1 Security

1.1 Risk Identification

1.1.1 Assets

The application consists of multiple services and tools that could all potentially be vulnerable to an attack. These are:

- Web application and stack
- Digital Ocean droplets
- ELK logging chain
- Metrics service (Grafana)
- Github source control (public repository)

1.1.2 Threat Sources

1. Web application and stack
    - SQL injection
    - cross site scripting
    - Open ports
    - Distributed Denial of Service
    - Big messages (DOS)
2. Digital Ocean droplets
    - Unauthorized access
3. ELK logging chain
    - Unauthorized access
4. Metrics service (Grafana)
    - Public metrics gives information about the application status
    - Unauthorized access
5. Github source control (public repository)
    - Unauthorized access

1.1.3 Risk Scenarios

We will now describe the threats and assets in terms of the threats impact on the assets and what security principles they break. To quickly summarize the three CIA security principles:

1. Confidentiality is the principle of keeping sensitive user data out of the hands of anyone but the user and the application.
2. Integrity is the principle of keeping data complete or whole. The data should not be cut, deleted or altered and can therefore be trusted.
3. Availability is the principle of keeping the service or data available without interruption or error.

Scenarios - Marked with corresponding security principle violation

C Attacker gains information about users due to incorrect request handling.

C/I Attacker performs cross site scripting to mine user data or fool users to submit their data

A Attacker sends very large message to bottleneck message-handling or fill database.

A Attacker performs cross site scripting to deny users access to the site

C/I/A Attacker performs SQL injection on web application to download/alter/delete possibly sensitive user data.

C/I/A Attacker finds vulnerability in a dependency listed on the public GitHub repository, potentially allowing full access to an arbitrary part of the system.

C/I/A Attacker gains access to any service through leaked secrets.

1.2 Risk Analysis

1.2.1 Determine likelihood

1. SQL injection, likelihood: very low
   The likelihood of this occurring is fairly small, as we sanitize the data received from users, this however should always be a strong consideration if new features where to be implemented.
2. cross site scripting, likelihood: low
   Same as the likelihood for SQL injections, the most important factor for this is to ensure proper sanitation of user input.

3. Open ports, Likelihood: very low
   This problem can be verified fairly easily by performing a scan on the machine to verify whether or not this threat exists. One thing that may be worth consideration is how to approach the issue when a project scales though services exist, such as Shodan, Metaspoilt, etc., to verify if the issue exists on larger networks.

4. Unauthorized access, likelihood: medium
   From examples early in the course we have seen that it is very important to stay vigilant on access to our services, as these can very easily be a either accessed by malicious actors, or completely shutdown.

5. Knowledge of entire stack (Dependencies and versions), likelihood: high
   This one most likely has a high chance of occurring as many web scrapers exist, that try to gain information of services through their repositories. Whether or not this is a problem is up for debate though, as obfuscation alone, is not a strong enough method for ensuring security. The likelihood of bad actors successfully finding useful vulnerabilities is a completely different case which is much less likely.

6. Distributed Denial of Service, likelihood: very low
   This could have a pretty big effect on our service, but in terms of likelihood, it isn't very high on the list. A distributed denial of service attack would require a level of resources from the attacker, that automatically excludes many adversaries. Because of this we rank it as very low.

7. Big messages (DOS), likelihood: very high
   The likelihood of a bad actor trying to upload big messages is very high, as they are prompted for input, making them curious how big that input can be.

1.2.2 Determine impact

1. SQL injection, impact: critical
   If an SQL injection exploit is found, that gives access to query the database, it could likely give an attacker access to all data, and modifying all data, because of the querying tool having full privileges to the database.

2. cross site scripting, impact: high
   Depending on the type of XSS attack, the impact may vary. If scripts are somehow deposited to the database and rendered by users, it has much higher impact than if an attacker gains access to a session cookie or can log user input.

3. Open ports, impact: high
   When a systems ports are open an attacker can access any vulnerable services running on those ports.

4. Unauthorized access, impact: critical
   An attacker gaining unauthorized access to any of our services will either grant large amounts of information about our data and application, or it will give unlimited access, in form of an API-key, that would let them close the service or take it over.

5. Knowledge of entire stack (Dependencies and versions), impact:very low
   While it is likely that someone will inspect the technology stack, it should not be a cause for concern as we use tools, such as Snyk, to check for vulnerabilities in our dependencies, and tools like SonarQube and CodeQL to check for errors and vulnerabilities in our application, either catching errors before they become problems or mitigating their impact by updating to more secure.

6. Distributed Denial of Service, impact: medium
   The impact of a DDOS attack is loss of availability, but usually not for extended periods of time. It is rare for these issues to persist longer than weeks or days.

7. Big messages (DOS), impact: medium
   Allowing users to send as large messages as they want will likely lead to bottle-necking some part of the request handling. In the worst case, an attacker can send many messages so large that our database is filled and will require more space which will, in turn, cost the team more money. If we in the future added censorship rules to messages, the application would also have to parse the content of these huge requests, taking many CPU-resources of our backend.

1.2.3 Risk Matrix

Likelihood \ Impact	very low	low	medium	high	critical
very low			DDOS	Open ports	SQL injection
low				XSS	
medium					Unauthorized access
high	Knowledge of stack				
very high			Big messages		

1.2.4 Solutions/mitigations

1. Sending big messages to saturate network connection can be mitigated heavily by imposing character limits on the length of messages. (To be implemented)

2. There is a multitude of port scanning tools available that would allow the team to ensure no port is left open.

3. The likelihood of leaking secrets is reduced by us enforcing required reviews on pull requests to the repository, making uploading keys less likely. Likewise the developers should ensure that no authentication information is shared through insecure methods. **This was implemented by the team.**

4. There are a few defence measures one can employ to reduce the impact of DDOS attacks such as rate limiting everyone so that applications to crash, detecting bot patterns and banning/blacklisting bad actors. The second

measure is usually harder to build, and is therefore usually outsourced to companies like cloudflare or others.