Case Tracking System with Access Control

Team Members: Vlad Dubrovenski, Sabrina Djeddi, Dawn Dixon, Nikoloz Dzidzava

Google Firebase Link: https://case-tracking-system.web.app/

Github Link: https://github.com/vladi7/Web-Project-Exam

Prepared for Web and Mobile Design Class, Summer 2020

The Motivation Behind the Idea and Significance:

Many commercial, medical, and legal software solutions require enhanced security. While the unauthorized access of the parties are being handled pretty efficiently(even though not always, we all are aware of recent data leakages), the unauthorized access by the insiders of the companies present an even bigger complexity. The following questions have to be answered:

- Do all of the members have the same access to each resource at any time? The answer to this question is no most of the time. For example, a medical software system. Nurses, staff, and doctors should all have various access control.
- Should the access privileges change after a particular action? Let's take a tax company as an example. Given two parties use the same company, and they have a conflict of interest with each other. Once a CPA accesses files of one person, this CPA should not be able to access another case of a person who has a conflict of interests. The above may not seem like a pretty serious situation, however, if we are talking about two different companies with high revenue then this situation may lead to great risks of litigation between those companies, as well as between those companies and the CPA who handled them.

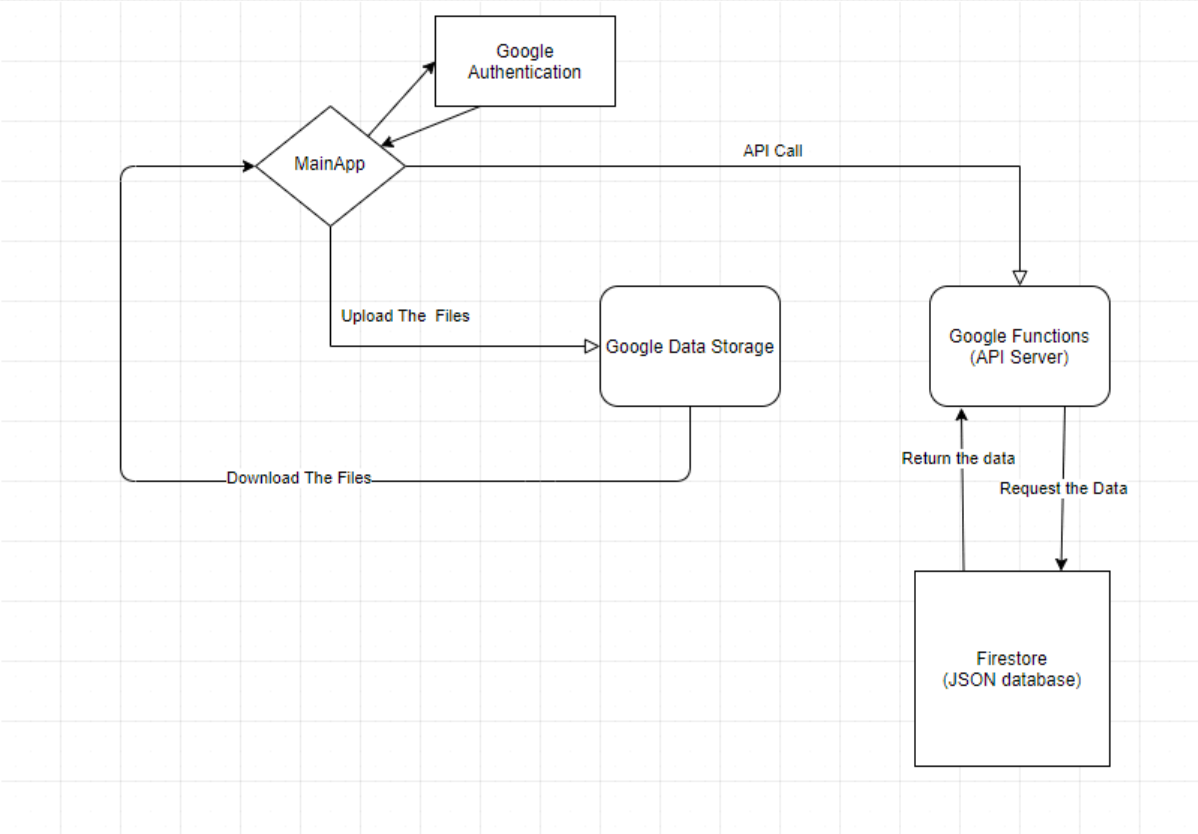
Objectives, Features, and Approaches:

Our group determined that we will be able to solve both of the issues described in the previous section. Because of the timeframe, we were not able to create a complete fine-grained access control system due to the complexity of such a project. That type of project is

very time consuming and uses some kind of standards to check whether a person has the access and can dynamically change them. Instead, we decided to implement a small but useful system that can be used to track cases using MEAN stack. After doing some research, we decided to actually host the application on firebase, as well as hosting another server on firebase that is responsible for the database access. Our app supports multiple cases per user and multiple users per case, as well as dynamic change of access rights in case of a conflict of interest. To store files, we use firebase data storage. We utilize the firebase authentication sign-in, sign-up, and restore password functionalities. We used a guide (1) to make those pages and then styled and changed them. We also utilize the following guide (2), to upload the files efficiently (multiple files at the same time). As we are using the firebase to host the website, we also used the following article (3) to create a serverless API utilizing Google Functions. It is not really serverless, it just has many servers and decides for itself which server to use to handle the request. And, of course, we used a reference to firebase documentation. And the most significant was about how to structure our database to handle the calls efficiently (4).

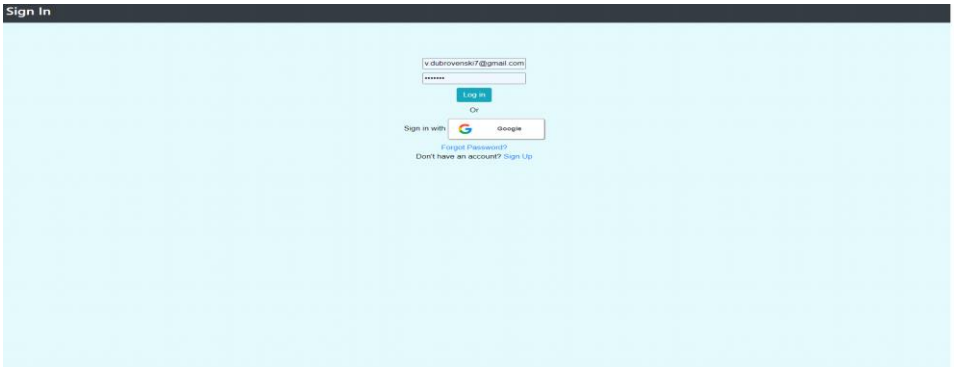
Workflow:

The following diagram explains the workflow of our project. In there you may see that the server for api calls is a separate project, that has its own database. The only database that is used by the main project is used for authentication and user data. As we mentioned before, we utilize the google storage to store the files for our project. We make a lot of api calls to data storage, to the actual server that we implemented, as well as google authentication.

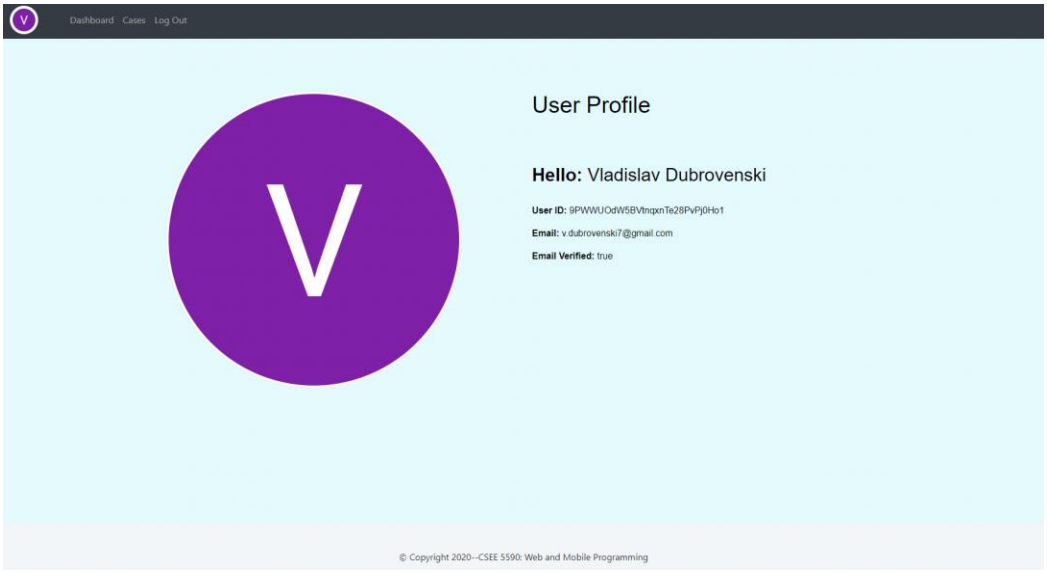


Screenshots of the Project:

Sign-In Page:

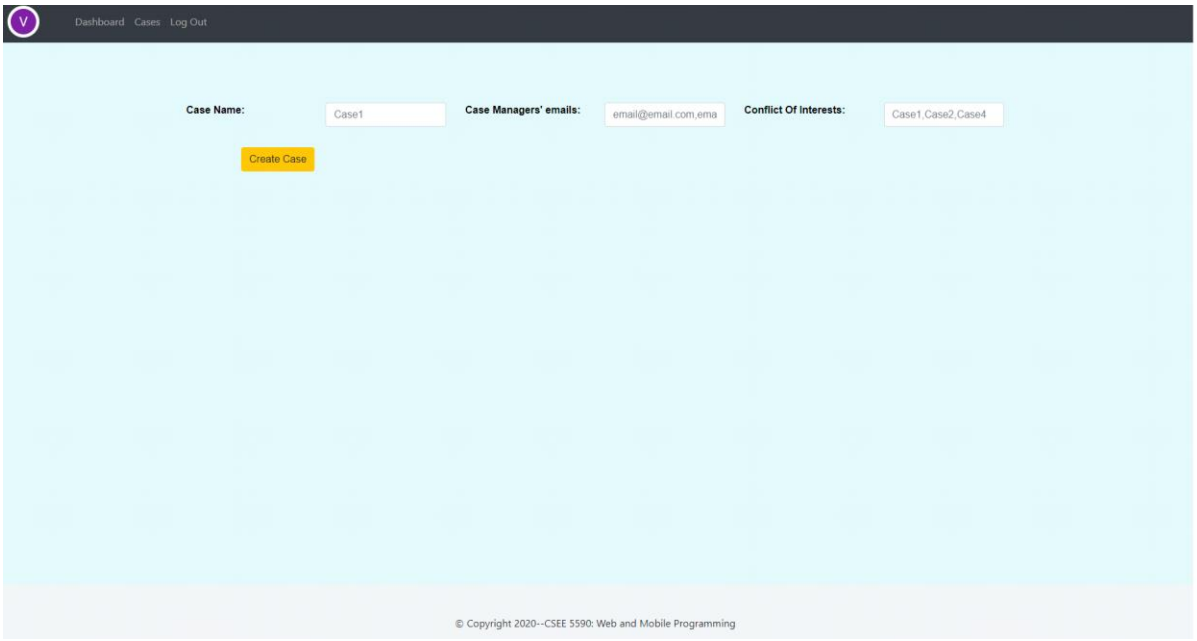


User Dashboard Page:

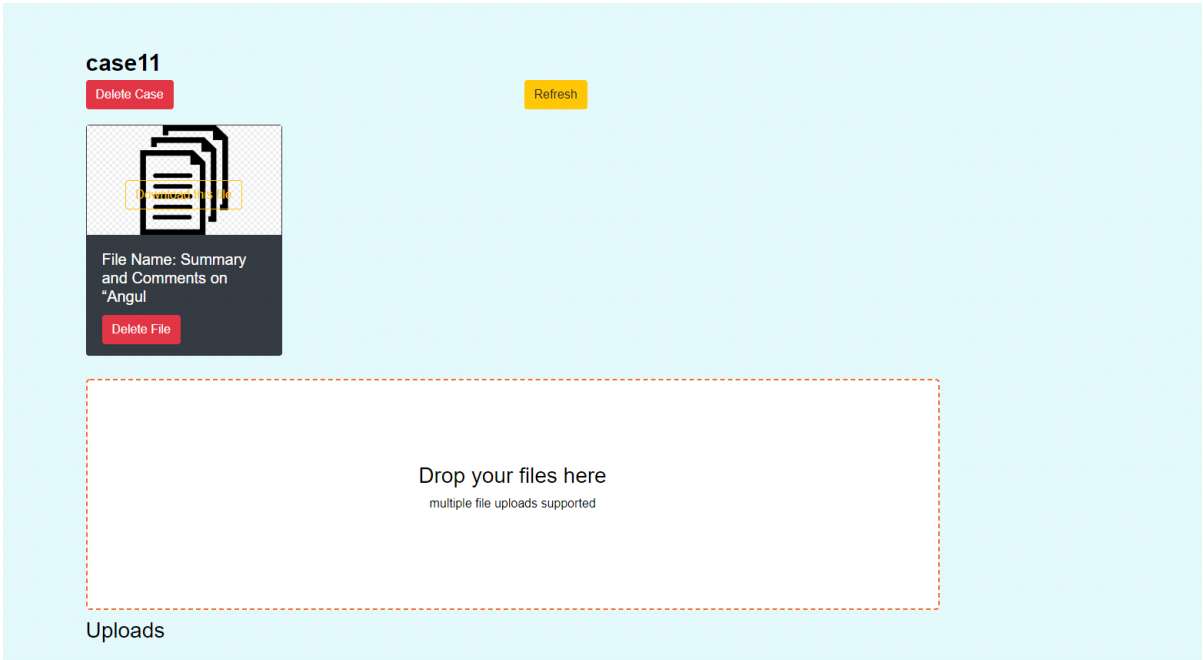


Page with Case Files:

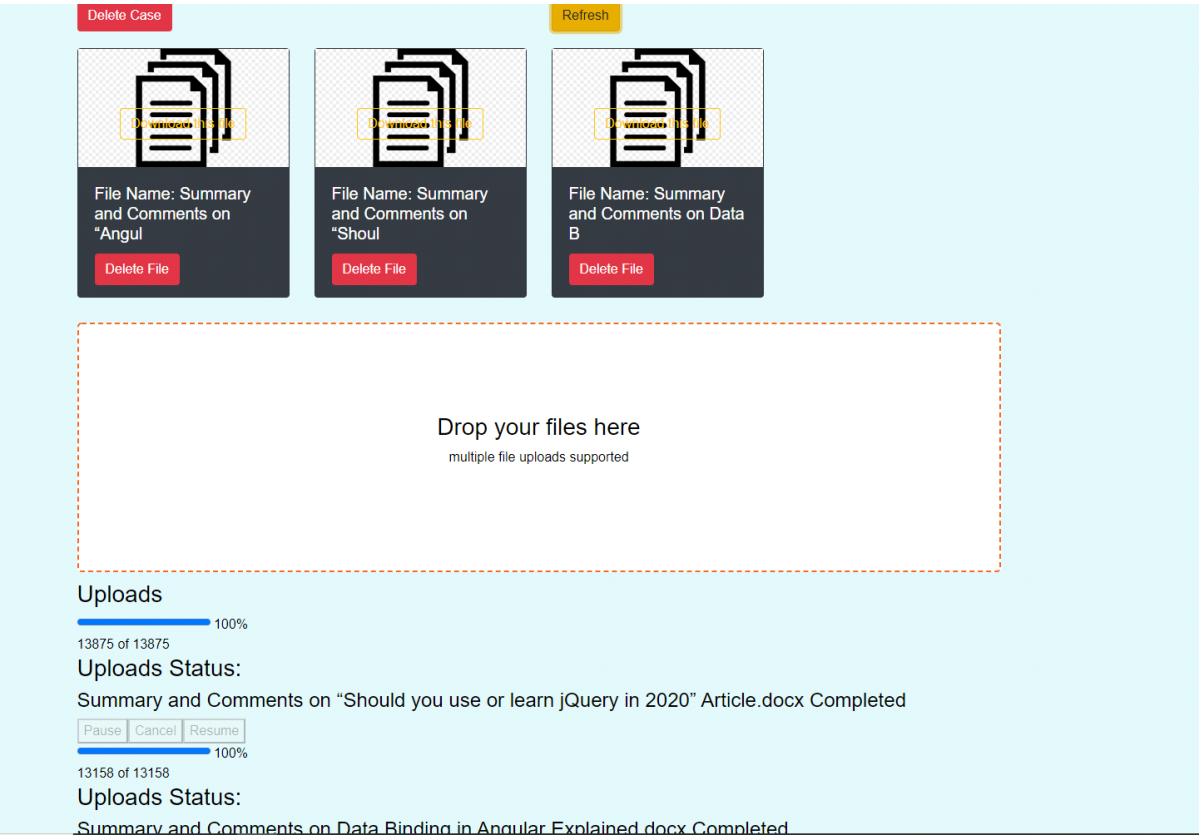
Page for Creating a Case:



Page for File List and File Upload:



Page for Uploading the Files:



Work Sharing Between Teammates:

There was a lot of work done in this project and we shared it pretty equally. Dawn Dixon was creating the design for the case view, sign up, and forgot password pages. Sabrina Djeddi handled the navigation bar and user dashboard page. Nikoloz Dzidzava was handling the login page. Vlad Dubrovenski handled the logic for case list and file list pages as well as back end for the project.

Issues and Blockages with the Project:

The greatest challenge was to handle the CORS (Cross-Origin Resource Sharing). CORS is, if simplified, used to access the resource from one domain in another one. As Google Functions

prohibits the wildcard for CORS, we had very complex issues handling that. The following three solutions were utilized during the development to handle those issues:

- First, after multiple failed attempts to handle CORS, we used the following plugin for Google Chrome: Moesif Orign & CORS Changer. This plug in let us develop the app in the early stages to avoid dealing with CORS and focus on the requirements. It is not a good solution because it is horrible to make the user of the app install any plug ins, which are not even available for mobile devices.

- The second solution was to append https://cors-anywhere.herokuapp.com/ in front of our api request. We would have continued to use this solution, however, a day before the due date it just quit working with google chrome for about an hour. Also, this is not good to transmit the data through a third-party service as we never know what the do with our data. Given all of that, we decided to switch to the third solution.

- The problem was actually easily fixed with the following line of code in our back end(express.js code). The following line fixed the issue: “app.use(cors());” We did install the cors library with npm since we wrote the backend in type script and the solution for this problem was not readily available for this language, unlike for the javascript. As we mentioned before, we used (3) to build our api server using typescript with express.js

Conclusion:

We believe we created a good app and were able to demonstrate why access control systems are important in almost every data sensitive industry. Our solution does have a flow, such as access control is handled on our front end which is not very reliable. A better

way to handle that would be to write a server with defined policies that will handle the api requests. However, the new access control policy, that the author of this paper has experience working with, is written in java, so we would have to write a tomcat servlet server to handle those requests. This is a very time-consuming process and we would not finish even after 3 months of writing the code since it is very complex. Otherwise, our solution presents a great case study of access control handling using only front end. Maybe if we filtered the results in express code in our api server that would have been a more secure solution, and it could be accomplished if given one more week.

Bibliography

1. “Full Angular 7: 8|9 Firebase Authentication System.” *PositronX.io*, 2 June 2020, www.positronx.io/full-angular-7-firebase-authentication-system/.

2. Delaney, Jeff. “Upload Multiple Files to Firebase Storage with Angular.” *Fireship.io*, Fireship.io, 18 Feb. 2019, fireship.io/lessons/angular-firebase-storage-uploads-multi/.

3. Velázquez, Levi. “Beginner's Guide for Creating a Serverless REST API Using NodeJS over Google Cloud Functions.” *DEV Community*, DEV Community, 22 Mar. 2020, dev.to/levivm/creating-a-serverless-rest-api-using-google-cloud-functions-firebasefirestore-in-10-min-37km.

4. “Structure Your Database Firebase Realtime Database.” *Google*, Google, firebase.google.com/docs/database/android/structure-data.