**Popular Session Management Techniques:**

When managing sessions across browser tabs in an ASP.NET MVC application, especially with an eye towards scalability and cleanliness, you're primarily looking to move session state out of the traditional in-memory ASP.NET session and into a more robust, shared store.

1. Distributed Cache (Recommended for Scalability and Performance)

This is generally the best approach for high-performance, scalable, and clean session management.

**How it works:** Instead of storing session data in the web server's memory, you store it in a dedicated, external distributed cache system. ASP.NET's session provider then retrieves and stores this data from the cache.

Key Benefits:

**Scalability:** Easily scales horizontally. You can add more web servers without worrying about session affinity (sticky sessions) because all servers access the same central cache.

**Performance:** In-memory caches are extremely fast.

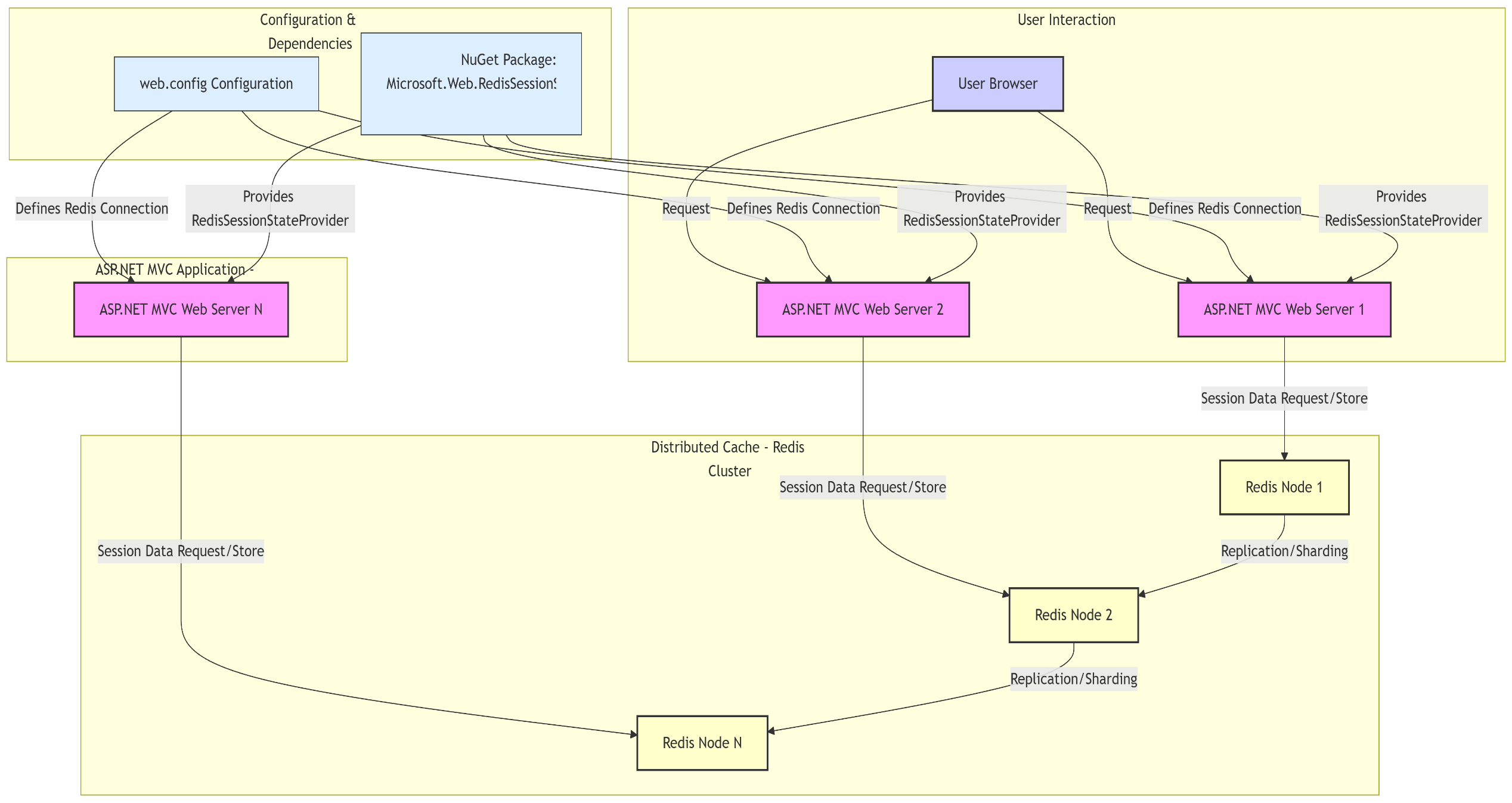
**Crosstab/Cross-Server:** Session data is accessible by any web server instance and thus implicitly works across browser tabs (if they share the same session ID).

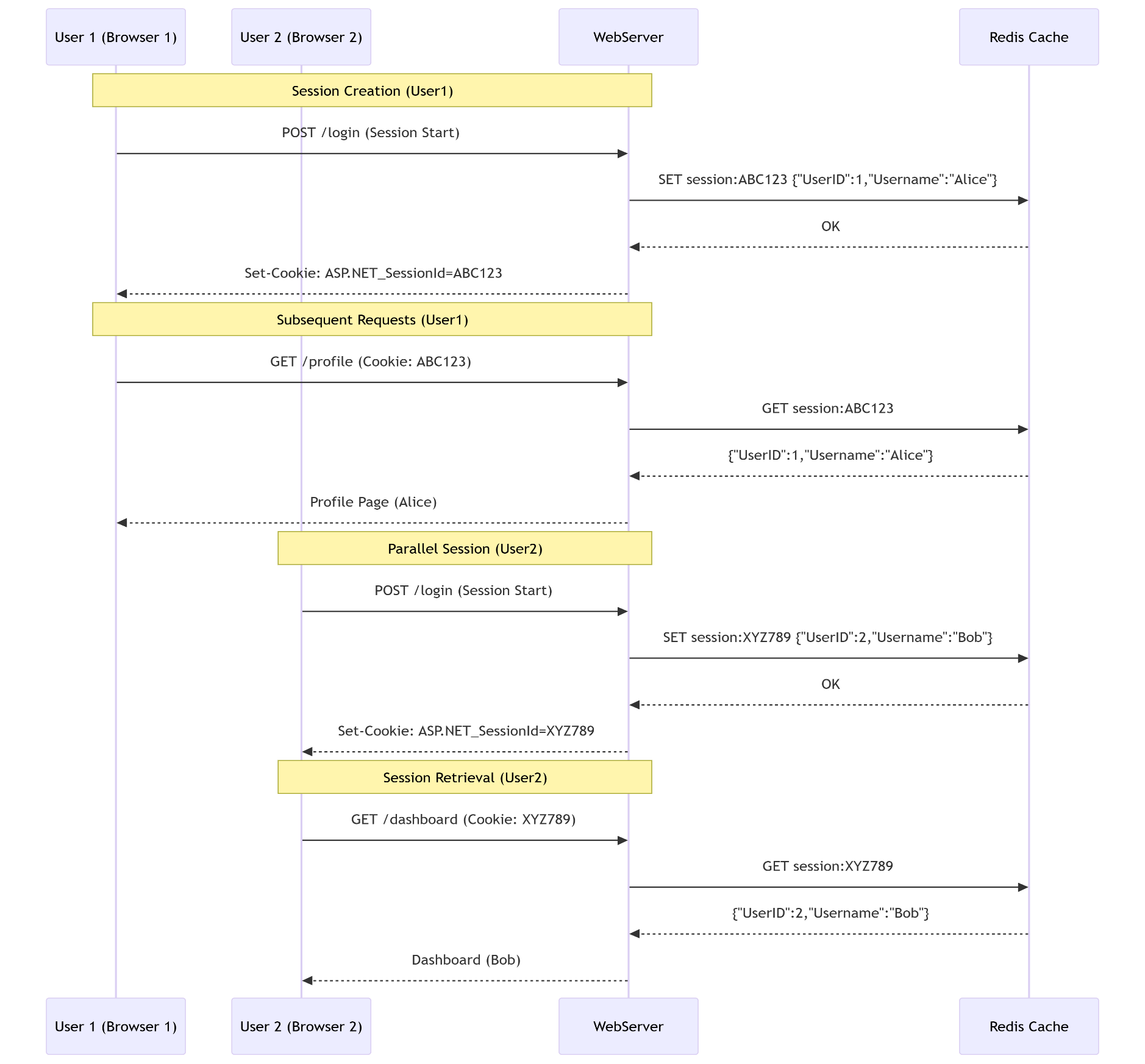
Popular Options:

Redis (Highly Recommended):

**Pros:** Extremely fast, versatile (can be used for other caching needs), supports various data structures, open source with commercial support options. ASP.NET has a dedicated RedisSessionStateProvider

**Cons:** Requires setting up and managing a Redis instance (can be done on-premises or via cloud services like Azure Cache for Redis).





Real-World Example:

1. **User A** logs in from Chrome → Session stored in Redis
2. **User B** logs in from Firefox → Separate session stored

**Unique Session IDs**, **Centralized Storage**, **Data Isolation**:

User 1 cannot access User 2's session data

Sessions stored as key-value pairs in Redis:

session: ABC123 → {"UserID":1,"Username":"Alice"}

session: XYZ789 → {"UserID":2,"Username":"Bob"}

* Both sessions persist even if: Web servers’ restart or Traffic shifts between load-balanced servers

**Implementation Steps:**

Step1: Install the appropriate NuGet package (e.g., Microsoft.Web.RedisSessionStateProvider).

Step2: Configure your web.config to use the Redis session state provider, specifying your Redis connection string.

<system.web>

<sessionState mode="Custom" customProvider="RedisSessionProvider">

<providers>

<add name="RedisSessionProvider" type="Microsoft.Web.Redis.RedisSessionStateProvider" host="yourredisinstance.redis.cache.windows.net" port="6380" ssl="true" throwOnError="true" password="your\_redis\_key" applicationName="YourMvcApp" />

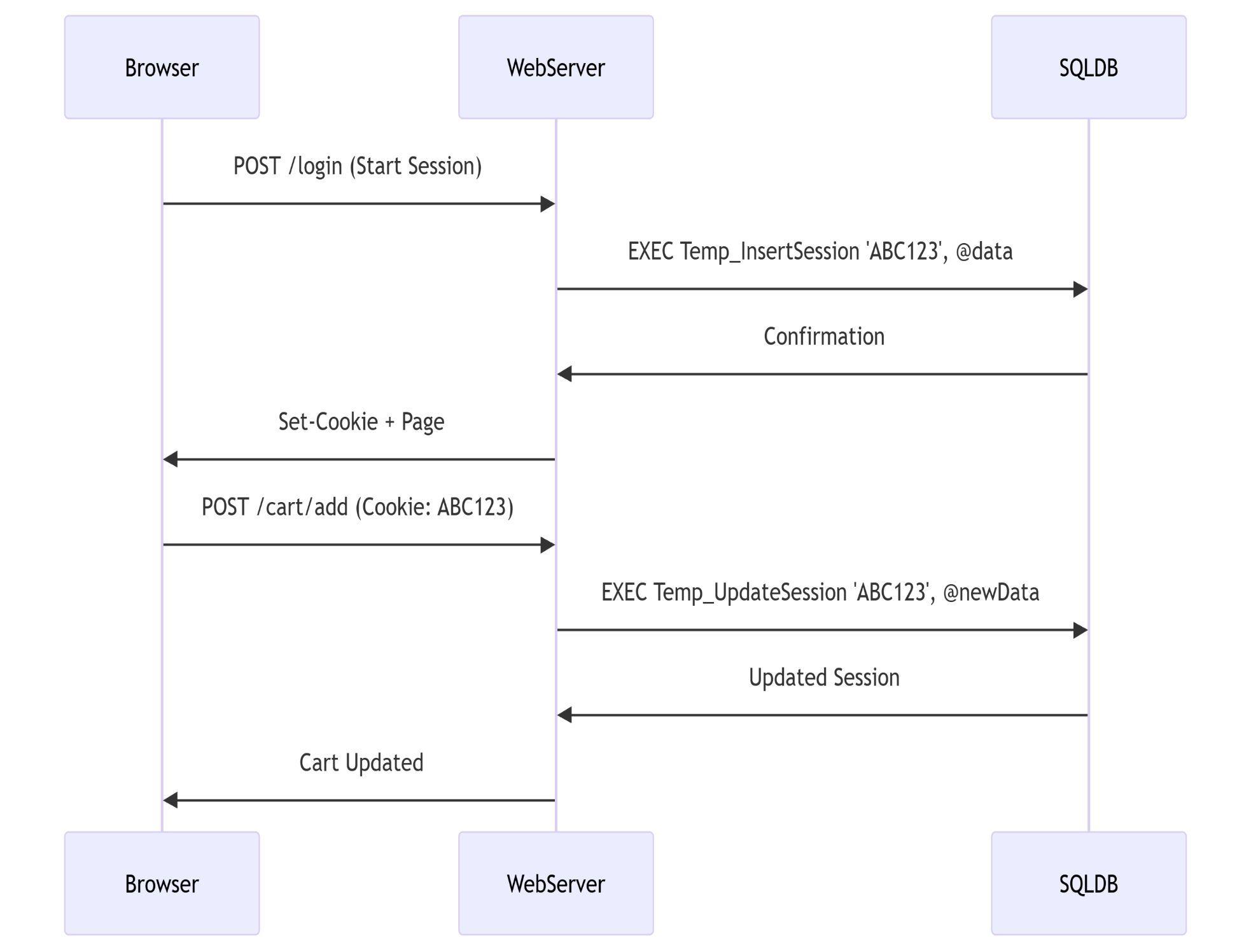
</providers>

</sessionState>

</system.web>

1. Database Session State Provider

**How it works:** Session data is serialized and stored in a relational database



* *User A* logs in on Chrome → Session stored in SQL DB
* Uses ASPStateSessions table (created by InstallSqlState.sql)
* **When to Use This**: When you need **transactional consistency** (e.g., banking apps)

Implementation Steps:

Step1: ASP.NET provides a built-in SqlSessionStateProvider.

Step2: You'll need to run a script (e.g., InstallSqlState.sql found in your .NET Framework installation directory) to create the necessary tables and stored procedures in your SQL Server database.

Step3: Configure your web.config:

<system.web>

<sessionState mode="SQLServer" sqlConnectionString="Data Source=your\_sql\_server;Initial Catalog=your\_database;Integrated Security=True" timeout="20" />

</system.web>

1. State Server (ASP.NET State Service)

**How it works:** An out-of-process service (the ASP.NET State Service) running on a separate machine stores session data.

Key Benefits:

**Scalability:** Better than InProc, as multiple web servers can connect to a single state server.

**Reliability:** Session state survives IIS restarts on the web servers.

Considerations:

**Single Point of Failure:** The State Server itself can be a single point of failure if not configured with redundancy.

**Performance:** Slower than InProc or distributed caches due to serialization/deserialization and network communication.

**Limited Scalability:** Not as horizontally scalable as distributed caches



Real-World Healthcare Example:

**Scenario**:

* *Doctor* logs in to view patient records → Session stored on State Server
* *Nurse* updates vitals → Accesses same State Server for session

**Key Components**:

1. **State Service**
   * Windows service (aspnet\_state.exe) running on dedicated machine
   * Listens on **port 42424** (default)
   * Stores serialized session data in memory

**When to Use This**:

* When you need **session persistence** during IIS restarts
* For **medium-traffic** applications

Implementation Steps:

Step1: Ensure the ASP.NET State Service is running on the designated server.

Step2: Configure your web.config:

<system.web>

<sessionState mode="StateServer" stateConnectionString="tcpip=your\_state\_server\_ip:42424" timeout="20" />

</system.web>

1. Token-Based Authentication (for API-driven scenarios/SPAs)

While not "session management" in the traditional sense, for modern applications (especially Single Page Applications or APIs), token-based authentication (like JWT - JSON Web Tokens) is a clean and highly scalable alternative that effectively replaces the need for server-side session state for user authentication and authorization.

**How it works:**

Step1: User authenticates, and the server issues a signed JWT containing user claims.

Step2: The client stores this token (e.g., in Local Storage or a HttpOnly cookie).

Step3: For subsequent requests, the client sends the token in the Authorization header.

Step4: The server validates the token's signature and expiration, extracts claims, and authorizes the user *without needing to look up session state*.

Key Benefits:

**Stateless:** The server doesn't need to maintain session state, making it highly scalable horizontally.

**Scalability:** Ideal for microservices architectures.

**Cross-Domain/CORS Friendly:** Tokens can be sent across different domains.

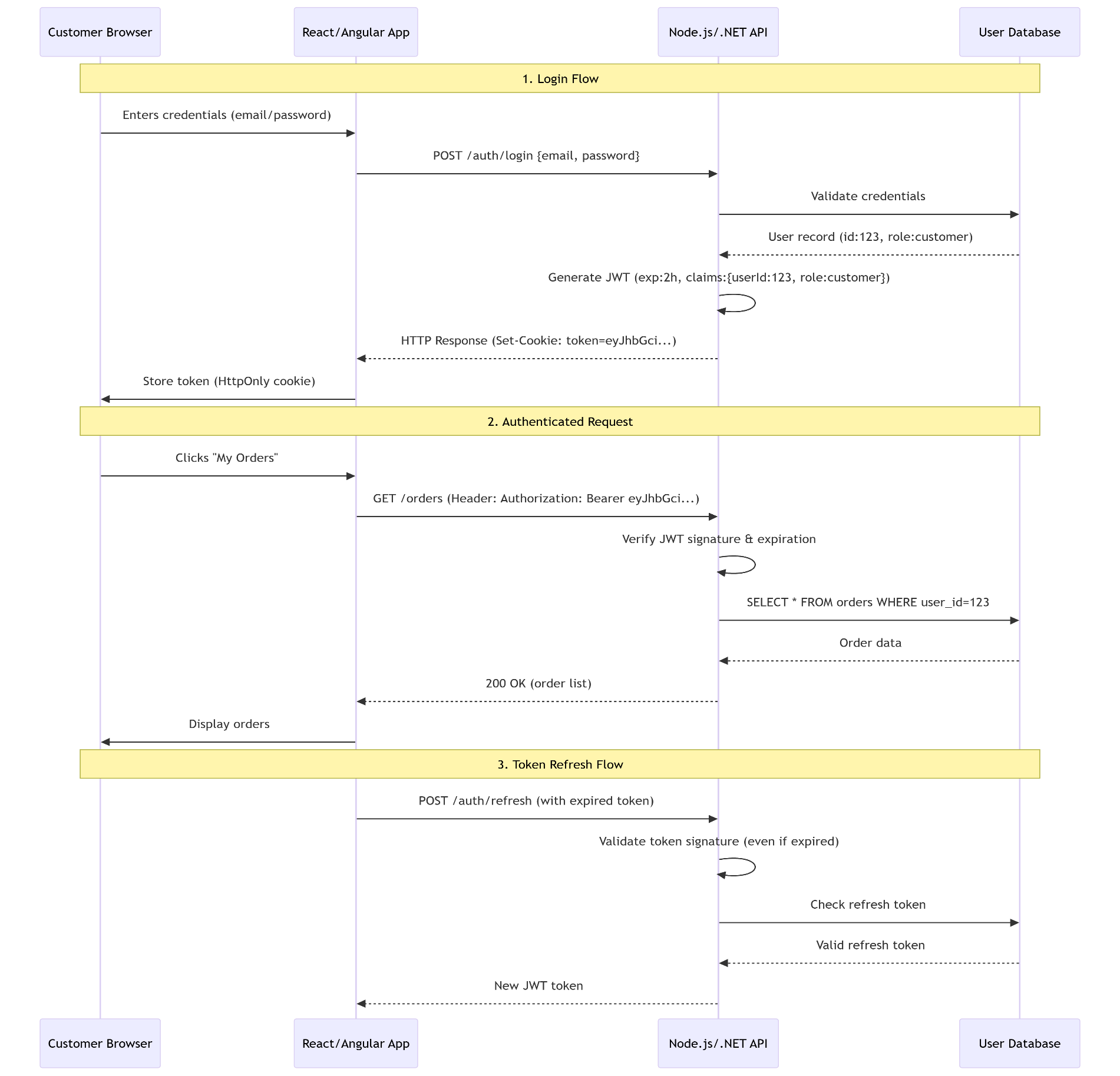
**Cleanliness:** Decouples authentication from server-side session management.

Considerations:

**Revocation:** Revoking tokens before expiration can be complex (requires a blacklist or short token lifetimes with refresh tokens).

**Token Size:** Can become large if too many claims are included.

**Security:** Needs proper handling (e.g., using HttpOnly cookies for tokens to prevent XSS, strong secret for signing).



Summary and Recommendation for ASP.NET MVC:

For a **better, scalable, and clean solution for session management across browser tabs in an ASP.NET MVC application**, the **Distributed Cache approach (especially Redis)** is generally the top recommendation.

**Scalability:** Achieved by moving session state out of individual web servers

**Cleanliness:** Architecture is cleaner as session state is decoupled.

**Performance:** Redis offers excellent read/write speeds.

**Reliability:** Redis can be configured for high availability.

If your application is evolving towards a more API-centric or SPA model, seriously consider a **Token-Based Authentication** system as it eliminates the need for server-side session state entirely for authentication purposes, offering the ultimate in scalability and statelessness. You might still use a distributed cache for other application-specific caching needs.

Impact on Your Current Application

1. **Session Attributes**: The [NHibernateMvcSessionContext] attributes will continue to work as they manage NHibernate sessions, not ASP.NET sessions
2. The change from inProc to SQLServer session state won't directly affect your NHibernate configuration

or the [NHibernateMvcSessionContext] attributes, as they manage a different aspect of session handling

1. **Session Usage**: Review all uses of Session, TempData, and PersistencyProperty to ensure they'll work with serialized sessions
2. **Dependencies**: Any objects stored in session must be serializable

**1. Prepare the Session State Database**

Step 1: Need to create the ASP.NET session state database

1. Open command prompt and navigate to - cd C:\Windows\Microsoft.NET\Framework\v4.0.30319
2. Use the command:

aspnet\_regsql.exe -S yourServer -E -ssadd -sstype p

* + -S: SQL Server instance
  + -E: Use Windows authentication
  + -ssadd: Add session state
  + -sstype p: Persistent (creates dedicated database)

Example : aspnet\_regsql.exe -S segotn15488.rds.volvo.com -E -ssadd -sstype p

For LocalDB: (localdb)\MSSQLLocalDB

Create a New DB : ASPState

ASPState DB is specific to session state storage

1. Run the command below

aspnet\_regsql -ssadd -sstype p -S (localdb)\MSSQLLocalDB -E

SELECT \* FROM sys.procedures WHERE name = 'TempGetVersion';

DECLARE @version int;

EXEC dbo.TempGetVersion @ver = @version OUTPUT;

SELECT @version AS Version;

Step 2: Modify Web.config

Change the <sessionState> element in your web.config:

<system.web>

<sessionState

mode="SQLServer"

sqlConnectionString="Data Source= (localdb)\MSSQLLocalDB;Initial Catalog=ASPState;Integrated Security=True;Connect Timeout=30"

cookieless="false"

timeout="180" />

</system.web>

Impact on the application:

1. **Session Attributes**: The [NHibernateMvcSessionContext] attributes will continue to work as they manage NHibernate sessions, not ASP.NET sessions
2. **Session Usage**: Review all uses of Session, TempData, and PersistencyProperty to ensure they'll work with serialized sessions
3. **Dependencies**: Any objects stored in session must be serializable

Implementation Considerations for the current code:

1. **Open SQL Server Management Studio (SSMS)**

* Launch **SSMS**.
* Connect to the SQL Server instance where you want the session state database.

1. Create a New Database for Session State

In **Object Explorer**, right-click **Databases** → **New Database…**

Name it something: **ASPState**

Click **OK** to create it

1. Create the Session State Tables and Stored Procedures

The GUI doesn’t have a wizard for session state, so you’ll need to run Microsoft’s official script in SSMS.

1. **Download the ASP.NET SQL scripts** from:

C:\Windows\Microsoft.NET\Framework\<version>\aspnet\_regsql.exe

cd C:\Windows\Microsoft.NET\Framework\v4.0.30319

1. Open **Command Prompt** (or PowerShell) and run:

aspnet\_regsql.exe

This launches the **ASP.NET SQL Server Setup Wizard**.

1. Use the Wizard
   1. **In the wizard, select:**

* **Configure SQL Server for application services**
  1. **Click Next.**
  2. **Enter:**
* **Server name (your SQL Server instance)**
* **Authentication method (Windows or SQL authentication)**
* **Database → Select your ASPState DB.**

** 4. Click Next → Finish.**