CS433: Mid-Term Project Report

Topic: Location Restricted Form

Darshi Doshi (21110050)
Nidhi Kumari (21110139)
Rohit Raj (21110179)
Animesh Tumne (21110227)

Approaches to the Problem

While ideating, we came upon various ways to advance into the question. We had thought of three approaches to the problem at hand -

1. IP address filtering

Filtering the incoming responses by checking upon their source IP. The Database maintenance of the MAC and IP addresses of each router was ideated. The application will check if the client IP is sourced from the desired router or not, and will take appropriate steps.

2. Local hotspot network

The form owner runs our application on his local hardware, and provides clients with a hotspot network to connect to, from which they can then access the form link shared with them.

3. Geolocation capturing, recognition and filtering

The form we design will capture the current location of the client, and determine whether the client should be granted access or not based on their location.

Drawbacks/Benefits of each approach

After discussion with the course TAs and prof. Sameer, we came to the conclusion that the first and second ideas were infeasible to implement, taking into account the short time at hand. The detailed discourse is provided below:

1. IP address filtering

The idea felt feasible when we discussed it as a group. We collectively thought that each classroom had its own router and, thus, its own IP address. Later, through our conversation with the TA, Mr. Dhyey Thummar, we came to know that the classrooms do not have their own routers but a WAP (Wireless Access Point). A router is allocated

to each academic block instead of each classroom, thus rendering any classroom-based filtering impossible via this method.

2. Local Hotspot Network

After the aforementioned discussion, we came to the second idea - based on a local hotspot network. But, once again, meeting with prof. Sameer Kulkarni, we found it was infeasible to implement, as he informed us that the work we would have to do to implement this idea would be very tedious and enormously time-consuming. He also suggested meeting with the TA, Mr. Hari Hara Sudhan, who then guided us on how to implement our project.

3. Geolocation capturing, recognition and filtering

As compared to the other approaches that we thought of, geolocation capturing was the most feasible solution in terms of both implementation and the methodology. We discussed this idea in detail with the TA, Mr. Hari Hara Sudhan and ideated further on this topic.

There was one way of determining the location of the client (suggested by Mr. Dhyey) - we check its distance from the host machine, and set a limit to how far a client can be. But, after discussing this idea amongst ourselves, we found it challenging to implement on a large scale basis, and this idea would essentially fail in large spaces, such as the Jasubhai Auditorium, since it would take into account a circular area which may include the outside foyer spaces, and leading to inaccuracy.

The basic idea that we finalised is that the client will be able to access the form on our server only when they are within the defined boundary (e.g. of a rectangular room). We came to realise that GPS location does not give the exact coordinates of the user, rather it is accurate within the range of 3m to 7m. This issue is solved by padding the boundary by some amount of the actual area of the classroom that we wish to consider.

Workflow

The final idea, deriving from all the discussions and meetings, was decided to be based on geolocation filtering (geofencing in technical terms). A few options as to how we could approach this - single point distance method (distance of client from the host device/hardware), geo-boxing (the user location is checked to be inside a rectangular area), multiple point distance (distance from two or more host hardwares is checked) - were thought by us, but the geo-boxing method was chosen as the final approach (reason described above, implementation described below).

Work Proceedings

Work Exploration and Implementation:

The final implementation was approached in various ways, some were discarded due to many perilous shortcomings. The finalised method of geoboxing has been expanded into the following described web application.

The basic design and work-flow of the web application:

1. Prompt the user to grant location access for the web-application. Mandatory for the end-user (client, e.g. the student) to grant it.
2. Client's location is fetched using the Geolocation API in the form of values like latitude, longitude and altitude - as geographical coordinates.
3. Client coordinates are used as a filtering criteria based on defined boundary coordinates, including the altitude. The altitude value adds another layer to the filter by allowing the users present only at specific floors of a building.
4. Client in allowed region/room- redirect to the forms page where the user can interact with the form (write their responses).
5. After submitting the form, the page displays a "successful" message.
6. Client not in allowed region/room - redirect to a page specifying an "Access Denied".

The Geolocation API successfully works in fetching the required data for a user's geographic location. We are also able to redirect the client correctly based on their coordinates. The next step was to implement the forms feature in our application. We tried different approaches for the same as described below:

1. **Google Apps Script - discarded due to excessive unnecessary work**
    a. Google Apps Script used to create and interact with Google Forms. Made possible by the functions defined in the app script document having access to methods which directly interact with the created form.
    b. We can call these functions from our backend server to fill in the fields in the created form via the Apps Script.
    c. We can access the created form from the form URL. But we still need to implement the forms into our webpage without exposing the original link. This can be approached without using the Apps Script, as described ahead.

2. **Embed the Google form into our webpage - discarded due to insecurities**
    a. On the Google Forms interface, after creating the form, we can obtain the "embed HTML" element for the same form. This element when pasted into our HTML structure embeds the form as a window of a specified size. The user can then interact with the form from this internal window of the webpage, without needing to get redirected.
    b. But, inside the embedded window, the form requires the user to sign-in for a QUIZ, clicking on which redirects to the original form link on the browser, which is again undesirable.

c. Without signing-in, a Google Form QUIZ can't be accessed. If the user is already signed-in on the browser, they don't have to sign-in again and can directly access the form. We thought of using this fact in order to avoid such redirects by implementing a sign-in (auth) logic on our client application itself, but this will create a headache for the allowed users to enter their credentials, instead of a direct access to the form.
d. However, even after applying the above logic in our code (point a.), the goal of hiding the original forms link won't be reached. Any tech savvy user could easily cheat the system by using the inbuilt developer tools in the browser to access the original Google Form link and share it directly to anyone, regardless of location.
e. Thus, form embedding as described above is not a feasible approach.

3. **Custom Frontend - Finalised Approach (Future objectives, tentative)**
    a. We will be designing a custom form UI corresponding to the created Google form.
    b. For implementing a custom frontend we need an organised structure of all the fields in the form which can be parsed by our program. This is provided by a JSON object fetched from a created Google Form using the *forms.get()* API provided under the Google Forms API.
    c. The user interacts with the presented form on our webpage and is able to submit the form.
    d. The defined "submit" button calls the google forms action URL. This URL submits the user responses into the Google Forms database available to the form-creator (host, e.g. the Professor). This URL is obtained from the developer tools under the "Elements" section.
    e. This approach resolves the issues faced in the above methods.

Implemented Code (Explanation)

Link to our Github repository: https://github.com/rohitt-raj/Location-Restricted-Form
For simplicity, we are only taking the latitude and longitude of the user, and then displaying them onto the webpage as a check for the Geolocation API. The values are then passed into the filtering logic. Here, we have assumed that the classroom comes as a rectangular fit under the longitude and latitude axes, hence the location can be verified by just a two-coordinate system – the coordinates of the diagonal opposite corners (this function is inherently flawed, needs changes). For testing, we have taken the Kyzeel hostel's geolocation address (two coordinates) from Google Maps. The user is allowed to access the form if they lie inside the defined rectangle. The main google form is accessed by redirection to another page (Form/index.html) using *window.location.href*.
We have also assigned a 5 seconds window before the form opens just to give some time for us to confirm the fetched location is indeed our true location. The "Form" folder will contain the implemented logic (custom frontend) for the user to access the form.

Future Work

General work:

- Design the frontend layout of the form to create a user-friendly and visually appealing interface that can be used conveniently.
- Integrate the application with the Google API to facilitate the submission of forms through our form.
- Develop a feature that allows professors (hosts) to input their original Google Form link and generate a new link with ease, which can then be distributed amongst the students (clients) to access. Or, design a new interface that allows the hosts to input their questions and answers, and in the end receive a link that can be distributed to the clients.
- Ensure that the new link generated embeds all the location-based access control features we have designed. This includes geolocation filtering, and accurate access control.

Geoboxing implementation:

- Initially, we will predefine the four coordinates for each classroom that we will be considering for our implementation. We will be padding the rectangle formed by these coordinates by some distance (approx 5m).
- Next, we will find the coordinates of the user and apply the algorithm to find whether the coordinates lie inside the padded rectangle or not.
- Assumption: Since the size of the room is very small as compared to the size of the Earth, all our calculations will be based on the assumption that the rectangular room lies in a 2D plane, i.e. the angular distance between any two coordinates is zero.
- Algorithm: Since the coordinates are in the form of degree of latitude and longitude, we will first convert them into the coordinates in XY Plane. Then, consider the coordinates of the padded rectangular room as A, B, C, and D. Consider the coordinates of the user as M. Hence, the point M must satisfy the following two conditions -

  $$0 < AM.AB < AB.AB \text{ and } 0 < AM.AD < AD.AD$$

  where '.' means the dot product of the two lines.
- The final implementation of this above algorithm will be approached using various functions which would take input the client's location and determine the actuality of location - whether the client is inside the allowed region or not, and then thus take further steps.

Finalised Server-side Program (Code):

1. Before being deployed for the end-users, the server needs to know the allowed location and form details (fields). So, the server will require the form-creator (e.g. the Professor) to provide the following information:
    a. Allowed location in terms of block and room numbers (e.g. AB 7/208)
    b. URL of the Google Form which the Professor created to be used by our web-application

2. The server will parse the provided room location details and select the corresponding latitudes, longitudes and altitude value of the room to be used in the location filter algorithm.
3. The Google Form link provided by the form-creator is utilised by the *forms.get()* method to fetch the corresponding JSON object containing all fields (questions, options, etc). This JSON object is parsed by our frontend algorithm to generate the custom form UI for the user.
4. The user fills the presented form and submits it. The user then gets a successful message for the same.
5. The *submit* button calls the forms action URL (described earlier) which submits the responses into the forms database visible to the form-creator.

Path To Completion (Tentative Timeline)

Week of 16th-22nd October & 23rd-29th October: FrontEnd/BackEnd development

The first two weeks will be spent on developing the front end environment and the backend that supports it. We will test our works locally during this time i.e. on local hardware, to ensure completeness and to debug if the need arises. The code will be tested rigorously to make sure that no shortcomings are observed that make the location restriction is bypassed.

Week of 30th October - 5th November: Code/Deployment

The third week (from now) will be reserved for deployment, but if a need arises we may also need to continue the testing phase (as described above) in this week too.
We will (eventually) need to deploy the form code on (atleast) a university-wide basis, thus making sure that what we have implemented is working and helpful to the faculty on campus. The code also has to be bug-free to ensure no unknown complications arise on deployment, thus we would also need to test the code beforehand on a local deployment basis.

Week of 6th-12th November: testing/modification

The last week will be mainly spent on testing our website for different locations. We would mainly correct the logical errors that arise upon considering various cases. After successfully testing it for different classrooms, we will further modify our website such that it tracks the location at every few intervals (say 30 sec) and exits upon finding the location outside the determined area of the user to access the form.

Acknowledgements

We would like to thank the TAs for their continued guidance throughout this project along with constant support and encouragement to proceed, and steady help whenever we needed. We would also like to express our sincere appreciation to our esteemed professor, prof. Sameer G Kulkarni, for introducing us to this topic and helping us reach our goal as a team.

References

1. https://www.w3.org/TR/geolocation/
2. https://www.youtube.com/watch?v=0udw0nol6Po
3. https://developers.google.com/apps-script/reference/forms
4. https://developers.google.com/forms/api/guides
5. https://math.stackexchange.com/questions/190111/how-to-check-if-a-point-is-inside-a-rectangle
6. https://www.gps.gov/systems/gps/performance/accuracy/