# Software Requirements Specification for Focus Enhancement Tool

**Team 31**
**Title: Focus Enhancement Tool(Growin)**
**Chengyuan Wen (wenc15)**
**Qinquan Wang (wangq198)**
**Chiyu Huang (huanc10)**
**Jingyao Sun (sun250)**
**Zhecheng Xu (xu562)**
**Zikai Lu (luz98)**

## 1. Versions, Roles and Contributions

### 1.1 Version：

Version 0（Created SRS and Project Development Plan）

### 1.2 Table of Contributions

| Group Member | Contributions (Sections) |
|---|---|
| Chengyuan Wen | 11.13 |
| Qinquan Wang | 1.2.3.4.9 |
| Chiyu Huang | 6 |
| Jingyao Sun | 10.12 |
| Zhecheng Xu | 7.14 |
| Zikai Lu | 5, 8 |

## 2.List of Abbreviations, Notations, Naming Conventions, and Definitions

### 2.1 Abbreviations

- **PoC** – Proof of Concept: a prototype demonstrating feasibility.
- **UI** – User Interface: visual components that allow user interaction.
- **UX** – User Experience: the overall usability and satisfaction perceived by the user.
- **DB** – Database: SQLite storage used to save focus-session data and logs.
- **API** – Application Programming Interface: communication interface between frontend and backend.
- **CI/CD** – Continuous Integration and Continuous Deployment: automatic build and test pipeline via GitHub Actions.
- **MVP** – Minimum Viable Product: earliest version with core functions such as timer and whitelist.
- **WPF** – Windows Presentation Foundation: C# framework for Windows GUI integration.
- **EF Core** – Entity Framework Core: ORM library for managing database operations.
- **WS** – WebSocket: real-time, bidirectional communication protocol.
- **QA** – Quality Assurance: testing process to ensure reliability.

### 2.2 Notations

- **≤ / ≥ / ≈** – Mathematical comparisons used in requirement statements.
- Represents a milestone on the project schedule.
- **<feature-name>** – Placeholder for Git branch names (e.g., feat/gacha-system).
- **CamelCase / snake_case / PascalCase** – Variable and class naming styles used in source code.

### 2.3 Naming Conventions

- **Variables:** use camelCase (e.g., focusTimer, sessionCount).
- **Classes:** use PascalCase (e.g., FocusSessionManager, RewardEngine).
- **Constants:** use UPPER_CASE_WITH_UNDERSCORE (e.g., MAX_SESSION_TIME).
- **Database Tables:** use snake_case (e.g., user_profile, focus_history).
- **File Names:** lowercase + underscore (e.g., focus_timer.cpp, reward_system.cs).
- **Git Branches:** feat/<feature>, fix/<issue>, docs/<update> (e.g., feat/gacha-system).
- **Commit Messages:** follow *Conventional Commits* style (e.g., feat: add gacha reward logic).
- **UI Component IDs:** prefix with the page name (e.g., dashboard_timerLabel, profile_saveButton).

### 2.4 Definitions

- **Focus Session:** A timed period during which non-whitelisted apps are blocked to improve concentration.
- **Whitelist / whitelist:** Lists of allowed or blocked apps and URLs for focus control.
- **Dashboard:** Main interface showing focus statistics and session history.
- **Token:** Virtual currency earned after each focus session to use in the Gacha system.
- **Gacha System:** Gamified reward feature where users draw random collectibles.
- **Achievement:** Reward for reaching defined goals (e.g., a 7-day streak).
- **Chrome Extension:** Browser module that sends activity data to the desktop app via WebSocket.
- **Local Storage:** SQLite database storing logs, tokens and statistics locally.
- **Focus Mode:** Active state in which the system blocks distractions and tracks progress.
- **Reward Mechanism:** System that connects focus performance with in-app rewards.
- **Frontend:** User-facing UI implemented in React / Electron for Windows desktop.
- **Backend:** C# (.NET 8) logic layer handling data access and communication.
- **Our project**：The project of COMPSCI 4ZP6 Team 31. The Focus Enhancement Tool
- **Our team**：The team 31 of COMPSCI 4ZP6
- **Growin:** The nickname of our project.

## 3. The Purpose of the Project

Modern individuals face increasing difficulty maintaining focus in digital environments filled with distractions and fragmented content consumption.
Focus Enhancement Tool is a PC-based application that trains and reinforces users' ability to sustain attention across all computer activities—including study, work, and entertainment.

The system integrates custom focus modes, distraction blocking, and a gamified reward mechanism that grants collectible cards as positive reinforcement for successful focus sessions.
Unlike traditional Pomodoro-style tools,  it embraces inclusive focus scenarios such as gaming or watching shows, transforming them into structured and mindful experiences.
Objectives:

- Encourage users to develop habitual, deliberate focus through consistent practice.
- Reduce stress and anxiety caused by distraction and inefficient task-switching.
- Demonstrate how gamification and positive feedback can sustain mental discipline in daily digital activities
  focus enhancement tool

**4. The Client, and other Stakeholders**

**Client:**

- **Course Instructor**（**Mehdi Moradi**）**and Teaching Assistant (Main TA**：***Amirhossein Sabour*****)** – They act as the official clients of this project. They define the project requirements, provide feedback during development, and evaluate the final deliverable.

**Stakeholders:**

- **Project Team Members**（**Team 31**）– Responsible for planning, designing, implementing, and testing the software. They directly influence the project's quality and success.
- **End Users** – The individuals who will use or interact with the developed system. Their needs and feedback help guide the system's design and functionality.
- **Course Administration / Department** – Oversees the capstone program and ensures that the project meets academic standards. The results may be used as a reference for future student projects.

**5 Project Constraints**

**5.1 Platform Scope & Deployment**

- PC-only (Windows desktop) for Phase 0. Work/study focus typically occurs on PCs. Mobile/tablet/macOS/Linux are out of scope for now.
- Local desktop app with optional browser integration for tab/activity signals. Core focus timing works offline; any sync is optional.

**5.2 Functional Boundaries**

- Assistive tool, not a lockdown system. The app supports self-regulation (timers, whitelists, soft blocks). Complete prevention of distraction is not a goal.
- Whitelist model by design. Improves usability but allows bypass if users whitelist distracting apps/sites. We will make this trade-off explicit in onboarding.

- Single user, single machine. No enterprise admin/MDM, no parental/school controls in Phase 0.

### 5.3 Technical Constraints & Choices

- Use only documented, user-consented OS/browser APIs. No kernel drivers, no intrusive hooks. Active-window/app and URL detection accuracy depends on OS/browser capabilities.
- Resource-light requirement. Low CPU/RAM, so the app doesn't become a distraction itself.
- Local-first storage. Session logs, focus stats, and rewards are kept locally; future cloud sync (if any) must be opt-in and conflict-aware.
- No content capture. We store app/site names and timestamps as needed; no page contents, no keystrokes, no screenshots.

### 5.4 Usability & Behavior Constraints

- User intent is required. The tool assumes the user wants to focus; we provide nudges and friction rather than hard enforcement. No matter how we design the tool, the user will always have thousands of ways to bypass the restrictions if they want.
- Low-friction controls. Quick start/stop; short grace period to recover from false positives (temporarily using a non-whitelisted app).
- Accessibility. Readable defaults, non-disruptive notifications.

### 5.5 Privacy, Ethics, and Compliance

- Minimum necessary data. Only what's required for focus timing and basic analytics.
- Transparency & control. Users can view/export/delete local data. Any cloud features are opt-in with clear settings.
- No stealth operation. User-installed, visible, and user-controlled at all times.

### 5.6 Out of Scope (Phase 0)

- Mobile, macOS, and Linux builds may be considered in the future.
- Enterprise/MDM controls; parental/school enforcement.
- Second-device/VPN/proxy detection and other anti-circumvention tech.
- Deep behavioral analytics beyond basic usage/focus statistics.
- Online/social systems (accounts, friends, rankings, cloud leaderboards) will be considered in the future.

### 5.7 Roadmap-Aware Constraints (After phase 0)

If/when we add more mini-games, achievements, friends/rankings, or mini tools:

- We'll require account & sync (optional), implying auth, data retention policies, and moderation.

- Leaderboards/friends introduce fair-play/anti-cheat constraints, rate-limits, and abuse prevention.
- Mini-games must remain opt-in and non-distracting during focus sessions (e.g., only accessible in break time).
- Additional legal/privacy reviews (e.g., age gating, ToS/Privacy Policy updates) will be needed before enabling social features.

## 6. Functional Requirements

This section specifies the functional requirements of **Growin**. The goal of the system is to help users regulate their use of entertainment and social media applications and instead allocate dedicated time to study or pre-planned productive activities. Each requirement is assigned a priority level (P0–P4), where **P0** indicates essential core functionality and **P4** indicates additional features that enhance user engagement but are not critical for the minimum viable product (MVP). Both frontend and backend aspects are described, and the data requirements for each feature are explicitly stated.

### 6.1 Data Requirements Overview

- **User input data**: focus duration, profile details (signature), friend requests.
- **System data**: application whitelist, focus session logs, user statistics.
- **Generated data**: gacha tokens (earned through focus completion), collectible cards, achievements triggered by focus history.

### 6.2 Functional Requirements Table

| Feature Name | Priority | Frontend / Backend | Data Requirements | Description |
|---|---|---|---|---|
| **Core Timer & App Whitelist** | P0 | **Frontend**: user sets focus duration; **Backend**: enforces timer logic and manages blocked apps | User-defined focus duration; list of installed apps selected for whitelisting | Users can configure a focus session with a specific duration. During this session, applications included in the Whitelist are locked. After the timer ends, blocked apps are released automatically. |

| | | | | |
|---|---|---|---|---|
| **Gacha (Card Draw) System** | P1 | **Frontend**: gacha interface and animations; **Backend**: token generation, draw logic, result storage | Tokens earned after completed focus sessions, card pool data | Upon completing a focus session, users earn tokens that can be used to draw collectible character cards. This feature adds gamification and long-term engagement. |
| **Achievements & Collection System** | P2 | **Frontend**: achievement panels and card gallery; **Backend**: check conditions and update progress | Focus history (total focus time, consecutive days), card collection data | Users unlock achievements by meeting predefined conditions (e.g., total focused hours, consecutive days of use). Collected cards and unlocked achievements are displayed in a gallery. |
| **User Profile & Dashboard** | P3 | **Frontend**: display user information; **Backend**: store and update profile data | User account details (UID, username, signature), cumulative focus time | Provides a personal dashboard showing username, unique ID, customizable signature, and total focus time statistics. |

| | | | | |
|---|---|---|---|---|
| **Friend & Social Ranking System** | P4 | **Frontend**: friend management and leaderboard view; **Backend**: manage friend relationships and ranking logic | User IDs, friend connections, individual focus time records | Enables users to add friends and view a leaderboard comparing focus times, fostering social motivation and competition. |

### 6.3 Frontend and Backend Responsibilities

- **Frontend**: provides the user-facing interface including the timer, whitelist settings, dashboard, friend list and ranking display, gacha draw screen, and achievement gallery.
- **Backend**: ensures correct execution of timer and blocking logic, handles user data storage and synchronization, processes friend relationships, manages gacha draw algorithms and probabilities, and evaluates achievement conditions.

### 6.4 Priority and Development Phases

The functional requirements are aligned with incremental development phases:

- **P0 (Core Timer & whitelist)** represents the fundamental logic without which the application cannot fulfill its purpose. This forms the MVP.
- **P1 (Achievements & Collections)** provides extended engagement and visual appeal through additional design-heavy elements.
- **P2 (Gacha System)** introduces gamification for motivation.
- **P3 (User Profile & Dashboard)** introduces personalization and progress tracking, enhancing usability.
- **P4 (Friend & Social Ranking)** adds community features to encourage long-term usage.

In practice, the implementation sequence will follow the priority ranking from P0 to P4, ensuring that essential functionality is delivered first while leaving space for enhanced features in later iterations.

## 7. Non-functional requirements

### 7.1 Operational and Environmental Requirements

- Windows 10/11 (64-bit) desktop app; Chrome v120+ for the extension.
- Baseline hardware: ≥ 2 CPU cores, 8 GB RAM, SSD, 1366×768 display.
- Core timing, whitelist checks, and local logging work without the Internet.

- User-level install (no admin). Updates may prompt; uninstall keeps data unless “Remove data” is chosen.

### 7.2 Usability and Humanity Requirements

- First-run: user can set a whitelist and start a session in ≤ 5 minutes and ≤ 10 actions.
- Non-whitelisted activity shows a visible countdown (≤ 60 s) before stopping.
- Presets provided: Pomodoro (25/5), HIIT (15/2), Ultradian (90/20); custom durations persist.
- Core screens support keyboard navigation with clear focus states and labels.

### 7.3 Performance Requirements

- Cold start ≤ 2.0 s on an SSD / 8 GB RAM laptop.
- Foreground/whitelist checks run ≥ 4 Hz with average CPU < 5% when idle.
- Local database writes p95 ≤ 20 ms; UI remains responsive.
- Localhost WebSocket p95 latency ≤ 50 ms; auto-reconnect ≤ 3 s.
- Memory budget: app ≤ 300 MB; extension adds ≤ 50 MB.

### 7.4 Maintainability and Support Requirements

- Documented module layout and naming; target cyclomatic complexity ≤ 10 per function.
- Core logic unit tests achieve ≥ 70% coverage; CI runs on every commit/PR.
- No hard-coded paths or secrets; logs rotate with a size cap; config is externalized.

### 7.5 Security Requirements

- Minimal data stored: app names/domains, timestamps, durations, mode IDs (no page titles or contents by default).
- Local IPC over 127.0.0.1 WebSocket with origin checks and a random per-boot token.
- Data at rest uses an encrypted DB or a user-protected directory; no secrets in source.
- Extension requests only required permissions and supports “on-click” site access.
- Users can view, export, and delete records; default retention ≤ 12 months.

## 8. Risks & Issues Predicted

### Over-blocking

- *Cause:* Coarse rules; limited context of allowed tools/sites.
- *Solution:* A more detailed whitelist for users to manage
- *Quick Measures:* Safe defaults; short grace period; per-mode whitelists.

**Privacy concern**

- *Cause:* Misunderstanding of data captured/stored.
- *Solution:* Local-first storage; no content/keystrokes; clear privacy page; data viewer/export/delete.

**Performance overhead**

- *Cause:* Monitoring loops or heavy polling.
- *Signs:* High CPU/RAM, UI lag.
- *Solution:* Profiling; debounce/poll intervals; lightweight window/URL checks.
- *Quick Measures:* Low-power mode; toggles to disable heavy checks.

**Accessibility gaps**

- *Cause:* Missing keyboard nav, low contrast, poor semantics.
- *Solution:* A11y checklist; focus rings; ARIA; font/contrast controls; ≥44×44px targets.

**Legal/regulatory ambiguity**

- *Cause:* Perception of “monitoring” in workplaces; regional rules.
- *Signs:* User/employer inquiries; store rejections.
- *Solution:* “Personal-use only” positioning; clear ToS/Privacy Policy.

**Cheating on rankings**

- *Cause:* Spoofed stats/tampering.
- *Signs:* Implausible scores; user reports.
- *Solution:* Server-side validation; signed events; rate limits.
- *Quick Measures:* Audit/reset tools; anti-cheat reviews.

**Social abuse & moderation load**

- *Cause:* Friends/leaderboards misuse.
- *Signs:* Spam/inappropriate names; abuse reports.
- *Solution:* Report/block; profanity filters; no messaging function.
- *Quick Measures:* Human moderation queue; clear policies.

**Cloud/account compliance**

- *Cause:* GDPR/age-gating/retention requirements.
- *Signs:* Legal tickets; app-store delays.
- *Solution:* Data minimization; DSR endpoints; age gate; regional toggles.
- *Quick Measures:* ToS/PP updates; legal sign-off.

**Infra cost & reliability**

- *Cause:* Leaderboards/real-time services.
- *Early signs:* Latency/cost spikes; throttling.
- *Solution:* Simple periodic leaderboards; caching; quotas.
- *Quick Measures:* Graceful degradation; kill-switch.

**9. Team Meeting and Communication Plan**

1. Meeting Schedule
   - Weekly team meeting on Discord.Monthly meeting with TA on Microsoft Teams.
   - Extra ad hoc meetings before major submissions (e.g., SRS, Proof-of-Concept).Members record notes and uploads to Google Drive.
2. Communication Tools
   - Discord: daily discussions and coordination.
   - Microsoft Teams: formal communication with the instructor and TAs.
   - Email: official announcements and scheduling.
3. Document & Code Sharing
   - Google Docs for reports and documentation.All docs auto-saved in Google Drive and synced for offline access.
   - GitHub repository:Each member works on their own branch.Pull requests reviewed before merging.
4. Project Management
   - Jira (Kanban) board with “To Do / In Progress / Meeting / Done”.And tasks prioritized with P0–P4 ranking.Progress reviewed in weekly meetings.

**10. Team Member Roles**

Each member is assigned specific functional modules and/or non-functional quality attributes based on their expertise. Collaboration occurs primarily during system integration and testing.

| Name | Responsibilities |
|---|---|
| Chengyuan Wen | Project manager and lead developer for **Core Timer** and **whitelist**, designing session logic and integrating the background monitoring system. Supervises architecture consistency across all phases.<br>Ensures system reliability, code maintainability, and compliance with project documentation standards. |
| Jingyao Sun | Responsible for **frontend interface design**, including User Profile and Dashboard, onboarding tutorial, and layout of focus and analytics screens.<br>Ensures usability, interface consistency, and accessibility compliance. |

| | |
|---|---|
| Zikai Lu | Backend engineer for **data storage**, session logs, and analytics dashboard implementation.<br>Ensures data consistency, local database efficiency, and secure handling of user information. |
| Chiyu Huang | Develops **Gacha System**, integrating reward algorithms and randomized drop tables.<br>Responsible for performance optimization, responsiveness, and cross-module testing. |
| Zhecheng Xu | Responsible for **testing and quality assurance**, including unit testing, integration testing, and user acceptance testing of all core functions.<br>Oversees reliability verification, bug tracking, and regression test documentation. |
| Qinquan Wang | Handles **Achievements & Collections**, coordinating animation, asset integration, and overall UI polish.<br>Manages version control discipline and knowledge transfer readiness. |

**11. Workflow Plan**

**a. General github workflow:**
Daily workflow summary
1. Pull latest main
2. Create feature branch
3. Commit and push changes
4. Open PR → review → merge
5. Sync local main and delete old branch

**b) Agile Methodology**
We follow a 2-week Scrum cycle using GitHub Projects.
Each sprint includes planning, development, demo, and retrospective.
Team members work on separate feature branches instead of the main branch to avoid conflicts and keep the codebase stable.
Tasks are tracked on a Kanban board (To-Do → In-Progress → Done).
This method ensures organized collaboration, stable integration, and continuous progress tracking.

**c) Data Storage**
- Local SQLite database for app data and logs.
- ML models stored locally (/models) or tracked via Git LFS/DVC.
- Optional encrypted cloud backup for user data.

**d) Compute-Heavy Tasks**
- Run locally on developer machines (NVIDIA GPU if available).

- Heavy ML tasks: Google Colab or AWS EC2 GPU instances.
- Inference runs locally or via cloud API fallback.

**e) Tools & Methods for Performance Targets**
- CI/CD: GitHub Actions for build, test, and code quality.
- Testing: xUnit (backend), Vitest/Playwright (frontend).
- Performance: Async I/O, caching, optimized DB queries.
- Metrics: UI latency ≤150 ms, focus accuracy >99%.
- Security: Localhost-only WebSocket, optional encryption for sync.

## 12. Proof of Concept Demonstration Plan

The purpose of the proof of concept demonstration (PoC) is to verify that the core real-time timer and whitelist/whitelist system (P0) can operate reliably and interact smoothly between frontend and backend components. This phase addresses the primary technical risk: maintaining responsive background activity monitoring and non-intrusive blocking while preserving usability and data persistence.
The PoC will include a working coded prototype that demonstrates:

1. Timer Core Functionality (P0 – Core Timer)
    - Start, pause, and reset a focus session timer.
    - Persist session data (start/end time, duration) using local files.
    - Display real-time countdown updates in the UI.
2. Whitelist Management (P0 – Whitelist)
    - Editable whitelist configuration interface (simple input form).
    - Background script that detects non-whitelisted applications or URLs and triggers a soft block message (“Stay focused!”).
3. Minimal Data Dashboard (Preview of P1)
    - Basic text or chart output summarizing total focus time and distraction count.
    - Demonstrates data flow from local storage to frontend display.

## 13. Technology

### 1.Front end

- Framework: Electron + React (or Vue/Svelte) for a desktop web-like UI.
- UI Library: Ant Design or MUI for consistent, polished components.
- Communication: IPC (Inter-Process Communication) or WebSocket to the backend.
- Key Roles:
  - Provides the visual interface, timers, focus statistics, and card collection.
  - Displays achievements and levels .
  - Handles user interaction, notifications, and animations.
- Advantages:
  - Cross-platform compatibility (Windows, macOS, Linux).
  - Rapid development using web technologies.
  - Smooth integration with existing Chrome extension logic.

- **Browser extension** (MV3): TypeScript + Chrome APIs
  - Build: Vite
  - Messaging: WebSocket → localhost app
  - (Optional) small React UI for the extension's options page

**Backend** (local)
- .NET 8 (C#) service inside the WPF app (single-process)
  - WebSocket endpoint via ASP.NET Core/Kestrel bound to 127.0.0.1:<port>
  - Windows active-window/app detection via Win32/PInvoke
  - Persistence: SQLite (via Entity Framework Core — preferred — or Dapper)

**Development Environment**
- Frontend: VS Code + Node.js 20
- Backend: Visual Studio 2022 + .NET 8 SDK
- Database: SQLite (shared local database file)
- Code Quality Tools:
  - Frontend: ESLint + Prettier
  - Backend: Roslyn Analyzers + StyleCop

**Unit / integration testing** — YES (we will)
- C# unit tests: xUnit (+ FluentAssertions, Moq for mocking)
  - Target: Focus state machine, whitelist policy engine, rewards math, DB repositories
- C# integration tests: xUnit host fixtures to run EF Core with SQLite InMemory and simulate WS events end-to-end
- Extension tests: Vitest (unit) + Playwright (E2E in a Chromium profile)
- CI: GitHub Actions (build, test, packaging artifacts)

Why: deterministic core logic (state machine, rarity tables) must be verifiable; prevents regressions as we add AFK, strict mode, quests, etc.

2.**Machine learning** is optional in Growin. Potential use cases:
1. Card Art Generation: Python microservice using PyTorch and Diffusers (Stable Diffusion / Flux models) for AI-generated rewards.
2. Analytics: Future enhancement with scikit-learn for user habit clustering or focus pattern prediction.

If integrated directly in C#, small models can be run via ONNX Runtime.

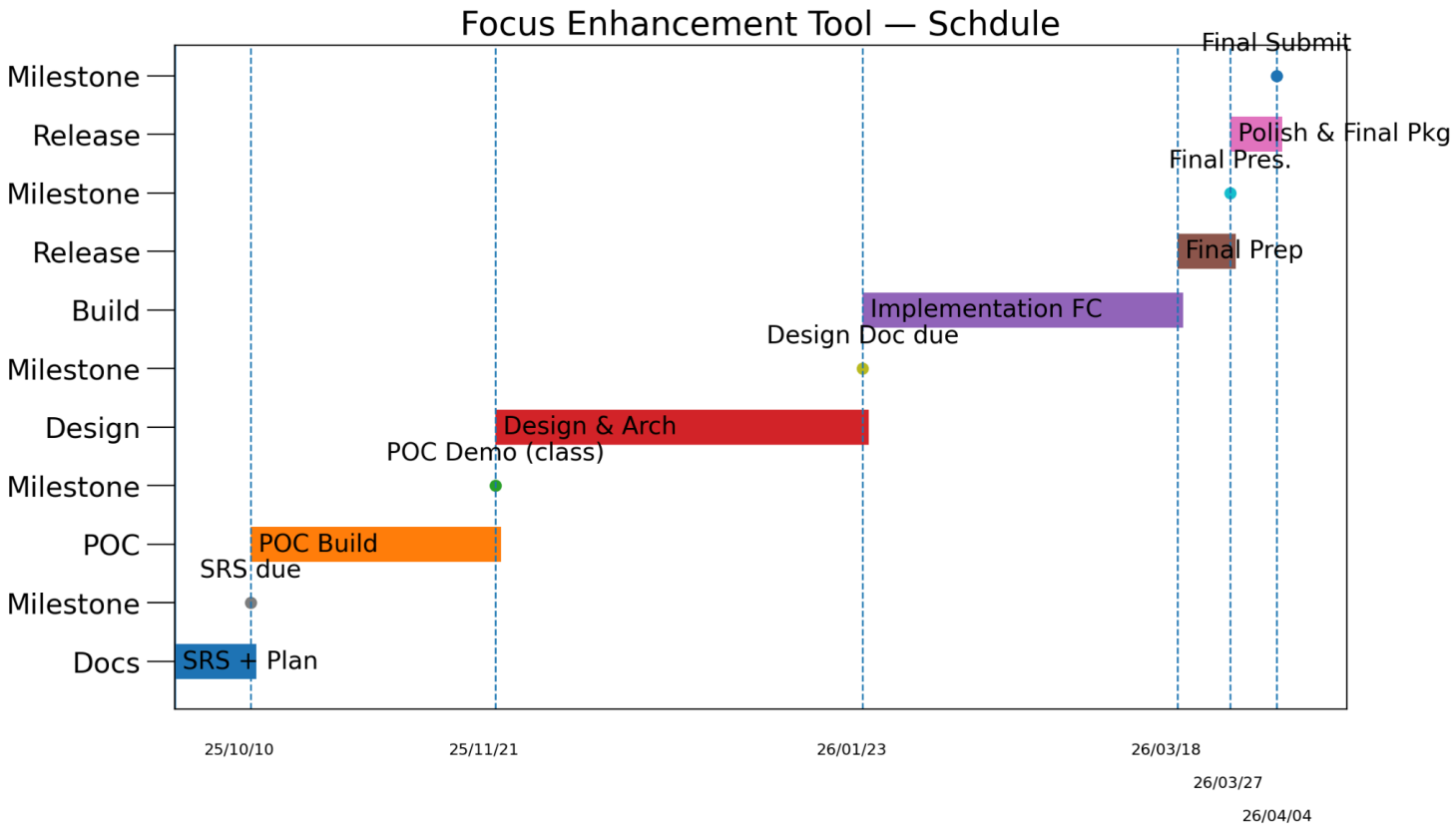
3.Gamification Extensions

Planned features such as  (achievement system) and  (level system) can use lightweight ML models or heuristic scoring to adapt rewards and progression curves.

c. **Will you use GPU? Any other relevant technology aspects.**

Core Application: No GPU is required for normal operation (focus tracking, timers, rewards, and user interface).

## 14. Project Scheduling




- **Horizontal bars: long phases**
  - **SRS + Plan (Docs)** — drafting and finalizing the Software Requirements Specification and project plan.
  - **POC Build (POC)** — build a minimal end-to-end path proving the core idea works (timer + blocking + basic UI).
  - **Design & Arch (Design)** — system/module design, privacy/accessibility decisions, interfaces, and data model.
  - **Implementation FC (Build)** — implement features until *feature complete* (timer logic, whitelist, analytics, onboarding, rewards).
  - **Final Prep (Release)** — prepare demo deck, rehearse talk, polish flows to presentation quality.
  - **Polish & Final Pkg (Release)** — bug fixes, UI tidy-ups, documentation, packaging the final deliverables.
- **Points (●) : one-day milestones**
  - **SRS due** — submission of the SRS + Project Plan.
  - **POC Demo (class)** — in-class proof-of-concept demo.
  - **Design Doc due** — hand-in of the full design document.
  - **Final Pres.** — in-class final presentation.
  - **Final Submit** — final code + docs package submission.