**Introduction**

* Information about the project short description of the system
* Teams set-up
* Attitude to legacy code and back-end

**Self-presentation (English)**

* Ask candidate to present himself
* Describe last/current project: domain, size, teams, practices used
* Why he wants to change his work, expectations from new place
* Achievements

**OOP**

* What is the Object Oriented Programming
* Main principles (encapsulation, inheritance, polymorphism, abstraction)
* Inheritance vs Composition
* Other programming paradigms and comparison

**.NET Framework**

* What is .NET Framework, which parts it consists of
* What is CLR, its responsibilities
* What is IL code
* Managed vs Unmanaged code
* Do we have pointers in C#? What do you know about unsafe code?
* What is assembly, GAC? Can you load assembly dynamically?

**Basics of C#**

* Reference and Value types:
  + System.Object and System.ValueType
  + Heap and stack
  + Lifetime of reference and value type objects
  + What if you need to represent an int that can hold null? Is nullable type a reference type or value type?
  + Which datatype is more precise and why, float or decimal. What numeric data type you will use to store money operations and why?
  + What’s immutable type? Ask to give some examples of immutable types. Are structs immutable by default? Why should they be implemented as immutable?
  + What's the default implementation of Equals method for classes and structures? If you override Equals method what other one you have to override and why? What is the purpose of GetHashCode method?
* Inheritance from Value types
  + Can we inherit from structure?
  + Can we implement methods in structure?
  + Can we define abstract or virtual methods in structure?
* Boxing and unboxing
  + Describe each operation, compare performance
  + What will happen if we call interface method of structure
* Interfaces and abstract classes
  + What is interface, what can be defined there (methods, properties, events)?
  + Can you specify the accessibility modifier for methods inside the interface?
  + Compare interface and abstract class
  + Explicit Interface Method Implementation
  + Do we always need to implement all the methods of abstract class?
  + "Virtual" and "new" keywords
* Classes
  + What is the static class and why do we need them (examples)
  + What is constructor? Can object be created without usage of ctor?
  + What is the property (IL code generated behind the scenes)
  + How method call looks like in IL? How virtual method call looks like in IL?
  + Properties with parameters (indexers)
  + Calling virtual method inside base class constructor – why is it dangerous?
  + How we can prevent a class from being inherited?
  + What is the nested class?
  + What is the anonymous type?
  + How we can extend the functionality of sealed class (extension methods, patterns)
* Generics
  + What are, what problems they are designed to solve
  + Constraints ("where")
* Collections
  + Difference between array and collection
  + What collections did you use? Examples of usage scenarios
  + LINQ: name familiar operators (Select, Where etc), describe deferred execution of LINQ
  + Concurrent collections
  + How to make your own type enumerable (IEnumerable and do we really need it)
* Events/delegates
  + What is event, what is delegate? Difference
  + Do you know what Func and Action delegates are?
  + Can we call event outside the containing class?
  + What will happen if one of the methods subscribed to an event will throw exception? How to avoid this?
  + Events under the hood
  + Delegates under the hood: MultiCastDelegate, InvocationList, Delegate.Combine()
  + Lambdas, closures in lambdas
* Garbage Collection
  + Describe the process (phases, roots, generations)
  + Force garbage collection (GC.Collect)
  + Destructor vs Finalizer vs IDisposable
  + Finalization queue and FReachable queue
  + What types of resources are disposed by Finalize and Dispose methods?
  + When and who can call Finalize and Dispose
  + "using" keyword and Dispose()
  + What tools will you use to detect memory leaks in your application?
  + What's the difference between small object heap and large object heap?
* Exception handling
  + Why do we need exception handling, compare to other languages (op codes, statuses etc)
  + Try/catch/finally
  + Multiple catch blocks (interception from most to less specific exception)
  + "Throw" vs "throw ex"
  + "Using", "foreach" and try/finally blocks
  + Exception in "finally" block – what will happen
  + How to break execution of "finally" block
* Reflection
  + What kind of task can be solved by using reflection, basic use cases
  + What is metadata
  + What is attribute and how can we define our own attributes
  + Performance of reflection
  + CodeDom and Expressions vs Reflection
* Multithreading
  + Which classes can be used to run code asynchronously? (Thread, ThreadPool, Async delegates, TPL, timers, BackgroundWorker, etc)? Describe the difference
  + What's the difference between foreground and background threads? What happens if exception is thrown on background thread?
  + Cancellation of tasks (CancellationToken, CancellationTokenSource, IsCancellationRequested)
  + How we can declare a set of actions that need to be executed exactly after a specific task (t.ContinueWith, OnlyOnException, OnlyOnCompletion etc)
  + Async/await (state machine behind the scenes)
  + Parallel LINQ: Parallel.For, Parallel.Invoke
  + What happens when you await a task that throws exception? What happens if task throws exception, but you never wait for it to end?
* Synchronization
  + What synchronization primitives do you know (lock, Monitor, Interlocked, ReaderWriterLock, Events, Mutex, Semaphore etc)?
  + Deadlock
  + Lock on "this", "typeof(MyType)", string, value type?

**Database & SQL**

* Types of Join. Describe Right/Inner/Full Join
* What is index? Why it's not always good to have too many of them?
* What is trigger, view, stored procedure?
* Transaction, ACID (Atomicity, Consistency, Isolation, Durability), Isolation levels
* Let's assume that you have SQL query which performs badly, what would you do? What is query execution plan?
* ORM frameworks (describe one that was used or known, if any)
* Code First and Model First approaches in Entity Framework (if familiar). Experience and impression about ORM frameworks.
* NoSQL (MongoDB, Maria DB, graph databases etc)

**Code quality**

* Ask how the candidate ensured code quality in his latest/current project
* Unit vs Integration vs Acceptance tests
* Test Doubles: Mock vs Stub vs Dummy
* Test frameworks used (NUnit, xUnit, Moq, NSubstitute)
* What is a good test for you?
* TDD, best practices to improve code quality
* Code review and its role

**Design and Architecture**

* Patterns used or known (at least 5, the more the better). Ask to describe 1-2 patterns
* SOLID
* High cohesion and loose coupling
* MVC vs MVVM (if familiar)
* Dependency Injection, ways of injecting dependencies

**DevOps, version controlling, CI**

* Continuous integration vs Continuous delivery. Common instruments for CI (TeamCity, Jenkins etc)
* Describe delivery pipeline applied on current/last project
* Version Control Systems. Git vs SVN
* Merge vs Rebase
* Git Flow
* Containers (if familiar)
* PowerShell, scripting and Windows Management in general

**SDLC**

* Describe SDLC process on current/last project
* Functional vs non-functional requirements
* Software Development methodologies (waterfall, iterative vs incremental model, agile methodology)
* SCRUM vs Kanban
* Roles in SCRUM (PO, SM, Devs)
* Planning, retrospective, daily standups – why they are needed, describe
* How team decides which stories should be included to sprint?

**Legend:**

**Orange – Regular must know**

**Red – Senior must know**

## Detailed skill matrix (from 0 to 5)

|  |  |
| --- | --- |
| Self-presentation |  |
| English |  |
| OOP |  |
| .NET Framework |  |
| Multithreading |  |
| C# |  |
| Database & SQL |  |
| Code quality |  |
| Design and Architecture |  |
| DevOps, version controlling, CI |  |
| SDLC |  |
| Overall impression |  |