TEAM 4: RISK REGISTER

This process helped our team identify potential risks or considerations we need to take to protect our application and data. We started by identifying potential vulnerabilities in our system and connections between the different parts of the system. Essentially, we looked at the trust boundaries between our components. For instance, if the user's Chrome account is compromised, or if our backend is compromised, what are the mitigations we could take. As for the user being hacked or their device being stolen, there is minimal mitigations we can take. However, if we assume that the user is an adversary by default, we can add protections to our backend to ensure that their range of potential attacks or influence is limited while still allowing them to use the product as intended. For our backend, it is all hosted on AWS, so we did some research into what protections we can include, and what protections they could provide. Although these considerations do not change our design, it made us realize that there are things we need to add or implement on top of our system.

There weren't distinct or serious challenges that arose from this deliverable, but rather considerations as to what we can realistically do and implement. For instance, we can't ensure that every user is going to use our product for good, there will be users that will try to break our application. The challenge is then figuring out the steps or precautions we can take to ensure that normal users can still use our application while preventing the malicious users from accessing data or taking down our application. As for the backend and AWS, we had to look at the offerings available that are free and that would be relevant to our application. We had to identify features for our different instances that we can leverage without hurting the performance of our application. For example, rate limiting our API Gateway could potentially block users from getting their passwords in a timely manner. Although we can't specify specific metrics at the moment for this example, it is something we would need to test and find a balance between speed and protection.

Link Risk Register (Or see below):
https://docs.google.com/spreadsheets/d/1iYboAG0T5YNln0yAgqYpAT7teah8g03ncTmfkU7WHRY/edit?gid=0#gid=0

ID	Risk Description	Likelihood	Impact	Risk Rating	Migitation/Action Steps	Review Date
1	An adversary is able to gain access our RDS database on AWS. This implies that they are able to see all fields and entries in our database.	Unlikely	Moderate	Low Medium	Our database is encrypted using AES, so that an adversary would not be able to see the fields in the database. Each user has their own AES encryption key which is their base password; there is no master key, so even if an adversary is able to get a key, only one user would be compromised. Given that AES-256 is currently immune to differential/statistical attacks and the most optimized brute force is 2^254.4, we do not feel that an attacker cracking AES is an issue. Our database is on AWS, which requires 2FA to increase the security of our accounts, thus making account integrity high.	2/18/2025
2	An adversary attempts to cause a Denial-of-Service, specifially through a DDoS attack. One way this can be set is through a user creating an account and using the generated token to add as many as entries as possible incuring high usage of our system. Another way would be if an adversary gets the URL to our API and tries to send as many requests as possible.	Possible	Significant	Medium High	We can set rate limiters in API Gateway such that a minimal amount of requests can be handled per second, this would prevent individuals from trying to process the same endpoint hundreds of time. In addition, AWS has built in DoS protections and firewalls to prevent large scale attacks. They provide a free service called AWS Shield which has monitoring systems and firewalls to further limit DDoS exploitations. Given that users may try to flood our database, we could potentially include a timer for each user where they can only register a certain number of sites in a given time (i.e. one website every 5 seconds). As for an adversary getting the URL, in API Gateway, we can parse the header and block any requests that don't contain authentication key which we will provide to users.	2/18/2025
3	Given that anyone that creates an account with our extension gets an authenticated token, any user can try to implement SQL injection on our database.	Very Likely	Moderate	Medium High	We will have limited priviledges such that if an adversary drops to insert a DROP or DELETE, the database will reject the command as they will be set to read-only. In addition, we will use prepared statements with parametrized queries. This means that we create the query and add fields from the users json fields. This means that the parameters passed from the user will only be treated as data and not as code. We can also employ filters and input validators that ensure that any code is converted to data or if there are characters used in commands (i.e. ;), they are removed from the data.	2/18/2025