.NET Core & React.js Interview Questions with Detailed Explanations (Mid-Level)

🔹 .NET Core Questions (Mid-Level)

1. What is Middleware in .NET Core? How does it work?

Explanation: Middleware in .NET Core is a component that handles HTTP requests and responses. The request pipeline is built as a sequence of middleware components where each one can:

- Process the request
- Call the next middleware
- Short-circuit the pipeline

Example:

```
app.Use(async (context, next) => {
    // Logic before next middleware
    await next.Invoke();
    // Logic after next middleware
});
```

This allows powerful chaining for tasks like logging, authentication, etc.

---

2. Difference between AddScoped, AddSingleton, and AddTransient?

Explanation: These determine the lifetime of services in Dependency Injection:

- **Singleton:** One instance for the entire app lifetime
- **Scoped:** One instance per request
- **Transient:** New instance every time it's requested

Use Singleton for stateless/shared services, Scoped for DB contexts, and Transient for lightweight operations.

---

3. How do you handle global exceptions in ASP.NET Core?

Explanation: Use `UseExceptionHandler()` middleware or custom middleware. For API, it can return a standard JSON error response.

```
app.UseExceptionHandler(errorApp => {
  errorApp.Run(async context => {
    context.Response.StatusCode = 500;
    context.Response.ContentType = "application/json";
    await context.Response.WriteAsync("{\"error\":\"Something went wrong\"}");
  });
});
```

This keeps your API consistent and debug-friendly.

---

4. How do you implement role-based authorization?

Explanation: Define policies or roles in `Startup.cs` and decorate controllers/actions:

```
[Authorize(Roles = "Admin")]
```

Configure roles in the identity system and map them with claims.

---

5. How do EF Core migrations work?

Explanation:

- `Add-Migration` generates a migration script based on model changes
- `Update-Database` applies the changes to the DB

EF Core tracks changes to your model classes and compares them to the current DB state.

🔹 React.js Questions (Mid-Level)

1. What are React Hooks? Explain useState and useEffect.

Explanation: Hooks are functions that let you use state and lifecycle features in functional components.

- `useState`: Declare state variables

```
const [count, setCount] = useState(0);
```

- `useEffect`: Run side-effects (like API calls)

```
useEffect(() => {
    fetchData();
}, []); // empty array = run only once on mount
```

---

2. What are controlled and uncontrolled components?

Explanation:

- **Controlled:** Form elements tied to React state. Easier to validate and control.

```
<input value={name} onChange={e => setName(e.target.value)} />
```

- **Uncontrolled:** Accessed using `ref`, React does not control the value.

Use controlled for complex forms, uncontrolled for simple cases.

---

3. Explain React Context API vs Redux. When to use what?

Explanation:

- **Context API:** Best for lightweight, top-down global state (theme, auth)
- **Redux:** For complex, deeply nested state with many mutations and actions

Use Redux when:

- You need predictable state
- You're handling async operations (with middlewares like thunk/saga)

---

4. What is memoization in React?

Explanation: Memoization optimizes performance by caching values:

- `useMemo`: Cache expensive computed values
- `useCallback`: Cache functions
- `React.memo`: Prevents unnecessary re-renders

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

---

5. How does React Router work?

Explanation: React Router enables SPA navigation without reloading the page. Use components like:

```
<Routes>
  <Route path="/home" element={<Home />} />
</Routes>
```

Use `useParams` to extract route variables, `useNavigate` for programmatic navigation.

🔹 .NET Core + React Integration Questions

1. How do you call .NET Core APIs from React and handle CORS?

Explanation:

- Use `fetch()` or `axios` in React to call API endpoints
- Enable CORS in .NET Core:

```
services.AddCors(options => {
  options.AddPolicy("AllowAll", builder =>
    builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());
});
```

And add middleware: `app.UseCors("AllowAll")`

---

2. How do you handle JWT authentication across .NET Core and React?

Explanation:

- Backend: Issue JWT after login
- Frontend: Store token in `localStorage` or `sessionStorage`
- Send token via `Authorization` header on each request

Protect routes in React and decorate APIs with `[Authorize]`

---

3. How do you host React inside a .NET Core app?

Explanation:

- Build React app: `npm run build`
- Place contents of `build/` folder into `wwwroot`

- In `Startup.cs`, configure SPA:

```
app.UseSpa(spa => {
  spa.Options.SourcePath = "ClientApp";
});
```

This is useful for full-stack deployments.

Let me know if you want a quiz sheet or mock interview setup next. Let's go beast mode on this prep – no excuses.