Web - UI development

IoT Security Framework[1]

August 2019

AUTHOR:
Akash Gupta
(CERN Openlab Summer Student)

SUPERVISOR:
Pascal Oser
(Computer Security Team)

[1] The name of the framework is kept confidential for research purposes.

PROJECT SPECIFICATION

The IoT security framework is a computer security platform designed to assess the risks of various heterogeneous IoT devices. The framework is currently being developed at CERN and analyses different IoT devices connected to CERN's General Purpose Network (GPN). The GPN mostly consists of devices that are accessible to everyone at CERN and are meant for general use, but currently are also used for IoT devices like, for example, printers, routers and CCTVs. Some of these devices are also used in the large experiment networks hosted at CERN.

The goal of this project is to design a web user interface for the existing IoT security framework with the following objectives in mind

a) Improve the user experience;
b) Separate the user interface from the back-end services;
c) Make the application responsive, modular and open to extensions;
d) Test and evaluate the latest technologies for web–UI development.

The student will get familiar with a full cycle of project development: from analysing and getting the user requirements to the implementation and deployment of a final solution.

ABSTRACT

The objective of the project was to create a web user interface for the existing IoT security framework that displays the risk of every IoT device used in the GPN at CERN. The framework allows CERN users to have a security check on their personal as well as professional devices and also provide necessary remedies to improve their device risk score.

One of the main goals was to develop a web-interface that is user-specific, modular, responsive and open to extensions. The project is also part of a research study based on IoT device security.

The first part of the report includes an introduction to CERN's Openlab summer student programme, the Computer Security Team in the IT department and the development team. The main part of the report contains an introduction to the project, requirements and system architecture. The last chapter includes results and conclusions for the project.

ACKNOWLEDGEMENT

I want to express my sincere thanks to my supervisor Pascal Oser and the computer security team for giving me helpful tips and insights on my project, and making me feel very welcomed during my 11 weeks as a summer student at CERN. Also, I want to thank CERN Openlab for giving me the opportunity to join the great environment at CERN, and work on interesting and challenging topics.

During the programme, I got valuable experiences and knowledge that I can use in my future studies and career. The programme has also given me many good memories and friends from all around the world. I am very grateful for the opportunity and I truly recommend it for others! The programme has exceeded all my expectations, and it has been one of the most amazing and memorable summer!

TABLE OF CONTENTS

A. CERN OPENLAB SUMMER STUDENT PROGRAMME

I. Introduction 08

II. The computer security team 08

III. The development team 09

B. PROJECT DETAILS

I. Introduction 11-12

II. Requirements

i. Landing page 13-14

ii. Device selection page 14

iii. View for non-technical users 15

iv. Views for technical users 16-17

III. Toolkit used 18-19

IV. Architecture

i. User log-in and device assessment query 20

ii. Backend processing 21

iii. Getting results 22

C. RESULTS

I. User log–in and device assessment query 24-25

II. Getting results for a requested scan 26-27

D. CONCLUSIONS

I. Problems faced and solutions 29-31

II. Summary 31

LIST OF FIGURES

Figure No.	Caption	Page No.
1	Total number of devices at CERN	11
2	Landing page	13
3	Device selection page	14
4	View for non-technical users	15
5	Default view for technical users	16
6	Elongated view for technical users	16
7	User logs in and generates a query	20
8	Backend processing	21
9	User getting results from database	22
10	User making request with CURL command	24
11	CERN SSO(Single–Sign On)	25
12	IoT security framework landing page	25
13	User getting results using CURL command	26
14	User getting results from the framework	26
15	Results for non–technical users	27
16	Results for technical users	27
17	Using React Hooks	30
18	CORS error	30
19	Using NGINX as proxy server	31

CERN OPENLAB SUMMER STUDENT PROGRAMME

1. Introduction

The CERN Openlab summer student programme is a yearly programme where around 40 students from all over the world come to CERN to work on different research projects based on cutting edge technologies for two months during the summer. CERN Openlab[2] is a public-private partnership, allowing CERN to collaborate with leading ICT(Information and communications technology) companies like Google, Oracle, Intel and others. The CERN Openlab summer student projects are based on this collaboration, which makes an interesting and unique package for the programme.

Besides working on their projects, the CERN Openlab summer students also get the opportunity to attend interesting IT lectures on relevant topics given by experts in different fields like Artificial Intelligence, Quantum Computing, Neuromorphic Computing, Security, etc. They also get to visit the CERN experimental areas and particle accelerators like the Compact Muon Solenoid (CMS) and ATLAS. Other highlights of the programme are the visits to external companies, for instance, IBM Research (Zurich, Switzerland) and Open Systems and universities like ETH Zurich.

2. The Computer Security Team

The summer student Akash Gupta worked in the Computer Security Team at CERN for a duration of 11 weeks of the CERN Openlab summer student programme. The mandate[3] of the Computer Security Team is

a) to define rules and conditions that apply to the use of CERN's computing facilities.
b) to recommend hints and practices and provide training courses to help CERN users get the necessary knowledge to improve computer security.
c) to conduct awareness presentations discussing general risks, past incidents and suitable solutions.

Besides this, the Computer Security Team has also deployed various sophisticated protection and detection mechanisms to ensure a secured network environment.

[2] https://openlab.cern/about-us
[3] https://cern.ch/security/home/en/index.shtml

3. The development team

The development team for the project consists of CERN Openlab summer student Akash Gupta, and his CERN supervisor, Pascal Oser.

Name	E-mail address	Role
Akash Gupta	aksgupta3697@gmail.com	Web developer
Pascal Oser	p.oser@cern.ch	Supervisor

Akash Gupta is an undergraduate student of Thapar Institute Of Engineering and Technology, India. As a summer student, he worked on developing the web-UI for IoT security framework.

Pascal Oser, is a PhD student at Ulm University, Germany. He is currently working at CERN in the field of IoT Security. As a supervisor, he has provided necessary guidance and has helped Akash for the whole duration of 11 weeks of the Openlab summer student programme.

PROJECT DETAILS

1. Introduction

The European Organization for Nuclear Research (CERN) is home to the largest particle physics laboratory in the world. It has around 2,500 scientific, technical and administrative staff members[4] working at CERN and a community of more than 17,500 users hosted on the organization's network. Users connect their work as well as personal devices to use CERN's computing facilities for different purposes.

CERN has a large network consisting of two main categories – the General Purpose Network (GPN) and the Technical Network (TN). With approximately 384,000 registered devices and 3000 IoT devices on the network, CERN is no stranger to cyber-attacks. Some of these devices and systems collect a lot of data, for example, data from large experiments, general information, etc.

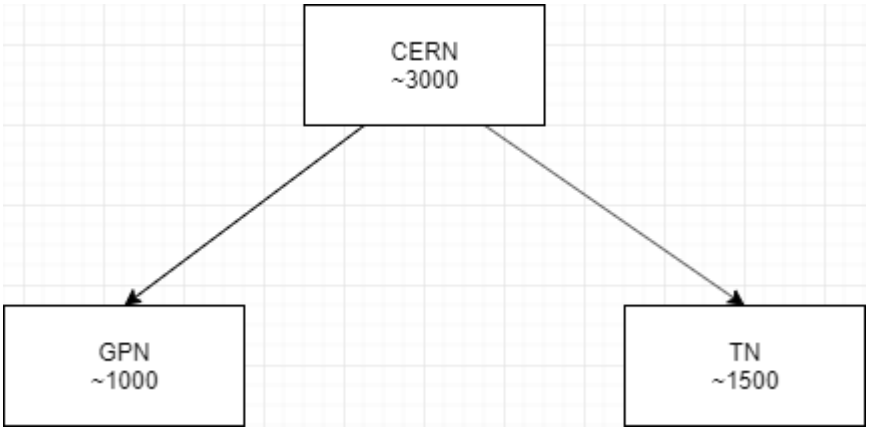


Fig 1. Total number of IoT devices at CERN[5]

As IoT devices are often task-specific, these embedded devices do not have much resources and functionality. In most cases, these devices are optimized to minimize processing cycles and memory usage and do not have extra processing resources available to support traditional security mechanisms.

[4] https://home.cern/about/who-we-are/our-people
[5] This is approximate data only estimated from past experimental tests.

As a result, standard PC security solutions will not solve the challenges of embedded devices. In fact, given the specialized nature of embedded systems, PC security solutions will not even run on most embedded devices.

The IoT security framework has been designed to assess the risks of all the IoT devices connected to the GPN at CERN. The framework outputs device risk score showing the security risk of the IoT device and also provides user with various remedies to improve the device's risk score.

The framework has been developed in two parts -

a) IoT security framework backend[6]

The already existing backend server that consists of 3 main processes:

1) Device identification
2) Vulnerability enrichment
3) Risk scoring

a) Web–UI

The web user interface for the framework developed with following objectives in mind:

1) Trigger new scans for single IPs and subnets.
2) Search for the security level of already scanned devices by filtering for category, manufacturer, model, firmware version
3) Show the device overview page.
4) Show the security results for identified devices for non-technical users.
5) Show the security results for identified devices for technical users.
6) Add visualizations to the metrics.

[6]**NOT PART OF THE PROJECT**

2. Requirements

The initial step of the project was to define the user requirements for the web-UI. Various mock-ups were created and discussed allowing the team to define the interface views and structure.

Below we list the mock-ups consisting of different requirements for web pages.

a. Landing page

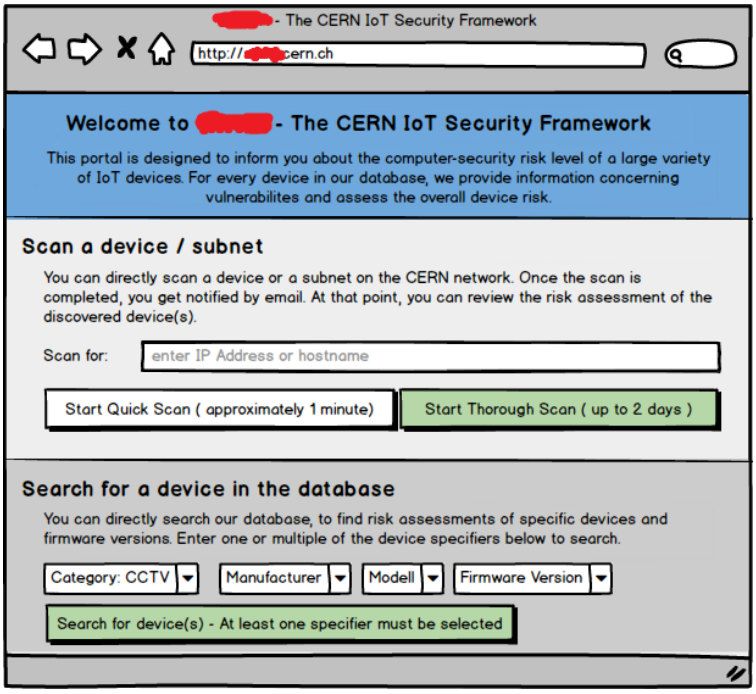


Fig 2. Landing Page

Components:

1. Radio Buttons:
   1.1 IP Address/Hostname
   1.2 Subnet
2. Search Bar: to input IP Address/Hostname of the device.
3. Buttons:
   3.1 Start Quick Scan: to initiate a quick scan that takes less time.
   3.2 Start Thorough Scan: to initiate a full scan that takes more time.
   3.3 Search for device(s): to search for a device from the database with the selected device specifiers.

4. Drop–Down List:
   4.1 Category: device category
   4.2 Manufacturer: name of the manufacturer
   4.3 Model: device model number
   4.4 Firmware Version: version number of the current device firmware

b. Device selection page

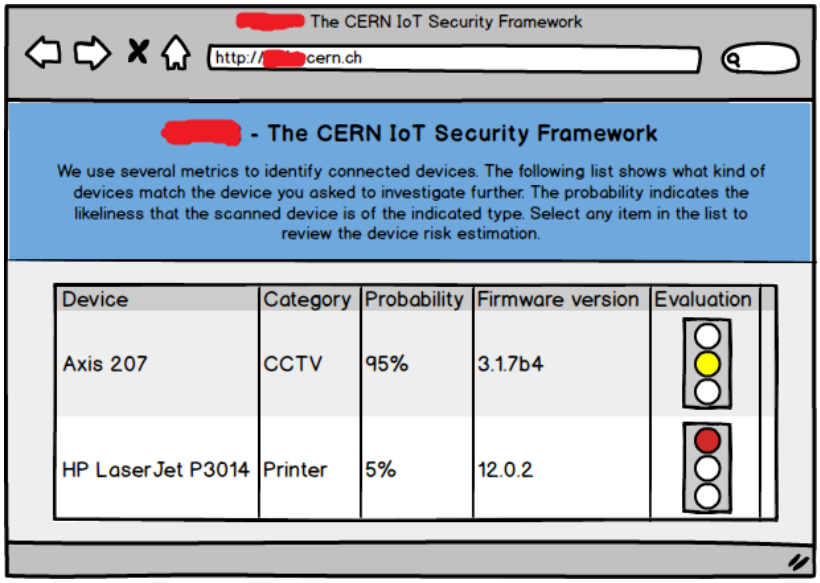


Fig 3. Device selection page

Components:

1. Device–List Table: listing information about the devices.
   1.1 Columns:
      1.1.1 Device: device name with a clickable-link to open a pop-up window showing details about the device.
      1.1.2 Category: device category
      1.1.3 Probability: probability that the listed device is of the indicated type
      1.1.4 Firmware Version: version number of the current device firmware
      1.1.5 Evaluation: traffic light showing the device risk score.
         1.1.5.1 Red signal: high risk
         1.1.5.2 Yellow signal: medium risk
         1.1.5.3 Green signal: low risk
      1.1.6 Expert Mode: a toggle button to switch between views for technical and non-technical users

c. View for non-technical users

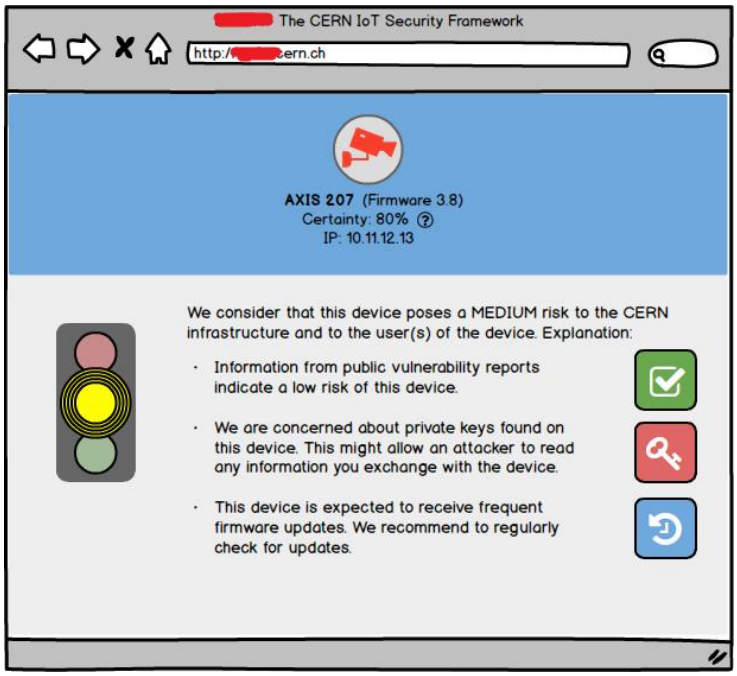


Fig 4. View for non-technical users

Components:

1. Labels:
   1.1 Device name
   1.2 Probability: probability that the listed device is of the indicated type
   1.3 IP Address/Hostname
2. Text and Images:
   1. Traffic-light: indicating the device risk score.
   2. Explanation for risk score: contains information about
      2.2.1 Public vulnerability reports
      2.2.2 Private keys
      2.2.3 Firmware updates

d. Views for technical users

DEFAULT

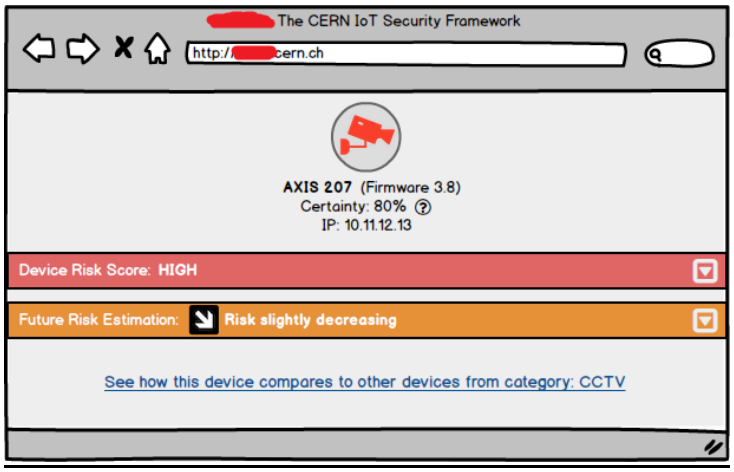


Fig 5. Default view for technical users

ELONGATED

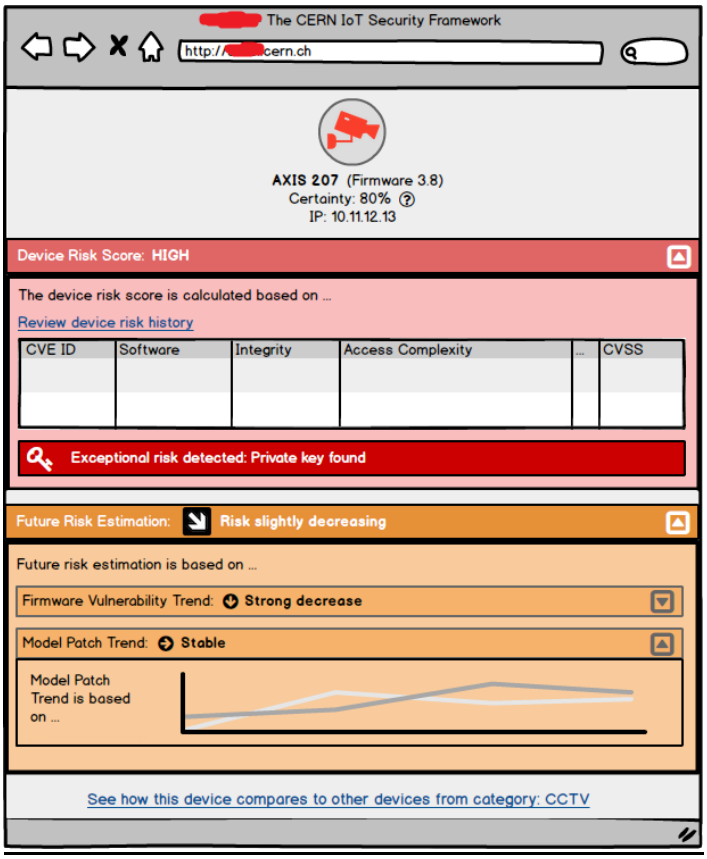


Fig 6. Elongated view for technical users

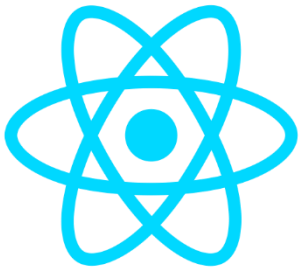
Components

1. Labels:
   - 1.1 Device name
   - 1.2 Probability: probability that the listed device is of the indicated type
   - 1.3 IP Address/Hostname
2. Tabs: to display detailed information when clicked
   - 2.1 Device Risk Score: displays the current device risk
     - 2.1.1 Links:
       - 2.1.1.1 Review device risk history: to compare different firmware versions of the device.
       - 2.1.1.2 Vulnerability Table: listing information about Common Vulnerabilities and Exposures(CVEs)
     - 2.1.2 Columns:
       - 2.1.2.1 CVE ID
       - 2.1.2.2 Software
       - 2.1.2.3 Integrity
       - 2.1.2.4 Access Complexity
       - 2.1.2.5 CVSS
     - 2.1.3 Labels:
       - 2.1.3.1 Exceptional Risks: displays any exceptional risks detected, for example, unsecured private keys.
   - 2.2 Future risk estimation: displays the future device risk
     - 2.2.1 Tabs: to display detailed information when clicked
       - 2.2.1.1 Firmware Vulnerability Trend: displays graph for Number of vulnerabilities v/s Year, for 4 types of vulnerabilities
         - 2.2.1.1.1 Critical
         - 2.2.1.1.2 High
         - 2.2.1.1.3 Medium
         - 2.2.1.1.4 Low
       - 2.2.1.2 Model Patch Trend: displays graph for Number of patches released v/s Year
3. Links:
   - 3.1 See how this device compares to other devices from the category: to compare different devices belonging to the same category

3. Toolkits Used

Various toolkits are used in the development and deployment of the web application for the framework. Each component acted as a separate independent module in the project architecture.

1. React –

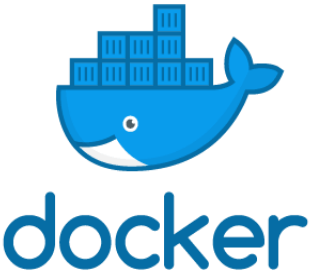


React[7], a JavaScript Library, maintained by Facebook, has been used for the development of web application. React provides various tools that are necessary to manage rapidly changing web pages. It also helps to control the routing between web pages. For complex applications, React Hooks, a new addition to React v16.8, is used for state management. It is supported by various other GUI libraries for displaying the web pages.

2. NGINX –

NGINX[8] is an open source application that allows the user to create a web server which can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. In the project, it is used to host the web application and proxy any AJAX calls from the client-side to the backend server in order to mitigate CORS (Cross–Origin Resource Sharing) issue.

3. DOCKER –



Docker[9] is a set of platform-as-a-service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating-system kernel and are thus more lightweight than virtual machines. In the project, it is

[7] https://reactjs.org
[8] https://www.nginx.com
[9] https://www.docker.com

used to make the web-application easily deployable, platform independent and also open to extensions.

4. OPENSHIFT –

OpenShift[10] is a multifaceted, open source container application platform from Red Hat Inc. for the development, deployment and management of applications. OpenShift provides developers with an integrated development environment (IDE) for building and deploying Docker-formatted containers, and then managing them with the open source Kubernetes container orchestration platform. In this project, it is used to deploy the containerized web application.

5. GROMMET –

Grommet[11] is a React-based framework that provides accessibility, modularity, responsiveness, and theming in a tidy package. It provides various components like tables, graphs, buttons, text fields, etc. to be displayed on the web page.

6. MATERIAL-UI –

Material-UI[12] is a React based framework that provides components for faster and easier web development. It has also been used to display other specific components in the project including icons, scroll bars, etc. in the project.

[10] https://www.openshift.com
[11] https://v2.grommet.io
[12] https://material-ui.com

4. Architecture

The user interface has been developed using React, HTML and CSS. NGINX, is used as a web server and also to handle AJAX calls from the client-side. The whole web application is wrapped in a docker container which is deployed using OpenShift. In the following, one can see the workflow architecture of the web application that is how the front-end interacts with the user and the backend.

a. User log-in and device assessment querying

The user logs into the web application using CERN's SSO (Single–Sign On) service and gets routed to the main page. The user initiates a scan by typing a hostname/IP Address through the web interface. The NGINX server transfers the query to the backend server, resolving the Cross–Origin Resource Sharing (CORS)[13] issue. The backend server, on receiving the query, gets triggered and initiates the scan. During the scan, the backend server responds with a unique scan key. The key is used to extract results from the database.

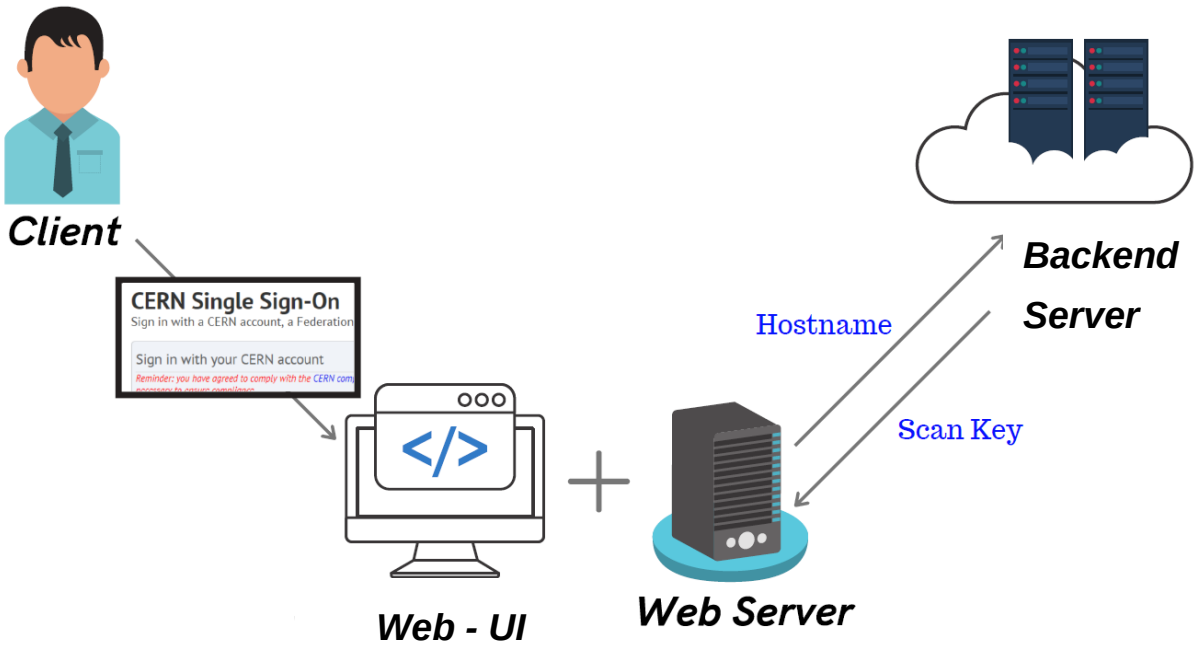


Fig 7. User logs in and generates a query

[13] Refer page no. 30

b. Backend processing

The backend processes initiate as soon as the backend server receives the query from the client-side. The server initiates three processes:

a) Device identification

It includes the process of IoT device identification. Based on the corresponding hostname/IP Address given by the user, the specific IoT devices are scanned.

b) Vulnerability enrichment

It includes the backend processes where the framework analyses the vulnerabilities of the IoT device.

c) Risk scoring

It includes providing risk score to the IoT device specifying the device security risk.

After processing, the data is pushed to the database by the backend server which can be accessed in the future.

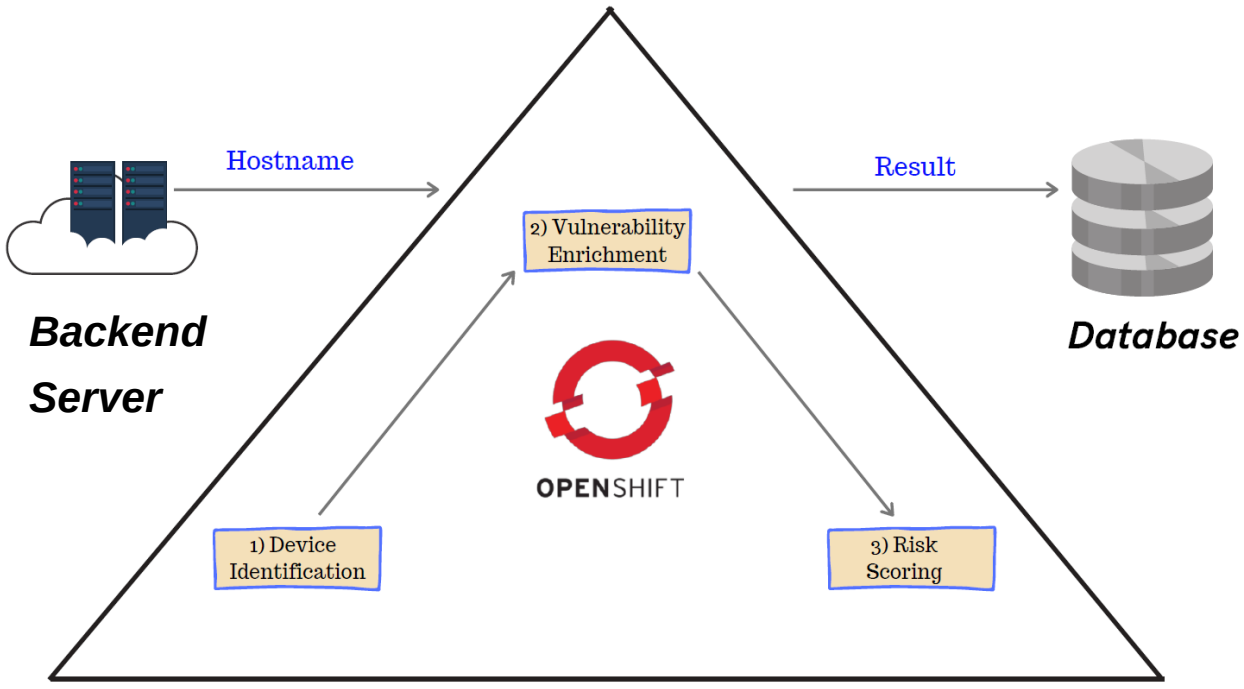


Fig 8. Backend processing

c. Getting results

After the data is pushed in the database, the client–side is notified with a status keyword. The web application now triggers an AJAX call to the database and accesses the results. This information is then parsed and results are displayed on the web page.

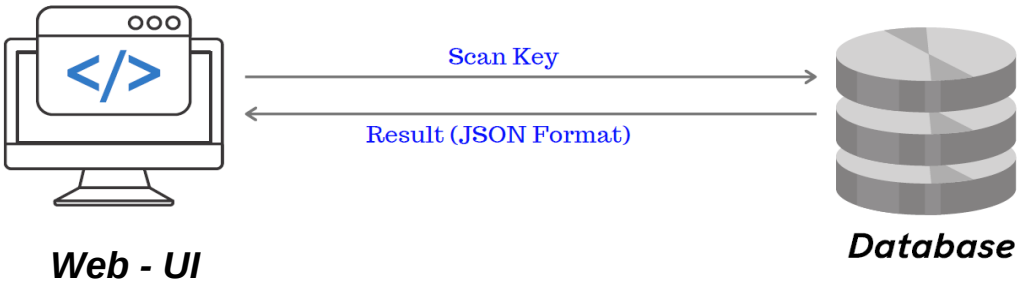


Fig 9. User getting results from database

RESULTS

a. User log-in and device assessment querying

BEFORE

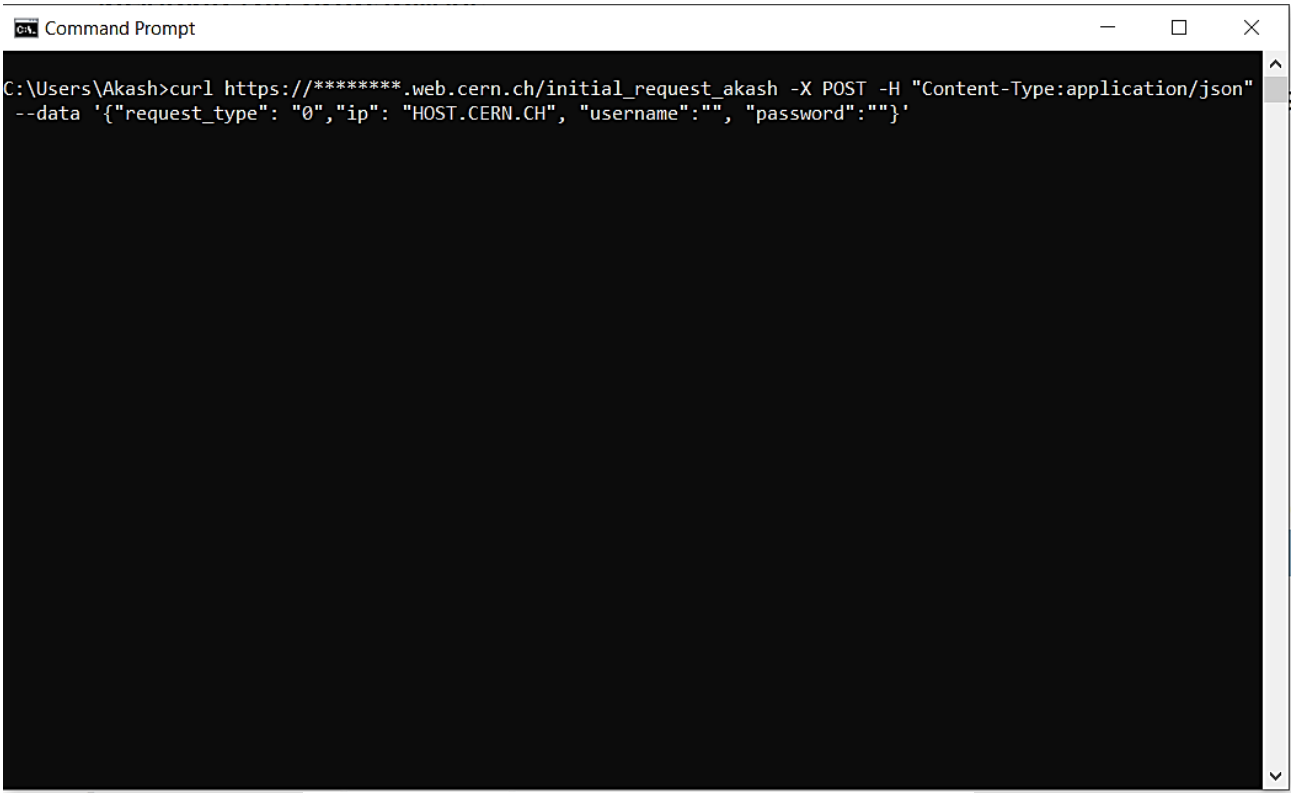


Fig 10. User making request with CURL command

The user had to write a curl command in order to make a request to the backend server for IoT device scanning.

AFTER

Fig 11. CERN SSO(Single–Sign On)

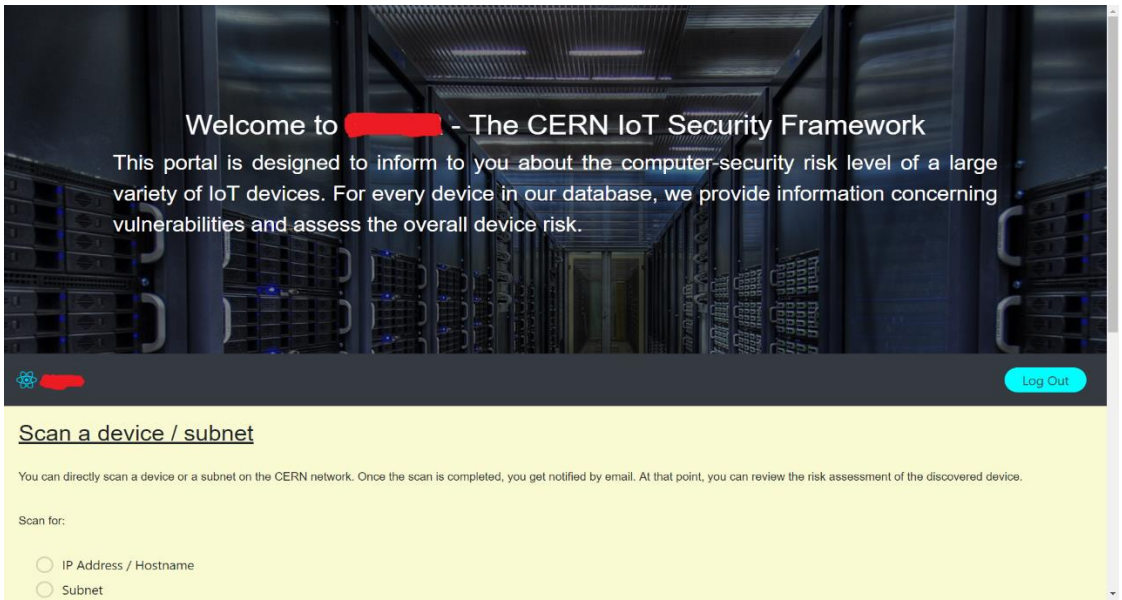


Fig 12. IoT security framework landing page

The user can easily generate a query by typing in the IP Address/hostname through the text field provided in the web–UI.

b. Getting results for a requested scan[14]

BEFORE

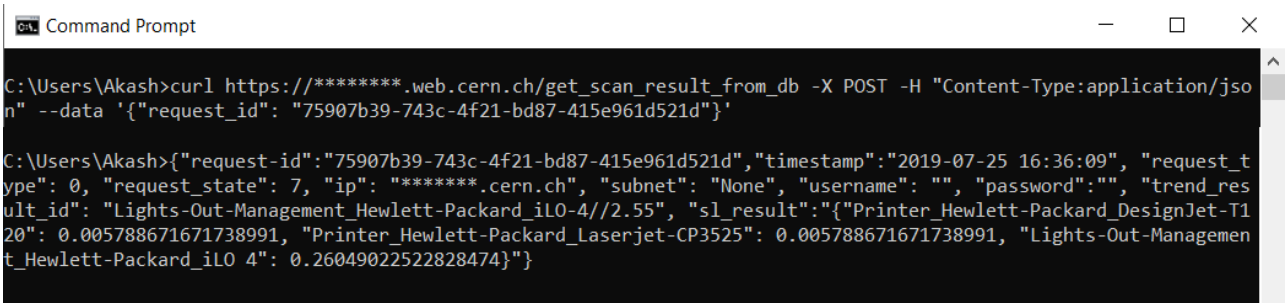


Fig 13. User getting results using CURL command

AFTER

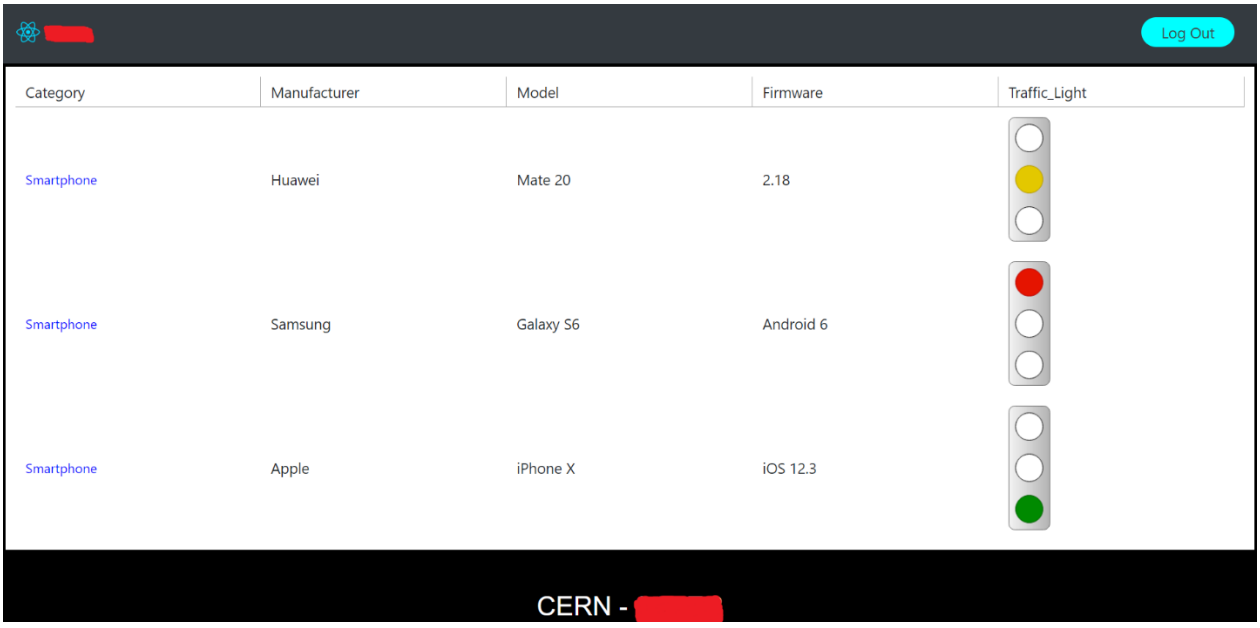


Fig 14. User getting results from the framework

The results are displayed in a user-friendly table with the 'traffic lights' indicating the risk of the IoT device.

[14] These are mock results. Only for presentation purposes.

The application displays user-specific views for each IoT device.

For Non-Technical Users

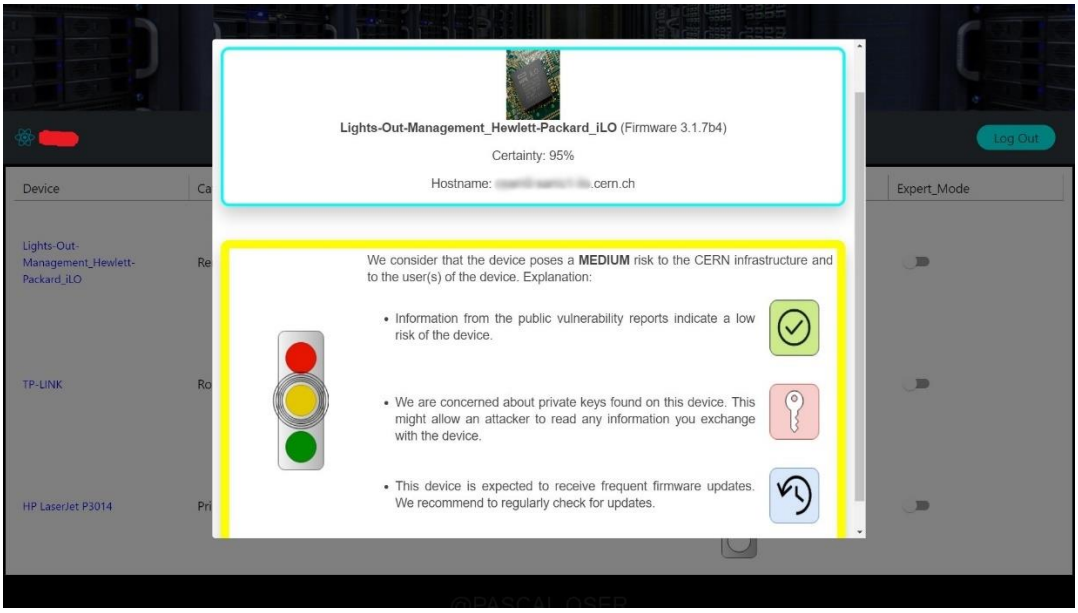


Fig 15. Results for non–technical users

For Technical Users

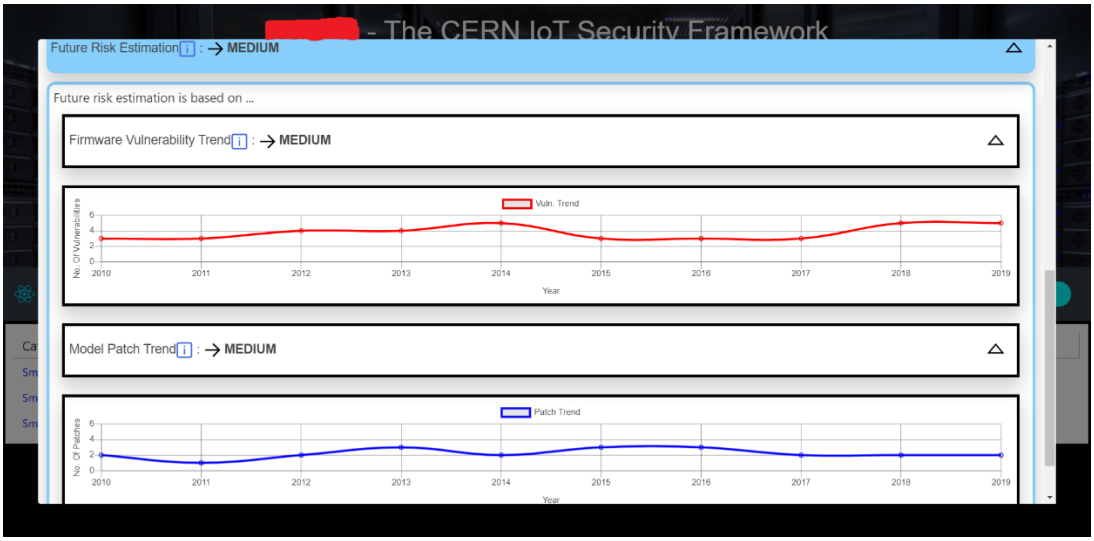


Fig 16. Results for technical users

CONCLUSIONS

a. Problems Faced and Solutions

? Problem with state variables

A state is an object that holds some dynamic information to control the behaviour of the React component. In other words, the state contains information that may change over lifetime of the component. But working with too many states is cumbersome and the following issues are faced during development:

a) It's hard to reuse stateful logic between components as we need to share state information between different component classes.

b) Complex components become hard to understand with increasing functionality. It's not possible to break these components into simple ones as stateful logic is all over the place. It's also difficult to test them.

➢ Solution

React Hooks[15], are used to solve the problem with state management. They allow to use state and other React features without writing a class. Other features include:

a) reusing stateful logic without changing the component hierarchy. This makes it easy to share states among many components.

b) splitting one component into smaller functions based on what pieces are related (such as setting up a subscription or fetching data), rather than forcing a split within the component that disrupts the functionality.

References

[15] https://reactjs.org/docs/hooks-intro.html

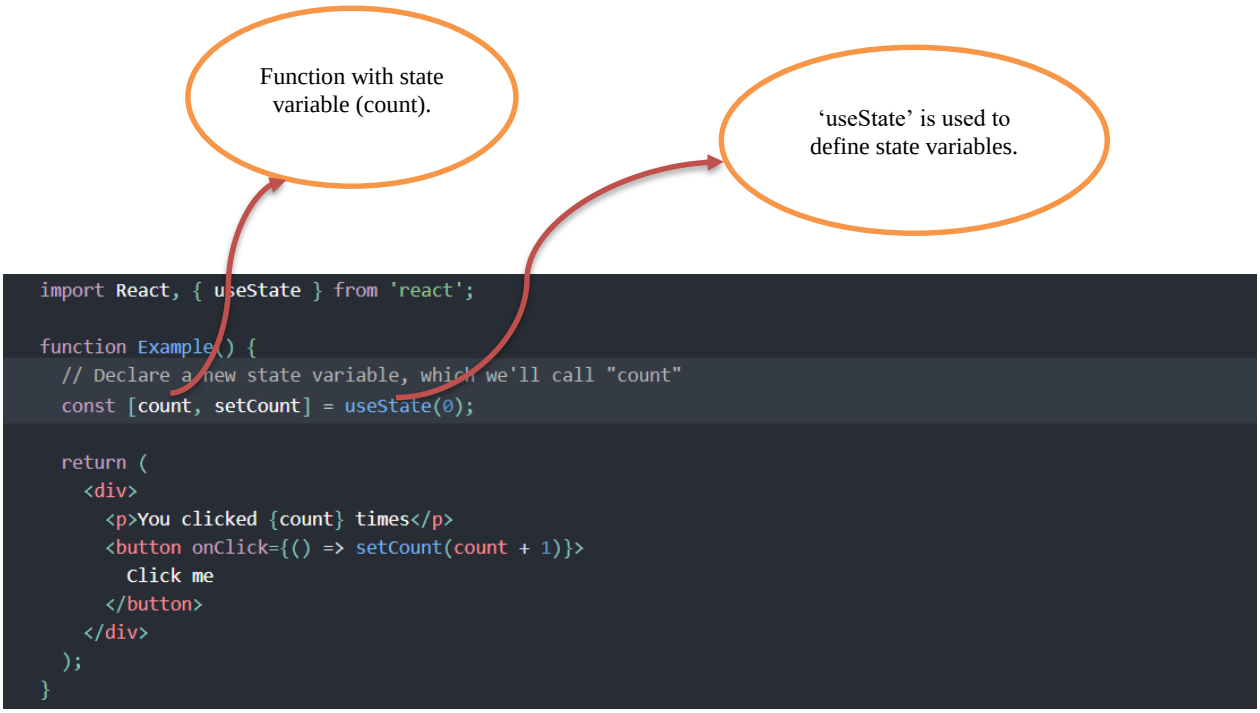


Fig 17. Using React Hooks[16]

? Cross–Origin Request Blocked

CORS[17] (Cross–Origin Resource Sharing) is a standard that allows a server to relax the same-origin policy. This is used to explicitly allow some cross-origin requests while rejecting others. Due to CORS policy, the web application from client-side fails to trigger AJAX calls directly to the Backend server at different location.

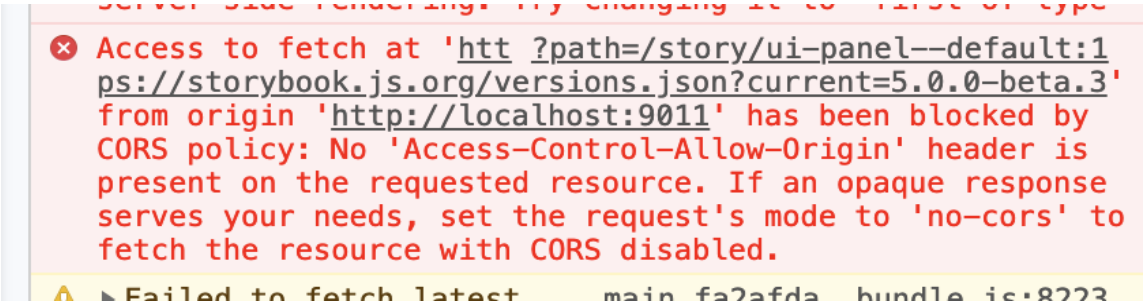


Fig 18. CORS error[18]

[16] Image Source: https://reactjs.org/docs/hooks-intro.html
[17] https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
[18] Image Source: https://github.com/storybookjs/storybook/issues/5823

➢ Solution

NGINX, is used to provide proxy for AJAX requests from the client-side to the backend server. This serves as an intermediary while making any AJAX requests. After user generates a query, the web–application requests the proxy server which instead forwards the request to the remote backend server resolving the CORS issue.

Fig 19. Using NGINX as proxy server

b. Summary

During the two months of the CERN Openlab summer student programme, Akash has gained valuable experience and knowledge from a hands-on and full cycle IT development project. He worked with various new technologies for web–application development in a very inspiring, motivating and international environment with colleagues from all around the world. The developer has been able to take part in the whole development cycle with analysing and getting the user requirements, as well as development, continuous integration and deployment.

Web - UI development

IoT Security Framework

August 2019