MSSE672 – Component Based Software

Student: John Michael Kreski

Instructor: Mohammad Abu Matar

Assignment: Week 6 – Authentication

Date: 08/10/2025

File Name: HWExecution.doc

## Summary

This assignment extended the GeometryApp by adding a socket-based authentication server, implemented as a new AuthController and AuthService component within the existing Spring Boot framework. While Spring Security was not used, the design emulates production-ready authentication using HTTP sessions to track logged-in users across requests.

A new AuthService interface was introduced with an in-memory implementation (InMemAuthServiceImpl) to validate username-password credentials. This service stores session attributes upon login and provides methods to authenticate, verify session state, and logout. The AuthController exposes /auth/authenticate and /auth/logout endpoints to manage authentication workflows.

The QuadController was updated to enforce authentication across all its endpoints by checking for a valid session before allowing access. Unauthorized access attempts now return a structured 401 Unauthorized response with an explanatory message.

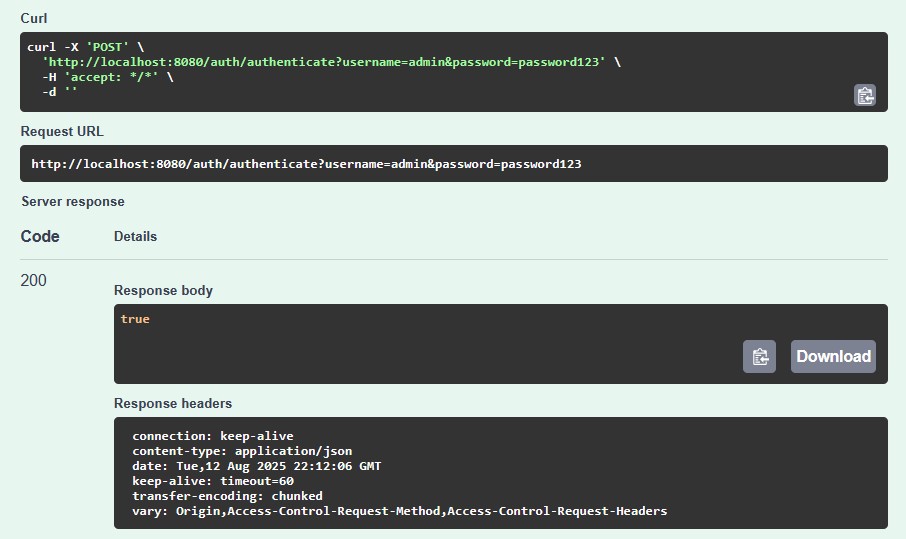
To facilitate testing, new JUnit integration test classes were created to validate the full authentication flow, including session tracking and endpoint-level access control. Additional logging was added to key authentication methods to support traceability during testing and debugging.

This work models a secure application foundation and lays the groundwork for future enhancements, including token-based authentication, security filters, or Spring Security integration.

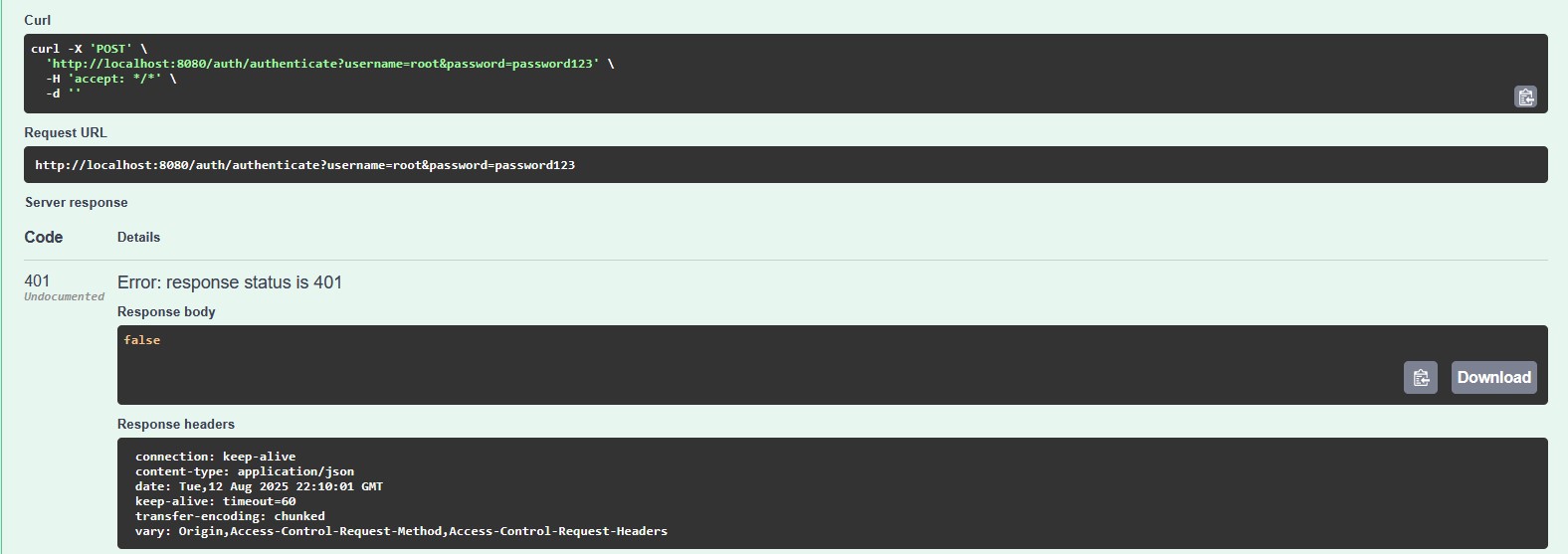
## Application Bootup

## Swagger UI – Authentication Flow

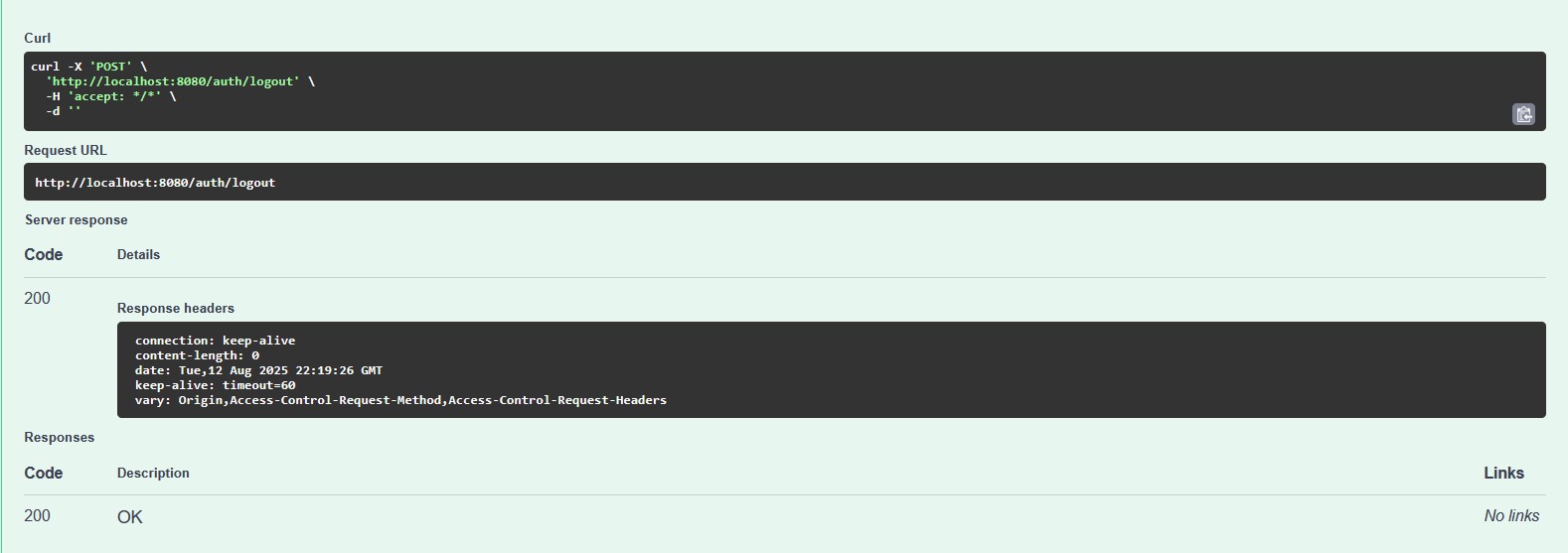
## Screenshot 1

*Screenshot of successful /auth/authenticate call using valid credentials (200 OK).*

## Screenshot 2

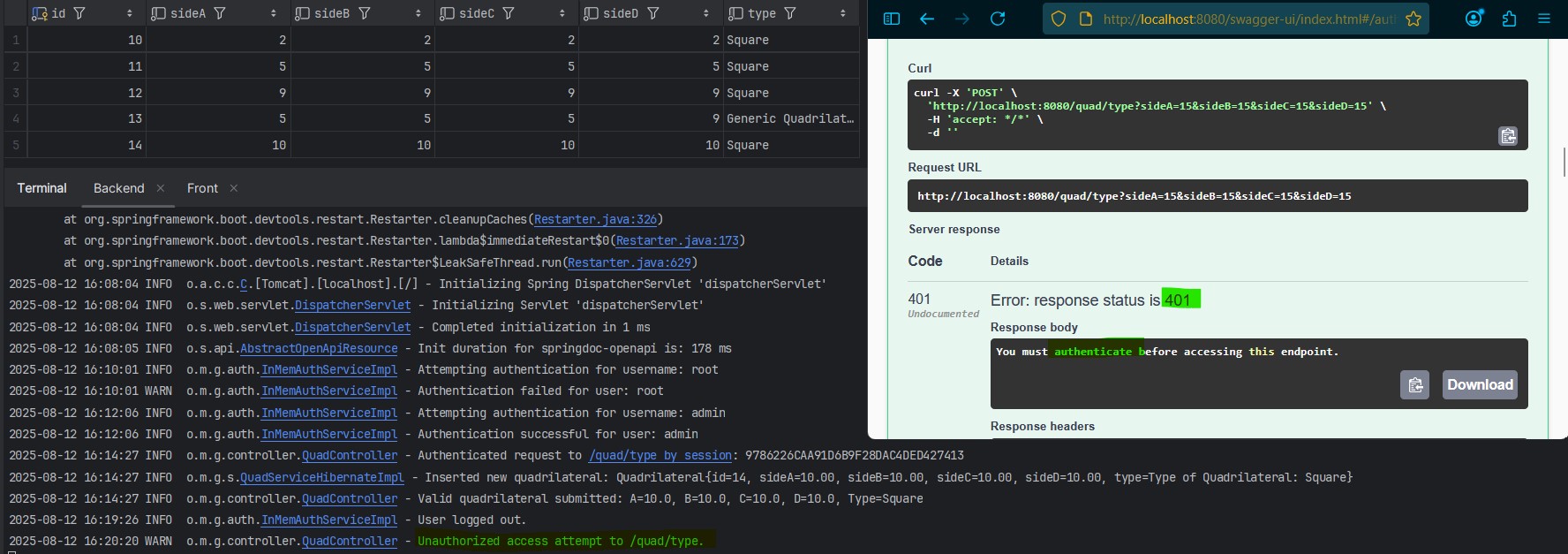
*Screenshot of a failed /auth/authenticate call using invalid credentials (401 Unauthorized).*

## Screenshot 3

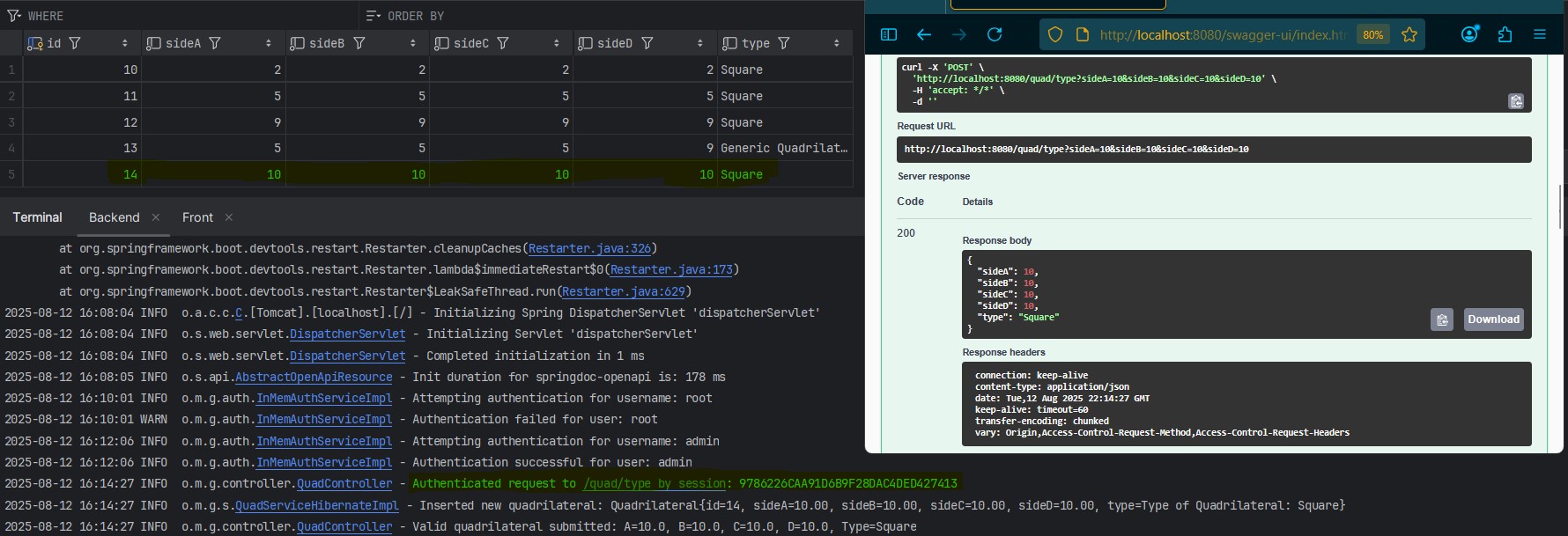
*Screenshot of a successful /auth/logout call.*

## Swagger UI – QuadController Authentication Enforcement

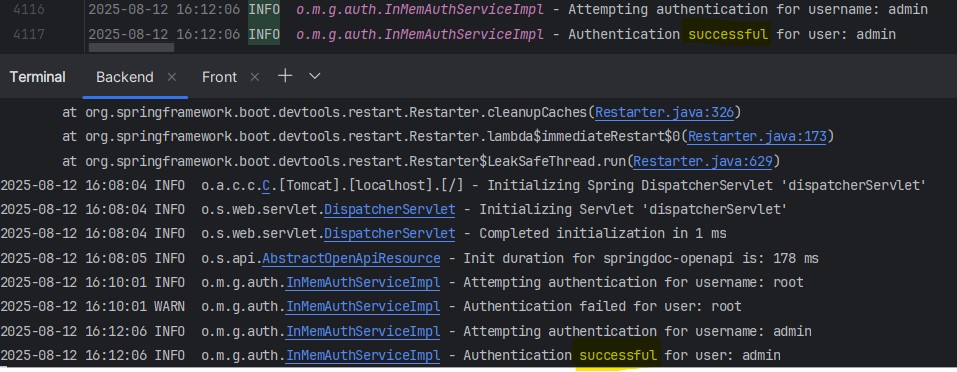
## Screenshot 1

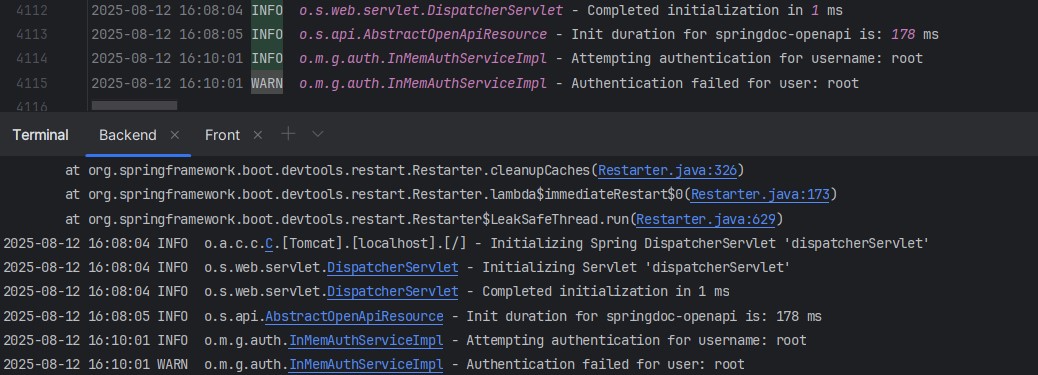
*Screenshot of /quad/type POST endpoint returning a 401 Unauthorized before login.*

## Screenshot 2

*Screenshot of the same POST endpoint returning 200 OK after authenticating successfully.*

## Console/App Logs

****Screenshot 1***Log showing successful authentication: Authenticated user: admin*

****Screenshot 2***Log showing unsuccessful authentication.*

****Screenshot 3***Log showing rejected quad access due to missing authentication: Unauthorized access attempt.*

## Refactored Files & Added Files

* AuthService.java
  + Interface defining authentication methods including login, logout, and session-based checks.
* InMemAuthServiceImpl.java
  + In-memory implementation of AuthService. Handles authentication logic using hardcoded credentials and stores session attributes to track user login state.
* AuthController.java
  + REST controller exposing /auth/authenticate and /auth/logout endpoints. Manages authentication flow via HTTP session.
* QuadController.java
  + Modified to enforce session-based authentication. All existing endpoints now check session validity before proceeding and return 401 Unauthorized if the user is not authenticated.
* AuthControllerTest.java
  + JUnit test class that validates successful and failed authentication attempts using MockMvc and session state assertions.
* InMemAuthServiceImplTest.java
  + Unit tests verifying the authentication logic, logout handling, and session-based checks within the service layer.
* AuthQuadControllerTest.java
  + Integration tests validating that authenticated users can access protected QuadController endpoints, and unauthenticated users are denied access with appropriate status codes.

## Known Limitations & Future Enhancements

* Session Scope: Current authentication is stored in the HTTP session, which is valid only for the current browser session or test context. This is sufficient for demonstration but lacks persistence across restarts or multi-client access.
* Hardcoded Credentials: The InMemAuthServiceImpl uses hardcoded username/password pairs (e.g., "admin"/"password123") for simplicity. Future enhancements could include secure password hashing, user storage in a database, or integration with Spring Security.
* No Role-Based Access Control (RBAC): All authenticated users have the same access rights. Adding role-based restrictions (e.g., admin vs. guest) could provide finer-grained endpoint access in the future.
* No Frontend Enforcement: While the backend now enforces authentication on all /quad endpoints, the existing Angular frontend does not yet redirect users or restrict access based on session status. This could be addressed with token/session tracking on the client side.
* Limited Socket Abstraction: While the assignment was framed around sockets, this implementation uses standard HTTP over Spring Boot. A future implementation may involve a lower-level NIO-based server for deeper socket programming practice.

## Notes

* The pom.xml build descriptor exists in the root directory of the project and supports full reproducibility, dependency management, and testing. It was not previously included in the submission since it was not explicitly required.
* A README.md is present at the project root level with build and usage instructions but was not submitted separately per earlier expectations.
* This assignment is delivered as a .zip containing .java source files and a Docs folder per course requirements.
* Each week's JUnit test documentation covers only the scope of the associated activity. The full src/test directory (including prior and current test classes) is included in every weekly .zip submission to provide a complete test history.
* Screenshots can be found in:
  + Docs/Week6/assets/