Varsity Sports System: JWT Authentication Prototype

Project Repository: https://github.com/Jaskirat-007/vss-auth-prototype

Date: November 25, 2025
Team Member: Jaskirat Singh
Project: Varsity Sports System Capstone - Backend Authentication & Authorization

---

Executive Summary

This document outlines the JWT (JSON Web Token) authentication prototype built for the Varsity Sports System capstone project. The prototype implements a secure, role-based access control system that manages user authentication, subscription verification, and livestream content access. The complete working code is available on GitHub for team testing and collaboration.

What We Built

1. Core Authentication Service (AuthService)

We created a centralized authentication service that handles JWT token generation with embedded user information. The system does NOT require database lookups for every request—instead, it encodes everything needed into the token itself.

Key Features:

- Generates secure JWT tokens with user ID, role, subscription status, and permissions
- Calculates user permissions dynamically based on role and subscription plan
- Supports three user types: Admin, Premium Subscriber, and Basic Subscriber
- Token expiration set to 15 minutes with automatic refresh capability

How It Works:
When a user logs in, the AuthService creates a token containing:

- User ID and email
- Role (Admin, Premium, Basic)
- Subscription plan
- Permissions list (stream:create, stream:manage, stream:view:paid, stream:delete)
- Expiration timestamp

This token is then used for all subsequent API requests. No database query needed each time—the token itself proves the user's identity and permissions.

2. Security Middleware

We implemented three custom middleware attributes to protect API endpoints based on user credentials:

RequireAuthAttribute

- Blocks unauthenticated requests (returns 401 Unauthorized)
- Verifies that incoming JWT tokens are valid and not expired
- Applied to: All protected API endpoints

RequireAdminAttribute

- Blocks requests from non-admin users (returns 403 Forbidden)
- Ensures only administrators can access sensitive operations
- Applied to: `/api/admin/*` endpoints

RequireSubscriptionAttribute

- Blocks users without active subscriptions
- Allows admins and premium subscribers to access paid content
- Returns 403 Forbidden if user is not subscribed
- Applied to: `/api/streams/[slug]` endpoints

3. Stream Access Controller

The StreamController enforces subscription requirements for viewing livestreams:

- GET /api/streams/{slug} → Returns stream details if user is subscribed or admin
- POST /api/admin/streams → Creates new stream (admin only)
- DELETE /api/admin/streams/{id} → Deletes stream (admin only)

Acceptance Criteria Met:

- ✅ Stream details indicate locked/unlocked status based on subscription
- ✅ Admin users bypass all subscription restrictions
- ✅ Free subscribers can only view free content
- ✅ Premium subscribers can view all content

4. Clerk Integration

We integrated Clerk for server-side user verification. The ClerkService validates frontend authentication tokens against Clerk's API, ensuring user identity is verified before issuing our internal JWT.

Current Status: Clerk API integration configured and ready to connect once credentials are available.

Complete Project Structure

```
VSSAuthPrototype/
├── Models/
│   ├── User.cs # User data model
│   ├── Stream.cs # Livestream data model
│   └── DTOs/ # Data Transfer Objects
│   ├── JwtTokenResponse.cs
│   ├── UserClaimsDTO.cs
│   └── StreamDTO.cs
├── Services/
│   ├── IAuthService.cs # Authentication interface
│   ├── AuthService.cs # JWT generation & permission logic
│   ├── IClerkService.cs # Clerk integration interface
│   └── ClerkService.cs # Clerk API communication
├── Controllers/
│   ├── AuthController.cs # Login & token generation endpoints
│   └── StreamController.cs # Stream access control endpoints
├── Middleware/
│   ├── RequireAuthAttribute.cs # JWT validation
│   ├── RequireAdminAttribute.cs # Admin-only routes
│   └── RequireSubscriptionAttribute.cs # Subscription verification
├── Repositories/ # Data access layer (Entity Framework ready)
├── Data/ # Database context configuration
├── appsettings.json # Configuration & JWT settings
├── Program.cs # Application startup & service registration
└── VSSAuthPrototype.csproj # Project file
```

How to Test Locally

Prerequisites

- .NET 8.0 or later
- Git
- VS Code or Visual Studio

Step 1: Clone the Repository

git clone https://github.com/Jaskirat-007/vss-auth-prototype.git
cd vss-auth-prototype

Step 2: Restore Dependencies

dotnet restore

Step 3: Run the Application

dotnet run

The API will start at https://localhost:5150 (or similar).

Step 4: Test via Swagger UI

Open your browser to: https://localhost:5150/swagger/index.html

Sample Test Requests:

1. Generate a JWT Token (Mock Login)

- Endpoint: POST /api/auth/mock-login
- Request Body:
  {
  "email": "admin@varsity.com",
  "role": "Admin",
  "subscriptionPlan": "premium"
  }
- Response: Returns JWT token valid for 15 minutes

2. View Stream (Requires Subscription)

- Endpoint: GET /api/streams/game-final-2025
- Header: Authorization: Bearer {YOUR_JWT_TOKEN}
- Response: Stream details with locked/unlocked status

3. Admin-Only Operations

- Endpoint: POST /api/admin/streams
- Header: Authorization: Bearer {ADMIN_JWT_TOKEN}
- Request Body:
  {
  "slug": "championship-2025",
  "title": "Championship Game",
  "isLocked": true
  }
- Response: Creates new stream (admin only)

What Still Needs to Be Done

Immediate (Ready to Implement)

- Database Integration: Connect to Azure SQL Database with existing schema
- Real user & stream storage in database (currently using mock data)
- Token refresh endpoint for extended sessions

Waiting For

- **Database Connection String** - From Carson
- **Clerk Production Credentials** - To enable real user verification
- **Dacast API Keys** - For livestream encoding and playback

Future Enhancements

- Rate limiting on authentication endpoints
- Audit logging for all permission checks
- Multi-factor authentication (MFA) support
- Subscription status real-time verification

Technology Stack

Component	Technology	Purpose
Framework	.NET 8.0	Backend runtime environment
API Type	RESTful Web API	HTTP endpoints for clients
Authentication	JWT Tokens	Stateless session management
Authorization	RBAC Middleware	Role-based access control
Third-Party Auth	Clerk	User identity verification
Database	Azure SQL (ready to connect)	User & stream data
ORM	Entity Framework Core	Data access layer

Key Architectural Decisions

JWT Tokens Over Sessions

- **Stateless:** No server-side session storage needed
- **Scalable:** Works across multiple servers without shared state
- **Standards-Based:** Industry standard for modern APIs
- **Efficient:** Reduces database queries per request

Role-Based Access Control (RBAC)

- **Simple:** Three tiers (Admin, Premium, Basic) match business model
- **Secure:** Explicitly defines what each role can do
- **Flexible:** Easy to add new roles or permissions as business grows

Middleware Attributes for Protection

- **Reusable:** Apply same protection to multiple endpoints
- **Clear Intent:** Security requirements visible in controller code
- **Consistent:** Uniform error responses (401, 403) across all routes

Clerk Integration for Frontend Verification

- **Third-Party Reliability:** Delegates authentication to industry leader
- **Security:** Frontend tokens verified server-side before granting access
- **Scalability:** No need to build custom authentication from scratch

---

Team Collaboration & Next Steps

For Team Members Cloning the Repo

Clone the repository

git clone https://github.com/Jaskirat-007/vss-auth-prototype.git

Navigate to project

cd vss-auth-prototype

Restore NuGet packages

dotnet restore

Run the application

dotnet run

Open Swagger UI in browser

Navigate to https://localhost:5150/swagger

Integration with Main VSS Project

Once testing is complete, the authentication logic will be merged into the main Varsity2024/VSS repository for production deployment on Azure.

Feedback & Contributions

- Test the endpoints and provide feedback on Discord
- Report any issues via GitHub Issues
- Submit improvements via Pull Requests

Repository & Access

GitHub Repository: https://github.com/Jaskirat-007/vss-auth-prototype

Visibility: Private (team access only)

Branches:

- `main` - Production-ready prototype code

How to Access:

1. Clone via HTTPS (if you have credentials)
2. Share feedback on Discord
3. Ask for contributor access if making changes

References & Resources

- JWT (JSON Web Tokens) Specification
- ASP.NET Core Authentication & Authorization
- Clerk Documentation & API
- Role-Based Access Control Patterns
- Entity Framework Core Documentation

Questions or Issues? Contact the team on Discord or create an issue on GitHub.

Last Updated: November 25, 2025
Status: Ready for Team Testing ✅