Explain basic security terms like authentication, authorization, confidentiality, integrity, SSL/TLS and provide examples of how you have used them.

- **Authentication**
  Confirming the identity of a client (via some kind of login procedure).
- **Authorization**
  Determining whether an authenticated client is allowed to receive a service or perform an operation.
- **Confidentiality (privacy)**
  Protection from disclosure to unauthorized persons.
- **Integrity**
  Maintaining data consistency (data cannot be modified).
- **SSL/TSL**
  Secure **S**ockets **L**ayer and **T**ransport **L**ayer **S**ecurity are used to authenticate servers and clients across untrusted networks and encrypt data sent between authenticated parties.

Example: se SSLNodeExpress.js

Explain, at a fundamental level, the technologies involved, and the steps required to initialize a SSL connection between a browser and a server and how to use SSL in a secure way.

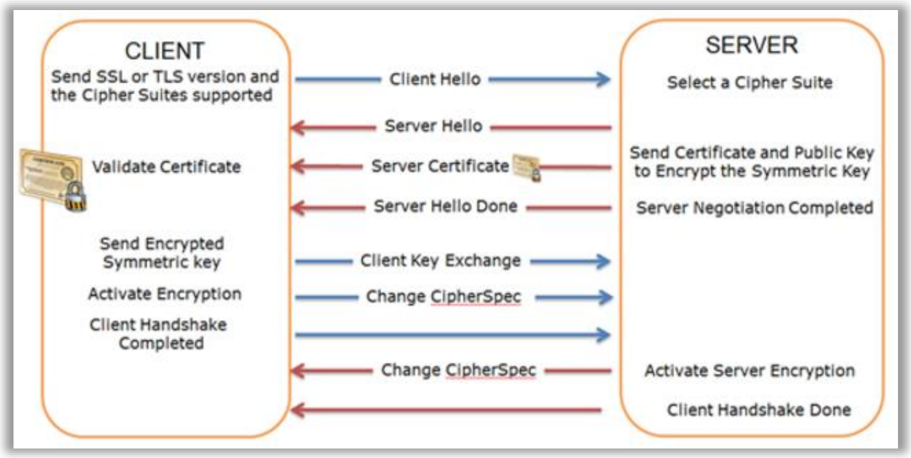
The steps required initialize a SSL connection between a browser and a server:

1. The browser will Request the web server identify itself.
2. This prompts the web server to send the browser a copy of the SSL Certificate.
3. The browser checks to see if the SSL Certificate is trusted.
4. If the SSL Certificate is trusted, then the browser sends a message to the Web server.
5. The server then responds to the browser with a digitally signed acknowledgement to start an SSLencrypted session. This allows encrypted data to be shared between the browser and the server. You may notice that your browsing session now starts with https (and not http).

How to use SSL in a secure way:

Using the Default WebLogic Server Host Name Verifier As a function of the SSL handshake, WebLogic Server compares the common name in the SubjectDN in the SSL server's digital certificate with the host name of the SSL server used to accept the SSL connection. If these names do not match exactly, the SSL connection is dropped.
The SSL client is the actual party that drops the SSL connection if the names do not match.
If anything other than the default behavior is desired, either turn off host name verification or configure a custom host name verifier.
Turning off host name verification leaves WebLogic Server vulnerable to man-in-the-middle attacks. Oracle recommends leaving host name verification on in production environments.
If you are using the default WebLogic Server host name verifier, host name verification passes if both of the following conditions exist:

- The host name in the certificate matches the local machine's host name.
- The URL specifies localhost, 127.0.01, or the default IP address of the local machine.


CLIENT
Send SSL or TLS version and the Cipher Suites supported
Validate Certificate
Send Encrypted Symmetric key
Activate Encryption
Client Handshake Completed
Client Hello
Server Hello
Server Certificate
Server Hello Done
Client Key Exchange
Change CipherSpec
Change CipherSpec
SERVER
Select a Cipher Suite
Send Certificate and Public Key to Encrypt the Symmetric Key
Server Negotiation Completed
Activate Server Encryption
Client Handshake Done

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 problem

XSS enables attackers to inject client-side scripts into web pages viewed by other users.
A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy.
Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities documented by Symantec as of 2007.
Their effect may range from a petty nuisance to a significant security risk, depending on the sensitivity of the data handled by the
vulnerable site and the nature of any security mitigation implemented by the site's owner.

A way to prevent or at least limit this is to:

1. Encode every data that is given by a user. If data is not given by a user but supplied via the GET parameter, encode these data too.
   Even a POST form can contain XSS vectors. So, every time you are going to use a variable value on the website, try cleaning for XSS.
   These are the main data that must be properly sanitized before being used on your website:
   The URL, HTTP referrer objects, GET parameters from a form, POST parameters from a form, Window.location, Document.referrer,
   document.location, document.URL, document.URLUnencoded, cookie data, headers data and database data, if not properly validated on user input.
2. Sanitize HTML markup with a Library designed for the job.
3. Prevent DOM-based XSS

SQL Injection is an attack method where hacker insert malicious SQL code into a Web form to gain access to resources, applications or databases. It has been on the rise with the advancement of automated exploit tools, and the attack method, which can enable data manipulation and the spread of malware, is becoming more advanced and popular among attackers.

An attacker uses SQL injection to manipulate a site's Web-based interfaces and force the database to execute undesirable SQL code, enabling data manipulation and spreading malware.

In order to prevent these types of attacks:

1. Enterprises must implement secure coding best practices and limit Web application coding privileges.
   Ensure employee security awareness: Make sure that employees who have a hand in website development (as well as dedicated Web developers) are aware of the SQL injection threat and know best practices to keep your servers safe.
2. Reduce debugging information.
   When a Web server experiences an error, make sure details of the error aren't displayed to the user, since this information could help a hacker commit malicious activity and gain the information he or she needs to successfully attack the server.
3. Test Web applications regularly.
   Test Web applications and check Web developers work by sending data through the Web server; if the result is an error message, the application might be susceptible to an SQL injection attack.

SQL Injection is it possible with NoSQL databases like MongoDB?

NoSQL definitely does not imply zero risk. In fact, NoSQL databases are vulnerable to injection attacks, cross-site request forgery (CSRF) and other vulnerabilities.
One of the common reasons for a SQL injection vulnerability is building the query from string literals which include user input without using proper encoding. The JSON query structure makes it harder to achieve in modern data stores like MongoDB. Nevertheless it is still possible.

Ex. Let us examine a login form which sends its username and password parameters via an HTTP POST to the backend which constructs the query by concatenating strings.

string query = “{ username: ‘” + post_username + “’, password: ‘” + post_password + “’ }”

But with malicious input this query can be turned to ignore the password and login into a user account without the password, here is an example for malicious input:

username=tolkien’, $or: [{}, { ‘a’:’a&password=’ }], $comment:’successful MongoDB injection’

This attack will succeed in any case the username is correct, an assumption which is valid since harvesting user names isn’t hard to achieve.
The password becomes a redundant part of the query since an empty query {} is always true and the comment in the end does not affect the query.
Another reason is that Javascript execution exposes a dangerous attack surface if un-escaped or not sufficiently escaped user input finds its way to the query.
Furthermore the REST API can expose the NoSQL database to CSRF attacks allowing an attacker to bypass firewalls and other perimeter defenses.

In order to prevent these types of injection attacks:

1. Security scanning to prevent injections -

It is recommended to use out of the box encoding tools when building queries. For JSON queries in MongoDB have good native encoding which will terminate the injection risk. It is also recommended to run Dynamic Application Security Testing (DAST) and static code analysis on the application in order to find any injection vulnerabilities if coding guidelines were not followed

2. REST API exposure

To mitigate the risks of REST API exposure and CSRF attacks, there is a need to control the requests limiting their format.

*It is here important to notice that databases like MongoDB have many third party REST API's which are encouraged by the main project, some of these are really lacking in the security measures we described here.

3. Access Control and Prevention of Privilege Escalation

Utilizing proper authentication and role management mechanisms is important for two reasons. First, they allow enforcing the principle of least privilege thus preventing privilege escalation attacks by legitimate users. Second, similarly to SQL injection attacks, proper privilege isolation allows to minimize the damage in case of data store exposure via the above described injections.

This can be done by making the data accessible via a web application authorized with a "user" role, while the sensitive entries require the "admin" role, which is never granted via the web interface. This allows scoping the damage in case of attack, ensuring that no administrators' data is leaked.

DOS-attacks

Denial of Service (DoS) are attacks against web sites occur when an attacker attempts to make the web server, or servers, unavailable to serve up the web sites they host to legitimate visitors. For some time, it was thought that these types of attacks were generally used against large corporations, government sites, and activist sites as a form of protest to disrupt their web presence.

Denial of Service attacks can result in significant loss of service, money and reputation for organizations. Typically, the loss of service is the inability of a particular network service, such as e-mail, to be available or the temporary loss of all network connectivity and services. An HTTP Denial of Service attack can also destroy programming and files in affected computer systems.

In some cases, HTTP DoS attacks have forced Web sites accessed by millions of people to temporarily cease operation.

In order to reduce the risk of this type of attack:

1. Purchase a lot of bandwidth.

This is not only the easiest solution, but also the most expensive. If you simply have tons of bandwidth, it makes perpetrating a DoS attack much more difficult because it's more bandwidth that an attacker has to clog.

2. Use DoS attack detection technology.

Intrusion prevention system and firewall manufacturers now offer DoS protection technologies that include signature detection and connection verification techniques to limit the success of DoS attacks.

3. Prepare for DoS response.

The use of throttling and rate-limiting technologies can reduce the effects of a DoS attack. One such response mode stops all new inbound connections in the event of a DoS attack, allowing established connections and new outbound connections to continue.

DoS protection is more "art" than science, requiring a combination of techniques to limit the impact of such an attack on your organization.