Software Requirements Specification for ProgName: subtitle describing software

Team #25, The Crazy Four

Ruida Chen
Ammar Sharbat
Alvin Qian
Jiaming Li

30.09.2025

Contents

Revision History

Table 1: Revision History

Date	**Developer(s)**	**Change**
9.29	Jiaming Li	Purpose of the Project
9.30	Ruida Chen	Section3, 10 ,11, 12
9.30	Jiaming Li	Scope of the Product
9.30	Jiaming Li	FR
9.30	Alvin Qian	Section 4,6,7
10.1	Alvin Qian	Section 14,16
10.9	Ruida Chen	Section 20,21,22
10.9	Jiaming Li	Section 17,18,19
10.9	Alvin Qian	Appendix and Reflection

1 Purpose of the Project

1.1 User Business

The purpose of this project is to design and implement an educational card game based on the traditional *Crazy 8s* rule set, but adapted to integrate the **Dozenal (base-12) number system**.

- This project addresses the lack of accessible and engaging tools that introduce alternative number systems in a playful and intuitive way.
- By combining a familiar card game mechanic with Dozenal representations and operations, users can gradually build comfort and intuition with the base-12 system.
- The primary business value lies in providing a lightweight, fun, and interactive educational tool for students, hobbyists, and anyone interested in number systems beyond decimal.

Additionally, the game offers an opportunity to evaluate how gamification can support mathematical learning, and whether abstract concepts (such as base conversions or divisibility in Dozenal) can be effectively taught through play.

1.2 Goals of the Project

The goals of this project are:

- Educational Integration: Seamlessly incorporate Dozenal concepts (symbols 0–B, factorization, arithmetic) into the gameplay, ensuring that players learn by playing without requiring formal prior knowledge.
- Gameplay Design: Deliver a working digital version of *Crazy 8s* that is intuitive, responsive, and enjoyable, while maintaining the familiar flow of the original game and introducing Dozenal-specific mechanics (e.g., matching rules, scoring, or special cards).
- Accessibility and Engagement: Create a user-friendly interface that lowers the barrier to learning, accessible for casual users while offering depth for learners who want to explore Dozenal further.

- Scalability / Stretch Goals: Explore the potential for extending the system to other educational card or board games, and investigate how different number bases can be taught through similar game mechanics.

2 Stakeholders

2.1 Client

Insert your content here.

2.2 Customer

Insert your content here.

2.3 Other Stakeholders

Insert your content here.

2.4 Hands-On Users of the Project

Insert your content here.

2.5 Personas

Insert your content here.

2.6 Priorities Assigned to Users

Insert your content here.

2.7 User Participation

Insert your content here.

2.8 Maintenance Users and Service Technicians

Insert your content here.

3 Mandated Constraints

3.1 Solution Constraints

- All source code must be managed using Github, with frequent commits and version control.
- The system must be developed using React for frontend and Node.js for backend.

3.2 Implementation Environment of the Current System

- The backend must operate on Node.js (LTS version, e.g., 18+) with a relational database such as PostgreSQL or MySQL.
- The application must be deployable on modern web browsers (Chrome, Firefox, Edge)

3.3 Partner or Collaborative Applications

- The team must use GitHub for code collaboration and issue tracking.
- Discord are required for real-time communication and coordination.

3.4 Off-the-Shelf Software

- The system must rely primarily on open-source libraries.
- Proprietary software must not be required for end users to run the application.

3.5 Anticipated Workplace Environment

- End users are expected to interact with the application primarily on web browsers.

3.6 Schedule Constraints

- The project must be completed within the academic deadline.
- Progress reports and milestone check-ins with the TA are mandatory.

3.7 Budget Constraints

- Total financial budget is under 500 Canadian Dollars.
- Open source and free tools are primary to be used.

3.8 Enterprise Constraints

- The project must comply with academic integrity and university policies.
- Data collection and storage must follow privacy, security, and ethical guidelines.

4 Naming Conventions and Terminology

4.1 Glossary of All Terms, Including Acronyms, Used by Stakeholders involved in the Project

- **Base-10 (Decimal)**: The standard numerical system using ten digits (0–9), commonly used in arithmetic and everyday calculations.
- **Base-12 (Dozenal)**: A numerical system using twelve digits (0–9, ↊, ↋), where ↊ represents 10 and ↋ represents 11 in decimal. Has divisibility advantages (divisors: 2, 3, 4, 6).
- **Base-8 (Octal)**: A numerical system using eight digits (0–7). Each octal digit represents three binary digits (bits). Commonly used in computing and digital systems due to easy conversion with binary.
- **Base-16 (Hexadecimal)**: A numerical system using sixteen digits (0–9, A–F), where A through F represent decimal values 10 to 15. Widely used in programming and computer memory addressing for its compact binary representation.

- **Crazy Eights**: A classic card game where players match cards by suit or rank, with the "8" card acting as a wild card allowing the player to declare a new suit.
- **MVP**: Minimum Viable Product, the initial version of the Crazy Eights software with core functionality, including two-player gameplay, classic rules, and dozenal scoring.
- **Functional Goals**: Features and behaviors the software must implement, such as gameplay mechanics and dozenal score display.
- **Non-functional Goals**: Quality attributes of the software, such as usability, performance, and stability.
- **Stretch Goals**: Optional features or enhancements, such as multiplayer support or advanced dozenal rule variants, to be implemented if time permits.
- **GitHub**: The platform used for version control, issue tracking, and Kanban board management.
- **Discord**: The communication platform for team coordination, quick updates, and voice meetings.
- **Kanban Board**: A project management tool in GitHub Projects, divided into stages (Backlog, In Progress, Review, Done) to track tasks.
- **CI/CD**: Continuous Integration/Continuous Deployment, an automated process for testing and deploying code changes.
- **SRS**: Software Requirements Specification, this document outlining the requirements for the Crazy Eights project.
- **PoC**: Proof of Concept, a prototype demonstrating core gameplay mechanics to validate feasibility.
- **UI**: User Interface, the visual and interactive components of the software, such as the game board and score display.

5 Relevant Facts And Assumptions

5.1 Relevant Facts

Insert your content here.

5.2 Business Rules

Insert your content here.

5.3 Assumptions

Insert your content here.

6 The Scope of the Work

6.1 The Current Situation

The current numerical system used around the world is predominantly decimal (base-10). This system, while widely adopted, has limitations in representing fractions cleanly, as its prime factors (2 and 5) result in recurring decimals for simple ratios 1/3 or 1/6. On the other hand, the dozenal (base-12) system, with divisors 2, 3, 4, and 6, offers a more intuitive and concise fraction representations, historically used in trade and measurement systems (e.g., dozens, hours). However, dozenal is underutilized in education and practice, leaving students, educators, and professionals reliant on decimal despite its inefficiencies for certain calculations. There is no current fun and engaging way to demonstrate the practical advantages of dozenal, such as a card game like Crazy Eights, to promote its adoption.

6.2 The Context of the Work

The project focuses on the software implementation of the Crazy Eights card game that incorporates dozenal (base-12) scoring and display to highlight the benefits of the dozenal system. The context includes:

- **Educational Context**: The project will simplify mathematical understanding for the user by showing dozenal's advantages in a familiar and fun game format.

- **Technical Context**: The software will be developed using a modern tech stack for web applications (JavaScript, TypeScript, Node.js, React, PostgreSQL) and use GitHub CI/CD pipelines for version control and testing.
- **Stakeholder Context**: Key stakeholders include students, educators, mathematicians, computer scientists, and the general public, all of whom could benefit from clearer fraction representations and easier mental arithmetic.

6.3 Work Partitioning

The workload is divided into the following major phases, aligned with the development plan:

1. **Problem Definition and Planning (Weeks 3–4)**: Draft problem statement, development plan, and initial proof of concept (PoC) to establish scope and feasibility.
2. **Requirements and Hazard Analysis (Week 6)**: Develop the SRS and identify potential risks and mitigation strategies.
3. **Verification and Validation Planning (Week 8)**: Define testing strategies to ensure the software meets functional and non-functional requirements.
4. **System Design (Weeks 10–16)**: Create and refine architecture diagrams, decompose the system into modules (frontend, backend, database, API), and document extensibility.
5. **Implementation and Testing (Weeks 11–19)**: Develop the MVP (two-player game with classic rules and dozenal scoring), conduct unit and integration testing, and prepare for demonstrations.
6. **Demonstrations and Refinement (Weeks 19–24)**: Conduct Revision 0 and final demonstrations, incorporating feedback to improve functionality and performance.
7. **Final Documentation and EXPO (Week 26)**: Finalize documentation and present a polished product at the EXPO.

Each task is tracked via GitHub Issues and the Kanban board, with responsibilities assigned to team members based on rotating roles (developer, reviewer, meeting chair, note-taker).

6.4 Specifying a Business Use Case (BUC)

BUC: Play a Game of Crazy Eights with Dozenal Scoring

- **Actors**: Two players, the software system.
- **Trigger**: A player creates a new game session using the UI.
- **Description**: Two players take turns matching cards by suit or rank, with the "8" card acting as a wild card that allows the player to declare a new suit. If no valid move can be made, the player draws from the stock pile. The game ends when one player discards all their cards, and the score is calculated and displayed in dozenal notation.
- **Preconditions**: The user is logged in. The software is running on a web interface, with a functional UI displaying the hand, discard pile, stock pile, and score tracker.
- **Postconditions**: The game concludes with a winner, points are displayed in dozenal and must be counted by the users to see the final score. The user can log Off or start a new game.
- **Basic Flow**:
    1. The system deals cards to both players and initializes the discard pile with a starter card.
    2. Players take turns, selecting a card to play (matching suit or rank) or drawing from the stock pile.
    3. If an "8" is played, the player selects a new suit via the UI.
    4. The system checks moves and gives immediate feedback for invalid moves.
    5. The game keeps going until one player has no cards left, triggering user score counting and calculation challenge in dozenal.
    6. The system displays the final score.

- **Exceptions**:
  - Invalid move attempted: The system highlights the error and prompts the player to select a valid card or draw.
  - Stock pile exhausted: The system reshuffles the discard pile to replenish the stock pile.
- **Assumptions**: Players are familiar with basic card game mechanics and the system supports a stable internet connection for play.

7 Business Data Model and Data Dictionary

7.1 Business Data Model

These are the key entities and relationships involved in the Crazy Eights game with dozenal scoring.

- **Entities**:
  - **Player**: Represents a game participant, holding a hand of cards and a score.
  - **Card**: Represents a single playing card with a suit and rank.
  - **Deck**: A collection of cards, divided into the stock pile and discard pile.
  - **Game Session**: Tracks the state of a single game, including players, current turn, discard pile, and scores.
  - **Score**: Tracks points in dozenal notation, calculated based on game rules.
- **Relationships**:
  - A **Game Session** has 2 **Players** (MVP) or 2–4 **Players** (stretch goal).
  - Each **Player** has a **Hand** (subset of **Cards**).
  - A **Deck** consists of 52 **Cards**, split into **Stock Pile** and **Discard Pile**.

- A **Game Session** produces a **Score** for each **Player** in dozenal notation.
- A **Card** played in a **Game Session** affects the **Discard Pile** and may trigger a suit change (if an "8").

7.2 Data Dictionary

- **Player**:
  - **ID**: Unique identifier for a player (integer).
  - **Name**: Display name for the player (string).
  - **Hand**: List of cards held by the player (array of Card objects).
  - **Score**: Player's cumulative score in dozenal notation (string, such as "1Ɛ" for 22 in decimal).
- **Card**:
  - **Suit**: One of four suits (Hearts, Diamonds, Clubs, Spades) (string).
  - **Rank**: Card value (2–10, J, Q, K, A, 8) (string).
  - **IsWild**: Boolean indicating if the card is an "8" (true/false).
- **Deck**:
  - **Stock Pile**: List of cards available for drawing (array of Card objects).
  - **Discard Pile**: List of played cards (array of Card objects).
- **Game Session**:
  - **Session ID**: Unique identifier for the game session (string).
  - **Current Turn**: ID of the player whose turn it is (integer).
  - **Status**: Game state (Active, Completed) (string).
  - **Starter Card**: The first card in the discard pile (Card object).
  - **Last Played Suit**: The active suit after an "8" is played (string).
- **Score**:

- **Player ID**: Links to the player (integer).
- **Dozenal Value**: Score in base-12 notation (string, such as "14" for 16 in decimal).
- **Calculation Method**: Rules for scoring (sum of remaining cards in the opponents hands) (string).

8 The Scope of the Product

8.1 Product Boundary

The product to be developed is a digital card game application based on the traditional *Crazy 8s* rules, modified to integrate the **Dozenal (base-12) number system**. The system boundary includes:

- A game engine that supports core Crazy 8s mechanics (drawing, discarding, turn-taking, winning conditions).
- Adaptations of rules, card values, and scoring to incorporate Dozenal arithmetic and representations.
- A user interface allowing players to interact with the game (play cards, view scores, receive feedback).
- Educational prompts or visual aids to help players understand Dozenal concepts.

External systems not included in the boundary are: general learning platforms, multiplayer servers beyond basic peer-to-peer/local play, and integrations with unrelated educational tools.

8.2 Product Use Case Table

The following table summarizes the primary product use cases (PUCs):

PUC #	Description
PUC-1	Player logs into the system or creates a new account to enable personalized features and multiplayer access.
PUC-2	Player views their personal profile, including gameplay history, win/loss record, and achievements.
PUC-3	Player starts a new Crazy 8s game with Dozenal-enabled deck.
PUC-4	Player takes a turn by drawing or discarding a card.
PUC-5	System validates whether the played card is legal (same suit, same Dozenal value, or sum = 12).
PUC-6	Player views scores and progress, displayed in both decimal and Dozenal.
PUC-7	System provides hints or explanations to support Dozenal learning.
PUC-8	Game ends when a player wins; final scores are calculated and displayed.

8.3 Individual Product Use Cases (PUCs)

UC1: Player Login

1. Player opens the game application and selects "Login" or "Create Account".
2. The system prompts the player to enter credentials (username and password) or choose guest mode.
3. The system validates login information against the stored database.
4. If credentials are valid, the player is granted access to personalized data (game progress, scores, and settings).
5. If the player selects guest mode, a temporary profile is generated for the session.
6. The system displays the main menu or lobby after login.

UC2: See Player Profile

1. The player selects "View Profile" from the main menu.

2. The system retrieves player data (username, past game records, win/loss ratio, and achievements) from the database.
3. The system displays profile information on screen.
4. If the player is logged in as a guest, the system displays a message such as "Profile unavailable for guest mode".
5. The player may choose to return to the main menu or edit available settings (if logged in).

UC3: Start a New Game

1. Player opens the application and selects "New Game".
2. The system initializes a Dozenal-enabled deck (0–B, 10).
3. The system shuffles the deck.
4. The system deals cards to each player.
5. The game state is displayed on the interface.

UC4: Take a Turn

1. The system indicates that it is the player's turn.
2. The player chooses either to play a card or to draw from the deck.
3. If the player chooses a card, the system checks its validity against the discard pile.
4. If valid, the card is placed onto the discard pile.
5. If invalid, the system notifies the player and the card remains in the hand.
6. If the player chooses to draw, the system gives one card from the deck to the player.

UC5: Validate Move

1. The player selects a card to play.

2. The system retrieves the top card from the discard pile.
3. The system checks if the move is legal under Dozenal rules:
   (a) same suit, or
   (b) same Dozenal value, or
   (c) sum of the two values equals 12 (base-12).
4. If the move is legal, the system accepts the card and updates the discard pile.
5. If not, the system rejects the move and notifies the player.

UC6: View Scores

1. A round ends or the player requests to view scores.
2. The system calculates points for each player.
3. The system converts the scores into both decimal and Dozenal.
4. The system displays the results on screen.

UC7: Provide Hints

1. The player hovers over or selects a card.
2. The system analyzes the current game state.
3. The system provides a hint (e.g., "This card is valid because its sum with the top card equals 12 (base-12).").
4. The hint is displayed as text or a visual highlight.

UC8: End Game

1. A player discards their last card.
2. The system checks if the game-ending condition is satisfied.
3. If satisfied, the system declares the winner.
4. The system calculates final scores in both decimal and Dozenal.
5. The results are displayed on the final game screen.

9 Functional Requirements

9.1 Game Manager

1. **Start new game:** Game manager shall allow the player to start a new Crazy 8s game with a Dozenal-enabled deck. (FR-1)
   Rationale: Players need a way to initialize the game state; starting a new game is the foundation for all other gameplay functions.

2. **Turn management:** Game manager shall manage player turns, ensuring that each player either discards a valid card or draws a card. (FR-2)
   Rationale: Turn-taking enforces fairness and ensures game flow consistency.

3. **Rule validation:** Game manager shall validate that each played card is legal under Dozenal rules. A valid move is defined as either: (a) same suit as the previous card, (b) same Dozenal value as the previous card, or (c) the sum of the two card values equals 12 in Dozenal. (FR-3)
   Rationale: Prevents illegal moves, guarantees consistency, and introduces the core educational mechanic of Dozenal arithmetic.

4. **Special cards:** Game manager shall implement special card effects (e.g., 8s are wild) while supporting extensions with Dozenal-specific effects. (FR-4)
   Rationale: Special cards increase engagement and add flexibility in teaching Dozenal-based rules.

5. **End of game:** Game manager shall determine when the game ends (e.g., when a player runs out of cards) and declare the winner. (FR-5)
   Rationale: A clear end condition is required for meaningful gameplay and reinforcement of learning objectives.

9.2 Score Manager

1. **Calculate score:** Score manager shall calculate points for each round in both decimal and Dozenal. (FR-6)
   Rationale: Displaying scores in both systems reinforces learning by encouraging comparison between familiar decimal and new Dozenal formats.

2. **Display score:** Score manager shall display both decimal and Dozenal results on the user interface. (FR-7)
   Rationale: Visual feedback supports player understanding and helps users internalize Dozenal representations.

9.3 Education Support

1. **Hints:** System shall provide hints or explanations when a player performs an action involving Dozenal arithmetic. (FR-8)
   Rationale: On-demand guidance lowers the learning curve and supports players with varying levels of familiarity.

2. **Highlight valid moves:** System shall visually highlight valid moves based on Dozenal rules. (FR-9)
   Rationale: Reduces frustration, ensures players stay engaged, and reinforces Dozenal rules through visual learning.

9.4 Login Manager

1. **Account creation:** The system shall allow players to create a new account with a unique username and password. (FR-10)

2. **Login/Logout:** The system shall allow players to log in and log out at any time without losing progress. (FR-11)

3. **Guest mode:** The system shall allow players to start a temporary session without login. (FR-12)

4. **Credential validation:** The system shall validate player credentials against stored records before granting access. (FR-13)

5. **Rationale:** Ensures users can have persistent profiles for tracking progress and enables multiplayer authentication.

9.5 Data Manager

1. **Data storage:** The system shall securely store user data, including usernames, game history, and Dozenal scores. (FR-14)

2. **Data retrieval:** The system shall allow retrieval of stored user data when logging in or viewing profiles. (FR-15)

3. **Data update:** The system shall update stored data after each game or when profile information changes. (FR-16)

4. **Data deletion:** The system shall allow players to delete their data permanently upon request. (FR-17)

5. **Rationale:** Supports persistent user experience, analytics, and compliance with data privacy expectations.

10 Look and Feel Requirements

10.1 Appearance Requirements

- **Layout**: The user interface must have a clean and minimalistic layout, ensuring readability and ease of navigation.

- **Responsiveness**: The system must be responsive, adapting automatically to different screen sizes.

- **Visual Hierarchy**: Key elements such as active player indicators, playable cards, and turn timers must be visually distinguishable using contrast, color, or animation.

- **Color Palette**: The color scheme should promote clarity and engagement.

10.2 Style Requirements

- **Consistency**: The application must follow a unified design language, maintaining consistent typography, iconography, and button styles across all screens.

- **Animations**: Smooth 2-D animations should be implemented for card movements, dealing, and discarding actions. These animations should be realistic but not distracting, maintaining short transition times. When a wild card is played, a distinct animation and color shift should visually emphasize its effect while maintaining overall style consistency.

- **In-Game Hints**: Hints should appear contextually (e.g., glowing borders, pulsing icons) rather than as intrusive pop-ups. They should guide the player without interrupting the flow of the game.

11 Usability and Humanity Requirements

11.1 Ease of Use Requirements

- **Simplicity**: The interface must minimize the number of steps needed to perform core tasks.
- Data collection and storage must follow privacy, security, and ethical guidelines.

11.2 Personalization and Internationalization Requirements

- **Language Options**: The system must support at least English and one additional language for international users.

11.3 Learning Requirements

- **Onboarding**A short interactive tutorial must explain the rules of the game and how counting in different bases works.

11.4 Understandability and Politeness Requirements

- **Friendly Wording**: Prompts like "Invalid Move" must be displayed as polite guidance.
- **Clear Instructions**: Rules and base-conversion explanations must be phrased in simple and non-technical terms.

11.5 Accessibility Requirements

- **Keyboard Support**: All main actions must be accessible via keyboard shortcuts

12 Performance Requirements

12.1 Speed and Latency Requirements

- **Low latency**: Game actions must update across all players' screens within 300 ms.
- **Fast loading**: The game lobby and first match must load within 5 seconds on a stable internet connection.

12.2 Safety-Critical Requirements

- **Cheat Prevention**: The system must prevent unauthorized manipulation (e.g., directly altering game state through client-side tools).
- **Data Integrity**: No game state (e.g. deck composition, player hand) should be lost or corrupted due to refresh or reconnect.

12.3 Precision or Accuracy Requirements

- **Card Rules**: Card-matching and counting rules must be enforced with 100% accuracy according to the chosen numeral base.

12.4 Robustness or Fault-Tolerance Requirements

- **Reconnection**: If a player disconnects, they must be able to rejoin within 30 seconds without losing progress.

12.5 Capacity Requirements

- **Concurrent Players**: The system must support at least 200 concurrent users during testing.

12.6 Scalability or Extensibility Requirements

- **Game Modes**: The system must allow for adding new rule variations (e.g., different numeral bases or wild card effects) with minimal code changes.

- **Server Scaling**: The backend must be deployable in a scalable environment (e.g., Docker, cloud hosting) to handle larger user bases.

12.7 Longevity Requirements

- **Maintainability**: The codebase must be modular and documented, so future developers can update rules easily.

13 Operational and Environmental Requirements

13.1 Expected Physical Environment

Insert your content here.

13.2 Wider Environment Requirements

Insert your content here.

13.3 Requirements for Interfacing with Adjacent Systems

Insert your content here.

13.4 Productization Requirements

Insert your content here.

13.5 Release Requirements

Insert your content here.

14 Maintainability and Support Requirements

14.1 Maintenance Requirements

- **Code Maintainability**: The codebase must be modular, with clear separation of concerns to facilitate updates and debugging. Inline comments for non-obvious logic and updated README/module documentation are required, as specified in the development plan.
- **Version Control**: All changes must be tracked via GitHub, with descriptive commit messages and pull requests requiring at least one peer review to ensure maintainability.
- **Extensibility**: The system must support adding new features without requiring significant refactoring. This is achieved by using a modular architecture and Object oriented Design principles.
- **Documentation Updates**: Any change in behavior, interfaces, or setup instructions must be reflected in the documentation to ensure future developers can maintain the system. Local deployment instructions documented in the README to simplify setup for developers and maintainers.

14.2 Supportability Requirements

- **Error Reporting**: The system must provide clear, user-friendly feedback for invalid moves. Logs or replays of game sessions must be available for debugging, as specified in the non-functional goals.
- **User Support**: A tutorial or visual guidance (stretch goal) must be provided to assist new players in understanding gameplay and dozenal scoring. This reduces the need for extensive manual support.
- **Cross-Platform Support**: The software must run on different web environments (Windows, macOS, or browsers).

14.3 Adaptability Requirements

- **Rule Flexibility**: The system must support toggling house rules (e.g., draw-until-playable, stacking eights) via a rule configurator (stretch

goal), allowing adaptation to different play styles without code changes.

- **Scalability for Players**: The architecture must accommodate extending from two-player (MVP) to 3–4 player games (stretch goal) by modifying session management and turn logic.
- **Numeric System Extensibility**: The scoring system must allow switching between dozenal and decimal displays, ensuring adaptability for users unfamiliar with base-12.

15 Security Requirements

15.1 Access Requirements

Insert your content here.

15.2 Integrity Requirements

Insert your content here.

15.3 Privacy Requirements

Insert your content here.

15.4 Audit Requirements

Insert your content here.

15.5 Immunity Requirements

Insert your content here.

16 Cultural Requirements

16.1 Cultural Requirements

- **Numeric System Accessibility**: The software must present dozenal (base-12) in an intuitive, non-disruptive way. This includes clear UI

elements for dozenal scores (using ↊ and ↋ for 10 and 11) and optional tutorials (stretch goal) to explain dozenal notation to users unfamiliar with it.

- **Inclusivity**: The system must avoid cultural biases in its design, ensuring that gameplay and terminology (suits, ranks) are universally recognizable across cultures familiar with standard playing cards. No culturally specific references or imagery should be used in the UI to maintain broad accessibility.
- **Educational Alignment**: The software must align with educational goals by demonstrating the practical benefits of dozenal in a game context, making it appealing to students and educators. This supports the cultural shift toward exploring alternative numeric systems, as advocated by stakeholders like dozenal enthusiasts.

17 Compliance Requirements

17.1 Legal Requirements

The game must comply with intellectual property laws and digital content distribution regulations. All assets (images, fonts, and sounds) used in the game must be open-source, royalty-free, or properly licensed.

17.2 Standards Compliance Requirements

The software shall follow standard software development practices such as version control (Git), documentation standards (LaTeX-based SRS), and accessibility guidelines (WCAG 2.1 Level AA). The codebase should be compliant with modern C++/Python style conventions (PEP8 or equivalent).

18 Open Issues

- Whether multiplayer mode will be implemented in the current phase or postponed to a future release.
- The extent of educational feedback (how deeply to explain Dozenal math).

- Balancing game difficulty to maintain both fun and learning value.
- Need to confirm visual design theme (educational vs. playful aesthetic).

19 Off-the-Shelf Solutions

19.1 Ready-Made Products

Existing Crazy 8s card games (e.g., UNO or digital clones) offer partial gameplay models, but none integrate an educational Dozenal arithmetic component. No direct off-the-shelf solution currently fulfills both entertainment and mathematical learning objectives.

19.2 Reusable Components

- Card rendering engines from open-source projects such as *Pygame* or *Godot* assets.
- Math libraries for number-base conversion (decimal $\leftrightarrow$ Dozenal).
- UI component libraries for game menus, card animations, and scoreboards.

19.3 Products That Can Be Copied

Some structural aspects (turn management, shuffling algorithms) can be adapted from existing open-source card game repositories, provided licensing terms (e.g., MIT, GPL) are respected. However, educational Dozenal mechanics must be developed in-house to align with project-specific learning goals.

20 New Problems

20.1 Effects on the Current Environment

- The system is a newly devleoped web-based game and does not replace or interfere with any existing enviroment, minimal impact is expected aside from standard hosting and network usage.

20.2 Effects on the Installed Systems

- No existing software systems are modified or replaced by this product. The game operates independently on standard browsers and web servers.

20.3 Potential User Problems

- Users may experience confusion if base-conversion rules are misunderstood. These issues will be mitigated through clear feedback, tutorials, and consistent UI design.

20.4 Limitations in the Anticipated Implementation Environment That May Inhibit the New Product

- The main limitation is dependency on stable internet connectivity. Poor network conditions may degrade real-time synchronization or cause delayed responses.

20.5 Follow-Up Problems

- As the system evolves, maintaining compatibility with browser updates and third-party libraries (e.g., React, Socket.io) may introduce future maintenance challenges.

21 Tasks

21.1 Project Planning

- The project life cycle will follow the deliverable outline for SFWRENG 4G06: Capstone. By April the project must be completed in its entirety.

21.2 Planning of the Development Phases

- Requirements and Design Phase - Gather functional and non-functional requirements, identifying key components.

- Prototyping Phase - Develop an early functional prototype focusing on core interactions and user interface flow.
- MVP Phase - Implement the main gameplay logic, user authentication, and basic scoring system as our MVP.
- Finalize Phase - Polish the user interface, improve performance, deploy the final version to the hosting environment and prepare presentation.

22 Migration to the New Product

Not applicable.

23 Costs

23.1 Assumptions

- Academic, non-commercial deployment targeting a small user group during the term.
- Free tiers are acceptable for hosting and CI where reliability is sufficient for demos.
- Budget ceiling: 500 CAD (see Section "Budget Constraints").

23.2 One-time Costs

Table 2: One-time Purchases (CAD)

Item	**Purpose**	**Unit**	**Total**	
Domain name (1 year)	Simple memorable URL for demos	1	20	
			Subtotal	**20**

23.3 Recurring Costs

- App hosting utilize free tier where possible
- DB hosting and Free tier Postgres, small instance
- Domain name renweal cost could be recurring if we plan on hosting it longer term

23.4 Total and Headroom

Estimated total spend for the term: **100 CAD**. This stays within the 500 CAD cap and leaves headroom for risk mitigation or stretch features.

23.5 Cost Control Plan

- Prefer free tiers for hosting, CI, and CDN. Upgrade only if load or reliability requires it.
- Reuse open-source UI kits and icon packs with permissive licenses.
- Cap incentives to pre-approved number of sessions.

24 User Documentation and Training

24.1 User Documentation Requirements

- **In-app help**: Contextual tooltips for controls, hover help for Dozenal terms, and an always-available "How to play" panel.
- **Quick Start**: Setup, start a match, play a turn, view scores. Written for first-time users.
- **Rules and Dozenal guide**: Clear rules of Crazy Eights plus a simple Dozenal primer with examples for A and B, and conversions between base 10 and base 12.
- **FAQ**: Troubleshooting (cannot play a card, reconnect flow, what counts as a valid move).

24.2 Training Requirements

- **Onboarding tutorial**: 3 to 5 steps covering play a card, draw a card, wild eight, and reading Dozenal scores. Skippable and repeatable.
- **Practice mode**: Solo mode against a simple bot to learn pace and rules without pressure.

25 Waiting Room

Insert your content here.

26 Ideas for Solution

26.1 Architecture Overview

- **Frontend**: React with TypeScript. State managed via a small store (Zustand or Redux Toolkit). WebSocket client for live turns. Component library and keyboard-first interactions.
- **Backend**: Node.js with TypeScript. HTTP API for auth and lobby. WebSocket for game sessions and events. Server-authoritative game state.
- **Data**: Postgres for user profiles, match history, and telemetry. In-memory room state with periodic snapshots to DB.
- **Game engine**: Pure TypeScript module with deterministic rules, validators, scoring, and base conversion utilities. No UI concerns.
- **CI/CD**: GitHub Actions for lint, tests, type-checks, and deploy to a free-tier host.

26.2 Implementation Options

Option	Summary, Pros, Cons
A. Local-first 2P hot-seat	*Summary*: Single browser, two players, no backend. *Pros*: Fast MVP, minimal infra. *Cons*: No remote play, no persistence.
B. Server-authoritative WebSocket	*Summary*: Lobby, rooms, turns via WS. *Pros*: Fair play, replay logs, reconnect support. *Cons*: More code and testing.

26.3 Recommended Path

1. Build the game engine as a pure library with full tests.
2. Ship Option A for fast validation and UI polish.
3. Upgrade to Option B for sessions, persistence, and reconnection.

26.4 Key Non-functional Techniques

- **Fairness**: Server-side validation for all moves. Shuffle with seeded RNG and record seed for audits.
- **Resilience**: Rejoin token and state rehydrate on reconnect.
- **Accessibility**: Semantic HTML, focus order, ARIA labels, and keyboard shortcuts for all core actions.

Appendix — Reflection

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

    - One thing that went well while writing the SRS was our team's ability to clearly define functional and non-functional requirements based on our project scope. We divided the document sections efficiently and maintained consistent formatting and terminology. Our discussions on user needs and system goals also helped refine the main features early, making later sections like use cases and functional requirements much easier to complete.

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. How many of your requirements were inspired by speaking to your client(s) or their proxies (e.g. your peers, stakeholders, potential users)?

    - Most of our system requirements were influenced by feedback from peers and proxy users who represent our target audience—students learning numeral base conversions, additional requirements, such as real-time multiplayer functionality, clear in-game hints, and visual base conversion indicators are added.

4. Which of the courses you have taken, or are currently taking, will help your team to be successful with your capstone project.

- SFWRENG 3A04: Software Design III - Large System Design (Git, designing software architecture, UML diagrams)
- SFWRENG 4HC3: Human Computer Interfaces (User-Centered Design, Usability Testing)
- SFWRENG 3RA3: Software Requirements (Git, Github Issues, Writing SRS, Requirements Elicitation)
- SFWRENG 3S03: Software Testing (Test Case Design, Automated Testing Frameworks)
- SFWRENG 4C03: Computer Networks and Security (Network Protocols, Security Best Practices)
- SFWRENG 2AA4: Software Design I (Git, Kanban Board, Object-Oriented Design, Design Patterns)

5. What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.

    - Ruida: Frontend Development: Deepening our understanding of React component architecture, animation design, and state management.
    - Alvin: Backend Development: Gaining hands on experience in Node.js, Express, and database management.

6. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

    - Ruida: Frontend Development (React & Animation) Approaches: (1) Following online React tutorials and official documentation; (2) Experimenting through prototype iterations and peer code reviews. I will choose the first approach - follow online react tutorials, since the official tutorial documentation of react is really

straightfoward and easy to learn, lots of code examples are provided.

- Alvin: Backend Development (Node.js, Express, Database Management) Approaches: (1) Completing online courses and tutorials on Node.js and Express (2) Reviewing course notes and project repositories from past relevant courses.