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

Growin helps users regulate entertainment and social-media use through a structured focus cycle—plan, focus, reward, reflect—organized across five development priorities (**P0–P4**). **P0** delivers the MVP end-to-end: an Electron-based UI featuring the core timer, widgets, a basic User Profile (name/avatar, focus totals, streaks), paired with a .NET backend that enforces focus logic, manages app whitelisting, monitors user activity, and stores data locally in SQLite. **P1** introduces gamified motivation with the **Gacha**, **Pet**, and **Music Player** systems—users earn tokens through completed sessions, draw collectibles, nurture virtual pets, and manage background music. **P2** expands engagement with **Mini-Games**, **Collections**, and **Achievements**, reinforcing focus habits through unlockable rewards and milestone tracking. **P3** adds analytics and visibility through personal **Dashboards** (heatmaps and trends) and an optional anonymized **Global Dashboard** displaying percentile-based rankings and global focus trends. **P4** layers social interaction via the **Friend System and Social Ranking**, enabling optional friend connections, leaderboards, and badges while incorporating moderation and privacy controls. Each requirement specifies **Frontend**, **Backend**, and **Data** objects (e.g., Session, Event, Whitelist, User Stats, Inventory) with measurable performance targets (e.g., latency, accuracy, fairness) to ensure functionality is testable, verifiable, and scalable.

6.1 Data Requirements Overview

- **User input data**: focus duration, profile information (username, signature), widget preferences.
- **System data**: application whitelist, focus session logs, user statistics, pet state.
- **Generated data**: gacha tokens, collectible items, achievement records, friend ranking data.

6.2 Functional Requirements Table

Feature Name	Priority	Frontend / Backend	Data Requirements	Description

Core Timer, UI, Widgets, Backend Monitoring, User Profile (basic)	P0	**Frontend:** start/stop timer, session setup, widget panel (clock/weather), basic profile view; **Backend:** timer enforcement, active app/URL monitoring, whitelist rules, local persistence	Focus duration, widget config, whitelist apps/sites, session logs (start/end/timestamps), basic user profile (UID, username, signature), cumulative focus time	Delivers fundamental focus management with a usable UI. Users configure sessions; whitelist controls block non-allowed apps/sites during focus; widgets show the current timer and warning. Profile shows basic identity and total focus time. Data stored locally; works offline.
Gacha (Card Draw), Pet, Music Player	P1	**Frontend:** gacha UI/animations, pet status panel, music controls; **Backend:** token generation after sessions, draw logic, pet state updates, audio playback control	Reward tokens, pet growth/affinity, music playlist/track state, session outcomes (to grant tokens)	Adds motivating, optional gamification. Users earn tokens from completed sessions, draw collectibles, see pet react/grow with positive habits, and play background music (break-friendly, can be muted).

Simple Games, Collection, Achievements	P2	**Frontend:** mini-game UIs (breaks only), collection gallery, achievement panels; **Backend:** game result storage, unlock logic, achievement triggers	Game scores, focus history (total hours, streaks), collection inventory, achievement states/timestamps	Expands engagement: light games during breaks only, collectible system, and achievements tied to focus goals (e.g., total hours, streaks). All surfaced in a gallery.
Global Dashboard (early competitive signals)	P3	**Frontend:** global dashboard views (e.g., anonymized top charts, global milestones, trends); **Backend:** periodic aggregation jobs, leaderboard computation (read-only), caching	Aggregated usage metrics (e.g., percentiles, anonymized top focus durations), deployment-wide stats	Introduces early competitive features without social graphs: read-only, anonymized global stats/leaderboards to encourage long-term retention. No friend relationships yet; opt-in if cloud is used.

Friend System & Social Ranking	P4	**Frontend:** friend management, social leaderboard, profile badges; **Backend:** friend graph, ranking logic, rate limits, abuse controls	User IDs, friend connections, per-user focus totals/streaks, moderation flags	Adds social + competitive features: add friends, compare focus stats on leaderboards, and show badges. Requires account/sync; includes basic moderation and abuse prevention.

6.3 Frontend and Backend Responsibilities

- **Frontend**: provides the user-facing interface including the timer, widget toolbox, whitelist settings, pet display, music player, mini-games, user dashboard, friend list, and ranking display, gacha interface, 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, updates pet states, records game scores, and evaluates achievement conditions.

6.4 Priority and Development Phases

The functional requirements are aligned with incremental development phases:

- **P0 (Timer, User Interface, Widgets, Backend Monitoring, User Profile)**: Delivers the fundamental logic that ensures the tool's primary purpose of focus management with UI implemented.

- **P1 (Gacha, Pet, Music Player)**: Adds engaging gamification and motivational features through gacha draws, pet interactions, and background music.

- **P2 (Simple Games, Collection, Achievements)**: Expands user engagement by offering light games, collectible items, and long-term achievement goals.

- **P3 (Dashboard)**: Introduces early competitive features by providing a global dashboard to promote long-term retention.

- **P4 (Friend & Social Ranking)**: Introduces social and competitive features to promote long-term retention and community building.

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 Look and Feel Requirements.

- The desktop app and extension present a clean, distraction-free interface with consistent spacing, typography, and iconography across Windows 10/11 and Chrome v120+. Core screens use high-contrast themes that meet WCAG 2.2 AA for text and interactive elements. Animations (e.g., gacha spins, pet reactions) are brief (≤300 ms entrance, ≤ 200 ms exit) and respect the OS “Reduce motion” setting; timer digits render with zero visual jitter at 1 Hz and never overlap other elements at 1366×768. Visual state changes (start, pause, block, unlock) appear within 100 ms with clear, color-safe cues plus text labels—no color as the only signal.

7.2 Usability and Humanity Requirements.

- A first-time user can create a whitelist and start a focus session in ≤5 minutes and ≤ 10 actions on the happy path; median time-to-first-session is ≤120 s in hallway tests (n ≥ 20) with ≥ 95% task success and ≤ 2 median mis-clicks. Keyboard navigation covers all core controls with logical tab order and visible focus rings; all controls have accessible names/roles for screen readers. When a non-whitelisted app/site becomes active, the UI shows a reasoned, human-readable countdown (≤ 60 s) that the user may cancel; enforcement never occurs without an explanation. Presets (Pomodoro 25/5, HIIT 15/2, Ultradian 90/20) are available and any custom duration persists across restarts. Copy is plain, localized, and avoids blame; errors are actionable and, where possible, offer a single-click fix.

7.3 Performance Requirements.

- On baseline hardware (≥ 2 cores, 8 GB RAM, SSD, 1366×768), cold start to an interactive home is ≤ 2.0 s and warm start ≤ 1.0 s. The timer maintains drift ≤ ±50 ms per minute; UI actions (start/stop/pause) reflect state in ≤100 ms. Foreground/whitelist checks run ≥ 4 Hz and block non-whitelisted targets in ≤ 250 ms. Local DB writes complete in ≤ 20 ms; daily compaction finishes in ≤ 1 min without UI jank. The localhost WebSocket used for IPC shows ≤ 50 ms latency and auto-reconnects in ≤ 3 s. Background CPU during an active session

averages <5% with peaks <20%; steady-state memory for the app is ≤300 MB and the extension adds ≤50 MB. Mini-games (breaks-only) deliver input-to-frame latency ≤50 ms and ≥30 FPS.

7.4 Operational and Environmental Requirements.

- The system comprises a Windows 10/11 (64-bit) desktop app and a Chrome v120+ extension. P0 features (timer, whitelist enforcement, clock widget, local logging, basic profile) work entirely offline; weather/music/art assets cache and display a “Last updated” stamp when the network is absent. Installation is user-level (no admin), updates may prompt, and uninstall preserves user data unless “Remove data” is selected, which deletes local records within ≤2 s per 1000 items. The product functions on single-monitor 1366×768 displays without clipping and scales correctly up to 200% DPI.

7.5 Maintainability and Support Requirements.

- Modules follow a documented layout and naming scheme; individual functions target cyclomatic complexity ≤ 10. Core logic maintains ≥ 70% unit test coverage, with integration smoke tests on every PR; the CI pipeline (lint, unit, package, basic e2e) completes in ≤10 minutes wall time. No hard-coded paths or secrets are allowed; configuration is externalized with safe defaults. Logs are PII-redacted, rotate with a total cap ≤ 50 MB, and can be toggled to “verbose” by the user; support captures include build hash and feature flags only.

7.6 Security Requirements.

- By default, only app names/domains, timestamps, durations, and mode IDs are stored; page titles or contents are excluded unless the user opts in. Local IPC binds to 127.0.0.1 with strict origin checks and a per-boot random token (≥ 128-bit) whose idle TTL is ≤ 24 h. Data at rest resides in an encrypted DB or OS-protected user directory with integrity checksums; secrets never appear in source or logs, and builds fail if scanners detect them. The extension requests least-privilege permissions and supports on-click site access. Users can view, export (CSV/JSON), and delete records by time range; default retention is ≤ 12 months. Friend graph and leaderboard endpoints apply rate limits (adds/removes ≤ 10/min/user; reads ≤60/min/user), and security SLAs patch High severities within 7 days and Medium within 30 days; releases are signed with reproducible build notes.

7.7 Cultural Requirements.

- The product is culturally neutral and avoids idioms or imagery that may not translate; celebratory visuals (e.g., gacha win, streaks) avoid gambling metaphors by default. All strings are i18n-ready with pluralization rules; dates/times follow system locale, and 12/24-hour format matches OS settings. Fonts include CJK and emoji coverage without layout shift; examples and defaults avoid region-specific services unless available locally.

7.8 Compliance Requirements.

- Privacy choices follow explicit opt-in for any optional content capture and default to data minimization; a clear privacy notice describes what is stored locally, what may sync (opt-in), and how to export/delete. Third-party licenses are tracked and a software bill of materials (SBOM) is generated per release. Telemetry, if enabled, is opt-in, anonymous, and never required for functionality. The design and accessibility conformance target WCAG 2.2 AA; encryption follows platform best practices, and cryptographic RNGs meet contemporary recommendations (≥128-bit entropy). Internal audits verify adherence to these requirements before each minor release.

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. 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. Ensures usability, interface consistency, and accessibility compliance.
Zikai Lu	Backend engineer for **data storage**, session logs, and analytics dashboard implementation. Ensures data consistency, local database efficiency, and secure handling of user information.
Chiyu Huang	Develops **Gacha System**, integrating reward algorithms and randomized drop tables. 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. Oversees reliability verification, bug tracking, and regression test documentation.
Qinquan Wang	Handles **Achievements & Collections**, coordinating animation, asset integration, and overall UI polish. 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 timer, widget, and background monitoring 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. Widget Integration (P0 – Widget)
    - A simple toolbox area containing mini-widgets.
    - Widgets are modular and can be toggled on/off by the user.
    - Displays the remaining time of the current session and shows the warning signs when distracted.
3. Background Monitoring (P0 – Monitoring System)
    - Ensures the app runs smoothly while the user engages in other tasks.

    - Background script that detects non-whitelisted applications or URLs and triggers a warning on the Widget with time remaining before the focus session fails.
4. Preview of P1 Features (Gacha, Pet, Music)
    - Basic text or chart output summarizing total focus time and distraction count.
    - A simple token reward display after finishing a timer session (to preview the gacha system).
    - Placeholder “pet” graphic or icon that reacts to session completion.
    - Basic music toggle to illustrate integration.

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)

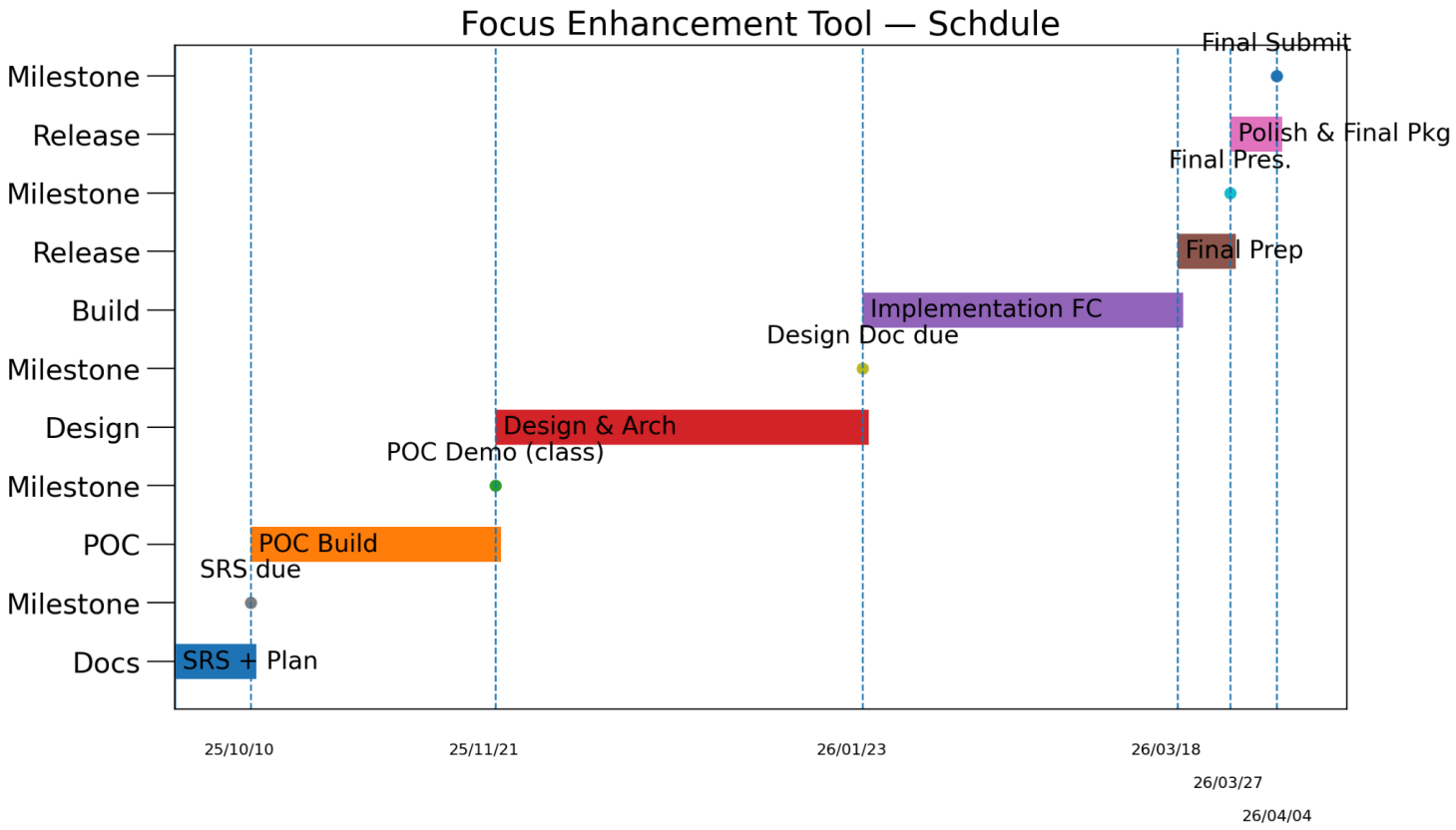
- C# unit tests: xUnit (+ FluentAssertions, Moq for mocking)
  - Target: Focus state machine, whitelist policy engine, rewards math, DB repositories
- C# integration tests: xUnit host fixtures to run EF Core with SQLite InMemory and simulate WS events end-to-end
- Extension tests: Vitest (unit) + Playwright (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.