CEBU INSTITUTE OF TECHNOLOGY UNIVERSITY

CEBU INSTITUTE OF TECHNOLOGY
UNIVERSITY

COLLEGE OF COMPUTER STUDIES

Software Project Management Plan

for

Darknet Duel

TABLE OF CONTENTS

1. Overview

1.1. Project Summary

1.1.1. Purpose, scope and objectives

The purpose of the Darknet Duel project is to design, develop, implement, and test a complete web-based application (frontend and backend) to support the Darknet Duel cybersecurity-themed card game. This system manages user authentication, player lobbies, card and deck data, game initialization, core gameplay logic, and the user interface for interaction. The project is implemented using a modern JavaScript/TypeScript stack: Node.js, Express, TypeORM, MySQL, boardgame.io, React, and supporting libraries.

The scope of this project encompasses all development activities, including frontend UI/UX design and implementation (React/TypeScript), backend API and service creation (Node.js/Express/TypeORM), game server logic (boardgame.io), database interaction (MySQL), implementation of game rules, integration between frontend and backend, and associated testing and documentation. Activities outside the scope include creation of game assets (e.g., card art), detailed network infrastructure setup beyond the application level, and post-deployment operational maintenance. Specifically, it includes the development of modules for user management, lobby handling, card/deck management, turn-based gameplay mechanics, store/currency, and admin/moderation tools.

The primary objectives of this project are:

- Deliver a robust, scalable, and secure application capable of supporting multiplayer Darknet Duel games.
- Successfully implement all features and functionalities outlined in the Software Requirements Specification (SRS) and the Software Design Document (SDD).
- Ensure the final product adheres to high-quality software engineering standards, including DRY, KISS, YAGNI principles, and OWASP security best practices.
- Provide clear and comprehensive API documentation using Swagger/OpenAPI for seamless integration with frontend clients.
- Deliver all specified application components and documentation according to the project plan.

This project provides the foundational logic for the Darknet Duel game. It is designed to integrate a React-based frontend, a Node.js/Express backend, and a boardgame.io-powered game server, all communicating via REST and WebSocket APIs. The successful completion of this project will provide the necessary infrastructure for players to engage in the Darknet Duel card game. The official statement of product requirements can be found in the Software Requirements Specification (SRS).

1.1.2. Assumptions and constraints

This project is planned based on the following assumptions:

- The frontend client application for Darknet Duel will be developed using React/TypeScript and will interact with the backend and game server via REST and WebSocket APIs.

- The necessary development environment, including Node.js 20+, npm, a suitable IDE, Git, and access to a MySQL database, is readily available to the development team.
- The existing Software Requirements Specification (SRS) and Software Design Document (SDD) provide accurate and stable definitions for the project's requirements and high-level design.
- The development team possesses the required technical expertise in JavaScript/TypeScript, Node.js, Express, React, boardgame.io, REST API design, and MySQL.

The project operates under the following constraints:

- **Technology Stack:** The application must be implemented using Node.js 20+, Express, TypeORM, MySQL, React, boardgame.io, Zustand, and supporting libraries. Swagger/OpenAPI is used for API documentation.
- **Development Principles:** Development must strictly adhere to DRY, KISS, and YAGNI principles. OWASP best practices must be followed for security aspects.
- **Architecture:** The application must follow a modular architecture, with clear separation between backend (API, persistence, admin), game server (real-time game logic), and frontend (UI/UX, state management).
- **Interfaces:** Interaction between frontend, backend, and game server will be exclusively through RESTful and WebSocket APIs, documented with Swagger/OpenAPI. No direct coupling between backend and frontend code is permitted.
- **Timeline:** As a university capstone project, the primary constraint is the academic calendar. The detailed schedule is outlined in section 1.1.4.
- **Resources:** The project relies on the assigned student development team members, university resources (if applicable), and standard open-source libraries and tools. Budgetary constraints are minimal, primarily related to time allocation and potential minor expenses for software licenses or cloud services if required for deployment beyond local testing, which are expected to be negligible or covered by student/university resources.

1.1.3. Project deliverables

The successful completion of the Darknet Duel project will yield the following deliverables:

- **Software:**
  - Complete application source code (Node.js/TypeScript/React) integrated with the defined technology stack.
  - Executable application artifacts (e.g., built frontend, backend, and game server).
  - Database schema definitions (TypeORM entities and migrations).
- **Documentation:**
  - Software Requirements Specification (SRS).
  - Software Design Document (SDD).
  - This Software Project Management Plan (SPMP).
  - Automatically generated API Documentation via Swagger/OpenAPI.
  - Final Project Presentation/Demo materials.

- **Delivery:** All deliverables will be provided electronically via access to the project's Git repository

1.1.4. Schedule and budget summary

The project follows a strict academic schedule defined by the university course requirements. The key milestones and deliverables are aligned with the following timeline:

Week	Dates	Deliverable / Activity	Checking Period
1	Jan 20 - 25	Team Formation	Jan 20 - 25
2-3	Jan 27 – Feb 8	Review of Related Literature	Jan 27 – Feb 8
3	Feb 3 - 8	Patent Search	Feb 3 - 8
4	Feb 10 - 15	Software Project Proposal	Feb 10 - 15
5	Feb 17 - 22	Software Requirements Specification	Feb 17 - 22
6	Feb 24 - 29	Software Project Management Plan	Feb 24 - 29
7	Mar 3 – 8	Software Design Description	Mar 3 – 8
8-9	Mar 24 – April 5	Increment 1: User Management & Auth, Initial Lobby	Mar 24 – April 5
10-11	April 7 – April 19	Increment 2: Lobby, Matchmaking, Game Server Setup	April 7 – April 19
12-13	April 22 – May 3	Increment 3: Core Game Logic, Card System, Real-Time Play	April 22 – May 3
14-15	May 5 – May 17	Increment 4: Store, Currency, Admin/Moderation, Polish	May 5 – May 17
16-18	May 20 – May 31	Software Project Demo/Presentation	May 20 – May 31

[Table 1.1] Capstone 1 Project Schedule

Week	Dates	Deliverable / Activity	Checking Period
1	Aug 7 - 15	Requirements Refactoring - Streamlined functional and non-functional requirements	Aug 7 - 15
2	Aug 18 - 22	Revised SRS, SDD, SPMP - Updated docs	Aug 18 - 22
3	Aug 25 - 29	Software Test Document (STD) - Accomplished STD document	Aug 25 - 29
4-7	Sep 1 - 26	Development, Integration, Deployment & Testing - Deployed and running software	Sep 1 - 26

8-9	Sep 29 - Oct 10	Software Demo - Presentation and demo of software project	Sep 29 - Oct 10
10	Oct 13 - 17	Performance testing - Performance test result	Oct 13 - 17
11	Oct 20 - 24	Software Validation - Tool to be used in usability testing	Oct 20 - 24
12-13	Oct 27 - Nov 7	Usability testing - Field testing data	Oct 27 - Nov 7
14	Nov 10 - 14	User-Driven Improvements - Utilize user feedback for refinement and optimization	Nov 10 - 14
15-16	Nov 17 - 28	Final Presentation - Final demo	Nov 17 - 28
17	Dec 1 - 5	Research report - Completed research report	Dec 1 - 5
18	Dec 1 - 5	Docs & Source Code Submission - Finalized documentation	Dec 1 - 5

[Table 1.2] Capstone 2 Project Schedule

As of the original publication of this SPMP (September 20, 2025), the project is within the **Development, Integration, Deployment & Testing** phase. This phase focuses on completing the remaining development modules, integration testing, deployment setup, and comprehensive testing of the deployed software system.

Budgetary considerations are minimal for this academic project. Costs are primarily related to the time investment of the team members and potential minor expenses for software licenses or cloud services if required for deployment beyond local testing, which are expected to be negligible or covered by student/university resources.

1.2. Evolution of plan

This SPMP is a living document and will be updated by the project team as necessary throughout the project lifecycle, particularly if significant deviations from the plan occur. Scheduled reviews and potential updates may occur at the transition points between implementation increments (INC).

All updates to the SPMP will be managed through the project's GitHub repository. Version history will track changes made to the document. Significant updates will be communicated to all team members during regular project meetings or via designated communication channels.

Version	Primary Author(s)	Description of Version	Date Expected
0.1	Project Team	Initial draft based on project proposal	Feb 2025
1.0	Project Team	Final Version for submission	Feb 29, 2025

Version	Primary Author(s)	Description of Version	Date Expected
1.1	Project Team	Updated post-SSD development (Optional)	Mar 2025
1.2	Project Team	Reflecting progress up to INC 3 (This Update)	May 3, 2025
2.0	Project Team	Reworked SPMP to match the new rewrite, migrating away from Spring Boot	June 14, 2025
3.0	Project Team	Apply requirements refactor: Add lore video, interactive tutorial, and various additions	September 20, 2025

[Table 1.3] SPMP Evolution Plan

2. References

This section lists documents and standards referenced within this SPMP or relevant to the project.

1. **Darknet Duel Software Requirements Specification (SRS)**: Defines functional and non-functional requirements for the project.
2. **Darknet Duel Software Design Document (SDD)**: Outlines the high-level architecture and design decisions for the backend, game server, and frontend system.
3. **IEEE Standard for Software Project Management Plans (IEEE Std 1058-1998)**: Standard providing the structure and guidance for this document.
4. **Node.js Documentation**: (https://nodejs.org/en/docs/)
5. **Express.js Documentation**: (https://expressjs.com/en/5x/api.html)
6. **TypeORM Documentation**: (https://typeorm.io/)
7. **MySQL Documentation**: (https://dev.mysql.com/doc/)
8. **React.js Documentation**: (https://react.dev/)
9. **boardgame.io Documentation**: (https://boardgame.io/documentation/)
10. **Socket.IO Documentation**: (https://socket.io/docs/)
11. **Swagger/OpenAPI 3.0 Specification**: (https://swagger.io/specification/)
12. **OWASP Top Ten**: (https://owasp.org/www-project-top-ten/)

3. Definitions

This clause of the SPMP shall define, or provide references to, documents containing the definition of all terms and acronyms required to properly understand the SPMP.

AP	Action Points – The resource spent by players to perform actions during their turn.
JWT	JSON Web Token – A compact, URL-safe means of representing claims to be transferred between two parties, used for authentication.
REST	Representational State Transfer – An architectural style for designing networked applications, used for the backend API.
API	Application Programming Interface – A set of endpoints and protocols for communication between software components.
WebSocket	A protocol providing full-duplex communication channels over a single TCP connection, used for real-time updates.
Socket.IO	A JavaScript library for real-time web applications, enabling real-time, bi-directional communication between web clients and servers.
Red Team	The attacker role in the game.
Blue Team	The defender role in the game.
Infrastructure	Cards representing assets or systems that can be attacked or defended in the game.
ELO	A rating system used to calculate the relative skill levels of players.
Creds	In-game currency earned by playing matches, used for purchases in the store.
Crypts	Premium in-game currency, typically obtained via payment/top-up, used for special purchases.
boardgame.io	An open-source framework for building turn-based games in JavaScript, used for the game server logic.
TypeORM	An Object-Relational Mapping (ORM) library for TypeScript and JavaScript, used for database interaction.
Zustand	A small, fast state-management solution for React applications.
Vite	A frontend build tool and development server for modern web projects.
Tailwind CSS	A utility-first CSS framework for rapidly building custom user interfaces.
ESLint	A tool for identifying and fixing problems in JavaScript/TypeScript code.
PostCSS	A tool for transforming CSS with JavaScript plugins.
Koa	A web framework for Node.js, used in the game server for certain API endpoints.
Swagger/OpenAPI	A specification and set of tools for describing and documenting RESTful APIs.
TypeScript	A strongly typed programming language that builds on JavaScript, used throughout the project.

4. Project organization

This section describes the project's team structure, roles, and relationships.

4.1. External structure

As a university capstone project, the primary external interface is with the course instructor/faculty advisor(s), who provide guidance, evaluate deliverables, and oversee project progress according to academic requirements. The project may also reference university policies or resources related to capstone projects. There are no external commercial clients or subcontractors involved.

The application developed is intended to interface with a React-based frontend, a Node.js/Express backend, and a boardgame.io-powered game server, all communicating via REST and WebSocket APIs. It also relies on standard external libraries and a MySQL database instance.

4.2. Internal structure

The Darknet Duel capstone team consists of five members:

- **Project Lead:** Matthew Emmanuel O. Echavez
- **Members:**
  - Brian Steve E. Pila
  - Kenjie B. Bentain
  - Ephraim Jay A. Solasco
  - Scott Benzer Gitgano

The team operates collaboratively, with the Project Lead responsible for overall coordination, task assignment tracking (using SPEAR), ensuring adherence to the schedule, facilitating communication, and leading the integration of modules. All members contribute to development, testing, and documentation activities.

4.3. Roles and responsibilities

While all members participate across activities, primary responsibilities are assigned as follows:

TEAM MEMBER	ROLE & RESPONSIBILITIES	
	SEM 1	SEM 2
Matthew Emmanuel O. Echavez	Project Lead	Project Lead/Backend Developer
Brian Steve E. Pila	Developer	Frontend Developer
Kenjie B. Bentain	Developer	Tester/QA
Ephraim Jay A. Solasco	Developer	Designer/Artist
Scott Benzer Gitgano	Developer	Tester/QA

5. Managerial process plans

This section details the plans for managing the project execution, covering start-up, risk management, work planning, control mechanisms, and closeout procedures.

5.1. *Start-up plan*

This initial phase outlines the planning for project estimation, staffing needs, resource acquisition, and any required training.

5.1.1. *Estimation plan*

Project estimation is primarily based on the breakdown of work defined in the academic schedule (Section 1.1.4) and the complexity outlined in the Software Requirements Specification (SRS) and Software Design Document (SDD). Each major deliverable (SRS, SPMP, SDD, Increments 1-4) has allocated timeframes. Estimation within the implementation increments considers the complexity of features within the assigned modules, dependencies, and the anticipated effort for coding, testing, and documentation. The team's collective experience and collaborative nature are factored in. Estimates are refined collaboratively during weekly team meetings as work progresses.

5.1.2. *Staffing plan*

The project is staffed by the pre-defined 5-member capstone team, as listed in Section 4.2. All members are expected to dedicate sufficient time as required by the capstone course requirements throughout the project duration. The team composition is fixed for the project's duration.

5.1.3. Resource acquisition plan

Required resources include:

- **Hardware:** Each team member will use their personal computers equipped for software development.
- **Software:** Standard development tools (Node.js 20+, npm, IDEs like VS Code or WebStorm, Git, MySQL Community Server or equivalent), communication tools (primarily Microsoft Teams), and a task management tool (Trello and SPEAR). All core development software is open-source or available via free licenses (e.g., student licenses for IDEs).
- **Information:** Access to university library resources for literature review, online documentation for technologies used (Node.js, Express, React, boardgame.io, etc.), and guidance from faculty advisors.

Resource acquisition involves installing necessary software on personal machines. No significant procurement is anticipated.

5.1.4. Project staff training plan

No formal training plan is required, as team members are assumed to possess the necessary foundational knowledge in software development, JavaScript/TypeScript, and related technologies required for the capstone project based on prior coursework. Skill gaps or learning related to specific framework features (e.g., advanced boardgame.io or TypeORM techniques) will be addressed through self-study, peer-to-peer knowledge sharing within the team, and referencing online documentation and tutorials.

5.2. Work plan

This section specifies the detailed work activities, scheduling, resource allocation, and budget considerations for the Darknet Duel project.

5.2.1. Work activities

The project work is broken down according to the major phases defined in the academic schedule (Section 1.1.4). The core development work is structured into four implementation increments, each corresponding to a set of modules and features as defined in the SRS and SDD. The Work Breakdown Structure (WBS) is as follows:

Phase	WBS ID	Task / Module	Key Activities
Setup & Planning	1.0	Initial Setup & Documentation	Literature Review, Proposal, SRS, SPMP, SDD, Env Setup
Increment 1	2.0	User Management & Authentication	Registration, Login, Profile, Role-based Access, Initial Lobby
	2.1	Module 1 Implementation	Code, Unit Tests, Integration Tests, Peer Review
Increment 2	3.0	Lobby & Matchmaking, Game Server Setup	Lobby Browser, Create/Join/Leave Lobbies, Real-time Lobby Updates, Game Server Bootstrapping
	3.1	Module 2 Implementation	Code, Unit Tests, Integration Tests, Peer Review
Increment 3	4.0	Core Game Logic, Card System, Real-Time Play	Game Creation, Turn Logic, Card Play, State Sync, Disconnection Handling
	4.1	Module 3 & 4 Implementation	Code, Unit Tests, Integration Tests, Peer Review
Increment 4	5.0	Store, Currency, Admin/Moderation, Polish	In-game Currency, Store, Admin Tools, Moderation, Final Polish
	5.1	Module 5, 6, 7 Implementation	Code, Unit Tests, Integration Tests, Peer Review
	5.2	Comprehensive E2E Testing	Expand test coverage, Run full suite
	5.3	Final Documentation & Packaging	Update README, Final checks, Build artifact
Demo & Delivery	6.0	Project Demonstration	Prepare presentation, Conduct demo, Final Submission

[Table 5.1] High-Level Work Breakdown Structure

Detailed task breakdowns within each module implementation are managed by the team, using a task board or list in Trello or a similar tool, ensuring coverage of all required modules including real-time communication aspects.

5.2.2. Schedule allocation

The allocation of schedule adheres strictly to the timeline provided in Section 1.1.4. Each INC spans approximately two weeks, requiring focused effort from the team members to complete the planned modules within that timeframe. The Project Lead is responsible for monitoring progress against this schedule during weekly meetings.

5.2.3. Resource allocation

Team members (human resources) are allocated across all implementation increments. Specific task assignments within each increment are managed by the Project Lead. Software and hardware resources are available throughout the project duration. No specialized resources requiring specific scheduling constraints are anticipated.

5.2.4. Budget allocation

As stated in Section 1.1.4, the project operates with a minimal budget. No specific budget allocation per phase is required beyond managing team time effectively.

5.3. Control plan

This section outlines the metrics, reporting methods, and control procedures used to manage the project's requirements, schedule, budget (implicitly time/effort), quality, risks, and eventual closeout, consistent with the collaborative and academic nature of the project.

5.3.1. Requirements control plan

Control over product requirements is maintained through adherence to the defined Software Requirements Specification (SRS) and Software Design Document (SDD). Any proposed changes or clarifications identified during development are discussed within the team during meetings. The Project Lead holds the final authority on accepting requirement modifications. Approved changes that impact scope or design must be reflected in updates to the SRS and/or SDD. The potential impact of any change on schedule and effort is assessed during these discussions. All changes to requirements documentation and related code are tracked via Git commits, providing traceability. The primary mechanism for control is the Project Lead's oversight and the explicit goal of implementing the features as documented, avoiding scope creep unless formally approved and documented.

5.3.2. Schedule control plan

Schedule control focuses on monitoring progress against the module completion milestones outlined in Table 1.1. Progress within a module is measured by the completion of features and associated test cases. At the end of each module development cycle (approximately every 2-3 weeks), the Project Lead assesses the achieved functionality against the plan. Objective criteria include successfully running the module's test suite and demonstrating the core features. If significant delays occur (e.g., a module overruns by a week or more), the Project Lead and team analyze the cause and determine corrective actions, which might involve adjusting priorities for the next module or simplifying non-critical features if permissible within the overall requirements. Progress tracking is mainly informal through regular interaction and observing completed tasks, supplemented by the milestone reviews.

5.3.3. Budget control plan

Human resources (team members' time and effort) are the primary controlled resource. The Project Lead monitors task completion and workload distribution during weekly meetings to ensure resources are utilized effectively and to identify potential bottlenecks or overburdened members. Software and hardware resources are generally available and require minimal control beyond ensuring necessary tools are installed and functional.

5.3.4. Quality control plan

Quality control is implemented through several mechanisms integrated into the development process. Adherence to coding standards (DRY, KISS, YAGNI) and security best practices (OWASP) is a primary focus, enforced through Project Lead guidance and review of team members' output.

Comprehensive automated testing (unit, integration, end-to-end) forms a core part of the QA strategy, providing continuous feedback on code quality and functional correctness. Verification activities, particularly Project Lead code reviews and design adherence checks, are integral to assuring quality. Validation testing ensures the product meets user needs.

The QA process is directly linked to V&V activities, configuration management (ensuring code integrity), and reviews (Project Lead oversight). There is no separate QA team; quality is a shared responsibility, primarily driven by the Project Lead's standards and oversight and team members' adherence to instructions and best practices.

5.3.5. Reporting plan

Progress reporting occurs primarily through:

- **Weekly Team Meetings:** Each member reports on completed tasks, current work, and any impediments. The Project Lead summarizes overall progress against the schedule.
- **Version Control Commits:** Git commit messages provide a granular log of development activity.
- **Task Management Tool Updates:** If used (e.g., ClickUp or GitHub Projects), the status of tasks provides visibility into progress.
- **Milestone Deliverables:** Submission of major documents (SRS, SDD, SPMP) and demonstration of functionality at the end of increments serve as formal progress indicators for faculty advisors.

Formal reports beyond the required academic submissions are not planned.

5.3.6. Metrics collection plan

Metrics collection will be kept simple for this academic project:

- **Schedule Adherence:** Tracking completion of major milestones and increment goals against the planned dates (Section 1.1.4).
- **Functionality Completion:** Tracking the number of planned features/user stories implemented per increment.
- **Test Results:** Monitoring pass/fail rates of unit, integration, and end-to-end tests.
- **(Optional) Code Metrics:** Basic metrics like lines of code or cyclomatic complexity might be tracked using IDE plugins or build tools if deemed useful by the team, but are not a primary focus.

Metrics are reviewed during weekly team meetings to inform progress assessment and identify potential issues.

5.3.7 Risk management plan

Risk management involves the ongoing identification, analysis, and mitigation of potential issues that could negatively impact the project. Initial risks identified include: technical challenges in specific modules, integration difficulties between components developed by different members, inaccurate estimations leading to schedule slippage, team member availability issues (due to other coursework or personal matters), unclear requirements interpretation, and environment/tool setup problems.

Risks are managed proactively. The Project Lead and team identify emerging risks during development discussions. The Project Lead prioritizes risks based on potential impact and likelihood. Mitigation strategies may involve breaking down complex tasks into smaller steps, allocating additional time for research or complex implementations, performing careful integration testing, strictly controlling scope via the requirements control plan, leveraging team members' ability to quickly refactor or generate alternatives, and maintaining clear communication. High-priority risks and their status are discussed as needed. This risk management process is embedded within the regular development workflow rather than being a separate periodic activity.

5.3.8 Project closeout plan

Orderly project closeout occurs upon completion of all development and final testing activities, aligned with the end of the project timeline. The closeout plan includes several key steps. First, the Project Lead ensures all final code, tests, and documentation updates are committed to the Git repository. A final execution of the complete test suite verifies the system's state. The Project Lead performs a final review of key documentation, particularly the README.md for completeness and accuracy of setup/run instructions, and ensures Swagger/OpenAPI documentation is up-to-date. The project's Git repository serves as the official archive for all source code, documentation, and configuration files. Finally, an informal post-mortem or reflection may occur among the team to discuss lessons learned during the project. No formal staff reassignment or extensive archiving procedures beyond securing the Git repository are required.

6. Technical process plans

This clause specifies the development process model, technical methods, tools, techniques, infrastructure, and the product acceptance plan for the Darknet Duel project.

6.1 Process Model

The development process follows a collaborative, iterative model adapted for a student capstone team. The project is broken down into distinct modules based on the features outlined in the Software Requirements Specification (SRS) and the structure defined in the Software Design Document (SDD). Development proceeds through the four major implementation increments defined in the schedule (Section 1.1.4).

Within each increment, the workflow typically involves:

1. **Task Assignment:** The Project Lead assigns specific features or sub-modules to developers.
2. **Development:** Developers implement the assigned functionality, adhering to coding standards and design principles.
3. **Unit/Integration Testing:** Developers write and execute tests for their code (using Jest, React Testing Library, Supertest, etc.).
4. **Code Review:** Peer reviews are conducted among team members (e.g., using Git pull requests or pair programming sessions) to ensure quality and knowledge sharing.
5. **Integration:** Code is merged into the main development branch.
6. **End-to-End Testing:** As functionality becomes available, end-to-end tests are developed and executed.
7. **Iteration Review:** The team reviews progress at the end of minor cycles (e.g., weekly) and adjusts plans as needed.

Communication and collaboration occur through regular team meetings, shared code repositories (Git), and communication channels (e.g., group chat, shared workspace).

6.2 Methods, tools, and techniques

The project employs a combination of modern development methodologies, languages, tools, and techniques to ensure a high-quality application.

Methods and Techniques

Category	**Methods and Techniques**
Project Methodology	Agile-inspired iterative development, Frequent team feedback/communication
Development Principles	DRY, KISS, YAGNI, OWASP Security Best Practices
Requirements Elicitation	Analysis of SRS and SDD documents, Team discussions within shared workspace

Category	**Methods and Techniques**
Design	Modular Architecture (Backend/Game Server/Frontend), Component-based Architecture (Frontend), UML/PlantUML Diagrams
Implementation	Backend: Node.js/Express/TypeORM, Game Server: boardgame.io, Frontend: React/TypeScript, Zustand, DTO pattern
Testing	End-to-End API Testing with Swagger and Test Pages for WebSocket testing
API Documentation	In-code annotations using Swagger/OpenAPI
Database Interaction	Object-Relational Mapping (JPA), JPQL for custom queries

[Table 6.2.1] Methods and Techniques

Tools

Category	**Tools**
Operating System	Developer's Preferred OS (Windows, macOS, Linux)
Languages	JavaScript (ES2022+), TypeScript, SQL (via TypeORM), Python 3 (for testing scripts)
Frameworks/Libraries	Node.js, Express, TypeORM, MySQL, React.js, Zustand, boardgame.io, Jest, React Testing Library, Supertest, Socket.IO
Database	MySQL
Build Tool	npm
Design Notation	PlantUML (for diagram generation)
Configuration Management	GitHub
Documentation Tools	Swagger UI (for API visualization)
Development Environment	VS Code / WebStorm (or other IDEs), Postman (API testing)
Project Planning/Tracking	Communication Tools such as Microsoft Teams

[Table 6.2.2] Tools

The frontend application is built using **Vite** for fast development and optimized production builds, with **Tailwind CSS** for utility-first styling. **ESLint** is used for code linting, and **PostCSS** is used for CSS processing. TypeScript configuration is managed via tsconfig.json files. This setup ensures rapid feedback during development, consistent code quality, and modern, maintainable styling practices.

Technical standards governing development are derived from the established team preferences (coding styles, architectural patterns like DTO usage) and general Node.js/React best practices.

6.3 Infrastructure Plan

The project infrastructure primarily relies on the team members' local development environments and the capabilities provided by the shared workspace environment. The local setup consists of a Windows, macOS, or Linux operating system equipped with the necessary software: Node.js 20+, a suitable Integrated Development Environment (IDE) chosen by the team, Git client, and a running MySQL database server instance. The server-side application requires an environment capable of running the Node.js backend, boardgame.io game server, and supporting persistent WebSocket connections (typically handled by Socket.IO).

Establishment and maintenance of the local hardware and software environment, including operating system updates, IDE configuration, and database administration, are the responsibility of each team member. No dedicated physical servers, specialized hardware, or office facilities are provisioned specifically for this project beyond the team's standard setup. Network connectivity is assumed to be available for accessing dependencies via npm, utilizing the shared workspace environment, and maintaining stable WebSocket connections during testing and gameplay.
.

6.4 Product Acceptance Plan

Acceptance of the Darknet Duel application is determined through team validation upon the completion of each development module and following the final integration phase. The primary method for acceptance involves the successful execution of the comprehensive test suite associated with each module and the overall system, demonstrating functional correctness and adherence to requirements. Objective criteria include: 100% pass rate for all defined automated tests (unit, integration, end-to-end), team confirmation that the implemented features behave as specified in the Software Requirements Specification (SRS) and Software Design Document (SDD) through manual testing or review, and adherence to the agreed-upon coding standards and architectural patterns. Additional acceptance criteria from refactoring include: 100% REST endpoint coverage in Swagger (REQ-301), verified rate limiting policies (REQ-302), documented WebSocket protocol (REQ-303), audit logs recorded for admin actions (REQ-402), GDPR export/deletion flows (REQ-403), security headers/CORS present (REQ-404), session timeout enforced (REQ-405), CI security gate fails on High/Medium vulnerabilities (REQ-406), and frontend UX foundations validated (error boundary, toasts, offline grace, first-time video + tutorial) (REQ-201–REQ-206).

Formal acceptance is achieved implicitly when the team approves the state of a completed module or the final integrated product, indicating readiness to proceed or conclude the project. No separate formal acceptance documentation or sign-off ceremony is planned, relying instead on the iterative validation and ongoing communication within the shared workspace environment. Testing tools (Jest, React Testing Library, Supertest, Swagger, Git, and the team's chosen IDEs) and code inspection by the team are the core techniques used for acceptance verification..

7. Supporting process plans

This clause details the supporting processes essential for the successful execution of the Darknet Duel project throughout its lifecycle. These plans cover configuration management, verification and validation, documentation, quality assurance, reviews, problem resolution, and process improvement. Subcontractor management is not applicable and thus not planned.

7.1. Configuration management plan

Configuration management (CM) for this project is centered around the use of Git for version control, hosted on a platform like GitHub. All source code, documentation (SPMP, SRS, SDD), design diagrams, and test scripts are stored and managed within the shared Git repository.

Control is maintained through standard Git workflows. Developers typically work on feature branches and submit pull requests for review before merging into a main development branch (e.g., main or develop). The Project Lead may oversee the merging process for the main branch. Baselining occurs implicitly with significant merges or tagged releases corresponding to increment completions. Status accounting is provided by Git's log, history, and branch/tag features.

Change requests (e.g., requirement adjustments, significant design changes) are discussed within the team during meetings. The Project Lead facilitates the decision process, considering impacts on scope and schedule. Approved changes are documented (e.g., updating SRS/SDD) and implemented, with traceability maintained through Git commits.

7.2. Verification and validation plan

This subclause contains the verification and validation (V&V) plan for the Darknet Duel project, detailing the scope, tools, techniques, and responsibilities for V&V activities. V&V is integrated throughout the development lifecycle. The Project Lead oversees V&V activities, ensuring independence between development and verification where practical (e.g., code reviews). Validation is primarily achieved through testing against requirements and team acceptance.
Key V&V activities include:

- **Traceability Analysis:** Ensures requirements from the SRS are reflected in the SDD, implemented in the code (both frontend and backend), and covered by tests. Traceability is maintained via the module structure, documentation references, and Git commit history.
- **Evaluation:** Assesses work products for correctness, consistency, and adherence to standards. This includes Project Lead code reviews (checking against DRY, KISS, YAGNI, OWASP), design reviews against the SDD and diagrams, documentation reviews, and potentially static analysis provided by IDEs.
- **Interface Analysis:** Verifies the consistency and correctness of interfaces between components, primarily focusing on the REST API (defined via Swagger/OpenAPI) and WebSocket communication contract between the React frontend, Node.js backend, and boardgame.io game server.
- **Testing:** Employs various testing levels:
    - Component/Unit Testing: Verifying individual backend modules (Jest), React components (React Testing Library), and game logic (boardgame.io tests).
    - Integration Testing: Testing interactions between backend components (Supertest) and frontend service integrations.
    - System Testing: Evaluating the integrated system against SRS requirements (may

involve manual testing or broader automated E2E tests).

- End-to-End (E2E) Testing: Using Jest/Supertest or Python scripts (requests library, potentially WebSocket clients) to test full user scenarios through the API.
- Acceptance Testing: Implicitly performed by the team upon successful completion of module tests, feature validation against requirements, and the final project demonstration.
- Security Testing: Automated OWASP ZAP baseline scan in CI must report 0 High/Medium findings; pipeline fails otherwise.
- Documentation Testing: Swagger generation step must include all REST endpoints; missing annotations fail the pipeline.
- Coverage Gates: Jest coverage thresholds enforced in CI for lines/branches/functions.
-

The following table outlines the V&V plan for each major phase:

Phase	**V&V Input**	**V&V Tasks**	**V&V Output**
Requirements/Design	SRS, SDD, PlantUML Diagrams	Requirements/Design Traceability Analysis, Requirements/Design Evaluation, Interface Analysis (API/WebSocket Definition), High-level Test Plan Generation	V&V Task Reports (Informal), Initial Test Strategy Outline
Implementation	SDD, Code (Node.js, React), Design Documentation	Code Traceability Analysis, Code Evaluation (Reviews), Interface Analysis (Implementation Checks), Documentation Evaluation (Swagger, Comments), Unit/Integration Test Execution	V&V Task Reports (Informal), Test Results (Unit/Integration), Code Review Feedback (via Git/Workspace), Generated Swagger Documentation
Testing/Integration	Integrated Codebase, Unit/Integration Test Results, Test Cases/Procedures	System Test Execution (Manual/E2E), End-to-End Test Execution (Jest/Supertest/Python Scripts), Defect Reporting and Tracking, Regression Testing	V&V Task Reports (Informal), Test Results (System/E2E), Bug Reports/Issue Tracker Updates
Delivery/Acceptance	Final Integrated System, All Test Results, SRS/SDD, Final Documentation	Final Acceptance Testing (Team Validation/Demo), Final Documentation Review (README), Review of Test Coverage	V&V Task Reports (Informal), Final V&V Summary (Potentially part of Final Report/Presentation), Implicit Team Acceptance

[Table 7.2.1] V&V Plan by Phase

The primary responsibility for V&V lies with the entire development team. Developers write unit/integration tests for their code. Peer code reviews contribute significantly to verification. The team collaborates on developing and maintaining the end-to-end test suite. The Project Lead oversees the overall testing strategy and ensures adequate coverage. Tools employed include Jest, React Testing Library, Supertest, Swagger, Git, and the team's chosen IDEs.

7.3. Documentation plan

The documentation plan covers the creation and maintenance of both deliverable and non-deliverable work products essential for the project.

Non-deliverable work products, primarily used during development, include this SPMP, the Software Requirements Specification (SRS), the Software Design Document (SDD), detailed design diagrams, Git commit logs, and communication history within the shared workspace environment. These are maintained and updated as needed, primarily by the Project Lead with input or generation assistance from team members.

Deliverable work products include the complete application source code, generated Swagger/OpenAPI documentation, automated test suites (unit, integration, end-to-end), a comprehensive README.md file, and the final project report/presentation required by the course.

The Project Lead coordinates the creation and maintenance of major documents (SPMP, SRS, SDD, final report), with contributions from all team members. Developers are responsible for code comments, Swagger annotations for their APIs, and potentially documentation specific to complex modules they implemented. Peer reviews extend to documentation to ensure clarity and accuracy. All documentation is version-controlled in Git.

7.4. Quality assurance plan

Quality Assurance (QA) activities are interwoven with the development process to ensure both the final product and the development process itself meet the defined standards and objectives. The plan focuses on defect prevention and early detection rather than relying solely on post-development QA cycles. Key QA elements include strict adherence to coding standards (DRY, KISS, YAGNI) and security best practices (OWASP), enforced through Project Lead guidance and review of team members' output.

Comprehensive automated testing (unit, integration, end-to-end) forms a core part of the QA strategy, providing continuous feedback on code quality and functional correctness. Verification activities, particularly Project Lead code reviews and design adherence checks, are integral to assuring quality. Validation testing ensures the product meets user needs.

The QA process is directly linked to V&V activities, configuration management (ensuring code integrity), and reviews (Project Lead oversight). There is no separate QA team; quality is a shared responsibility, primarily driven by the Project Lead's standards and oversight and team members' adherence to instructions and best practices.

7.5. Reviews and audits

Project reviews are conducted through several mechanisms:

- **Peer Code Reviews:** Developers review each other's code (e.g., via pull requests) before merging to the main branch.

- **Weekly Team Meetings:** Progress is reviewed, issues are discussed, and plans are adjusted.
- **Milestone Reviews/Demos:** Functionality is demonstrated to the team and potentially faculty advisors at the end of implementation increments (INCs).
- **Faculty Advisor Reviews:** Formal reviews of deliverables (SRS, SDD, SPMP, final demo) are conducted by the course instructor/advisor according to the academic schedule.

No external audits are planned.

7.6. Problem resolution plan

Problems (bugs, defects, requirement misunderstandings) are identified through testing (automated or manual), code reviews, or team discussions. Issues should be reported promptly within the team on the designated Microsoft Teams chat.

Analysis and prioritization are done collaboratively by the team, led by the Project Lead. High-priority issues impacting core functionality or blocking progress are addressed first. Resolution involves developers fixing the issue on a branch, testing the fix, and merging it after peer review. Verification is done by the reporter or through automated tests. The Git history serves as the primary log for tracking fixes.

7.7. Subcontractor management plan

Not applicable. This is a self-contained student capstone project with no subcontractors.

7.8. Process improvement plan

Process improvement is handled informally through team reflection and discussion, typically during weekly meetings or post-mortems after major increments or the final demo. The team can identify bottlenecks (e.g., slow code reviews, inefficient testing) or successful practices and adjust their workflow accordingly for subsequent phases or future projects. Lessons learned will be documented as part of the final project report or presentation as required by the course.

8.0 Additional Plans

No formal additional plans beyond those integrated into the preceding sections are required for this project. Security considerations, particularly adherence to OWASP best practices, are embedded within the development methodology, quality assurance, and verification activities described in sections 6 and 7. Specific plans for installation, user training, data conversion, system transition, maintenance, or support are outside the scope of this project.

9. Plan Annexes

All essential supporting details and specifications are contained within the main project documents, referenced throughout this SPMP. No separate annexes are attached. Key reference documents include:

- Software Requirements Specification (SRS)
- Software Design Document (SDD)

10. Index

An index is not provided for this document.