ADDITIONAL QUESTIONS

- Still the same topics as in "Self-Check Questions".
- Give you some examples on how questions may be asked differently or indirectly.

CHAPTER 1

Q1. Identify the most appropriate **MVC component** (model, view or controller) for each of the following tasks. Justify the reason.

Task	Description
A	Check if inputs are valid.
B	Select records from database.
C	Layout with HTML and CSS.
D	Display records in UI.
E	Handle and process form submission.
F	Select view template and return response.

Chapter 1 Slide 27

- A = **Model** → Model handle input validations to enforce business rules.
- B = **Model** → Model connect to database and handle database operations.
- C = **View** → View generate layout using HTML and CSS.
- D = **View** → View display records and generate HTML (UI) dynamically.
- E = **Controller** → Controller handle application logic and form submission.
- F = **Controller** → Controller select view, pass data to it, then return response.

CHAPTER 2

Q2. Would you use **tag helpers** or **HTML helpers** to develop UI? Justify your choice.

Chapter 2 Slide 39

If you choose tag **helpers**:

- Written as HTML elements with custom non-standard attributes.
- Codes are more readable as they resemble typical HTML syntax.
- Codes are clean, concise and easier to understand.
- Friendly to web designers who know little about programming.
- Efficient in performance as they are compiled as part of the view.

[OR] If you choose **HTML helpers**:

- Written as C# methods. Work like function calls with parameters.
- Codes are less readable as HTML and C# codes are mixed together.
- Codes are usually more verbose and slightly complicated.
- Natural to web developers who familiar with programming syntax.
- Introduce minor overhead as they are executed as C# codes.

CHAPTER 4

Q3. Would you use **database-first** or **code-first** approach for a project with existing database? Justify your selection.

Chapter 4 Slide 11

Database-first approach:

- Develop database first. Then generate entity classes automatically.
- Popular for database administrators who focus on database development.
- Suitable when working with existing database.
- Recommended for large applications that involve extensive data processing (manual database optimization possible).
- Suitable for shared and centralized database that is accessed directly by multiple applications or systems.

Q4. Would you use **database-first** or **code-first** approach for a project with existing entity classes? Justify your selection.

Chapter 4 Slide 11

Code-first approach:

- Develop entity classes first. Then generate database automatically.
- Popular for programmers who focus on application development.
- Suitable when the database has not been created.
- Recommended for small to medium applications that do not involve extensive data processing.
- Suitable for application specific database, or when the data is shared via Web API (indirect access).

CHAPTER 5

Q5. Is **GET** or **POST** request method most appropriate for each of the following operations? Justify the reason.

CHAPTER 5 Slide 44, 46

EXTRA NOTE:

- **GET** → Appends inputs to the URL. It is bookmarkable.
- **POST** → Does not append inputs to the URL. It is more secure.

a) Select and display product records.

GET:

- It does not alter server state or data.
- No side effect → read-only operation.
- GET appends inputs to the URL. It is bookmarkable.

b) Insert, update or delete new product records.

POST:

- It alters server state or data.
- Has side effect → database is modified.
- POST does not append inputs to the URL. It is more secure.

c) Perform searching, filtering, sorting or paging.

GET:

- It does not alter server state or data.
- No side effect → read-only operation.
- GET appends inputs to the URL. It is bookmarkable.

d) Submit email and password for login purpose.

POST:

- It alters server state or data.
- Has side effect → login session and cookie are created.
- POST does not append inputs to the URL. It is more secure.

CHAPTER 6

Q6. Would you implement **client-side** or **server-side input validations** for an input form? Justify your reasons.

CHAPTER 6 Slide 21-23

- Implement **BOTH** client-side and server-side input validations.
- Get the benefits from both sides.

Client-Side Input Validations:

- Program using JavaScript. Run at browser (locally).
- Provide immediate response, improve responsiveness and user experience.
- However, unable to perform complex validation logic.
- Insecure, as JavaScript can be turned off.

Server-Side Input Validations:

- Program using C#. Run at server (remotely).
- Access full programming resources. Able to perform complex validation logic.
- However, involve roundtrip, thus is less responsive.
- Secure, server-side input validations always run.

CHAPTER 7

Q7. You want to add a service that needs to be reused and shared across all HTTP requests. Which **service lifetime** would you use? Justify your answer.

Chapter 7 Slide 38

Singleton:

- Only one instance is created per application.
- Live until the application stops or resets.
- Object instance created is shared globally.

Q8. You want to add a service that needs to maintain consistent state within a single HTTP request. Which **service lifetime** would you use? Justify your answer.

Chapter 7 Slide 38

Scoped:

- A new instance is created per HTTP request.
- Live until the HTTP request-response ends.
- Object instance created is shared within a request.

Q9. You want to add a stateless and lightweight service that do not require shared state. Which **service lifetime** would you use? Justify your answer.

Chapter 7 Slide 38

Transient:

- A new instance is created every time it is injected.
- Live until the calling method exits.
- Not shared. New object instance created every time.