Interview Preparation

<https://geektastic.com/blog/five-killer-interview-questions-for-developers?&ref=quuu&utm_content=buffer16252&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer>

What happens between typing a URL into your browser address bar, hitting enter and seeing a web page?

DNS, etc, Protocol, network knowledge

What are the things you should consider if you were writing your own database server?

ACID

Atomicity, Consistency, Independence, Durability

Computer security considerations

CIA

Confidentiality, Integrity, Availability

OWASP

Top 10 current threats

1. Injection (SQL Injection)
2. Broken authentication and session management: leaks in the authentication system, poor passwords management in all the areas, such as resets, etc, session management – not timing out, leaking data in session state info
3. Cross site scripting (XSS): Injecting scripts, e.g. javascript because input is not properly escaped, or validated
4. Broken Access Control: e.g. horizontal or vertical privilege escalation
5. Security misconfiguration: leaving default accounts around, unpatched flaws, unprotected files or directories
6. Sensitive data exposure: hackers stealing data, man-in-the-middle, stealing keys. Not encrypting sensitive data.
7. Insufficient attack protection: detecting and blocking attacks, rather than ignoring them, e.g. incorrect password lockout.
8. Cross site request forgery
9. Using components with known vulnerabilities: teams not keeping libraries up to date
10. Under protected APIs
11. Click-jacking
12. Denial of service
13. Information leakage, incorrect error handling
14. Malicious file execution
15. Unvalidated redirects and forwards

Some encryption knowledge in .NET – what is currently the best practice?

**Symmetric algorithms**

**AES** – Advanced Encryption Standard (Replaces DES). A subset of Rijndael but with a fixed block size of 128bits, the jkey can be 128, 192 or 256 bits.

DES – Data Encryption Standard (old – not very secure – can be brute forced)

RC2 – again old don’t use

Triple DES or 3DES – is a more secure version of DES, but cascading 3 instances of DES. It is slow.

Rijndael – block size can be specified in size o 32 bits (from 128 – 256), AES is based on it but with a fixed block size of 128 bit.

**Asymmetric algorithms**

Public and private keys are inverses of each other: public key enc – private key dec & private key end – public key dec. Asymmetric encryption is quite slow, so typically if encrypting a lot of data, you would use symmetric encryption and then use asymmetric encryption to encrypt the key that you provide for decryption.

**RSA**

**Hash algorithms**

MD5 – susceptible to collisions where 2 messages can have the same hash

SHA1 – still susceptible but not as much as MD5

**SHA2** – the most resistant – use this – in .Net this is SHA256, SHA384, SHA512

SHA3 – this is the newest standard but question marks over performance

<https://www.cryptologie.net/article/400/maybe-dont-skip-sha-3/>

BLAKE, SHAKE, BLAKE2

Consider how secure the data needs to be and also how quickly it needs to hash, because the less secure versions are faster.

<https://stackoverflow.com/questions/5554526/comparison-of-des-triple-des-aes-blowfish-encryption-for-data>

Block cipher: encryption is applied to blocks of data rather than one bit at a time.

Initialization Vector: this should be used with AES to prevent the same data being encrypted exactly the same way. So it add some random initial block which will change how the whole encryption array looks.

Public key to encrypt – private key to decript

Symmetric algorithms

Uses the same key to encrypt and decrypt

Hashing

TLS/SSL secure cookies, etc

<https://news.ycombinator.com/item?id=14642193>

Also, know more about secure cookies, and SSL certificates, etc.

**Microservices – patterns**

* MTTR – mean time to recovery – **most important one**
* MTTD – mean time to discovery
* MTBF – mean time between failures

Prerequisites for microservices

<http://philcalcado.com/2017/06/11/calcados_microservices_prerequisites.html>

1. Rapid provisioning of compute resources
2. Basic monitoring
3. Rapid deployment
4. Easy to provision storage
5. Easy access to the edge: edge gateway, rather than going through a monolith
6. Authentication/authorisation
7. Standardised RPC

Bitwise operations

Algorithms

* Sorting
* Searching
* Memoization
  + Ensures that a function doesn’t run more than once for the same inputs by caching previous results (e.g. in a dictionary).
* Bottom-up
  + Recursion often “starts from the end and works backwards”, bottom-up avoids recursion because it 2starts from the beginning”.
* Dynamic programming: breaking a complex problem into smaller units and using memorization and/or bottom-up.

Code quality

SOLID

Design Patterns

C#

* Streams
* Async
  + Task, Task<T>, async, await
  + Once **await** is executed, the point in the method is exited and returned to the caller. Once the Task is complete, the code in the method pics up from that point.
  + Task.
* Delegates
* Examples of all the built in Interfaces
  + IAsyncResult
  + ICloneable
  + IComparable
  + IConvertible
  + ICustomFormatter
  + IEquatable
  + IFormatProvider
  + IFormattable

**Javascript**

This binding types;

1 – default binding

2 – Implicit binding

3 – Explicit binding

4 – new binding

Alternative is to use lexical scoping

**Closures**

Class inheritance vs prototypal inheritance

Be able to explain prototypes

Domain Driven Design

Enterprise Design Patterns

**SQL Server**

* Performance
* System Databases
  + Master: records all system level information for an SQL Instance, e.g. logons, endpoints, linked servers, system configuration settings
  + Msdb: Used by sql server agent for scheduling alerts and jobs
  + Model: used as the template for all databases created on the instance of SQL Server. Modifications made to the model, e.g. database size, collation, etc are applied to all databases created afterwards.
  + Tempdb: a workspace for holding temporary objects or immediate result sets. It is shared by all users connecting to the instance.
  + Resource database: a readonly database that contains all the system objects such as sys.objects. It allows quicker upgrades as the new SQL server version just has to update this DB rather than drop and recreating system objects.

**DOTNET Core**

* dotnet new // creates a new little console app
* dotnet restore // restores nuget packages
* dotnet run // runs the application
* dotnet new –t lib // creates new library
* dotnet new –t xunittest // creates test project
* dotnet test // runs the tests

Project dependencies should be added in the project.json in the depencies section:

"dependencies": {

"System.Runtime.Serialization.Primitives": "4.3.0",

"xunit": "2.1.0",

"dotnet-test-xunit": "1.0.0-rc2-192208-24",

"Service": {

"target": "project"

}

}

Pipleline

ASP.NET Core is an HTTP pipeline, replaces the OWIN/Katana implementation of middleware.

ASP.NET Core has its own server called Kestrel.

**CCS**

* SASS: Syntactically awesome CSS
  + Pre-processing
  + Variables
  + Nesting
  + Mixins: groups of CSS ‘functions’ that can be reused throughout the site
  + Imports and partial CSS for reuse
  + Extend/Inheritance
  + Operators
* ITCSS
  + **S**ettings: global variables, feature toggles
  + **T**ools: Global mixins and functions
  + **R**eset: Normalize.css
  + **B**ase: global styles only on element selectors, e.g. h1, p
  + **O**bjects: skeleton; grids, spacing, positioning, no visual styling applied
  + **C**omponents: chunks of UI, anything with colours, borders, backgrounds, etc
  + **U**tilities: helpers, overrides, trumps
* BEM – Block Element Modified
  + Block: the sole root of the component (in its own file)
  + Element: A descendant of the block
  + Modifier: A variant or extension of the block

.c-car {}

.c-car\_\_wheel {}

.c-car\_\_wheel—alloy {}