## Explain basic security threads like: Cross Site Scripting (XSS), SQL Injection and whether something similar to SQL injection is possible with NoSQL databases like MongoDB, and DOS-attacks. Explain/demonstrate ways to cope with these problems

* XSS enables attackers to inject client side scripts into web pages viewed by others users. This can be used to gain access to cookies and other valuable stored data.
* SQL injection is where someone tries to gain access to a sites database. This is done by injection SQL into a login by writing “’admin’ OR 1=1 --“. Everything written after the two – are commands that can manipulate the database. This is pretty easy to secure against though. Instead of using a normal statement, a prepared statement should be used. Prepared statements doesn’t completely secure against SQL injections though as if the user input are not used as parameters in the statement you will still be vulnerable to SQL injections.
* A DOS (Denial of service) attack is where multiple requests are sent towards a server to deny the real traffic access to the server. This is usually done via a bot network.
* SQL injection on MongoDB: there is not something directly similar to SQL injection with a NoSQL database, but there are other security issues – e.g. with MongoDB you don’t have an admin password thereby having the database “exposed” if the communication port 27017 or 28017 is open on the server.

## Explain and demonstrate ways to protect user passwords on our backend and why this is necessary

By making sure, our passwords keep their integrity. This can be achieved by using a hashing method/function, and then adding some salt to secure the password even further.

## Explain about password hashing, salts, and the difference between bcrypt and older (not recommended) algorithms like sha1, md4 etc.

A hashing algorithm is a one-way function to “hide” our passwords in the backend. One-way means that it’s impossible to get the password after hashing it by running it through a function. This has been by-passed by people using a spreadsheet, where a lot of different words are being run through the hashing algorithm and saving the hashed value to the unhashed primary key value. So more security is needed.

This is where salt comes in. So before a password is created some “salt” is added to the password and then the password is hashed. It’s important that the function adding the salt is a slow function, so that even though the people we are securing against knows both the salt, and the hash it would take way to long to create a spreadsheet with the hashed and unhashed values.

## Explain about JSON Web Tokens (jwt) and why they are very suited for a REST-based API

A jwt is a token that can be used to authenticate a user just as a cookie, but has the added function of also being able to send data, because it has a body, called a payload. A jwt looks like this:

JWT eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.

eyJhdWQiOiJ5b3Vyc2l0ZS5uZXQiLCJpc3MiOiJ5b3VyY29tcGFueUBzb21ld2hlcmUuY29tIiwiaWF0IjoxNDYwNDQ0NDk0LjA5MywiZXhwIjoxNDYwNDQ0NTU0LjA5M30.

FIq8Dt61-5MIIe1q8mHgRdXCuE8B-hzFAyLhJ3AWUw8

This looks really dangerous but it’s using base-64 encryption and the different pars are separated by dots. Everything until the first dot is the header. Then everything until the second dot is the payload, and the rest is the signature.

## Explain and demonstrate a system using jwt’s, focusing on both client and server side