Project Plan: Clue-Less

1. Scope

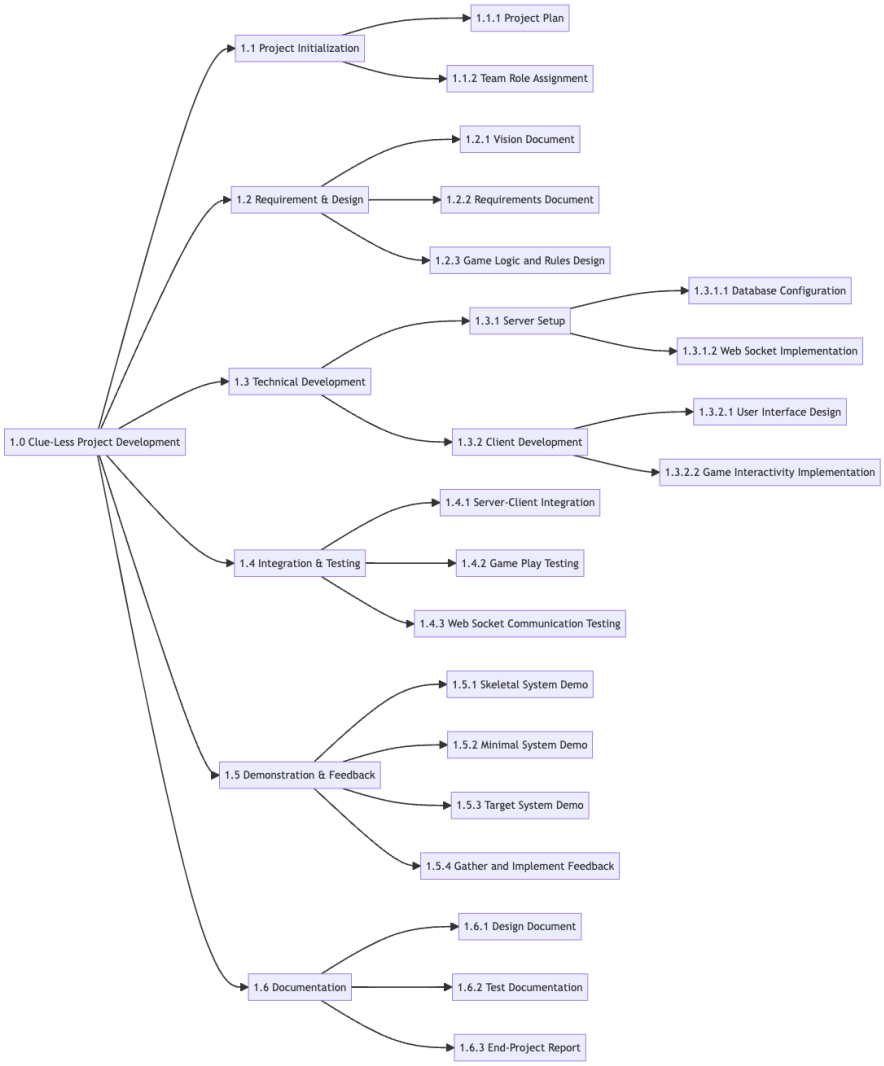
- **1.1 Project Objectives**
    1. Develop a computerized version of Clue-Less: Create a simplified webpage digital version of the classic board game, Clue®, with streamlined navigation and real-time multiplayer functionality.
    2. User-Friendly Interface: Design an intuitive and accessible interface that allows players to participate from different devices or browsers.
    3. Real-time Interaction: Utilize web sockets to enable synchronous gameplay and provide instant notifications on changes in the game state.
- **1.2 Waterfall Life Cycle**
    1. Requirements:
        - Collect and document game needs.
        - Output: Vision and Requirements Documents.
    2. Design:
        - Blueprint the game's architecture.
        - Output: Skeletal System.
    3. Implementation:
        - Develop the frontend and backend.
        - Output: Working game code.
    4. Testing:
        - Verify game functionality and fix bugs.
        - Output: Tested and integrated game.
    5. Deployment:
        - Launch the game for users.
        - Output: Live Clue-Less game.
- **1.3 Milestones**
    1. Complete Vision Document: October 9
    2. Submit Requirements Document: October 16
    3. Skeletal System Demo: October 23
    4. Finalize Design Document: November 6
    5. Minimal System Demo: November 13
    6. Quality Assurance Check: Before November 20
    7. Complete Testing: By December 4
    8. Target System Project Demo: December 11
- **1.4 Features of the Four Increments**
    1. ***Skeletal*** - Architecture:
        - Server-client architecture design

- Basic communication protocol between server and client
- Database structure (if required)

2. ***Minimal*** - Most Important Functionality:
    - Very basic UI (text)
    - Each player gets their cards at the beginning of the game.
    - Give a basic way to take notes or show what you've found out.
    - Rooms and starting squares are implemented.
    - Basic character movement (to hallway and rooms)
    - Suggestion mechanism
    - Make suggestion.
    - Notifications of suggestions made by others.
    - Player connectivity and session management
3. ***Target*** - One Semester Production:
    - Fully functional room-to-room movement and secret passages
    - Complete game rules as described in the original Clue-Less description.
    - Graphical User Interface (GUI) for the client side
    - Show where on the board where each character is.
    - Multiplayer functionality (supporting multiple users)
4. ***Dream*** - Wish List Features:
    - Advanced graphics and game animations
    - In-game chat functionality
    - Scoreboard or leaderboard
    - Player profiles with game statistics

2. Organizational Structure

- **2.1 Management/Team Structure**
    - Bohan leads the overall management of the project.
    - Jacob leads backend development.
    - Fabricio leads skeletal system design, including web socket.
    - Onur leads testing, deployment with Azure, and further improvements, e.g., frontend UI optimization.
- **2.2 Work Breakdown Structure**



3. Resources

- **3.1 Human Resources**
  - Project Manager: Manages timelines and communication.
  - Backend Developer: Handles Python/Flask server setup and integration.
  - Frontend Developer: Develops the UI using React.
  - QA Specialist: Manages testing and quality assurance.
  - DevOps Engineer: Manages deployment on AWS or Azure.
- **3.2 Technical Resources**
  - Backend: Python with Flask framework.
  - Frontend: React.
  - Version Control: Git, GitHub.

    - Database: Selection based on project needs (e.g., PostgreSQL, MySQL, MongoDB).
    - Hosting: AWS or Azure platforms.
- **3.3 Hardware**
    - Development computers/laptops.
    - Various devices for testing web responsiveness.

4. Quality Plan

- **4.1 Quality Assurance**
    - Conduct peer reviews to evaluate code and design documents to ensure they meet specified requirements and are free from errors.
    - Regular team meetings to ensure alignment. These meetings facilitate effective communication, encourage collaborative problem solving, and help in setting clear quality goals and expectations.
- **4.2 Testing**
    - Unit Testing: Testing individual units or components.
    - Integration Testing: Testing combined parts of an application to determine if they function correctly together.
    - System Testing: Validate the complete and fully integrated software product to ensure that it operates correctly and meets all specified requirements.
    - Performance Testing: Assess the scalability, speed, and responsiveness of the application under varying levels of load and stress conditions.
    - User Acceptance Testing (UAT): Conducting tests with actual users to ensure the system meets requirements.
- **4.3 Configuration Management**
    - Use version control systems like Git.
    - Maintain different branches for features, testing, and production.
    - Document software versions, database schema versions, and any third-party tool versions.

5. Project Estimates

- **5.1 Time Effort & Human Resources Allocation**
    1. **Basic Server Setup (1 week):**
        - Set up a basic Flask server to handle initial requests and responses.
        - Resources: 1-2 team members focused on server-side development.

2. **Establish Skeletal System (2 weeks):**
    - Implement foundational communication between subsystems.
    - Test basic message passing and endpoint checks.
    - Resources: 2 team members (1 focused on server-side, 1 on initial client-side setups).
3. **Detailed Backend Development (3.5 weeks):**
    - Develop server functionalities: game logic, database connections, and WebSocket enhancements.
    - Iterative testing for each new backend functionality.
    - Resources: 2-3 team members on server-side development, database, and WebSocket handling.
4. **Basic Frontend Development (2 weeks):**
    - Design a minimal text-based interface in React.
    - Ensure it communicates effectively with the backend.
    - Resources: 1-2 team members on client-side development.
5. **Deployment Setup (1 week):**
    - Choose between AWS or Azure.
    - Configure the server for deployment, set up the database in the cloud, and ensure security measures.
    - Initial deployment and testing in the cloud environment.
    - Resources: 1-2 team members, preferably those familiar with the chosen cloud platform.
6. **Integration, Testing, & Iteration (1.5 weeks):**
    - Merge frontend and backend components and test as a unit.
    - Debug integration issues.
    - Resources: Entire team.

- **5.2 Assumptions**
    1. **Team Expertise**: The team members possess the necessary skills in React for frontend development, Flask for backend development, and AWS/Azure for deployment.
    2. **Availability**: All team members are available and committed throughout the 11-week duration of the project.
    3. **Tools & Technology**: The chosen technologies (React, Flask, AWS/Azure) will suffice for the entirety of the project, without the need to switch or integrate additional technologies midway.
    4. **External Dependencies**: Third-party libraries or tools used will be maintained, and there won't be any unexpected deprecations or breaking changes during the project's timeline.
- **5.3 Constraints**

1. **Time Limitation**: The entire project, from conceptualization to deployment, needs to be completed within an 11-week timeframe.
2. **Resource Limitation**: The team comprises only four members, restricting the parallel development or multitasking capabilities.
3. **Experience Level**: There might be a steeper learning curve and potential delays in certain advanced functionalities.
4. **User Interface**: Initially, the project's frontend will be minimal, and text based. Enhancements in GUI will be considered only if time permits.

6. Project Schedule Estimate

- **6.1 Project Schedule**

  Week of Sep 25 - Oct 1:

  - **Tasks**: Kick-off meeting, assign roles, set project objectives, gather initial vision for the project, draft the Vision Document, and ***set up basic server structure (Backend).***
  - **Deliverable**: Draft of Vision Document.

  Week of Oct 2 - Oct 8:

  - **Tasks**: Finalize Vision Document, engage with stakeholders, submit the Vision Document, start gathering detailed requirements, define functionalities, constraints, use cases, and ***begin skeletal system's architecture planning****.*
  - **Deliverable**: Vision Document (Due Oct 9).

  Week of Oct 9 - Oct 15:

  - **Tasks**: Complete the Requirements Document, finalize it, define the architecture for the skeletal system, ***start foundational communication development between subsystems****.*
  - **Deliverable**: Requirements Document (Due Oct 16).

  Week of Oct 16 - Oct 22:

  - **Tasks**: ***Complete development for the Skeletal System Increment****.* Test the system to ensure basic functionalities, such as message passing, are working.
  - **Deliverable**: Skeletal System Increment Demo (Due Oct 23).

Week of Oct 23 - Oct 29:

- **Tasks**: Outline the Design Document, detail system design, components, modules, interfaces, database designs, and ***initiate detailed backend development with Flask, setting up APIs and WebSocket basics.***
- **Deliverable**: Draft of Design Document.

Week of Oct 30 - Nov 5:

- **Tasks**: Finalize Design Document, submit it, ***continue backend development with game logic integration, and start minimal frontend development (text-based UI).***
- **Deliverable**: Design Document (Due Nov 6).

Week of Nov 6 - Nov 12:

- **Tasks**: ***Complete the Minimal System Increment***, including finalizing minimal frontend integration with the backend, and test essential functionalities.
- **Deliverable**: Minimal System Increment Demo (Due Nov 13).

Week of Nov 13 - Nov 19:

- **Tasks**: Engage in 'Assuring Software Quality' module, focus on quality assurance methodologies, ***initiate deployment preparations for AWS/Azure,*** and test cloud compatibility.
- **Deliverable**: Assuring Software Quality module completion (Due Nov 20).

Week of Nov 20 - Nov 26:

- **Tasks**: Dive into software testing methodologies, prepare test cases/scripts, ***initiate comprehensive backend testing, and ensure system stability for deployment.***
- **Deliverable**: Software Testing module completion (Due Dec 4).

Week of Nov 27 - Dec 3:

- **Tasks**: ***Deploy the project to AWS/Azure, test live functionalities,*** finalize stages of Target System Increment development, and make final adjustments.
- **Deliverable**: Draft of Target System for internal review.

Week of Dec 4 - Dec 10:

    - **Tasks**: Finalize the target system, conduct a comprehensive review, ***ensure optimal user experience, and troubleshoot any live issues.***
    - **Deliverable**: Target System Increment Project Demo (Due Dec 11).
- **6.2 Deliverables List**
    - **Vision Document**: Due Oct 9 - Outlining the project's primary goals, objectives, stakeholders, and scope.
    - **Requirements Document**: Due Oct 16 - Detailed functionalities, constraints, use cases, and requirements for the system.
    - **Skeletal System Increment Demo**: Due Oct 23 - A basic architectural demo showing main functionalities.
    - **Design Document**: Due Nov 6 - A detailed system design, including components, modules, interfaces, and database designs.
    - **Minimal System Increment Demo**: Due Nov 13 - Demonstrating the main functionalities of the system.
    - **Assuring Software Quality Module**: Due Nov 20 - Focusing on software quality assurance, methodologies, and practices.
    - **Software Testing Module**: Due Dec 4 - Emphasizing on software testing methodologies, tools, and best practices.
    - **Target System Increment Project Demo**: Due Dec 11 - Demonstrating the complete system with all functionalities in place.

7. Risk Assessment Plan

- **7.1 Project Risk & Mitigation**
    1. **Scope Changes**
        - **Risk**: Changes in requirements after development begins.
        - **Mitigation**: Firmly finalize requirements before starting. Any changes must go through a strict change control process.
    2. **Lack of Familiarity with Tools/Technologies**
        - **Risk**: Team members might struggle with React, Flask, or AWS/Azure.
        - **Mitigation**: Provide training sessions or resources. Consider pair programming or mentorship within the team.
    3. **Integration Difficulties**
        - **Risk**: Issues when integrating the React frontend with Flask backend.

   - **Mitigation**: Regular integration tests and having a clear API contract.
4. **Deployment Issues**
   - **Risk**: Challenges deploying to AWS or Azure, especially with scalability.
   - **Mitigation**: Start with a basic deployment setup, test thoroughly, and iteratively add complexity. Utilize cloud platform's documentation and support.
5. **Performance Bottlenecks**
   - **Risk**: The game might face lag or performance issues, especially with many players.
   - **Mitigation**: Performance testing early on and optimize based on results.
6. **Security Concerns**
   - **Risk**: Data breaches or other security vulnerabilities.
   - **Mitigation**: Implement best security practices, regular security audits, and using secure coding guidelines.
7. **Unrealistic Timeframes**
   - **Risk**: Project milestones may be set too optimistically, leading to rushed work, or missed deadlines.
   - **Mitigation**: Regularly review and adjust timelines. Add buffer periods for unexpected delays.
8. **Inadequate Testing**
   - **Risk**: Bugs or issues might slip through due to insufficient testing.
   - **Mitigation**: Allocate enough time for thorough testing phases, including user acceptance testing.
9. **Poor User Experience**
   - **Risk**: The game interface might not be intuitive or user-friendly.
   - **Mitigation**: Conduct user testing sessions, gather feedback, and iterate on the design.
10. **Team Communication Breakdown**
   - **Risk**: Misunderstandings or miscommunications among team members.
   - **Mitigation**: Regular team meetings, clear documentation, and open channels for communication.