Fluent Api

What is Fluent API?

Fluent API (Fluent Application Programming Interface) is an approach to designing APIs that allows you to create code that reads like natural language. It provides a clear, flexible, and understandable way to define rules or configurations. Instead of using traditional annotations directly on model properties, you define validation rules or other configurations in a separate class using a fluent interface.

Benefits of Fluent API:

1. **Expressiveness: Fluent APIs make your code more expressive. By chaining methods together, you can create a sequence of actions that read like a sentence. For example, consider the following validation rule: "Check if the property is not null and its length falls within a specific range." With Fluent API, you can express this elegantly in code.**
2. **Readability: The method chaining approach in Fluent API makes it easy to understand the intent behind the code. Each method call adds a specific rule or configuration, and the sequence of calls provides a clear picture of what's happening.**
3. **Separation of Concerns: Fluent API allows you to separate the validation logic or other configurations from the model itself. This separation enhances maintainability and keeps your model classes clean and focused on their primary purpose.**

4. Main points that all students must know about fluent api what use and why?

Entity Configurations:

Fluent Api

1. **HasAlternateKey(): Defines an alternate key for the entity, providing a unique constraint other than the primary key.**

2. **HasIndex(): Specifies an index for the specified properties, optimizing query performance for those properties.**

3. **HasKey(): Sets the specified property or properties as the primary key for the entity.**

4. **HasMany(): Defines a one-to-many or many-to-many relationship, indicating that the entity contains a collection of related entities.**

5. **HasOne(): Indicates the navigation property representing the principal end of a relationship.**

6. **WithOne(): Specifies the navigation property on the dependent end of a relationship.**

7. **HasForeignKey<TDependent>():
Identifies the property in the dependent entity that serves as the foreign key for the relationship.

Fluent Api

8. **Ignore():** Instructs Entity Framework to ignore the specified class or property during mapping to a database table or column.

9. **ToTable():** Specifies the name of the table in the database to which the entity is mapped.

Property Configurations:

1. **HasColumnName():** Defines the name of the corresponding column in the database for the property.

2. **HasColumnType():** Specifies the data type of the corresponding column in the database.

3. **HasComputedColumnSql():** Maps the property to a computed column in a relational database.

4. **HasDefaultValue():** Sets a default value for the column mapped to the property in the database.

5. **HasDefaultValueSql():** Specifies a SQL expression for the default value of the column in the database.

Fluent Api

6. **HasField():** Indicates the backing field for the property.

7. **HasMaxLength():** Sets the maximum length of data stored in the property.

8. **IsConcurrencyToken():** Marks the property as an optimistic concurrency token.

9. **IsRequired():** Defines whether the property requires a valid value or can be null.

10. **IsRowVersion():** Configures the property for optimistic concurrency detection, typically used for row versioning.

11. **IsUnicode():** Specifies whether the string property supports Unicode characters.

12. **ValueGeneratedNever():** Indicates that the property cannot have a generated value when saving an entity.

13. **ValueGeneratedOnAdd():** Specifies that the property has a generated value when inserting a new entity.

14. **ValueGeneratedOnAddOrUpdate():** Specifies that the property has a generated value when saving a new or existing entity.

Fluent Api

15. **ValueGeneratedOnUpdate():** Specifies that the property has a generated value when updating an existing entity.

15. **IsUnique():** specify that the property values must be unique within the table. It doesn't have any relation to generating values when updating existing entities.