https://github.com/ASK-LOCK/ASKLOCK

INTRODUCTION:

Password managers are simple, easy, extremely beneficial, and allow the average consumer to protect and secure their information, either online or not, with very secure and strong passwords and can eliminate the need for the user to remember such a complex string of characters. ASK LOCK™, developed by team A.S.K.H.O.L.E consisting of team members Aaron Geronimo, Sunny Siu, and Kain Yogi as project developers, is a web application that is a secure service that safely stores passwords and can allow a specific user to access their own account credentials ranging from a list of passcodes to other relevant account data they have stored within the application. We seek to create a service that not only is secure on login but also utilizes a large range of security and privacy requirements that will show the user that their data is protected by our backend system. We currently have Visual Studio Community 2019 and IntelliJ as our IDEs with HTML/CSS, the semantic framework, and a backend we will choose later on in our current repertoire but we seek to expand as needed once the project grows.

REQUIREMENTS:

Security is why we are building this application and security will be a driving force with development as well. The types of security and privacy requirements will include a multitude of parts. First, our password manager shall require a separate master password in order to access the account on the application and the data stored. This master password will use a stronger and different key derivation function because this is the password that unlocks the application from the state. Our data encryption for the credentials that the user wants to be stored within each item will also use its own key derivation function, however, we are still researching possible functions to use. The master password set for our user accounts must also meet the set requirements to utilize it as a passcode. It should have at least 8 or more characters while also utilizing special characters and an uppercase letter along with basic functionality of locking access to the account for a few minutes when there are 5 to more erroneous attempts to log in. We also seek to implement a system where the application will log out automatically once the session has been idle after 15 minutes of not pressing any buttons on the page. For any and all issues and

issue-tracking, we will have an issue tracker on our Github.io page that, using the functionality provided by GitHub, should update our project board with issues so that we can provide the user with an updated view and any changes regarding the issue.

The application's login and storage of data will require threat modeling and security reviews. The most important part of our program that needs threat modeling and security reviews is the storage of the personal data gathered from the user and the initial login data to access our application. To determine how we will assess the security and privacy within our risk assessment plan, we will utilize a step by step process that includes finding all valuable assets, identifying potential consequences, identifying threats and their level, identifying vulnerabilities and assess the likelihood of their exploitation, calculate the risk and create a strategy, and finally define a mitigation process to combat this security and privacy risk.

For the security bug bar, the most critical state that our program would be in is when users have the ability to either execute arbitrary code or obtain more privilege than authorized. An important-level scenario would be cases where the attacker can locate and read information from anywhere on the system, including system information that was not intended or designed to be exposed, this is an important quality gate as we must manage a system to hide the data or utilize encryption so if the data was compromised the level of encryption would be the last wall to a full breach. A moderate-level scenario would be persistent storage or information disclosure as we seek to store the hashed data within a file, that is possibly hidden, after the user exits the application so they can access it again on login, this ties with our important level scenario as we must store this data onto the local machine for future use. A low-level scenario would be Lack of notice and consent, we seek to collect possible data such as usernames, emails, and other account-related credentials and store it locally, some users may not like this idea so we must get the users consent to their data.

DESIGN:

Earlier, we specified several security requirements that we want to meet with our application. This includes having a master password to access the account that is only valid if it has a minimum of 8 characters, an uppercase letter, and a symbol, while being locked out for

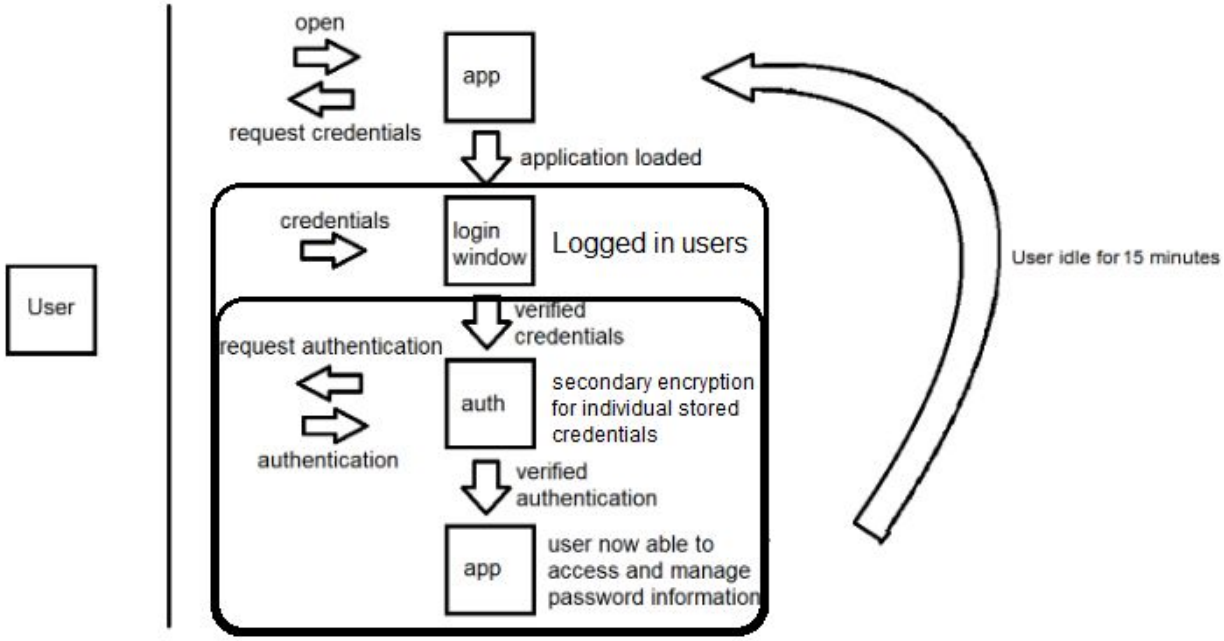
several minutes if 5 attempts are made with incorrect passwords. Users are also prompted with a second encryption for each individual credential they store and want to access. Lastly, the application will time out and automatically log the user out after 15 minutes of inactivity on the application. Our development team intends to implement these security requirements by prompting the user to make a password of at least 8 characters with at least one uppercase letter and a special character when first setting up the account. If the user inputs a password that does not meet the requirements, the password will be denied and the user will be re-prompted to create a password. To meet the second security requirement, our design team will decide on a secondary verification technique and our development team will implement it. As for the third security requirements, our development team intends to use a timer that resets at every button press, if no button has been pressed by the time the timer runs out, the application will automatically log the user out, thus keeping the user's stored credentials safe if they are away from the computer.

Our team has decided on 2 privilege levels for our application, logged in and not logged in. A user that has not logged in has little privilege in the application, limited to only being able to open the application, enter credentials to log in, and/or exit the application. A logged-in user however, has full privilege over the application. A logged in user can select and manage an account for which they want to view their credentials associated with it. They are able to add credentials, remove, and edit. However, users are only able to view, edit, or remove credentials after entering the individual encryption for that account. This also applies to the master credentials for logging into the application in the event that something must be edited. Many other password managing applications are web based, thusly have web-based vulnerabilities associated with them. This includes sending information such as the users credentials over unsecure connections and possibly having it intercepted. We intend to mitigate this vulnerability by having all credentials stored locally on the users computer and encrypting the information to prevent others from accessing it.

In this application, security threats may come from intruders possibly bypassing the first layer of security which is logging into the application. This could be done by an intruder somehow guessing or finding the users credentials for logging into the application. For this

threat, we mitigate it by adding a second layer of authentication after logging in with a users email or username and password. This ensures that even if an intruder were to get past the first layer of security, they are still unable to view sensitive information. Even before the second layer of security, intruders must successfully login to the application, if 5 attempts are made without success, all actions will be suspended for several minutes. Another possible threat may come from a user who forgets to log out of the application and leaves their computer where an intruder can see or in a worse case scenario, the users laptop is stolen and they are still logged into the application. This threat will be mitigated by an automatic log out after an idle time of 15 minutes. Meaning that if there is no activity on the application for longer than 15 minutes, the application will automatically log the user out and will prompt the user to log back in. These possible threats are equal in terms of threat level which we would consider as all high threat, leaving the relative attack surface of the application as 3.0.

APPLICATION DIAGRAM:



APPROVED TOOLS:

Compiler/ Tool	Minimum Recommended Version and Switches/Options	Optimal Recommended Version and Switches/Options	Comment
JavaScript HTML CSS Interpreter	IntelliJ IDEA Version 2018.1 / Visual Studio 2017 Community Version 15.9	IntelliJ IDEA Version 2019.3 / Visual Studio 2019 Community Version 16.4	
Browser Developer Tools	Chrome 80.0.3987.116/ FireFox 4 / Internet Explorer 9/ Opera 11	Chrome 80.0.3987.116/ Opera 16.0.1196.55/ Internet Explorer 11/ Firefox 73	
Database	mySQL MongoDB		May change based on need.
Web hosting service	BlueHost GoDaddy UH Unix Personal Webspace		May change based on need.
Source Code Control System	GitHub GitHub Desktop Subversion HostInger HostGator MeteorApp		Github is needed for the issue management tab and assignment submission.
Frameworks	Semantic UI React		May change based on need.

DEPRECATED/UNSAFE FUNCTIONS:

DEPRECATED/UNSAFE FUNCTIONS	ALTERNATIVES
Center tag (html)	text-align:center
Menu tag (html)	Ul tag
Isindex tag (html)	Form tag
Align attribute (html)	Text-align, vertical-align
Background attribute (html)	background-image:
Vspace attribute (html)	padding:
document.write() (html + javaScript)	*No Alternatives DO NOT USE* *deemed unsafe*
document.writeln() (html + javaScript)	*No Alternatives DO NOT USE* *deemed unsafe*
.innerHTML (html + javaScript)	*No Alternatives DO NOT USE* *deemed unsafe*
.outerHTML (html + javaScript)	*No Alternatives DO NOT USE* *deemed unsafe*
.insertAdjacentHTML (html + JavaScript)	*No Alternatives DO NOT USE* *deemed unsafe*
Dir tag (html)	Ul tag
Embed tag (html)	Object tag
Font tag (html)	Font-family, font-size, color
Type attribute (html)	list-style-type
Applet tag (html)	Object tag
Basefont tag (html)	Font style sheets
Height attribute (html)	Padding attribute
.eval (jQuery)	jQuery.globalEval

.append (jQuery)	*do not use from untrusted sources like form inputs*

STATIC ANALYSIS:

Compiler:	Code Inspection and Analysis
IntelliJ:	Closer Linter ESLint JSCS JSHint JSLint Standard Code Style
Visual Studio	HTMLHint

There were many code inspection and analysis tools that we looked into for each of the compilers that we are going to use. For IntelliJ, we have the closer linter, that enforces the guidelines set by the Google JavaScript Style Guide, it handles style issues so that you can focus on the code. ESLint, the severity level of the inspection is applied only for the 'Inspect code' action. Severity levels used in the editor can be configured in the ESLint configuration file for each individual rule. JSCS, JavaScript Code Style checker for specified JavaScript file. JSHint and JSLint are basically the same, a validator for specific JavaScript files. And Standard Code Style, which reports errors according to JavaScript Standard Style. The severity level of the inspection is applied only for the 'Inspect code' action.

For the IntelliJ users in our group, we chose to use ESLint because it was the most consistent after using it for a few times and we had the most experience with it before. It has configurable rules, including error levels, allowing us to decide what is a warning, error, or simply disabled. There are rules for style checking, which can help keep the code format consistent across our group. It also has the ability to write our own plugin and rules, which we are using a .eslintrc file with rules from our previous class.

For our Visual Studio user, they will be using HTMLHint. The HTMLHint extension will run HTMLHint on your open HTML files and report the number of errors on the Status Bar with details in the Problems panel. Errors in HTML files are highlighted with squiggles and you can

hover over the squiggles to see the error message. It has a default set of rules, however, we can modify the rules with a .htmlhintrc file just like what we did with ESLint.

DYNAMIC ANALYSIS:

Tool	Version
Iroh.js	
Chrome DevTools Canary	Version 82.0.4085.4 (Official Build) canary (64-bit)
React Development Tools	4.5.0

For the ASKLOCK project, our team, team ASKHOLE, researched many dynamic analysis tools for our web application and selected our tools based on criterias consisting of accessibility, adaptability, and level of difficulty for implementation. We have selected three tools, Iroh.js, Chrome Canary, and React Developer tools for the purposes of dynamic analysis.

Iroh.js is a dynamic code analysis tool for JavaScript. Iroh.js allows us to utilize the Iroh.js API to record our code in realtime, intercept runtimes information, and manipulate program behavior. The API consists of many commands that range from commands that handle basic functions such as setters and getters to more complex logical functions. At first, we used the functions provided by the API in Iroh.js to test basic JavaScript functions in the mock-up stages in our more HTML based page. We made sure that our project can handle the mouse clicks, menu dropdowns, and returns so we can implement them more in the live version of our website. We are using JSX files so the functionality of Iroh.js has stopped working for us for the basic pages we have now, however we will use this in future functions with ASKLOCK. The difficulties that we faced were generally the fact that the implementations of Iroh.js requires us to understand the API. We used the examples given on the corresponding github for Iroh.js to better understand the functionality however in terms of usability, our project needs to use other tools such as Chrome Canary and React Developer Tools to be more thorough.

We are also using the React Development Tools as a dynamic analysis tool. The Components tab shows you the root React components that were rendered on the page, as well as the subcomponents that they ended up rendering. By selecting one of the components in the tree,

you can inspect and edit its current props and state in the panel on the right. In the breadcrumbs you can inspect the selected component, the component that created it, the component that created that one, and so on. The Profiler tab shows you gathered profile data, which you will immediately see data for the first render that occured. For each render, you can see what components were involved and which ones took the most time to complete. The gray items didn't need to re-render at all. We used this for the Navigation Bar elements to debug them and tested that our drop downs work correctly. We used the tree to see our react components render and not render dynamically in the web browser extension. The issue that we faced is generally learning how to use the tool properly, we have to click down the tree multiple times to find our element and from there we decided to research the most optimal way to use this tool for this project. The input was ideal as well as output and we seek to use this as the project continues for the front end development.

JavaScript and CSS code coverage has arrived in Chrome Canary, which we will be using as our dynamic analysis tool for now as well. The feature uses dynamic analysis to provide runtime code coverage statistics. Unlike Chrome, Chrome Canary has a feature where it allows you to track the function that you've made through the process of browsing through your web page. Also you are able to browse through a websocket that is currently unavailable to the base Chrome Developer Tools. Through this socket we used this to look at the output and input parameters for the login function that implemented past Iroh.js. Iroh.js was able to give us basic functionality yet Canary was able to give us the hashed sequence for the password as well as any indexes for the outputs. This is really good for us as we can see the hased sequence and manipulate it so we can have further test cases. The difficulty that we have using Canary is the fact that this seems to only handle after we have implemented this on the webpage. Canary is our final stand for the back-end development before we publish our webpage to live so we must use this tool the most during our project development

ATTACK SURFACE REVIEW:

Compiler/ Tool	Minimum Recommended Version and Switches/Options	Optimal Recommended Version and Switches/Options	Changes	New Vulnerabilities
JavaScript HTML CSS Interpreter	IntelliJ IDEA Version 2018.1 / Visual Studio 2017 Community Version 15.9	IntelliJ IDEA Version 2019.3 / Visual Studio 2019 Community Version 16.4	IntelliJ IDEA 2019.3.3	No Vulnerabilities
Browser Developer Tools	Chrome 80.0.3987.116/ FireFox 4 / Internet Explorer 9/ Opera 11	Chrome 80.0.3987.116/ Opera 16.0.1196.55/ Internet Explorer 11/ Firefox 73	Chrome 80.0.3987.132 / Firefox 74	No Vulnerabilities
Database	mySQL MongoDB		No Updates	
Web hosting service	BlueHost GoDaddy UH Unix Personal Webspace		No Updates	
Source Code Control System	GitHub GitHub Desktop MeteorApp		Github Desktop 2.3.1	No vulnerabilities
Frameworks	Semantic UI React		No updates	No vulnerabilities

FUZZ TESTING:

Our first attempt featured a brute force method to break into our web application. We found a list of passwords from a source (shorturl.at/rwLW8) and utilized this for a brute force testing method. We set up a mouse and keyboard macro to constantly type in our admin account, admin@foo.com, and a password from this list. We did not record the amount of times the passcode was wrong yet we found that after a certain amount of time, this method was able to break in. We realized from this testing that we have no code in place currently to lock the account depending on the amount of tries to break in. We only have a timeout for quick and fast submissions for logging in. In response, we will look into creating code that will lock the account depending on the amount of incorrect login attempts. After this lock is made, we will email the user a notification about the lock on their account and seek a password change to relieve them of this lock.

SUCCESS RATE: **100%**

The next attempt that we tried was to change the password of a user account to something else and recheck that user account against this password list. We used our parameters including using special characters, numbers, and a certain length to help the user create this password. This is the password the user came up with, $hittyP@ssw0rd, and we utilized this password for our brute force method again. We later found that this password was unique enough to go through all of the passwords within the list, given the extraordinarily long time it took, and resulted in a secure account.

SUCCESS RATE: 0%

Another attempt that we tried to use was to locate the schema that is being used to store the account details. This schema holds all the account details for everyone that utilizes our web application. If someone finds this document, the entire web application is compromised. Looking at our project directory, we are looking for the config folder within the sources tab within Chrome Canary Developer Tools to locate this file. Meteor already utilizes secure steps to keep this file hidden yet for this method we are trying to circumvent it using Chrome Canary Tools. Upon searching constantly, even manipulating the directory within the URL, we were not able to

locate this file. Meteor is helping us with this regard as it already has standards for security and can hide files such as our schemas from the public eye.

SUCCESS RATE: 0%

The final attempt that we considered was the attempt to inject harmful HTML or JavaScript into our login input fields. We researched this topic heavily beforehand and came up with a list that we should never use due to the fact that infections are common with these pieces of code. These code references are above within the **UNSAFE FUNCTIONS** section of this report. Because of the fact that we were able to eliminate possible vulnerabilities within our code, I believe that this is why we were unable to effectively inject harmful code into our web application.

SUCCESS RATE: 0%

STATIC ANALYSIS REVIEW:

We utilized ESLint in our web page application with the goal and objective of making our code more consistent and being able to avoid more bugs in mind. With the implementation of ESLint, the coding of our application looked more neat and formal to our eyes. Because of the automatic filter of ESLint, no matter how we write our code, it will always end up formatted.

DYNAMIC ANALYSIS REVIEW:

Upon investigation our application through dynamic analysis, we have found that tighter security must be implemented. Utilizing the developer tools available on Chrome Canary, running our application and viewing the network messages being sent and received revealed a security vulnerability that we must address.

Here is an example of said message:


```
[,…]
        0:
        "{"msg":"method","method":"login","params":[{"user":{"email":"admin@foo.c
        om"},"password":{"digest":"057ba03d6c44104863dc7361fe4578965d1887360f90a0
        895882e58a6248fc86","algorithm":"sha-256"}}],"id":"3"}"
```


The content of this message was able to be viewed after recording the network data while logging in. The message content contains the users email and password. Though the actual password is not shown, the hash used to encrypt the password is, along with the hashing

algorithm used which is “sha-256”. Knowing this information, a simple google search yields a long list of hash decoders. The one we used in particular was https://md5decrypt.net/en/Sha256/ which was able to decrypt the password in 0.053s. We have been discussing ways to patch this vulnerability and are currently trying to decide which way is best for us to pursue. Since this is a high risk security vulnerability, this will be our primary focus for now.

INCIDENT RESPONSE PLAN:

For our plan of action in case of any future threats to software,we would have a privacy escalation team to deal with internal processes to communicate the details of a privacy-related incident to deal with breaches such as data, theft, failure to meet communicated privacy commitments, privacy-related lawsuits and regulatory inquiries, and/or contact media outlets regarding an incident.

Kain will be our project’s escalation manager. He will be in charge of dealing with bringing in our team’s attention about a major incident that has been escalated. The need for creating an escalation process is caused by the loss of a consumer base due to an emergency situation.He will also be in charge of analyzing and highlighting issues which will require immediate response, allowing us to track critical problems, monitoring it as well as managing the situation.

Our privacy escalation team’s legal and public relations representative will be Sunny. As a legal representative, he will have legal responsibility to ensure and watch over the standing of the company. He will be the person who is responsible for performing the duties and powers on behalf of a legal person in accordance with the law or the constituent documents of the legal person. As a public relations representative, his job will be to plan, direct, and coordinate activities which are designed to create and maintain a good public image for our software. Which includes developing and maintaining the company’s corporate image and identity, as in the use of logos and slogan. As well as drafting speeches for our company’s executives/ developers and arranging interviews for them.

Our Information Security Engineer will be Aaron. He will be in charge of helping us

safeguard our software's database. He will plan a set of security standards our software must follow and recommend security enhancements as needed. He will also help develop strategies for when we are responding and recovering from a security breach. Usually, he will be conducting tests to simulate an attack on our software to find weaknesses that might be exploited by others and patching them after.

Our contact email that users can reach our team is ask-lock@gmail.com, where you users and consumers will be able to contact us in a case of emergency as well as basic support.

A set of procedures that our team will go through everytime there is an an incident will be something like:

1. Preparation, it is the key to an incident response because a plan must be in place to support our team. eg. Develop IR policies, define communication guidelines, incorporate threat intelligence, conduct tests to find incidents that occur within our software, and assess our threat detection capability.
2. Detection and reporting, having our team monitoring security events in our software, detecting potential security incidents, creating incident tickets, documenting findings, and have regulatory reporting escalations.
3. Triage and analysis, classifying events, conducting correlation analysis, identifying the cause of an incident, analyzing intrusion artifacts and malware, and performing vulnerability analysis determining the risks.
4. Containment and neutralization, is the most crucial stage of incident response. The strategy for containment and neutralization is based on the intelligence and indicator that were gathered during the analysis phase. Wiping the infected devices and rebuilding the operating system from the ground up, as well as changing our hashing system. If we have identified domains or IP addresses that are leveraged by threat, we could request to block communication channels that are connected to those domains.
5. Post-incident activity, after an incident is resolved, it is important to properly document any information that could be used to prevent a similar incident from happening again in the future by completing an incident report. Even though one incident is over, our team

should closely monitor for activities post-incident to ensure the threats do not reappear again. Create new security initiatives to prevent future incidents.

FINAL SECURITY REVIEW:

For our final security review we will conduct the same tests that we conducted on a previous fuzz test. In these series of fuzz testing, because the developers are more familiar with the back end code, we should be able to identify a few attack vectors that other users may not have seen. The first we will test is the password test upon logging in.

Our first attempt was to use our UHM password that is using a special character, capital letters, and a number to create an account. Because of the way our backend works only one person could be working on this at a time. After we create an account and verify the hashing is done correctly using Chrome Canary, we fire up the macro and utilize the 1-million common password github file as our first test. This resulted in a success as we are unable to brute force break in to our own account. Plus, our account time-out worked.

For our static analysis, we utilize eslint along with general coding practices within IntelliJ to seek out depreciated code. From our previous run we did find some parts of code that was not up to par with EsLint standards and that has been fixed. After running through all of our code once again, EsLint was unable to target any errors with our code and we compiled normally.

Our final review is the review on the JSON file that holds some of the account credentials. This is a big security risk that we kept an eye on since the start of development however Meteor both hides the password and the username from the developers so that even they could not see the passwords such as my college password I have used on a previous test. As said before we scoured the developer tools on Chrome Canary as well as regular chrome and other browsers yet we could not access this JSON file. Meteor really came through for use in this part and this is why we are using this feature in our web application.

Our final grade would be **Passed FSR with exceptions**. Our reasoning for this was the fact that in the application we were unable to manage hiding this password within the UI of the web application. This is a giant security risk if you have anyone viewing your computer at the

same time you are using this feature. We are currently looking at this being remedied at this time however the overall function of saving, editing, and deleting is still able to be used.

CERTIFIED RELEASE & ARCHIVE REPORT:

Link: https://github.com/ASK-LOCK/ASKLOCK/releases

Features:

- **Store accounts and passwords**
- **Hashing stored password**

Features to be added:

- **Email verification**
- **Password generation**
- **Unique second factor authentication**
- **Rubix cube authentication**
- **Chess authentication**
- **Switching from web-based to actual application**

User Guide Step-By-Step

1. After completing the general setup for this web application, load up the localhost on any browser.

2.Upon loading the home page you are greeted with the ASK LOCK logo. Click on the lock to start the login or registration process. Continue to step 3 to continue the registration process. If you have already registered, continue to step 4

3.To register, enter a valid email and a password then hit submit, after registering you will be redirected to the Add Account page to begin using our service.

4.The Add Account page is simple, simply type in the name of the account you are using. Example FaceBook, Twitter, Discord.. etc. Then type in your username for that account along with your password. Hit submit when you are done.

5.To view, delete, or edit your account click on the ... button in the navbar. Click on edit to edit any field and click delete to delete the entire record of that account.

6.To log out, click on the icon at the top right then click sign out.

##

##

Closing thoughts written by all members of our team.

Challenges:

1. There has been little feedback to the past assignments so we were unable to discern if we are going in the right direction.
2. The general Corona outbreak led to a disruption in our organization.
3. Understanding the SDL and any other materials regarding user security. For all three of us this is our first security class.

Surprises:

1. Surprised by no one giving us feedback. There have been multiple assignments yet our current grade and direction of our project is a mystery
2. How hard is it to code hashing into our program.
3. The defaults on some softwares to help us create a secure webservice. Example Meteor.

Important Achievements:

1. We were able to understand animations better for web design.
2. We actually got a hash to work.
3. We started with prototypes to a minorly ready application.

Disappointments:

1. Too many