Answers

Ok I will provide answers by theirs numbers in order to make it clear ☺

1. I think current project is the one I can be proud of. Currently I working on web AddIn for MS office based on JS namely on React.js with .net core backend. The reason I’m so proud of cause I’ve convinced customer to move product from windows desktop version based on .net framework only to multiplatform version which is as I mentioned involves more technologies but in turn has poor office API. Despite it ultimately led to market to be extended and satisfied users on other platforms different from windows.
2. As far as I remember the log(x) is the function which always grows and has a derivative on all the set of values so the boundary conditions will be calculated on a specific range of values by first derivative

And the limit lim log x = -∞ for x -> 0 and x should always be x >0

1. I’m not sure I get the question right but if so the answer is 2 because we can check with let’s say x = -1 and x = 1 in a way they both preserve their sign which means there is an intersection with x axis so the function has a 0 value.
2. Basically, it’s specified by the process currently but in fact it’s no critical and up to 3-5 major bugs with workaround.
3. Interface defines an open api set of members or abstract members without implementation and it’s purely a specification or protocol for type it’s modeling. Delegate in turn it’s a class and has it’s own implementation and it’s nothing but the pointer to function. In common they define sort of a type for entity and function in respect. We need to use delegates at the points where we pass functions as arguments or callbacks which is the same or defining events. The interface in turn modeling further entity behavior like specifying level of abstraction instead of certain types.
4. It’s unusual project as for me I’m everything new geek so I would with pleasure try to apply my expertise to help project and join unique team ☺

Car Architecture

According to provided diagram by the customer it’s a composition association so Car class looks following

public class Car

{

public List<Wheel> Wheels { get; protected set; }

public Engine Engine { get; protected set; }

public Carburetor Carburetor { get; set; }

public Car()

{

Wheels = new List<Wheel>();

Engine = new Engine();

Carburator = new Carburator();

}

}

The issue here is that all entities have strong bind to Car class so the architecture is rigid to changes and very hard to control and test. It would be better to use aggregation association and sort of Dependency Injection pattern so our class could look like this

public class Car

{

public List<Wheel> Wheels { get; protected set; }

public Engine Engine { get; protected set; }

public Carburetor Carburetor { get; set; }

public Car(List<IWheel> wheels, ICarburator carburetor, IEngine engine)

{

Wheels = wheels;

Engine = engine;

Carburetor = carburetor;

}

}

So here Engine and Carburetor could be instantiated separately as a Singleton and wheels as required. In this case we keep working with abstraction instead and now it’s could be tested and easy to change and enhance.