# BattleZone Server

Adam Ingersoll, Dillon Ladd, Willow Leverone, Ryan Plaza, and Nicholas Schoenfeld

## INTRODUCTION

- BattleZone is one of two projects implemented by groups of students in UNH's CS619 Object Oriented Development and Design class, taught by Prof. Matthew Plumlee
- It is an online multi-player tank shooting game originally written a decade ago
- The project teaches Object Oriented Design patterns and strengthens teamwork-oriented development skills using tools like Git and GitLab
- The new and improved version seeks to modernize the existing codebase with new frameworks and design patterns
- The server and SQL database have been containerized using Docker for ease of deployment

## REQUIREMENTS

Functional:

- Account creation and login; login returns a unique number for authenticating future requests
  - Server serves game state (grid of terrain, tanks, bullets, etc.) to clients over gRPC
- Users can control their tank: move, turn, and shoot enemies
- Support up to twenty concurrent containers

Non-Functional

- Easy to spin up for testing via Docker (server + database)
- Well-documented starter code with comprehensive tests
- Stable and responsive; invalid requests return error codes without crashing or affecting other users

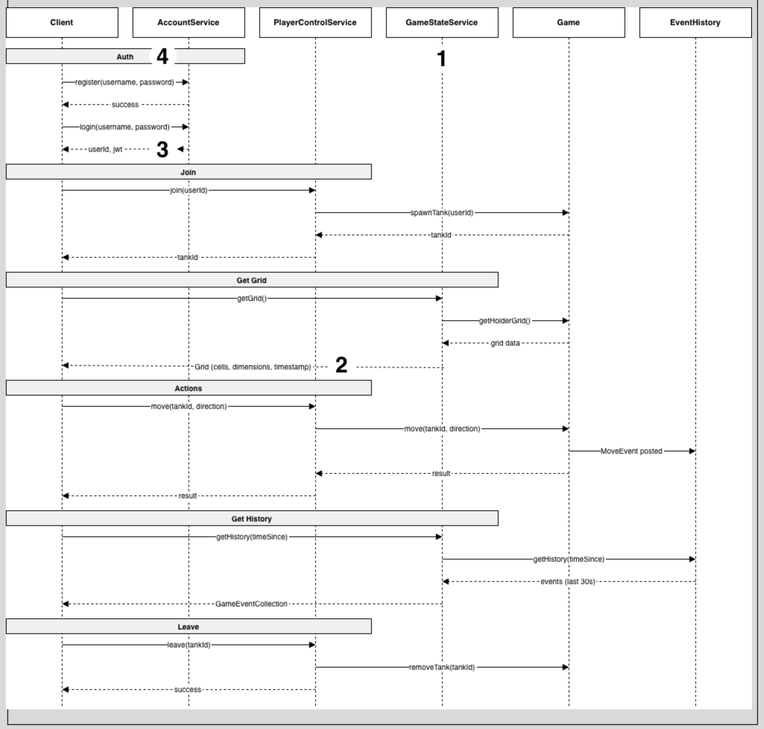
## CREDITS

Team: Adam Ingersoll, Dillon Ladd, Willow Leverone, Ryan Plaza, Nicholas Schoenfeld

Sponsor: Prof. Matthew Plumlee

Capstone Advisor: Prof. David Benedetto

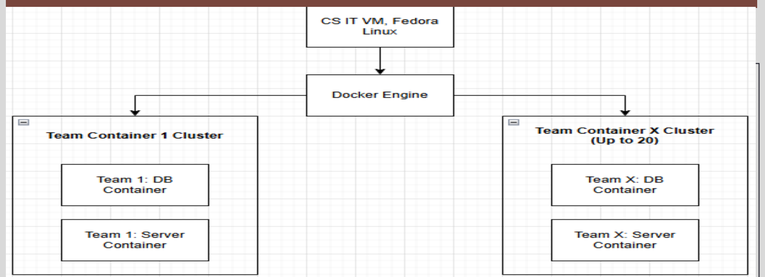
## SYSTEM SEQUENCE DIAGRAM




## TESTING

- JUnit for unit testing; Mockito to support unit tests
- Client-server connectivity verified using a dummy client
- Server endpoints tested for reachability and expected behavior
- Server and database containers integration tests
- CI pipeline via GitLab

## DOCKER DEPLOYMENT




## DESIGN

- (1) Replaced REST-based GameStateController with gRPC-based GameStateService
- (2) Protocol buffers provide a strongly typed, shared contract between server and client teams
- (3) Username/password login RPC returns a JWT session token
- (4) JwtServerInterceptor validates the session on every gRPC call transparently, without changes to each endpoint

## IMPLEMENTATION

- gRPC + Protocol Buffers: Replaced REST to eliminate type mismatches; .proto files auto-generate strongly typed client stubs
- Docker: Deploys server and database containers on the UNH IT VM; removes OS/dependency inconsistencies and enables TAs to pull and run student containers
- MariaDB: SQL database retained from the previous project
- GitLab: Source control and continuous integration

## EVALUATIONS & CONCLUSIONS

- Containerized application enables separation of game instances with different team code
- Dummy client, originally made to test protocol buffers, ended up being one of the most useful features, and much faster than spinning up an Android Studio instance
- Main features are working and have small test suites
- Outstanding work
  - Demo mode
  - UUID integration
  - Client repository integration

## Abstract

The 2025–2026 Battlezone Server Project is aimed at improving the Battlezone project in Professor Plumlee's CS619 course by modernizing the tools and protocols used, improving the usability of the starter code to better support students working on the project for the first time, and moving away from the Android platform toward a more portable and maintainable implementation by packaging the server in a Docker container to run on a virtual machine. This allows the backend to be deployed consistently across different environments to reduce the chance of platform specific issues that are hard to debug which students may have run into previously. The project refactors the existing codebase away from a REST API toward a more modern gRPC streamed connection, enabling more efficient real-time. In doing so, it removes unnecessary Android-specific dependencies, significantly reduces compile time, and decreases the overall size of the generated JAR file. Our project is being developed in parallel with the Battlezone Client Project, ensuring both components interoperate seamlessly while remaining independently deployable and loosely coupled.