MSSE672 – Component Based Software

Student: John Michael Kreski

Instructor: Mohammad Abu Matar

Assignment: Week 8 – Spring’s IoC/DI

Date: 08/24/2025

File Name: HWExecution.doc

## Summary

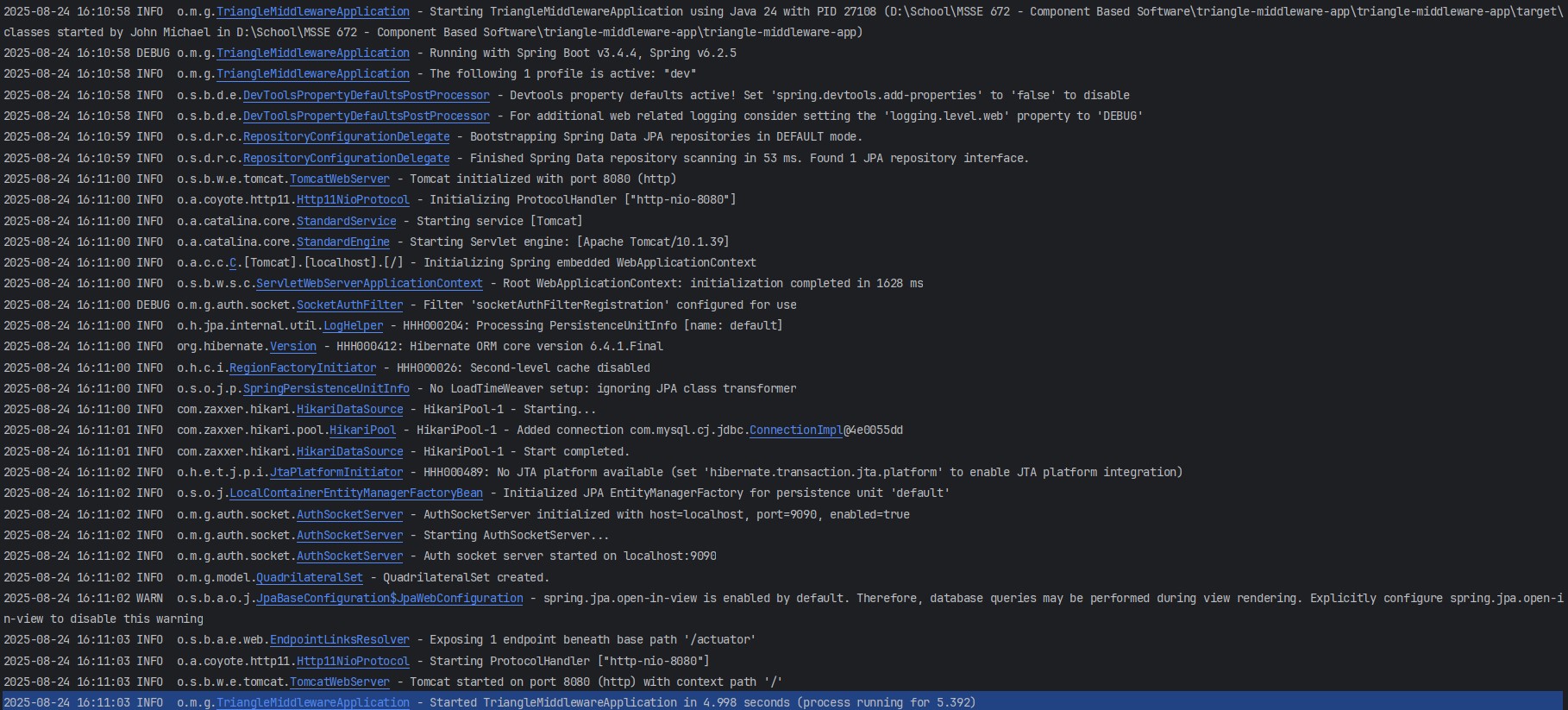
This week’s work focused on demonstrating the use of Spring’s Inversion of Control and Dependency Injection throughout the application, in alignment with Week 8’s programming objectives. Rather than refactor the educational ServiceFactory created for Week 5, I focused on strengthening the core architecture using Spring Boot’s native dependency injection model, as confirmed with the instructor.

**Key Enhancements:**

* **Spring-Managed Components**: All controllers, services, and filters are wired using Spring annotations such as @Component, @Service, @Configuration, and @Autowired, allowing Spring’s IoC container to manage lifecycle and dependencies automatically.
* **Constructor-Based Injection**: Where possible, constructor injection was used instead of field injection to promote testability, immutability, and better clarity of required dependencies.
* **Filter Configuration**: The SocketAuthFilter was registered as a Spring bean via AuthFilterConfig using FilterRegistrationBean with highest precedence. This filter securely intercepts requests to protected endpoints and validates authentication tokens via a socket call.
* **Configuration Properties**: The AuthSocketProperties class is bound to external configuration via @EnableConfigurationProperties and used in multiple components, demonstrating Spring’s configuration binding capabilities.
* **OpenAPI Security Integration**: The OpenAPI/Swagger configuration (OpenApiConfig) now includes a SecurityScheme that allows users to test secured endpoints by providing the X-Auth-Token in the Swagger UI.
* **Simplified Auth Flow**: The raw socket authentication system was extended to support Swagger sessions, while internal request filters abstract token validation logic across all secured endpoints.
* **Removed Legacy Logic**: The unused session-based authentication flow from Week 6 was fully removed. The ServiceFactory remains as an educational artifact but is not part of the runtime.

# Build Compilation

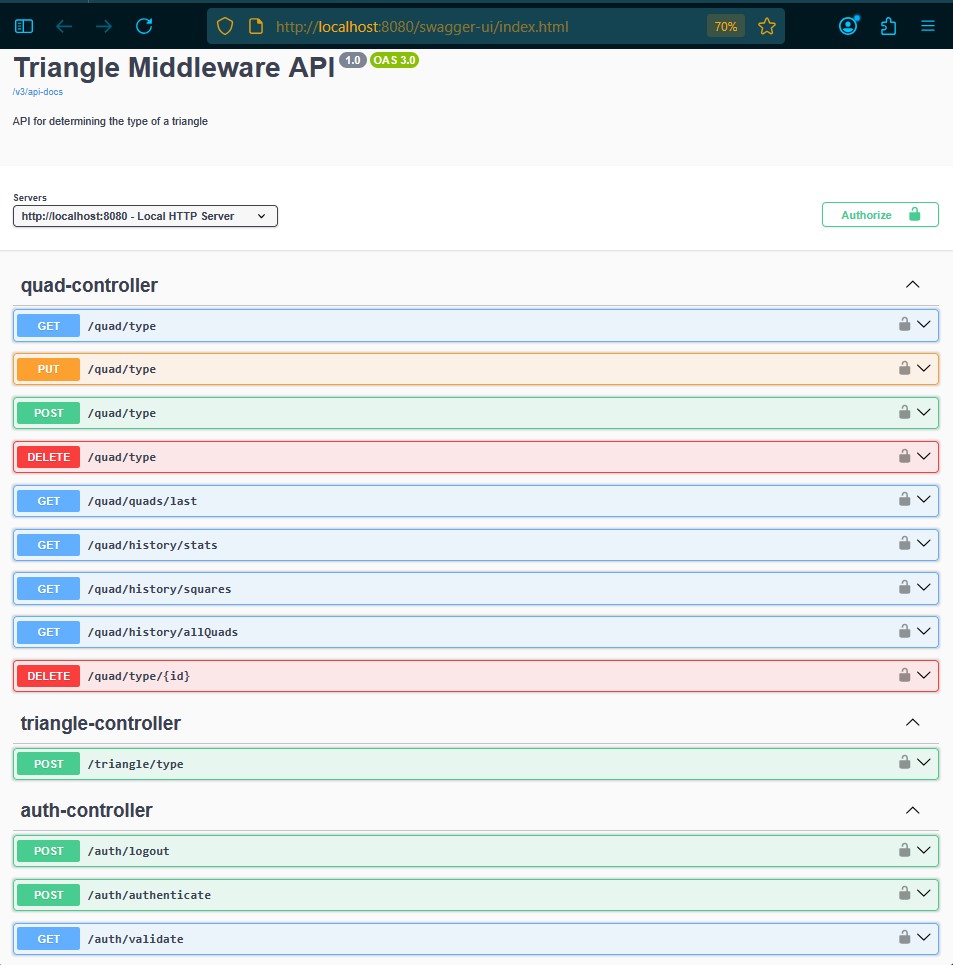
## Screenshot 1

*Successful Maven build output with no compilation errors.*

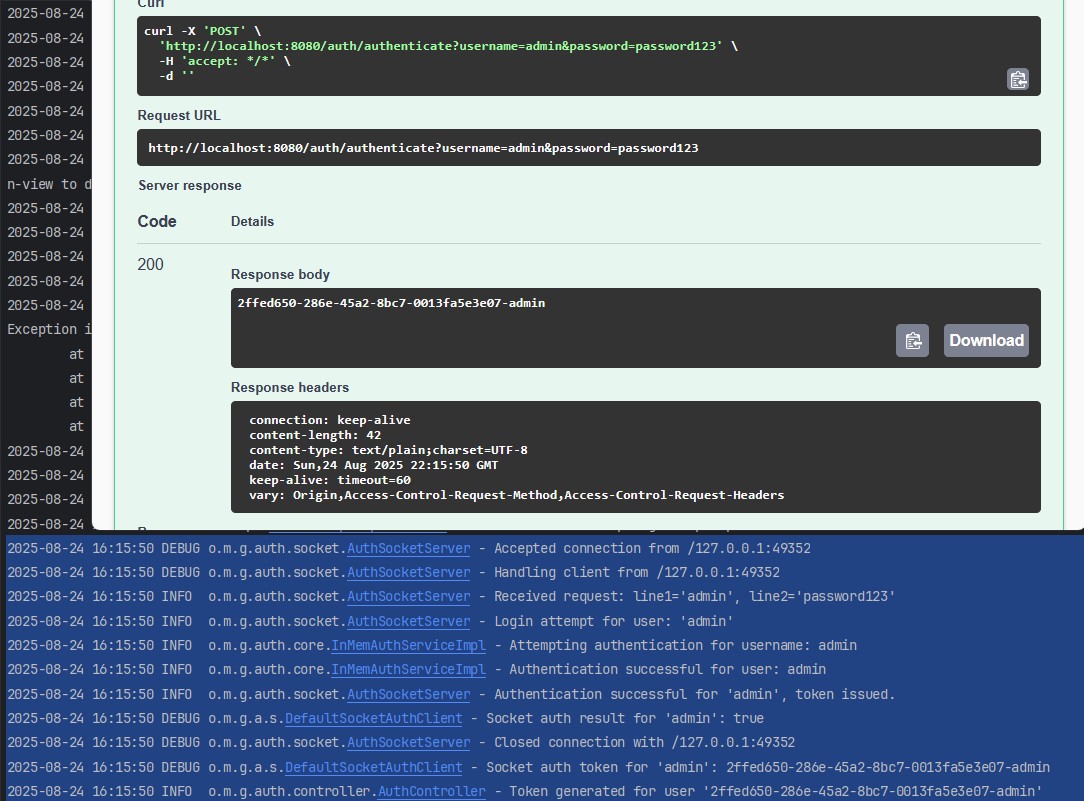
# Swagger UI

## Screenshot 1

*Swagger UI home screen showing available endpoints and custom API info.*

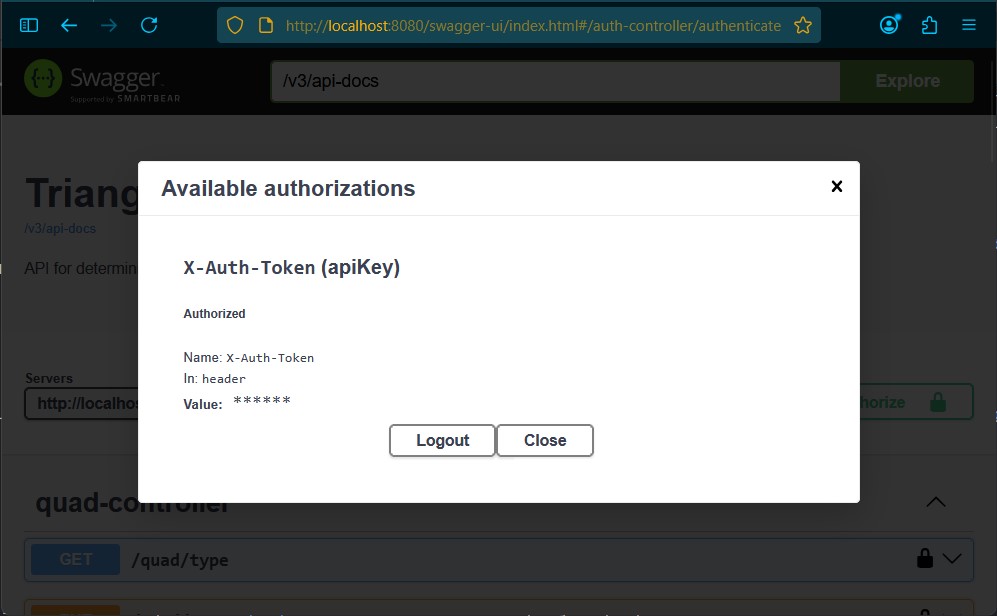
**

## Screenshot 2

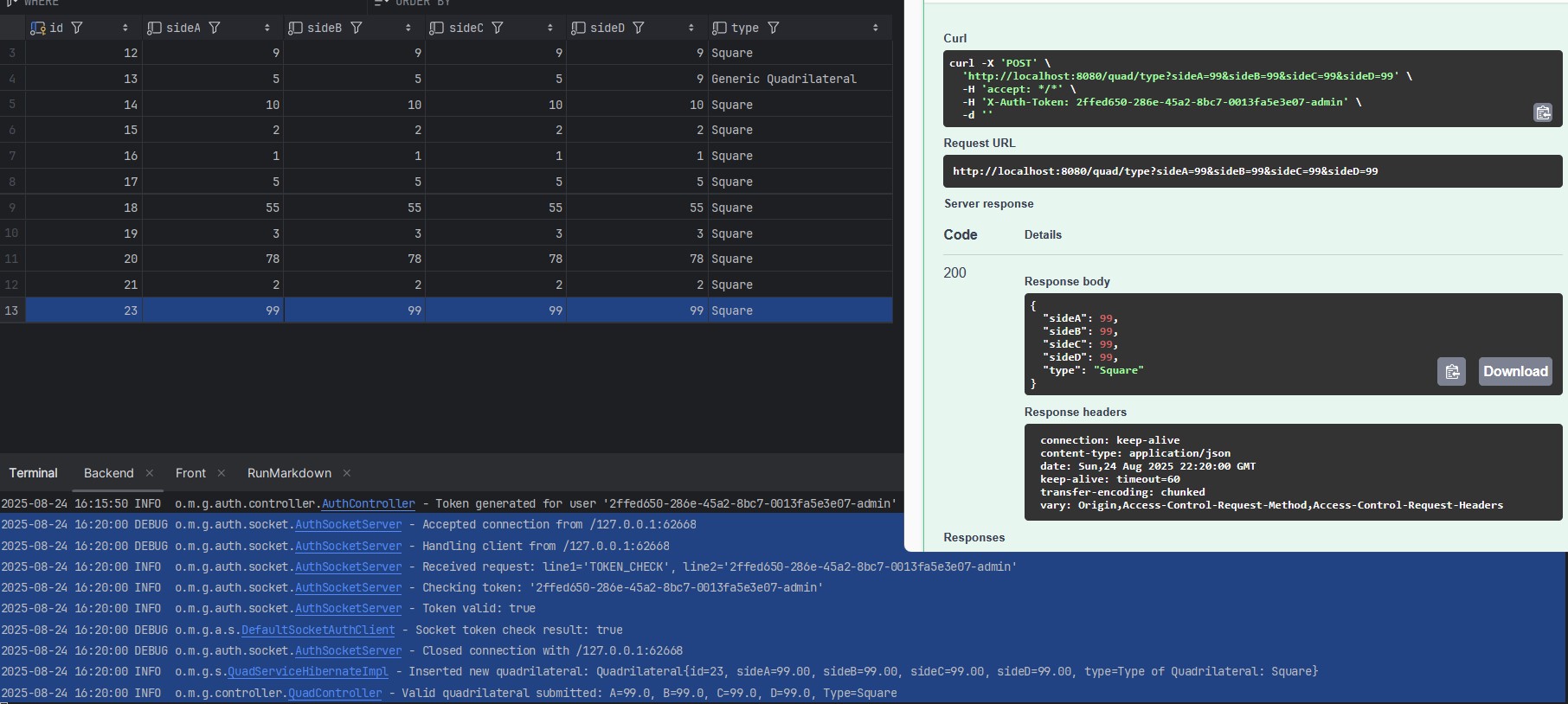
*Successful POST request to /auth/authenticate with returned UUID token.*

## Screenshot 3

*Swagger OpenAPI Config UI to allow custom token into controllers.*

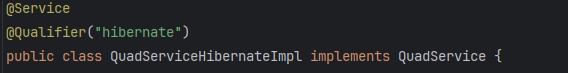
**

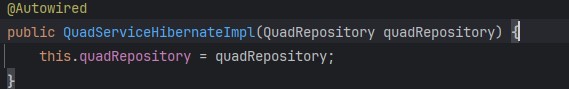
## Screenshot 3

*POST request to /quad/type with X-Auth-Token header set and valid response. Swagger UI showing X-Auth-Token configured as a security scheme in headers.*

# Spring Boot Injection & Config

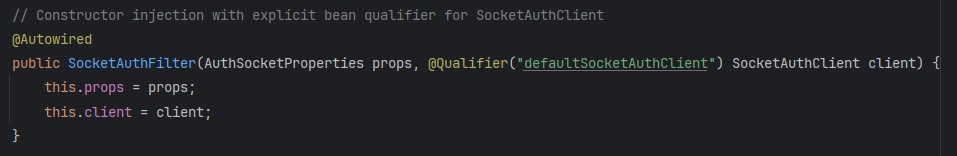
## Screenshot 1

*QuadServiceHibernateImpl with @Service annotation and constructor-based @Autowired injection.*

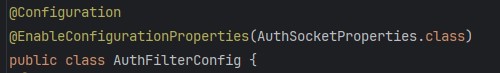
**

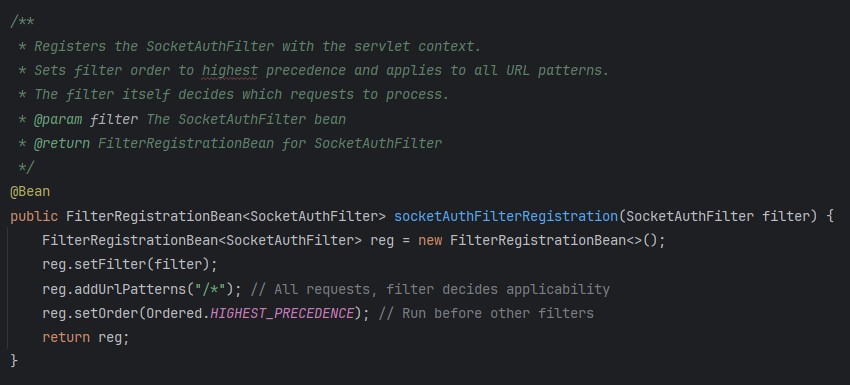
## Screenshot 2

*SocketAuthFilter class annotated with @Component and constructor injection.*

**

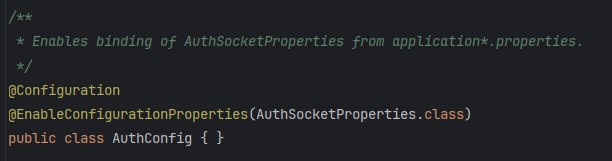
## Screenshot 3

*AuthFilterConfig class showing @Configuration and FilterRegistrationBean setup.*

**

## Screenshot 4

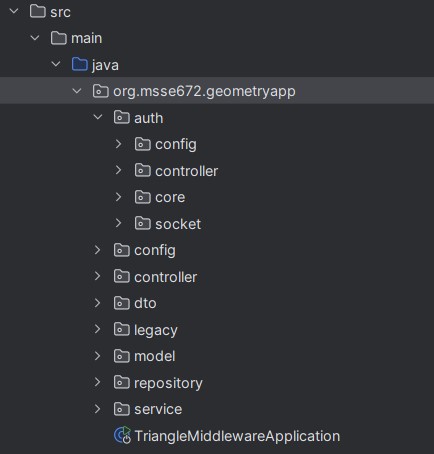
*AuthSocketProperties class bound via @ConfigurationProperties and injected into AuthConfig.*

****

# Project Structure

## Screenshot 1

*Expanded IntelliJ Project Explorer showing src/main/java/org/msse672/geometryapp and subpackages (auth, controller, config, legacy, etc.).*

**

## Refactored Files / Classes

* OpenApiConfig.java
  + Adds global header-based security (X-Auth-Token) to all Swagger endpoints using Springdoc.
  + Lets developers test secured endpoints via Swagger UI by entering their token once.
  + Adds server metadata and redirects root (/) to Swagger UI.
* README.md
  + Rewritten to reflect Spring Boot's use of Dependency Injection across the app.
  + Clarifies architecture, features, and testing flow.
  + Links to the new docs.md file and describes each week's academic objective.
* src/main/java/org/msse672/geometryapp/...
  + Restructured the source code to follow modular package conventions for clarity and scalability.
  + Organized by feature and concern, not just layer:
    - auth/ → all authentication logic (controller, service, filter, socket client/server).
    - Further divided into auth.socket and auth.config.
    - model/ → domain objects such as Quadrilateral, QuadResponse.
    - service/ → core business logic with multiple implementations (InMem, Jdbc, Hibernate).
    - controller/ → REST controllers (QuadController).
    - config/ → Spring configuration classes (OpenAPI config, filter config, application props).
    - legacy/ → standalone ServiceFactory and dummy classes used solely for educational or assignment-specific reasons (e.g., XML parsing in Week 5).
  + Created Docs/ directory in the root to hold Markdown-based documentation (README.md, docs.md) for transparency and future maintainability.
  + Added Javadoc-style comments to key packages and files to assist with future onboarding or review.

## Notes

* Spring IoC/DI is used throughout the application with minimal manual instantiation.
* Swagger now allows testing authenticated requests via token input.
* All authentication logic is socket-based and fully decoupled from web context.
* The ServiceFactory created in Week 5 remains outside the runtime by design.
* This submission includes all source files and a Docs folder per course requirements. Screenshots and logs of the threaded socket authentication in action are located under:
  + Docs/Week8/assets/

## Unit Testing

* No new unit tests were created for Week 8.
  + This week's focus was on organizing the codebase and validating the application of Spring’s Dependency Injection and IoC principles across existing components. Since no functional logic was changed and no new services or controllers were added, existing tests remain valid and sufficient.

<https://github.com/JohnMKreski/msse672-geometryapp-backend>

This submission demonstrates mastery of Spring’s DI principles across multiple layers of the application and aligns with the course's learning objectives. Supporting documentation (README.md and Docs/docs.md) has also been updated to reflect this architecture.