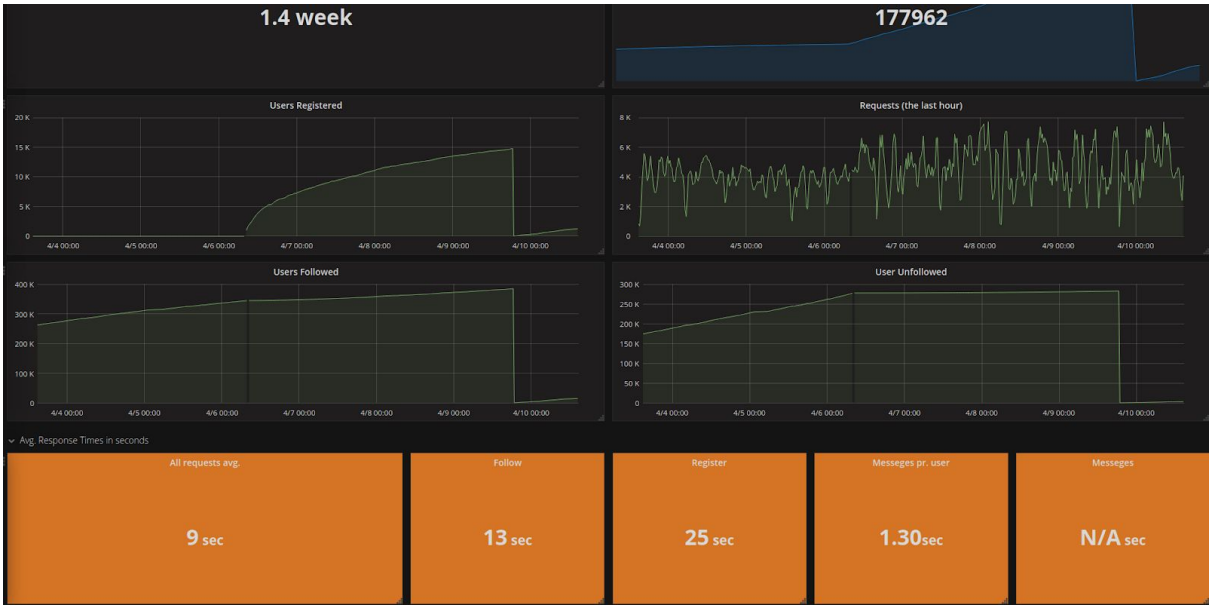
Monitoring group F

After looking through the SLA we received, we took note of what to look for before heading over to Grafana. After opening up Grafana (http://134.122.83.72:3000/dashboard/db/minitwit?orgId=1), we saw the following:



The metrics improved, but still remained in violation of the SLA after an update:

As we can see, this violates the SLA requirement we received from group F, as seen below:

The service performance level

The team strives for:

- uptime of 95%
- an avg response time under 2 seconds
- an avg time to recover on 1 hour from the time that the team is aware of the issue

The SLA clearly states that average response time should be “under 2 seconds”, and Grafana reveals that their average response time is 9 seconds, across all message type requests.

The specification in the SLA document for uptime, is not explained enough to be measured. The only data we can see for uptime is a number in Grafana. This does not provide any measurable information for us and thus we cannot measure the performance levels that the team strives for.

Security testing group F

09-04-2020:
We were unable to perform a penetration test of Group F's system, as we could not obtain access to their back or frontend from the links provided on the repositories.py files. Instead, we will outline a series of blackbox tests we would have performed, if the system had been available for us:

- NMap - Scan for open ports and hopefully obtain some information of the server
- Injection attacks
    - XSS - Attempt to insert javascript code (e.g. <script>alert("hello")</script>) in all user input fields to see if fields are being sanitized
    - SQL Injection - Attempt to gracefully inject the SQL database, i.e. read-only queries to not mess with their data.
- Cookie-based attack. See if their authentication cookies are using strong secrets.
- Metasploit - Use metasploit to find vulnerabilities based on the results of NMap
- Password attacks - Use wordlists to crack user passwords to see if the website enforces a strong password policy for its users

Additionally, it seems that TLS has not been implemented, making the website vulnerable to a Man-In-The-Middle attack.

We have sent group F an email regarding the missing access, and are awaiting a response.

15-04-2020:
We received feedback from group F regarding the missing access the 14-04-2020. They provided a new link to the frontend, so that we could access it. However, we received this information one day prior to the deadline, and have therefore been unable to conduct a penetration test given the time remaining for the deadline. Furthermore, the fault introduced in the system was also introduced as per the 14-04-2020, to our knowledge, so we have not been able to investigate it further.

Security testing our system

System assets:

1. Database = terrible
2. API in risk (authorization)

			probability of occurrence			
	RISK ASSESSMENT MATRIX		**very unlikely** little to no chance of occuring	**unlikely** rare combination of events would need to coincide for this to happen	**possible** small but significant chance of happening, still more probable not to happen then to happen	**probable** more likely to happen than not to happen
	low priority					
	moderate priority					
	high priority					
Impact	**minor** service impeded					
	moderate service impeded					
	serious service significantly disrupted, SLA violated, user privacy violated					
	major service has to be brought down					

How to read the matrix: for example, an attack on user authentication is very likely to happen (probability=probable) and if it succeeds it would violate user privacy (impact==serious), so this security concern is high priority. On the other hand, a hardware fault is unlikely to happen since we use Digital Ocean. If we were to have a hardware fault however and not able to access the droplet again we would lose the data, which is problematic.

To pen test our system we looked into known issues with our system. We have two issues from OWASP Top 10:

1. Insufficient logging and monitoring (owasp nr. 10)
2. Missing authentication (owasp nr 2)

We see that the missing authentication has high priority with respect to our risk matrix since it's very probable and a serious violation.

We have worked to remedy both of these issues first by introducing additional logging to the system in the following classes:

- AuthController
- LatestController

- TimelineController
- UserController

Specifically, all controllers now have logging implemented. We log information we deem important, such as exceptions, failed login attempts, as well as successful login attempts. We have implemented logging using SeriLog, Elasticsearch and Kafka to facilitate our logging needs.

We initially had authentication in the API however due to the difference in between the simulator and initial API-implementation we rewrote, we decided to omit authentication instead of reimplementing it. This meant we had authentication in place, however it was not enabled. To enable authentication we had to introduce both the initial authentication and an option to use the simulator authentication. To do this we included an additional statement to the authentication which validates the header:

```
(Request.Headers.TryGetValue("Authorization", out var header) &&
header.Equals(AuthorizationConstants.terribleHackAuth)))
```

We extract the authorization header value and compare with the value which the simulator uses. We have hardcoded the authorization header value, which still opens up the system, but given the circumstances it's the best we can do.