# Technical Documentation: Token Transmission Approaches

**Version:** 1.0  
**Date:** [Current Date]

## Table of Contents

1. [Goals](#X70af6ca7895521de59604591fd2d7f5677aa38b)
2. [Approach 1: Manual URL Redirection](#X318068d64b583c638c794a54c5fa6126072bcc3)
   * 2.1 [Workflow](#X53d8ea84eeb2ec062719f1118f2502c133b2e24)
   * 2.2 [Pros and Cons](#X52abd3251042535cce4314fdbb5bf79960e01c8)
3. [Approach 2: HTTP Client](#Xe4762c71b7fbdf35d7158059d0abde7ca08afd5)
   * 3.1 [Workflow](#Xaba0ecdfc8107350e7df99b8a718ed1991191be)
   * 3.2 [Pros and Cons](#Xcad5533f50d477f93bac15d8602b2104d4e568c)
   * 3.3 [Why Not Used](#X84edb9de3f9738f95931f4e6812286b7a368a84)
4. [Approach 3: WebSocket (Current Implementation)](#X889c21e91f09b3361b070de4bee115a6428f872)
   * 4.1 [Workflow](#X00c67760297e95fad779c2e62817fed1ff2a21a)
   * 4.2 [Pros and Cons](#X347583233816fca647a013c971f97f494d0f4f6)
   * 4.3 [Why It’s the Best Choice for Cross-Domain](#X49af1dfbbe99bf02827d01cf2928df6ea335408)
5. [Future Goals for Production](#X7c967fefbc6a14575013442df1d94ea70c89ce4)
6. [Implementing WebSocket in ASP.NET 4.7](#Xc503037373610c436eff6be12a61a08db082176)
   * 6.1 [Challenges](#X0b5ca4edc12fe4a428a4dc257cba15d3ae65364)
   * 6.2 [Possible Solutions](#X05937073130a93a4808995af4430016c1d4401a)
7. [Conclusion](#X19cad7d15381a9dfa000caf49ac5c69889e188d)

## 1. Goals

The primary objectives of this system are:

* **Secure Token Transmission:**  
  Securely transmit authentication tokens and user-specific data (DTO) from the API to the GUI.
* **Login Flow Confirmation:**  
  Ensure the login API waits for GUI confirmation before finalizing the login process.

## 2. Approach 1: Manual URL Redirection

### 2.1 Workflow

1. **API Generates Token:**  
   After a successful login, the API generates a JWT token.
2. **Redirection URL:**  
   The API returns a URL containing the token as a query parameter:

* https://gui.com?token=eyJhbGci...

1. **Manual Step:**  
   The user manually copies the URL and opens it in a browser.
2. **GUI Processes Token:**  
   The GUI extracts the token from the URL and stores it in cookies or sessionStorage.

### 2.2 Pros and Cons

**Pros:** - Simple to implement. - No dependency on real-time communication protocols.

**Cons:** - **Security Risk:** Tokens are exposed in URLs (visible in browser history, logs). - **Poor User Experience:** Requires manual steps, which can disrupt workflow. - **Limited Scalability:** Not suitable for automated or large-scale systems.

**Why Not Used:**  
Manual intervention is error-prone and insecure, making this approach unsuitable for production environments.

## 3. Approach 2: HTTP Client

### 3.1 Workflow

1. **API Generates Token:**  
   A JWT token is created upon successful login.
2. **Redirection URL:**  
   The API returns a redirection URL containing the token as a query parameter.
3. **HTTP Client Call:**  
   The GUI automatically follows the redirection using HttpClient:

* var response = await httpClient.GetAsync(redirectUrl);

1. **GUI Processes Token:**  
   The GUI extracts the token from the response and stores it in cookies or sessionStorage.

### 3.2 Pros and Cons

**Pros:** - Automates token handling. - No manual user involvement.

**Cons:** - **Limited Persistence:** Cookies are stored server-side, not in the browser. - **Token Loss:** Tokens are lost when the GUI is refreshed. - **Error Handling Issues:** Always returns HTTP 200, making errors harder to detect. - **Cross-Domain Issues:** Requires complex CORS configurations.

### 3.3 Why Not Used

* Token persistence issues.
* Inability to handle real-time updates effectively.

## 4. Approach 3: WebSocket (Current Implementation)

### 4.1 Workflow

1. **WebSocket Connection:**  
   The GUI establishes a persistent connection to ws://localhost:8181.
2. **Token Transmission:**  
   The API sends the token via WebSocket after a successful login.  
   Example message:

* TOKEN:eyJhbGci...

1. **GUI Processing:**
   * Stores the token in cookies or sessionStorage.
   * Fetches DTO data and sends confirmation via WebSocket:
   * DTO\_CONFIRMED:token:json

### 4.2 Pros and Cons

**Pros:** - **Real-Time Updates:** Immediate token transmission without polling. - **Improved Security:** Tokens aren’t exposed in URLs. - **Cross-Domain Support:** Works across domains with proper CORS configuration. - **Scalable:** Efficient handling of multiple clients.

**Cons:** - **Complex Setup:** Requires WebSocket server management. - **Firewall Restrictions:** WebSocket ports (e.g., 8181) might be blocked. - **Resource Usage:** Persistent connections consume more server resources.

### 4.3 Why It’s the Best Choice for Cross-Domain

* **Cross-Domain Flexibility:** WebSocket connections bypass traditional CORS limitations.
* **Real-Time Communication:** Ideal for dynamic applications needing live updates.
* **Enhanced Security:** Supports encrypted communication via WSS (WebSocket Secure).

## 5. Future Goals for Production

* **WebSocket Scaling:**  
  Use **Azure SignalR** or **Redis** for scalable WebSocket connections.
* **Security Enhancements:**  
  Implement AES token encryption and exclusively use **HTTPS/WSS** protocols.
* **Load Balancing:**  
  Utilize sticky sessions to maintain stable WebSocket connections.
* **Monitoring:**  
  Track WebSocket metrics such as message rates, connection lifetimes, and failure rates.
* **Fallback Mechanism:**  
  Implement long-polling as a backup for clients with restricted WebSocket access.
* **Token Expiry Management:**  
  Add support for refresh tokens to extend session durations securely.

## 6. Implementing WebSocket in ASP.NET 4.7

### 6.1 Challenges

* **No Native Support:**  
  ASP.NET 4.7 lacks built-in WebSocket support compared to ASP.NET Core.
* **Complex Configuration:**  
  Requires manual setup for handling WebSocket connections.
* **Performance Constraints:**  
  Less efficient than the WebSocket features available in ASP.NET Core.

### 6.2 Possible Solutions

* **Use SignalR for ASP.NET 4.7:**  
  SignalR offers WebSocket-like functionality with built-in fallback mechanisms.
* **Example Code:**
* public class MyHub : Hub  
  {  
   public void SendToken(string token)  
   {  
   Clients.All.receiveToken(token);  
   }  
  }
* **Third-Party Libraries:**  
  Libraries like **Fleck** or **WebSocketSharp** can provide WebSocket functionality.
* **Upgrade to .NET Core:**  
  Migrating to ASP.NET Core offers better WebSocket support and performance.

## 7. Conclusion

The current **WebSocket-based** implementation offers a secure, real-time, and scalable solution for token transmission. While this approach increases complexity, it effectively overcomes the limitations of manual redirection and HTTP client methods.

### Next Steps:

* Conduct load testing for WebSocket connections.
* Implement token refresh mechanisms for long-lived sessions.
* Set up real-time monitoring for connection performance.

**Prepared By:** [Your Name]  
**Reviewed By:** [Team Member Name]  
**Approved By:** [Stakeholder Name]