API and generates a desktop notification with help of Notification API.

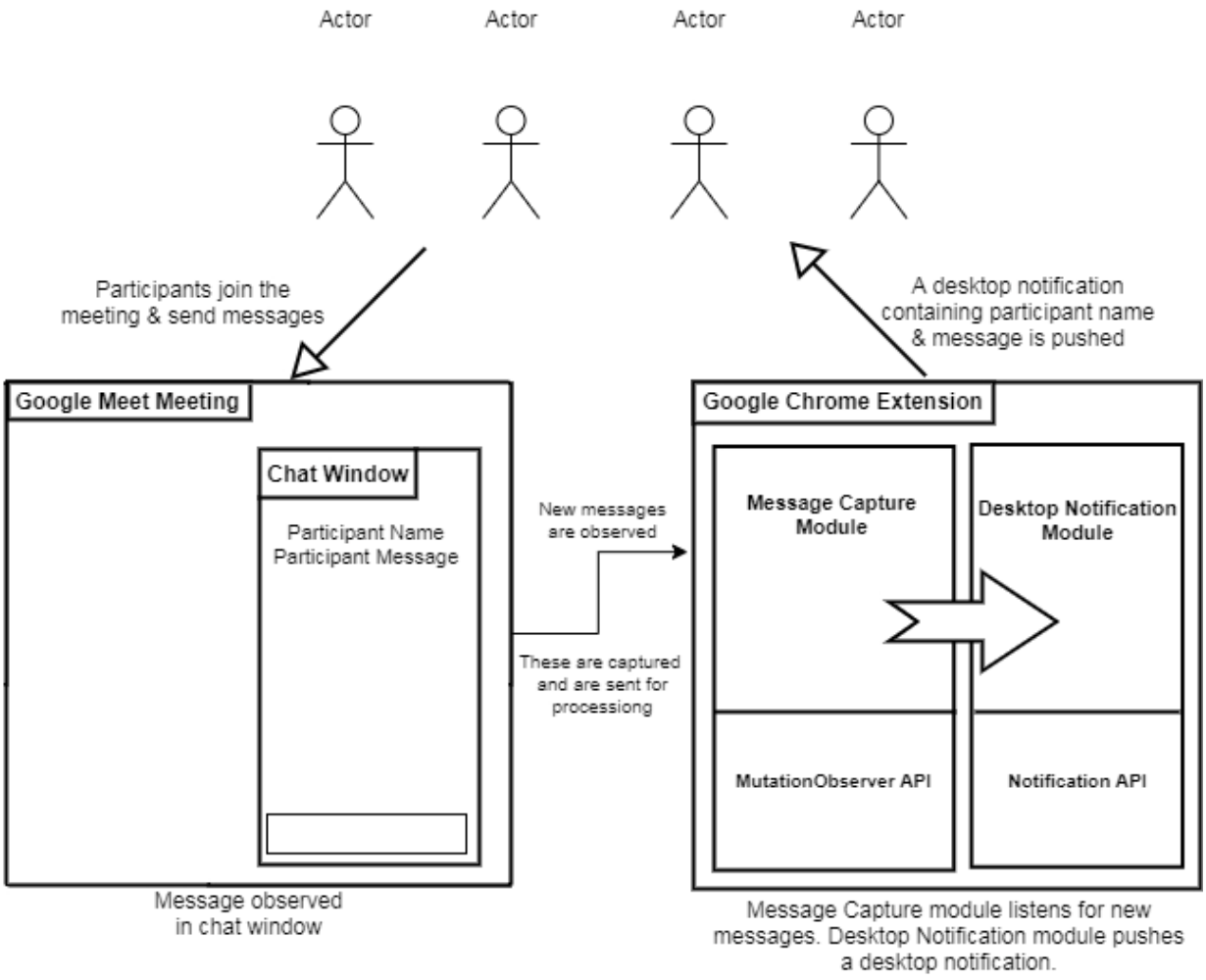


Fig.3.1: System Architecture for Desktop Notifications For Google Meet

3.2 Modules Description

3.2.1. Activating Chrome Extension

Since this application is only required to run during a meeting in Google Meet, it is vital to recognize if the user has entered the meeting to actuate our desired functionality. The chrome extension only activates on the Google Meet domain, and through the use of 'polling'. checks for existence of an end-call button, if exists, which implies that the user is in a meeting, it configures the message listeners.

3.2.2. Observing New Messages

In Google Meet, messages can be witnessed in two specific locations in the browser tab of the meeting. A new message will appear as a message bubble if the chat box is not open, or it will appear directly in the chat box otherwise. A point to note in the design of Google Meet is that the chat box node in DOM is not created until the user explicitly opens it. Prior to it, all messages are delivered as message bubbles.

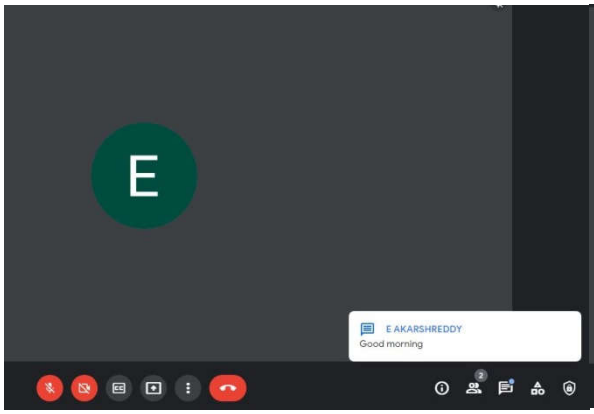


Fig.3.2.2.1: Message Bubble

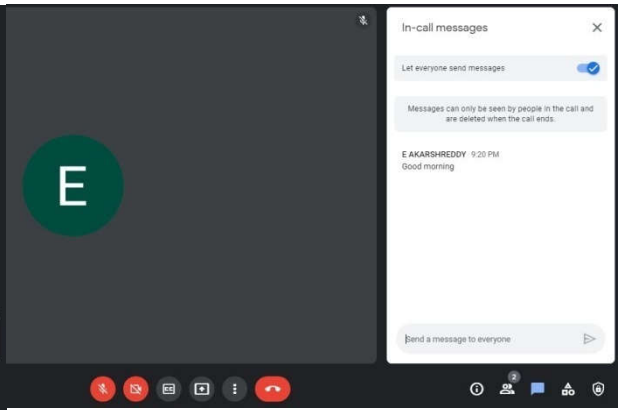


Fig.3.2.2.2: Chat box

3.2.3. Listening to New Messages

It is essential to identify the presence of a new message to capture its contents. It is done by setting up 'listeners' in regions where we can observe a new message(discussed above). These 'listeners' help in acknowledging the event of arrival of a new message and capture its contents. Listening to changes through the concept of polling can be computing intensive, since it repeats the checks of any addition/removal of node at specified intervals of time indefinitely. In complex web apps, DOM changes can be frequent. As a result, there are instances where your app might

need to respond to a specific change to the DOM. For some time, the accepted way to look for changes to the DOM was by means of a feature called Mutation Events, which is now deprecated. The W3C-approved replacement for Mutation Events is the MutationObserver API, a standard JavaScript API, is utilized by the 'listeners' to accomplish the task of observing mutations in respective regions. In addition to being the essential API for this application, it is chosen for the fact that it is a standard JavaScript API. No special/non-standard APIs are used and only standard JavaScript APIs are used to ensure wider compatibility and consistency among various browsers.

MutationObserver API

The MutationObserver API allows you to monitor for changes being made to the DOM tree. When the DOM nodes change, you can invoke a call-back function to react to the changes. MutationObserver allows you to provide a function that is called asynchronously when certain parts of the DOM change, such as adding a child to a node, changing an attribute on a node, or changing the text on a node. As the changes happen, the MutationObserver records them as MutationRecords and then calls a user provided call-back at a later time with all the MutationRecords that are pending.

The basic steps for using the MutationObserver API are:

- Define the call-back function that will execute when the DOM changes
- Create a MutationObserver object and pass the callback function object to the constructor
- Call the 'observe' method. The 'observe' method of the MutationObserver object has two parameters(target, observerOptions). The target is the root of the sub-tree of nodes to monitor for changes. The observerOptions parameter contains properties that specify what DOM changes should be reported to the observer's callback. The second argument is an options object of boolean attributes. Here is a list and description of each:
    - childList: Set to true if additions and removals of the target node's child elements (including text nodes) are to be observed.
    - attributes: Set to true if mutations to target's attributes are to be observed.
    - characterData Set: to true if mutations to target's data are to be observed.

- subtree: Set to true if mutations to not just target, but also target's descendants are to be observed.
- attributeOldValue: Set to true if attributes is set to true and target's attribute value before the mutation needs to be recorded.
- characterDataOldValue: Set to true if characterData is set to true and target's data before the mutation needs to be recorded.
- attributeFilter: Set to an array of attribute local names (without namespace) if not all attribute mutations need to be observed.

- Disconnect the MutationObserver using the 'disconnect' method to halt listening.

3.2.4. Capturing Contents Of New Message

The call-back function attributed to the MutationObserver object receives a array of MutationRecord objects which hold the information of all the mutations that occurred on sub-tree of nodes rooted at target element. The contents of the new message that triggered this call-back can be identified by scrutinizing the MutationRecord objects.

MutationRecord objects have the following properties:

- type – mutation type, one of
  - "attributes": attribute modified
  - "characterData": data modified, used for text nodes
  - "childList": child elements added/removed
- target – where the change occurred: an element for "attributes", or text node for "characterData", or an element for a "childList" mutation
- addedNodes/removedNodes – nodes that were added/removed
- previousSibling/nextSibling – the previous and next sibling to added/removed nodes
- attributeName/attributeNamespace – the name/namespace (for XML) of the changed attribute
- oldValue – the previous value, only for attribute or text changes, if the corresponding option is set attributeOldValue/characterDataOldValue.

If a new message is observed, it implies that a new node was added. Its contents such as participant's name and their message can be acquired from addedNodes of that corresponding MutationRecord.

3.2.5. Generating Desktop Notification

After extraction of necessary contents of the new message, the delivery to the desktop environment is implemented through the use of Notifications API, a standard JavaScript API that has wide compatibility with chromium-based browsers.

Notifications API

The Notifications API allows web pages to control the display of system notifications to the end user. These are outside the top-level browsing context viewport, so therefore can be displayed even when the user has switched tabs or moved to a different app. The Notification interface of the Notifications API is used to configure and display desktop notifications to the user. These notifications' appearance and specific functionality vary across platforms but generally they provide a way to asynchronously provide information to the user. The API is designed to be compatible with existing notification systems, across different platforms.

The following is an essential(non-exhaustive) list of properties of Notification object:

- title – The title of the notification as specified in the first parameter of the constructor.
- body – The body string of the notification as specified in the constructor's options parameter.
- image – The URL of an image to be displayed as part of the notification, as specified in the constructor's options parameter.

Notification event handlers:

- onclick – A handler for the click event. It is triggered each time the user clicks on the notification.
- onclose – A handler for the close event. It is triggered when the user closes the notification.
- onshow – A handler for the show event. It is triggered when the notification is displayed.

- onerror – A handler for the error event. It is triggered each time the notification encounters an error.

Notification API requires the user to 'allow' notifications for that webpage when requested.

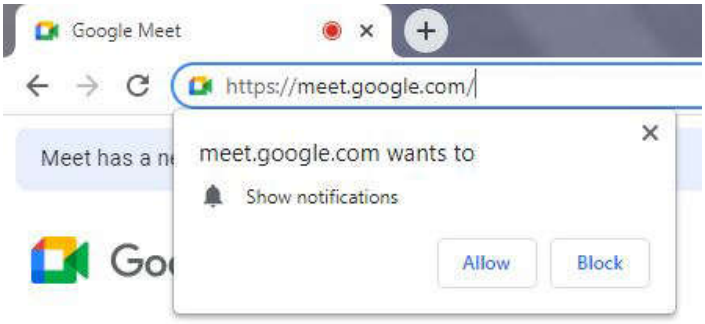


Fig.3.2.5.1: Permission to show notifications

Given the above permissions and that the Google Chrome extension is installed and active. If a participant sends a message in the Google Meet meeting, a desktop notification is generated over any existing screen. For example, if a presentation is being given, any query from any participant would be displayed over it.

Fig.3.2.5.2: Illustration of desktop notification over a presentation screen

Versatile compatibility of standard JavaScript APIs such as the Notification API, the following figure illustrates the look and feel of notifications on various versions of Windows.

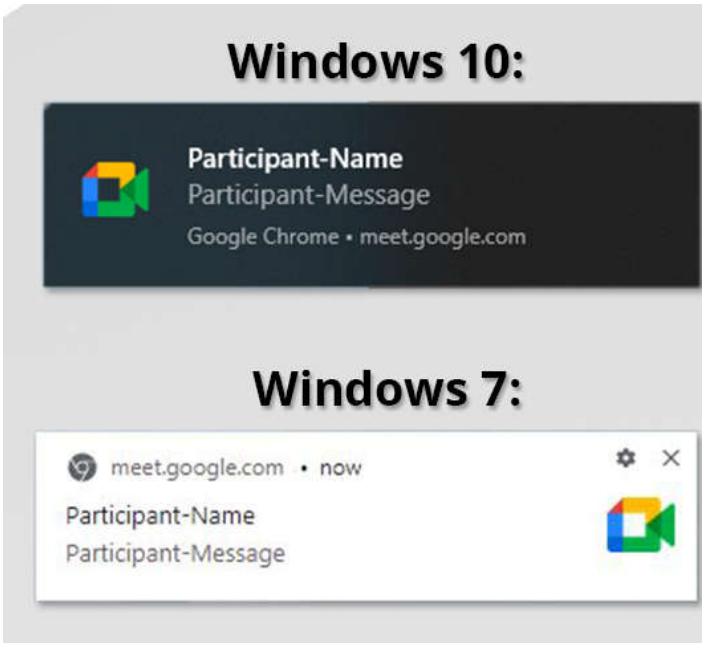


Fig.3.2.5.3: Notification instances on different versions of windows

3.3. Design Methodology

3.3.1. Activity Diagram

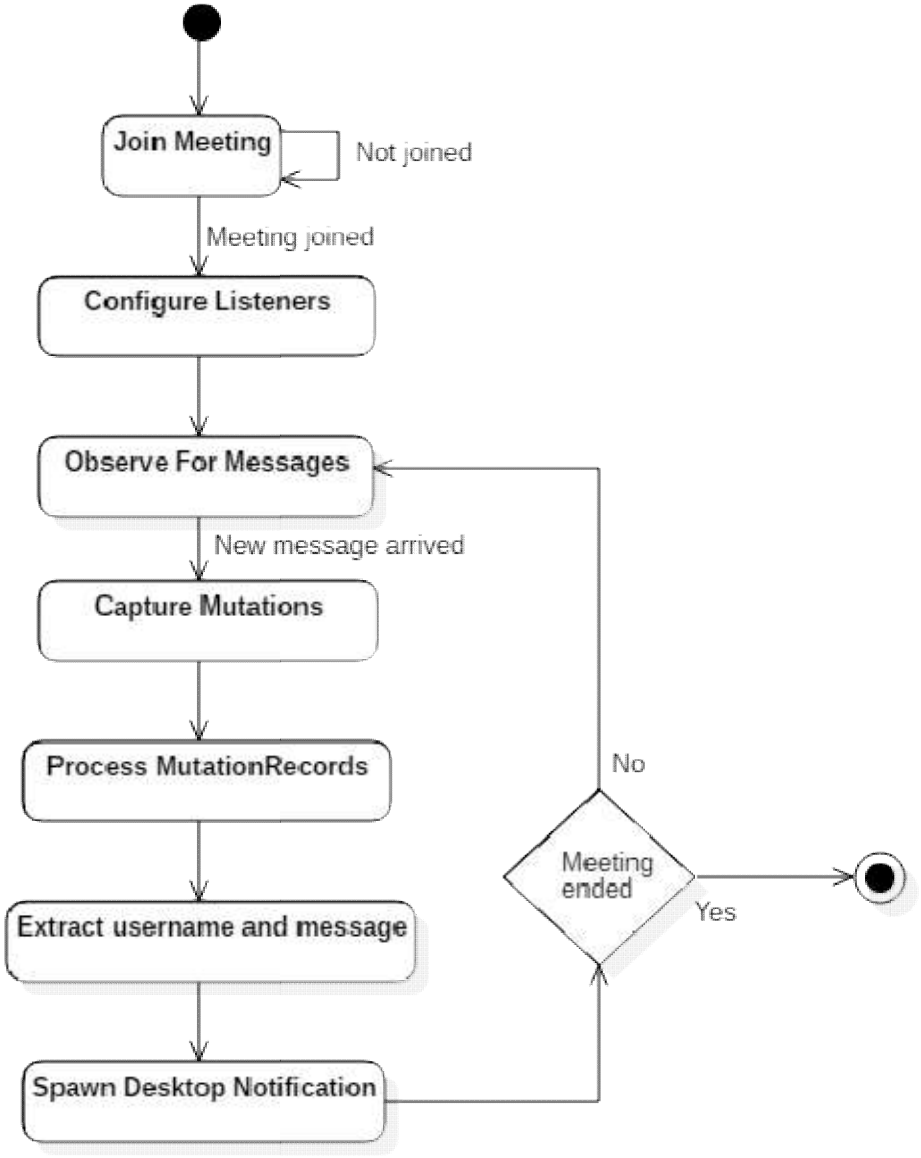


Fig.3.3.1: Activity Diagram – Decision Making and Implementation

Fig.3.3.1 signifies the activity flow in terms of actions and decisions responsible for the application's activity. The activity flow starts with checking if the user has joined the meeting or not.

If the user has entered a meeting, 'listeners' are established at regions where a new message can be observed with the capability to observe for any new message.

When a new message arrives, the associated call-back function of that listener is called by passing list of MutationRecords that contain the information of new mutations that took place in the DOM tree rooted at our target element.

These objects are scrutinized by the call-back function to extract the username and message text. This essential information is used and a notification is generated.

This activity flow is halted once the meeting is ended/exited.

3.3.2. Sequence Diagram

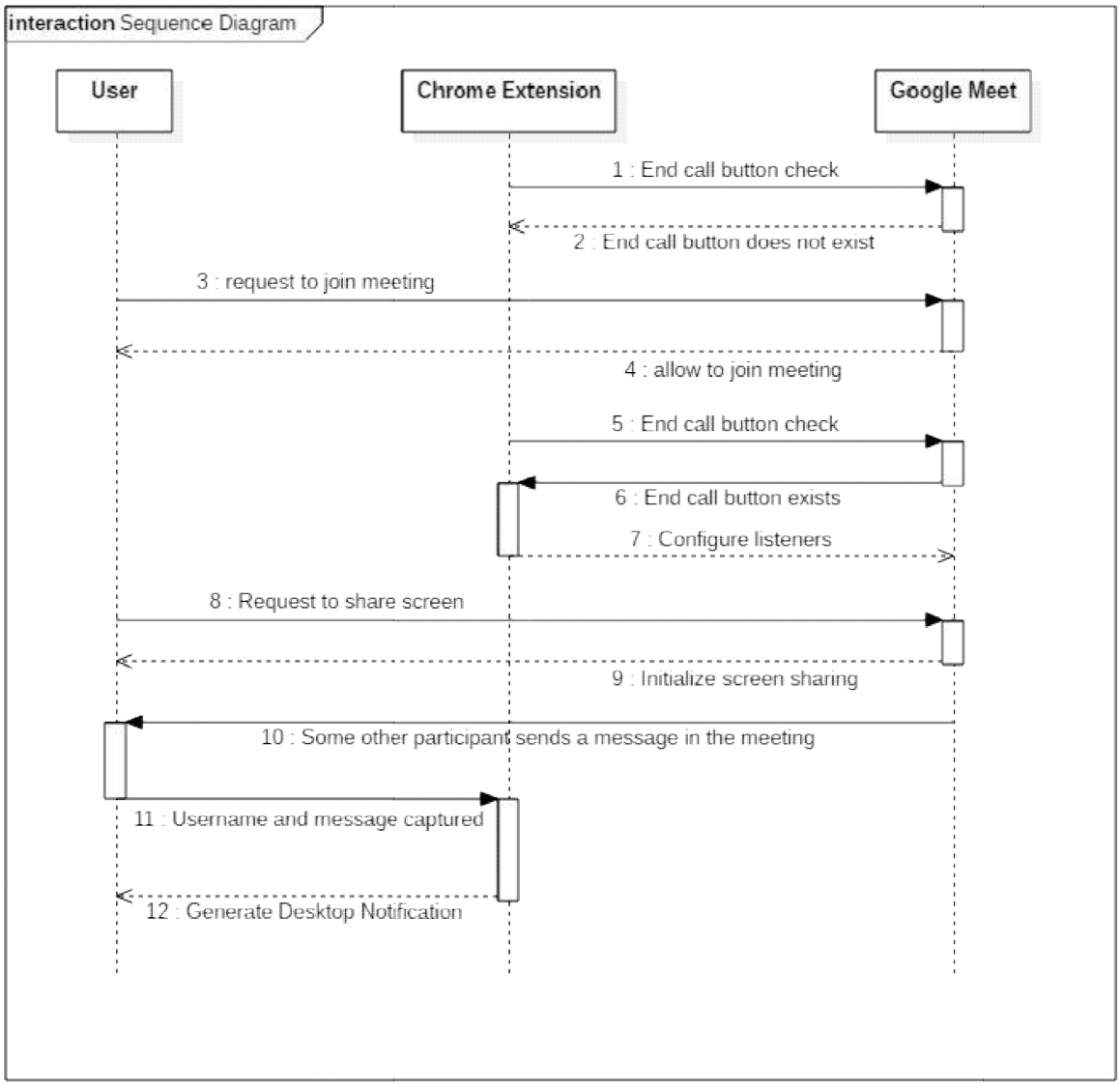


Fig.3.3.2: Sequence Diagram - Extension Interaction and Process Execution

Fig.3.3.2 exhibits the interaction among the user, Chrome Extension and Google Meet system. The 'user' i.e., the participant, joins the meeting. The Chrome Extension uses polling to inspect for the existence of end-call button to check if the user has joined the meeting. Once entered the meeting, the Chrome Extension sets up listeners. On arrival of a new message from other participants, a desktop notification containing the username and their message is generated.

3.3.3. Use-Case Diagram

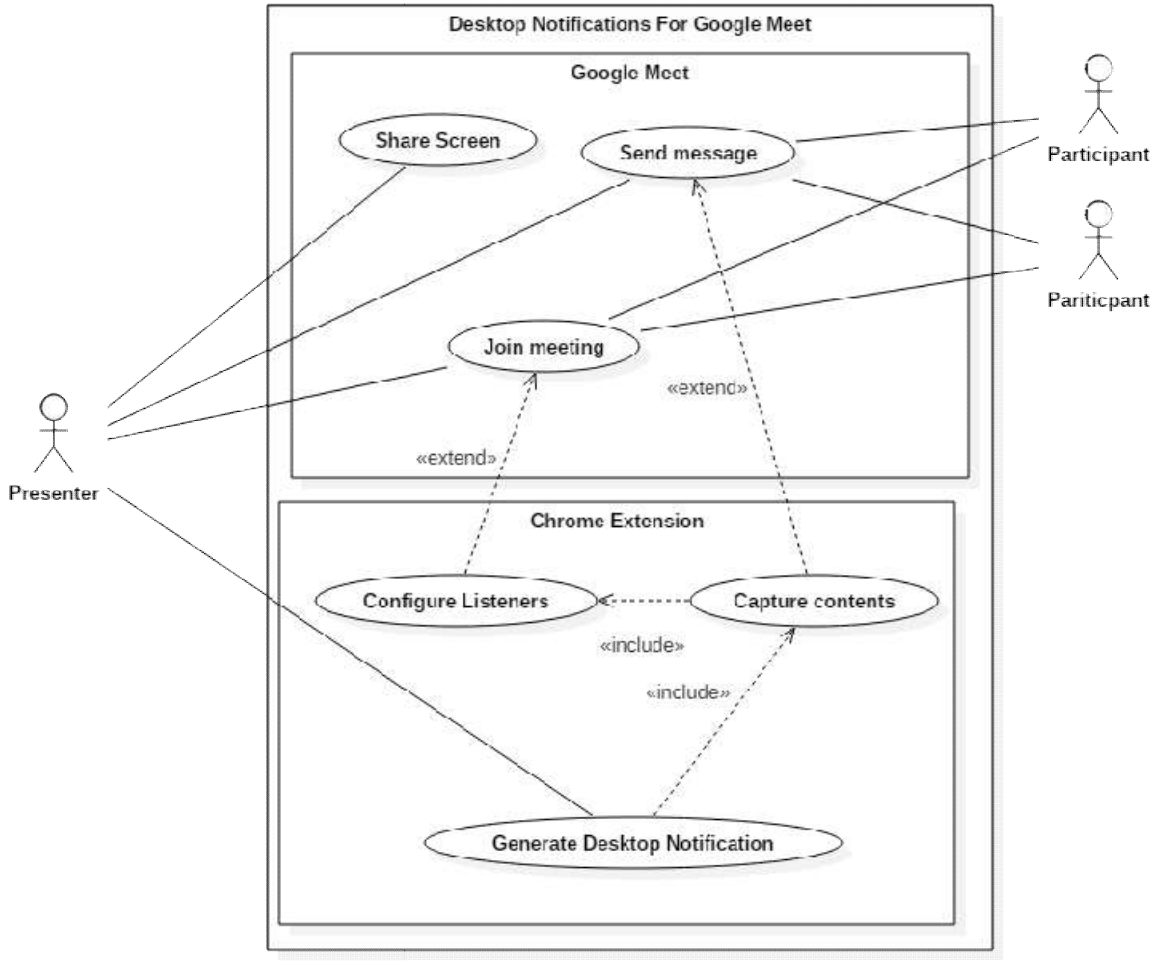


Fig.3.3.3: Use-Case Diagram - Use-Cases and Their Relationships

Fig.3.3.3 illustrates various use-cases, where one participant is assumed to the presenter i.e., sharing their screen. As discussed, Chrome extension configures the listeners when the user has joined the meeting. All the participants are allowed to join the meeting and send messages. The new message contents are captured. A desktop notification is generated to the presenter that forestalls the need to open the browser window to be able to read the new message.

4. Testing including test cases and results

4.1. Desktop Notifications For Google Meet Trail Runs

Chrome Extension testing is an inspection conducted to provide potential users with information about the quality of the product. It is essential to be able to publish on the Google Chrome Web Store since they interest only quality extensions. The system will be subjected to a series of field trials and functional testing where various test cases will be performed to validate the different features of the system.

4.1.1. Desktop Notifications For Google Meet Test Cases

The Chrome Extension has a distinctive architecture that requires cautious testing. The following situations are to be tested:

1. Desktop notification when notifications are allowed
2. Desktop notification when notifications are not allowed
3. Desktop notification when the presenter is on a full-screen application
4. Desktop notification on click
5. Chrome extension pop-up toggle
6. Chrome extension pop-up toggle off and chat-box is opened

S.No.	1
Name of test	Desktop Notification when notifications are allowed
Feature under test	Notification permission
Input	New message in Google Meet
Expected Output	If notification are allowed, desktop notification must be generated
Actual Output	Desktop notification is generated
Remarks	Module is working properly

Table 4.1.1.1: Test case: Desktop Notification when notifications are allowed

S.No.	2
Name of test	Desktop Notification when notifications are not allowed
Feature under test	Notification permission
Input	New message in Google Meet
Expected Output	If notification are not allowed, 'Notification' permission must be requested
Actual Output	'Notification' permission is requested
Remarks	Module is working properly

Table 4.1.1.2: Test case: Desktop Notification when notifications are not allowed

S.No.	3
Name of test	Desktop Notification when the presenter is on a full-screen application
Feature under test	Focus-assist on Windows 10. Focus-assist on Windows 10 blocks desktop notifications when the user is on a full-screen application.
Input	New message in Google Meet
Expected Output	Notification must be generated on a full-screen application when the focus-assist is turned off.
Actual Output	Desktop notification is generated
Remarks	Module is working properly

Table 4.1.1.3: Test case: Desktop Notification when the presenter is on a full-screen application

S.No.	4
Name of test	Desktop Notification on click
Feature under test	Tab focus
Input	New message in Google Meet
Expected Output	Clicking on Desktop Notification must focus the browser tab containing the meeting
Actual Output	Browser tab containing the meeting is focused
Remarks	Module is working properly

Table 4.1.1.4: Test case: Desktop Notification on click

S.No.	5
Name of test	Chrome extension pop-up toggle
Feature under test	Pop-up toggle
Input	Toggle switched off
Expected Output	No desktop notification when toggle is turned off
Actual Output	No desktop notification is generated
Remarks	Module is working properly

Table 4.1.1.5: Test case: Chrome extension pop-up toggle

S.No.	6
Name of test	Chrome extension pop-up toggle off and chat-box is opened
Feature under test	Pop-up toggle, chat-box
Input	Toggle switched off. chat-box is opened.
Expected Output	No desktop notification must be generated but pointers to latest message must be updated.
Actual Output	No desktop notification is generated and pointers are updated correctly.
Remarks	Module is working properly

Table 4.1.1.6: Test case: Chrome extension pop-up toggle off and chat-box is opened

4.2. Results

4.2.1. Google Chrome Extension

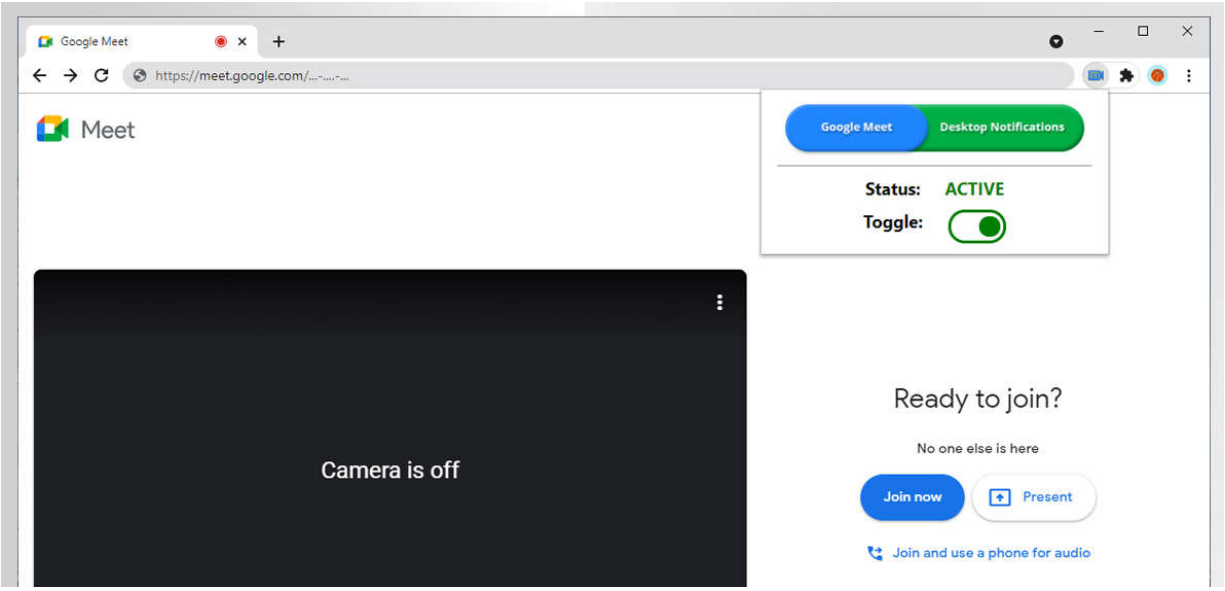


Fig.4.2.1: Chrome Extension Pop-up UI

Once installed on a chromium based browser, it activates automatically when the web page domain is of Google Meet. This is done in view of privacy concerns of extension unnecessarily running on different web pages. When on Google Meet domain, clicking on the extension icon will drop a simple UI(as shown in Fig.) that informs the user about the status and enables them to toggle the functionality. For the first run, if notifications are not allowed, they are requested. Further, the extension does not demand any explicit toggle or interaction every time to be able perform its desired function. It is completely automated to run and generate desktop notifications once notifications are allowed.

4.2.2. Generation of Desktop Notification for a new message

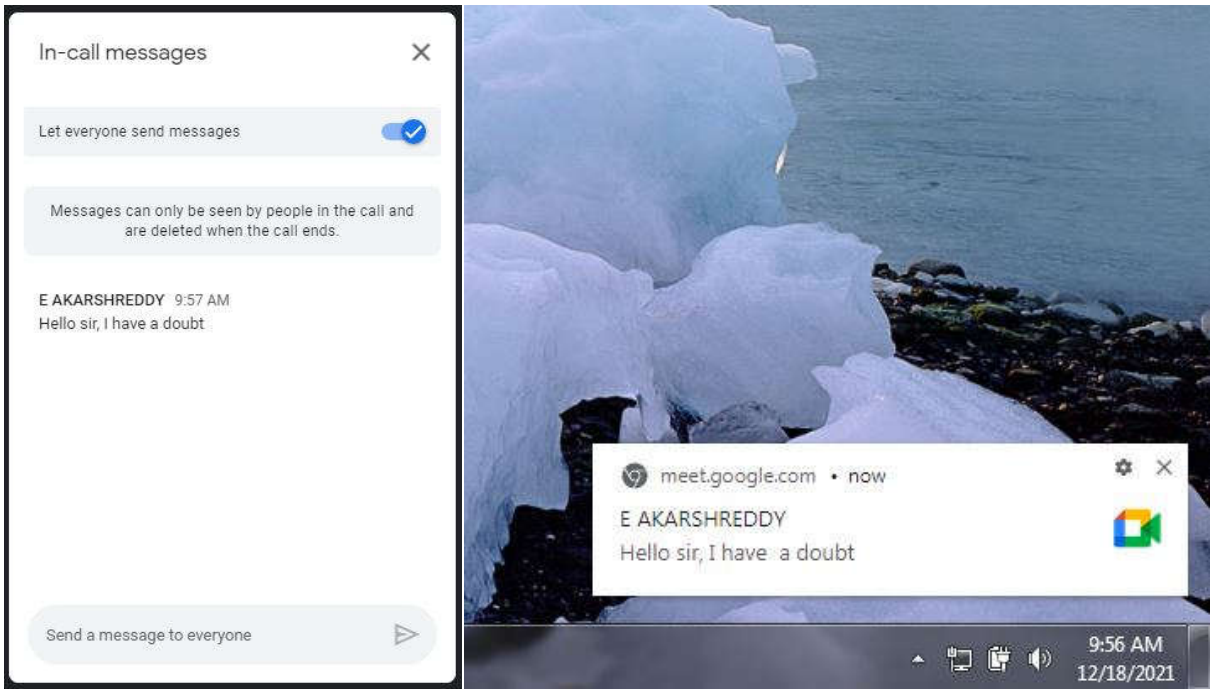


Fig.4.2.2: Desktop Notification for a new message

Fig.4.2.2. illustrates the outcome of the extension. When some participant sends a message in the meeting, its corresponding desktop notification containing the participant's name and their message is generated in the desktop environment that stays for five seconds. When clicked, it opens the browser window of Google Meet allowing faster navigation.

5. Conclusion and Future Scope

5.1. Conclusion

Desktop Notifications for Google Meet is a novel extension that is developed to add an essential feature to Google Meet. It is incorporated as a extension that can be installed on various chromium-based browsers and can be toggled with single click of a button. When a new message arrives, independent of the window the presenter is currently on, the new message is immediately notified to him by directing a notification containing the participant's name and message directly to the desktop environment of the presenter. The presenter need not to navigate to the browser window to view the new message and is notified of the new message immediately.

5.2. Future Scope

This extension is an implementation of a crucial feature that is missing in the current Google Meet system. Google integrating this feature into its system could improve their utility competence since some of its competitors in virtual meeting space such as Microsoft Teams have already incorporated this feature. This extension can be reconstructed for different architectures of non-chromium browsers to enable utilization of this feature on various browsers. It can be further extended to notify raised hands and give the user control on amount of time the notification to be shown.

Bibliography

[1] "Extensions - Chrome Developers", *Chrome Developers*. [Online]. Available: https://developer.chrome.com/docs/extensions/. [Accessed: 15- Dec- 2021].

[2] "MutationObserver - Web APIs | MDN", *Developer.mozilla.org*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver. [Accessed: 15- Dec- 2021].

[3] "Notification - Web APIs | MDN", *Developer.mozilla.org*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/notification. [Accessed: 15- Dec- 2021].

[4] "Publish in the Chrome Web Store - Chrome Developers", *Chrome Developers*. [Online]. Available: https://developer.chrome.com/docs/webstore/publish/. [Accessed: 15- Dec- 2021].

[5] P. Mehta, *Creating Google Chrome extensions*. Apress, 2016.

[6] J. Duckett, *JavaScript & JQuery: Interactive Front-End Web Development*. John Wiley & Sons, 2014.

[7] G. Booch, *Unified Modeling Language User Guide*. Addison-Wesley, 2017.

Appendix-A – Source code

contentScript.js

```
/*Immediately invoked function expression is used to avoid global name collisions */
(function () {
/*checkStart() checks if the user has joined the meeting by testing existence of end call button */
function checkStart(){
        var testElement = document.querySelector("#ow3 > div.T4LgNb > div > div:nth-child(9) > div.crqnQb > div.rG0ybd.xPh1xb.P9KVBf.LCXT6 > div.Kn8SEb > div > div.NHaLPe.kEoTPd > span > button");
/* polling until end-call button exists */
        setTimeout(function(){
                testElement?configure():checkStart();
            },
            1000);
        }

var balloonMessageObserver = null;
var chatWindowObserver = null;
var chatObserver = null;
const listenConfig = {childList:true,subtree:true};
const notifyIcon = chrome.runtime.getURL('googlemeet.png');
var firstRun = true;
var chatWindowOpened = false;
var ToggleButtonState = true;
var balloonMessageHolder = null;
var chatWindowPlaceHolder = null;
```

```
 /* Configure listeners */
function configure(){
/* Efficiency purposes – oneTimeSelector */
        var oneTimeSelector = document.querySelector("#ow3 > div.T4LgNb > div >
div:nth-child(9) > div.crqnQb");
/* Configure MutationObserver for ballon message */
        balloonMessageHolder =
oneTimeSelector.querySelector("div.NSvDmb.cM3h5d.Uir1pc");
        balloonMessageObserver = new MutationObserver(balloonCallBack);
        balloonMessageObserver.observe(balloonMessageHolder,listenConfig);
/* Configure MutationObserver to observe creation of chat-box object */
        chatWindowPlaceHolder =
oneTimeSelector.querySelector("div.R3Gmyc.qwU8Me");
        chatWindowObserver = new MutationObserver(chatWindowObserverListener);
        chatWindowObserver.observe(chatWindowPlaceHolder,listenConfig);
}

/* Check if chat-box object is created */
function chatWindowObserverListener(mutations,observer) {
        if((mutations[0].addedNodes[0].getAttribute("data-tab-id"))=="2")
        {
            try{
                    balloonMessageObserver.disconnect();
            }catch(ex){}
            balloonMessageObserver = null;
            chatWindowObserver.disconnect();
            chatWindowObserver = null;
            chatWindowOpened = true;
            chatCheckReady();
        }
```

```
}


function balloonCallBack(mutations, observer){
        if(mutations[0].addedNodes.length==1)
        {
            try{
                var name = mutations[0].addedNodes[0].querySelector("span > span > div
> div.UgDTGe").innerText;
                var message = mutations[0].addedNodes[0].querySelector("span > span >
div > div.mVuLZ.xtO4Tc").innerText;
                spawnNotification(name,message);
            }
            catch(ex)
            {
            }
        }
}
/* Check instantiation of chatbox */
function chatCheckReady() {
        var chatWindowSiblingMessage = document.querySelector("#ow3 > div.T4LgNb >
div > div:nth-child(9) > div.crqnQb > div.R3Gmyc.qwU8Me > div.WUFI9b > div.hWX4r > div
> div.z38b6.CnDs7d.hPqowe > div.v8W0vf");
        setTimeout(function(){
        chatWindowSiblingMessage?chatConfigure(chatWindowSiblingMessage):chatChec
kReady();
        },200);
}
var parentElement = null;
var element = null;
var current = null;
```

```
/* Configuring chat box – Adjusting pointers to latest messages in chat-box */
function chatConfigure(chatWindowSiblingMessage) {
        parentElement = chatWindowSiblingMessage.parentNode;
        chatObserver = new MutationObserver(chatCallBack);
        if(ToggleButtonState == true){
            chatObserver.observe(parentElement,listenConfig);
        }
        current = parentElement.lastChild;
        function chatCallBack(mutations,observer) {
            if(mutations[0].addedNodes.length==1)
            {
                    if(current.nextSibling){
                            current = current.nextSibling;
                    }
                    var name = current.firstChild.firstChild.innerText;
                    if(name!="You")
                    {
                            var message = current.lastChild.lastChild.innerText;
                            spawnNotification(name,message);
                    }
            }
            else if(mutations[0].removedNodes.length==1){
                    current = parentElement.lastChild;
            }
        }
}
/* Function that creates a desktop notification */
function spawnNotification(title,body) {
        var options = {
            body : body,
```

```
            icon : notifyIcon
        };
        var notification = new Notification(title,options);
        notification.onclick = function(){
            window.focus();
            notification.close();
        };
        setTimeout(function(){
            notification.close();
        },5000);
}
/* Function that is called when extension is toggled */
function pauseAll(){
        if(chatWindowOpened){
            chatObserver.disconnect();
        }
        else{
            balloonMessageObserver.disconnect();
        }
}
/* Listens to messages from background script */
chrome.runtime.onMessage.addListener(
        function(request, sender, sendResponse) {
            if(request.checkStatus == "checkStatus"){
                    sendResponse({"buttonState":ToggleButtonState})
            }
            else if(request.turnMessage=="turnOff"){
                    ToggleButtonState = false;
                    pauseAll();
            }
            else if(request.turnMessage=="turnOn"){
```

```
                ToggleButtonState = true;
                if(chatWindowOpened){
                        current = parentElement.lastChild;
                        chatObserver.observe(parentElement,listenConfig);
            }
            else
            {
        balloonMessageObserver.observe(balloonMessageHolder,listenConfig);
                chatWindowObserver.observe(chatWindowPlaceHolder,listenConfig);
            }
        }
})
/* Starting point */
if(firstRun){
        firstRun = false;
        chrome.runtime.sendMessage({"firstMessage":"activate_icon"});
        checkStart();
}
})();
```

Appendix-B – User manual

To install the extension, please head over to the Google Chrome Web Store. Find the chrome extension using the keywords 'Desktop Notifications for Google Meet AkarshR'.

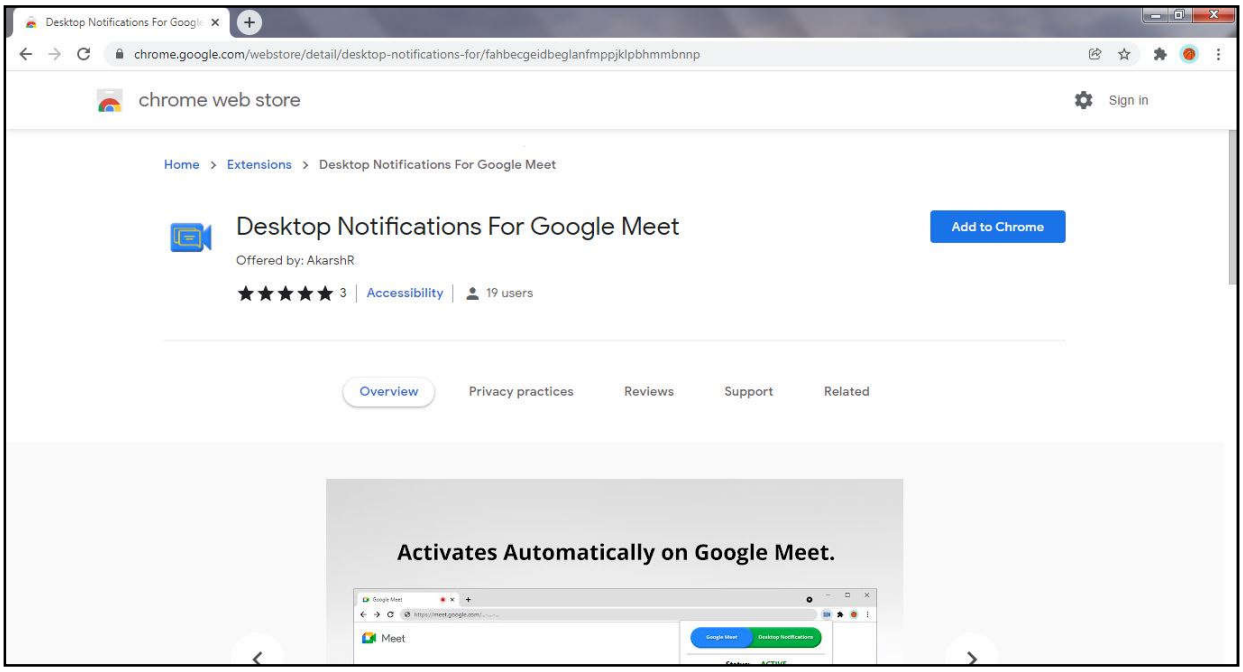


Fig.B.1: Chrome Web Store

Install the extension on any chromium based browser(Google Chrome, Microsoft Edge) by clicking on 'Add to Chrome'.

This extension requires the notification permission, please 'allow' when prompted.

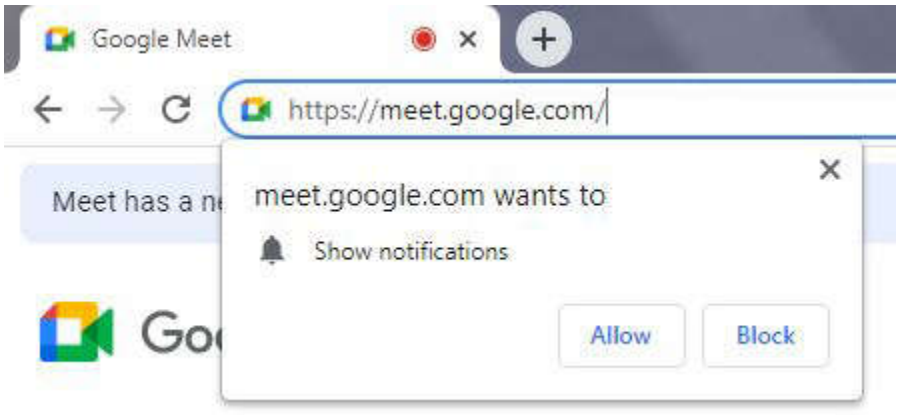


Fig.B.2: Allow Notifications

The functionality is automatically activated on a Google Meet domain. You can toggle its functionality to your requirement.

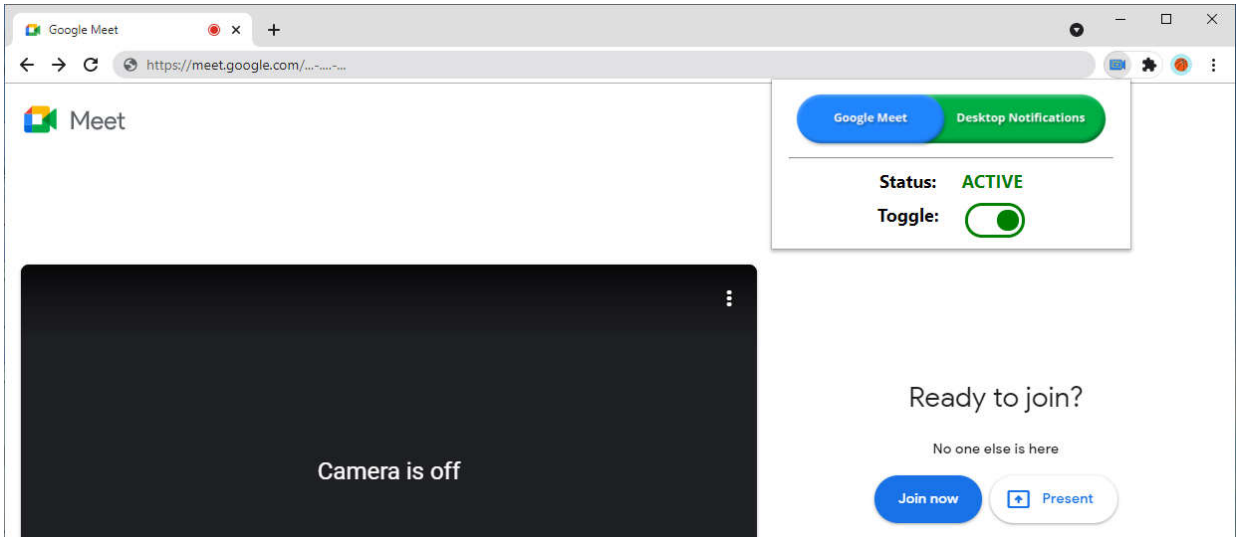


Fig.B.3: Chrome Extension Pop-up

For Windows 10 users,

Windows 10 uses 'Focus Assist' to block notifications when you're on a full screen application.(Could be your full screen presentation too). If you want notifications even when you're on some full screen application. Please follow the instructions:

1. Please goto Settings > System
2. Search for "Focus Assist"

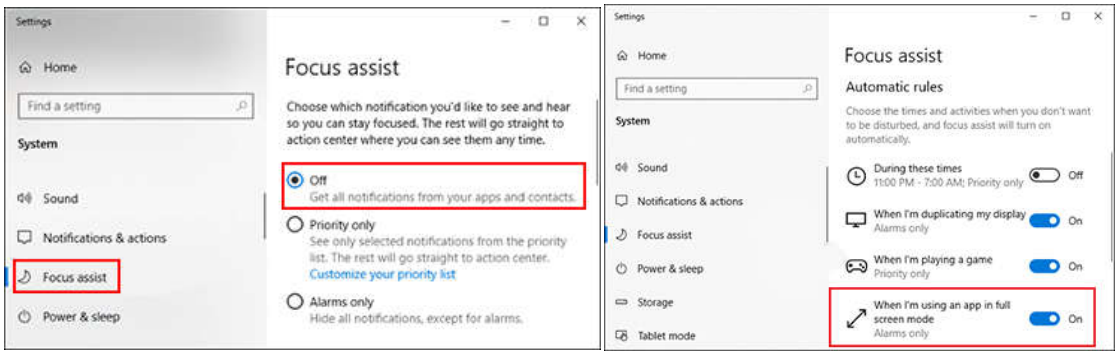


Fig.B.4: Turn off focus-assist