Backend Web Development

1)

One difference between a Relational Database Vs a NoSQL Database is the NoSQL kind are built around JSON Documents with key value pairs where it's easy to add new attributes to and modify whilst Relational Databases eg MySQL focus more on tables with fixed columns and rows (a nightmare to modify without breaking things, especially on a live site).

SQL & MySQL history learned from
NoSQL vs SQL databases (no date) MongoDB. Available at:
https://www.mongodb.com/nosql-explained/nosql-vs-sql (Accessed: 25 October 2023).

SQL databases came from the 1970s where there was a greater focus on avoiding data duplication and dealing with limited hardware storage issues. NoSQL databases were developed in the late 2000s. With a "a focus on scaling, fast queries, allowing for frequent application changes, and making programming simpler for developers."

2)

Frameworks help developers know how they should structure their code and what modules are essential for carrying out a lot of common tasks. Frameworks also have communities where developers can support each other and assist with security /bug patches & feature improvements. They can rapidly notify each other of any major security issues. Having devs gather around a common framework is good for networking and getting to meet up, I used to be a Drupal developer and would attend various Drupal camps.

The common community based documentation of frameworks is also helpful, if you're ill and someone has to take care of your code then that person can familiarize themselves with pre existing official documentation so they can understand how the non custom aspects work. Also great when hiring new people if they're already familiar with how a common framework works rather than having them dive into 100% uniquely made custom code.

Flask is similar to Sinatra and Express, it is a light weight unopinionated Application Server you'd normally use to handle RESTful requests.

Django is closer to Laravel & Ruby on Rails. They're large opinionated frameworks, in sacrificing some freedom you can use a powerful cli that can help structure your web applications in a very common manner to other developers at an incredibly fast pace. They provide ORM (Object Relational Mapper) for interacting with the database, authentication, email validation tools and other helpful libraries for common web development tasks.

Both Flask and Django use Python.

3)

Some security considerations you need to consider when designing a website/application.

- What is the minimum amount of permissions access I can provide to the user and do I need to create numerous roles that only have access to specific aspects of the site?
- What is the maximum amount of abstraction I can provide to external users not just for simplicity but also secrecy should an attacker find weaknesses (eg using an outdated framework)?

- Should it be public facing or just exist on an intranet?
- Will I need Cloudflare to handle DDoS attacks along with fallack databases and additional caching tools?
- How regularly should I be taking database backups, have I run drills to see how fast I can get the site back up if it's overwhelmed with bots and who should have access to these backups?
- Is it location specific so you can block out foreign IP addresses or will your uses be using VPNs?
- Is the framework or language I'm using still being maintained and getting regular security updates?
- Is it easy to carry out regular security updates on my code & server operating system without constantly breaking the site?
- Is it possible to carry out SQL Injections due to a lack of filtering in my webforms?
- How trustworthy are my hosting providers with my data?
- How susceptible are the third party libraries I'm using to malware?

4) Web Servers, Application Servers, and Proxy servers.

Web Servers are computers with internet connections that handle HTTP requests from web browsers & make HTTP responses. The Web Server provides the browsers making requests with relevant site information such as HTML, images, vidoes, pdfs, CSS & JavaScript. HTTP requests can occur whenever someone with a web browser clicks on a link, submits a form or does a search. If the request the Web Server receives is non static then is has to be sent to the Application Server.

Application Servers contain business logic needed to create resources dynamically that a web browser requests so if all you had was a purely static site there wouldn't be any need for an Application Server - all you would need is a Web Server. Application Servers rely on server side languages eg PHP, Java or Python and use frameworks to handle RESTful requests (old devices might have to handle XML based SOAP requests) for instance PHP could use Symfony or Slim.

Proxy & Reverse Proxy learned from
What is a reverse proxy server? (2023) *NGINX*. Available at:
https://www.nginx.com/resources/glossary/reverse-proxy-server/ (Accessed: 25 October 2023).

"A proxy server is a go-between or intermediary server that forwards requests for content from multiple clients to different servers across the Internet. A reverse proxy server is a type of proxy server that typically sits behind the firewall in a private network and directs client requests to the appropriate backend server. A reverse proxy provides an additional level of abstraction and control to ensure the smooth flow of network traffic between clients and servers."

A VPN like Proton VPN is a good example of a proxy service since when you access a website that website sees the IP address the VPN is providing rather than your own maintaining your privacy. So the VPN is sitting between your own web browser and the website you're trying to visit passing information back and forth between the two.

Cloudflare is a good example of a reverse proxy service, it sits in front of the website and prevents malicious bots from being able to take down the website with endless requests so only valid human users can access the website.

Clouflare simplified summary learned from
(No date) What is a reverse proxy? | proxy servers explained | Cloudflare. Available at:
https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/ (Accessed: 25 October 2023).

"A forward proxy sits in front of a client and ensures that no origin server ever communicates directly with that specific client. On the other hand, a reverse proxy sits in front of an origin server and ensures that no client ever communicates directly with that origin server."

5)

For simplicity's sake when explaing the steps involved in handling HTTP requests in a backend application I've omitted forward and reserve proxies.

A typical HTTP session learned from
MozDevNet (no date) A typical HTTP session - http: MDN, HTTP | MDN. Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Session (Accessed: 25 October 2023).

1) Client establishes a connection (often a TCP one, others might be SFTP or SSH).
2) Client sends request and waits for an answer.
3) Server processes the request and responds with a status code and appropriate data.

If this is just a static site then a Web Server handles the issues just responding in Html, CSS, JS and files/images.

However if it requires dynamic content then the Application Server gets involved with the backend. The client sends either Get or Post requests with a URL encoding data (sometimes form data) and possibly cookies. The Application Server steps in, carries out any business logic with its server side language to dynamically generate content and retrieves the relevant files or database information to handle the request (assuming the client meets both authentication and authorization permissions if they exist). The content is generated in Html and sent to the Web Server who sends the relevant data and response back to the client browser.