**Ticketing Domain – Asynchronous ASP.NET Core APIs**

1. What are the benefits and drawbacks of async programming?

The main benefit is the execution speed due to the fact there’s no need to wait for tasks to complete. This way responsiveness and high performance may be achieved – program remains responsive during heavy operations held asynchronously in background (the main tread is not blocked by time-consuming tasks and still able to manage other tasks efficiently). Async programming introduces the ability to perform multiple tasks concurrently. can be executed concurrently. Another benefit is improved end user experience: in applications with UI treads (web, mobile) end users don’t face with freezes of UI.

At the same time, there are a couple of drawbacks: increased complexity (understanding of tasks chains, execution order, debugging), unusual cases (like deadlocks, etc.), difficult error handling (exceptions can occur during unexpected behavior), not efficient for small tasks (in those cases asynchrony/parallelism can lead to increased execution time in comparison with synchronous variants – there is a point in a chart “execution time to amount of data” where sync/async processing reach same execution time, starting from which async becomes faster than its sync variant with increasing the data amount).

It's also important to remember that not all operations benefit from being asynchronous – async can be considered in case of I/O-bound tasks (database queries, HTTP requests, File I/O operations, etc), CPU-bound tasks (e.g. massive heavy computations) are not suitable for async (due to they don’t involve waiting for external resources)

1. How to make ASP.NET controller action support async flow?

The return type of the method should be a Task or generic Task (e.g. *Task<IActionResult>*). The method should be marked with *async* keyword. Use *await* in the method’s body on the awaitable (long-running) calls.

1. How does async flow influences ASP.NET request executions (life cycle)?

Async in ASP.NET Core allows to handle multiple requests efficiently without blocking threads. As the framework has the middleware pipeline, the request is processed in the configured order, but async middleware can return the control back to the server while waiting for external resources (database, API, etc) – for example this way routing, authentication and logging are handled. This way scalability may be achieved (multiple requests handling).

1. List at least 5 tips on ASP.NET API performance best practices?

Use asynchronous database operations (e.g. with EF Core – follow EF Core best practices, e.g. no-tracking queries).

Use pagination to chunk large data collections.

Avoid sync read/write on HttpRequest/HttpResponse body (that can lead to sync over async because Kestrel does NOT support synchronous reads).

Avoid IHttpContextAccessor.HttpContext stored in a field.

Do not access HttpContext from multiple threads.

Use Caching.

1. Vertical vs Horizontal scalability. Where to use each?

Vertical means increasing power capabilities of existing machines to meet demand (e.g. upgrading CPU, memory, storage). This type of scalability is preferred in cases of predictable growth of demand, easy for quick prototyping (there is only one unit that doesn’t need to be coordinated).

Horizontal means adding nodes (machines) to the infrastructure to handle increased demands. This is an analogy of real spread of work between employees (same to nodes), so the load and traffic may be distributed effectively. But here it’s important to remember about correct managing the nodes and their communication.

1. Explain why the PUT method was suggested for the book action on the order

There are three ways (HTTP verbs) for requesting updates of the resources: PUT, POST and PATCH.

POST is commonly used to create new resources. Usually returns the updated state of the entity (e.g. with the increased amount of something).

PATCH allows to update an existing resource (but unlike PUT which replaces the entire resource, this action will modify only specific fields or attributes), that’s why PATCH is more complex due to handling specific fields updates. This way PUT is preferred to update all seat states in a cart, but PATCH is for individual seats states updates.

PUT is used to modify (update) an existing resource completely, it’s idempotent in comparison with POST (repeated POST requests will create several entities, when the results of several PUT actions will be the same).

**Ticketing Domain – Persistence Level (SQL, NoSQL) + DAL**

1. What are your steps to start designing database?

To start designing database, it’s necessary to

- determine the purpose of the database: the kind information to be stored, its consumers (people who will use that information) and the way they will interact with it

- find and organize the information required: gather all of the types of information you might want to record in the database, create a description of the data type, size, and format

- start designing with division the information into subjects (or entities), each one then becomes a table.

1. When can we say that our database is modeled correctly?

A modeled-correctly database divides the information into subject-based tables (it helps to reduce redundant data), provides with the necessary details to gather the information from various tables as required, aids in maintaining and assuring the precision and consistency of the stored data and accommodates the stored data processing and reporting needs.

1. What is a Data Access Layer (DAL), and how does it simplify database interactions?

A Data Access Layer (DAL) is a part of the software application (module, project or any else), which is responsible for interaction with the database. In a three-tier layered application together with Business Logic (BL) and Presentation layers, forms a well-supported modular system, each part of which has its own responsibilities. DAL provides the simplified and consistent methods for accessing, retrieving and manipulating the stored data.

The simplification of the interactions is about:

- encapsulating of details of operations, leaving only a single interface

- single interface provision allows the BL layer’s services to be independent on the type of database used, so any changes to the DB structure (adding a new table, changing a data type) don’t affect the business logic. Same to complete change of DB (e.g. from MySQL to PostgreSQL)

- increased security to validate queries for SQL injections.

1. You need to implement a new service for a customer. How would you select database (SQL or NoSQL)?

The choice depends on

- budget bounds

- the way data should be structured (NoSQL will be more flexible and will handle any structural changes of data or storing the unstructured one)

- the speed/consistency needs (SQL databases follow ACID principles and its data is always consistent, but NoSQL doesn’t follow with having a speed and simplified scalability)

- needs to handle complex queries (SQL databases have powerful querying capabilities, when NoSQL are not good at dealing with complex queries, but cope with quick processing of large amounts of high-velocity, variable data).

**Ticketing Domain – Introduction**

1. What diagram types do you have on your current project?

There is a component diagram and a state diagram on my current project.

1. What are the pros and cons of UML diagrams?   
   – learning of UML may be a long process (due to variety of standard details and diagram types)

– usually take much time for creation  
– take time for post-creation support (keeping them up to date)   
– it’s important to keep the diagrams simple – that influences on their readability and understanding

+ standardization

+ don’t require technical knowledge to be reviewed by customers, clear for clients, designers and developers

+ allow to decompose complex systems into smaller pieces

1. What is the difference between a structural diagram and a behavioral diagram in UML?

As their names say, structural diagram shows the structure of the system components and their relationship & interaction.

A behavioral diagram describes the way of processes held between the components, how they work over time, the messages and events between the components and the way they are handled.

1. Name the relationship types in a use case diagram

Association, Extend, Include, Use case or actor generalization

1. What is the sequence diagram?

A type of UML diagram that shows the way and the order of objects operate with one another over time. The objects involved in the sequence are represented as a vertical dotted “lifelines”. The interaction between those lifelines is shown with horizontal arrows in the same order (from up to down) the actions are performed