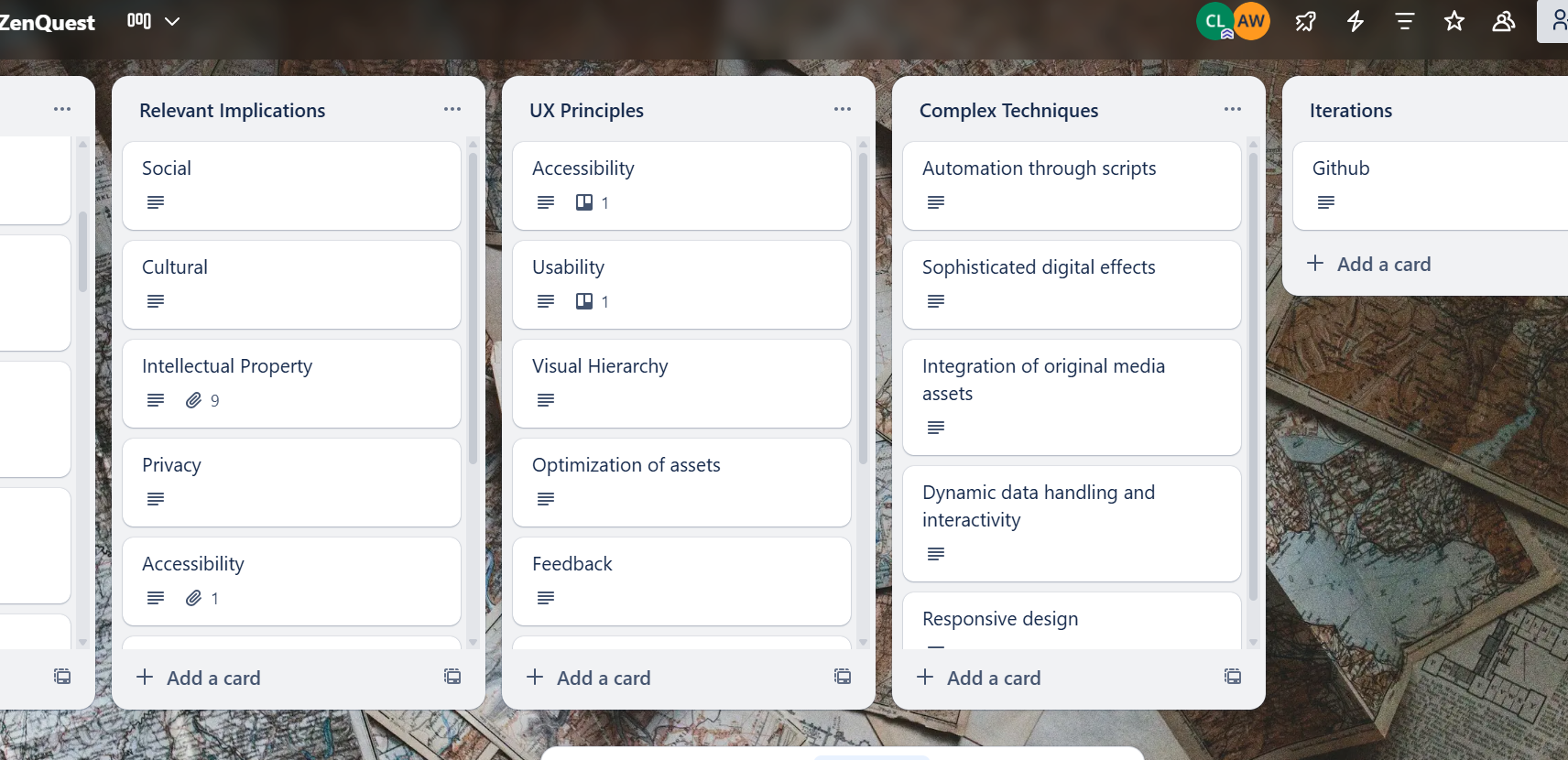
**This document contains the testing log, end-user feedback and a final evaluation**

**Trello:** [**https://trello.com/b/hnU8ma5N/zenquest**](https://trello.com/b/hnU8ma5N/zenquest)

Trello contains relevant implications, UX principles, complex techniques, iterations, etc.

Click on the colored labels to see meaning.

**A screenshot of a cell phone

AI-generated content may be incorrect. **

**Demo/testing video:** [**https://saintkentigern-my.sharepoint.com/:v:/p/c\_lau\_student/EclhRVlv6HJInifUFqt2VUUBATDEyxQXolkYdIyYoKn2sg?e=ix7Ogg&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJTdHJlYW1XZWJBcHAiLCJyZWZlcnJhbFZpZXciOiJTaGFyZURpYWxvZy1MaW5rIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXcifX0%3D**](https://saintkentigern-my.sharepoint.com/:v:/p/c_lau_student/EclhRVlv6HJInifUFqt2VUUBATDEyxQXolkYdIyYoKn2sg?e=ix7Ogg&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJTdHJlYW1XZWJBcHAiLCJyZWZlcnJhbFZpZXciOiJTaGFyZURpYWxvZy1MaW5rIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXcifX0%3D)

**Functionality Testing**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Description | Input | Expected Outcome | Actual Outcome (screenshots) | Pass |
| Daily quests - Expected | Any text in the daily quest entry field | Daily quest is added to list, and a toast message pops up (same for all quest types) |  | **✔** |
| Daily quests – Invalid | Empty entry field/ only spaces | Nothing happens. User is unable to add quest |  | **✔** |
| Habits - Expected | Any text in the habit entry field | Habit is added to list  Green label if positive; red label if negative |  | **✔** |
| Habits – Invalid | Empty entry field/only spaces | User is unable to add habit |  | **✔** |
| Main quests - Expected | Any text in the main quest entry field, with or without due date.  (User can only choose a future date from the calendar pop-up) | Main quest is added to list  If due date is selected, it is shown as a small label under the quest |  | **✔** |
| Main quest due dates - Expected | Main quest is added | Added main quests have a left border that corresponds to the due date selected.  Red (Pulsing) – Overdue  Orange (Pulsing) – Due tomorrow or today  Yellow – Due this week  Green – Due this month  Grey – Due date is more than a month away or not selected  Main quests are added in priority according to due dates. Orange (most recent due date) on top, grey on bottom. And then user can drag and drop as they wish. |  | **✔** |
| Main quests - Boundary | User manually enters a due date that is the day before current date  User enters current date or the following day | If the due date is a day before current date, error message shows up.  If due date is current date or the next day, accept |  | **✔** |
| Main quests - Invalid | Empty entry field/ only spaces  Any date before the current date is chosen. | Empty: user is unable to add quest  Invalid date: error message shows up |  | **✔** |
| Side quests – Expected | Any text in the side quest entry field  User clicks the dice button | Side quest is added to list  Dice: random side quest is generated (chosen from preset list) |  | **✔** |
| Side quests – Invalid | Empty entry field/ only spaces | User is unable to add quest |  | **✔** |
| Quest Completion - Expected | User clicks the tick/plus buttons | For main and side, quest is removed.  For daily and habits, quest counter increases by 1. Quest is not removed. A completed daily quest is blurred (this resets daily).  XP is added depending on quest type, and Zen Coins are added depending on chance (50% at level 1), with toasts indicating specific numbers). XP progress bar and Zen Coins display are updated.  For negative habits, HP is deducted (no rewards). HP progress bar is updated.  If XP is full, user levels up. Dismissible toast pops up. Restores full HP.  Some items have a critical chance stat. Critical hits give double the original rewards (blue toast). |  | **✔** |
| Quest Removal – Expected | User empties out an existing quest item | The quest is removed from the list, and a toast pops up |  | **✔** |
| Market – Custom Rewards – Expected | User enters a real-life reward with name, description and price  User clicks the X button | Reward is added to list  Reward is removed |  | **✔** |
| Market – Custom Rewards – Boundary | 0 is entered as price  1 is entered as price | 0: error message  1: accepted |  | **✔** |
| Market – Custom Rewards – Invalid | Any field is left empty/ only spaces | Error message pops up |  | **✔** |
| Market – Any Item (Weapons/  Equipment/  Potions) – Expected | User clicks Buy/Upgrade button with enough Zen Coins  (Upgrade button is shown if user already owns the item) | User successfully buys the item, and it is added to inventory. Zen Coins is deducted according to price. Item stats added to user.  User successfully upgrades item, and level of item increases by 1. Zen Coins are deducted according to price. Item stats added to user. |  | **✔** |
| Market – Any Item - Boundary | User has 99 Zen Coins and tries to buy an item that costs 100  User has 101 Zen Coins and tries to buy an item that costs 100 | 99: error message pops up  101: user successfully buys the item, and it is added to inventory. Zen Coins are deducted according to price. |  | **✔** |
| Market – Any Item – Invalid | User clicks Buy/Upgrade button with insufficient Zen Coins | Error message pops up |  | **✔** |
| Inventory - Expected | Weapons/Equipment: User clicks Equip button  Potions: User clicks Use button  User clicks Sell/Downgrade button | Equip: User gains extra bonuses depending on the weapon/equipment. If an item is already equipped, the new item is equipped in place of it.  Use: User gains bonuses from potions  Sell/Downgrade: Item is sold for half its price. If weapon/equipment, item stats are removed. |  | **✔** |
| Market & Inventory Item Sorting & Compact Mode - Expected | User chooses an option from dropdown menu  User clicks the compact mode button | Items are sorted according to the option chosen (Level, Price, Quantity, Alphabetical)  Items are smaller and more compact (can be toggled) |  | **✔** |
| Pomodoro timer - Expected | User clicks Start button  User clicks Pause  User clicks Reset  User clicks on Focus, Short Break or Long Break | Start: Timer starts countdown. ‘FOCUS/REST SPELL CASTED” message shown. Blue flame animation starts.  Pause: timer is paused. PAUSED message shown. Blue flame is extinguished.  Reset: timer is reset  Top buttons: timer duration changed depending on mode. Only Focus mode gives rewards on completion. |  | **✔** |
| Nav bar - Expected | User clicks a nav button | User is taken to respective page.  Quests -quest page  Market- market page  Items - inventory page  User – user page  Focus! – pomodoro timer in quest page | A screenshot of a video game  AI-generated content may be incorrect.A screenshot of a video game  AI-generated content may be incorrect. | **✔** |
| Mini HUD - Expected | User scrolls past the main HUD  User clicks the hide button at top right corner (hides header and main HUD) | The mini HUD shows up at top of the screen.  For User page, mini HUD is shown by default, as the main HUD is pretty useless there. |  | **✔** |
| Daily quest bonus & penalty - Expected | User completes 5 or more daily quests in one day  User fails to complete 5 or more daily quests | User gains extra XP and coins. Toast pops up to notify user.  User loses HP. Toast pops up to notify user at the start of each day. |  | **✔** |
| XP - Expected | User fills the XP bar | The XP bar resets to 0, and level increases by 1. XP required for next level increases.  Formula: xp required = 100+(level-1)\*(10^1.2)  Each level increases zen coin gain & zen coin chance  Formula: (coin gain)\*(ln(level+1))^1.05  Zen coin chance +1% per level (capped at +50%).  Show level up toast.  HP restored to max. (If max already, +0 HP)  If additional XP from previous level, carry over to next level. |  | **✔** |
| HP - Expected | HP reaches 0 (death)  HP is already max | Upon death, level -1 and coins -10%. (if level = 1 already, only decrease coins)  Show death toast.  Then reset to max Hp.  Any HP recovery will be HP+0. HP cannot go over max HP. |  | **✔** |
| Items - Expected | Potions:  User uses HP potion when not at full HP.  User uses XP potion.  User uses a Xp/Coin boost potion  Other Item testing done above (Market & Inventory) | Otherwise, HP gets recovered according to type of HP potion.  User gains XP. XP gain scales with stats.  User gains boosts for a limited duration of time. |  | **✔** |
| Items - Invalid | User tries to use HP potion when already at full HP.  User tries to upgrade items past their user level | Full HP: Error toast shown. Potion does not get consumed.  User is unable to upgrade the item. The button cannot be clicked. |  | **✔** |
| Feedback - Expected | User clicks feedback button on top right of user page | User is brought down to feedback section | A screenshot of a video game  AI-generated content may be incorrect. | **✔** |
| Feedback Form - Expected | User fills in all fields (except email) and clicks Send Feedback | The feedback is sent to my game email (zenquestrpg@gmail.com) through EmailJS. Success toast pops up along with XP reward. | A screenshot of a phone  AI-generated content may be incorrect.A green rectangle with white text  AI-generated content may be incorrect.A screenshot of a video game  AI-generated content may be incorrect.A screenshot of a computer  AI-generated content may be incorrect. | **✔** |
| Feedback Form - Boundary | User enters 9 or 11 characters | 9: error message  11: accepted | A screenshot of a message  AI-generated content may be incorrect. | **✔** |
| Feedback Form - Invalid | User enters less than 10 characters | Error message shown | A screenshot of a video game  AI-generated content may be incorrect. | **✔** |
| HTML/CSS Validation | HTML: <https://validator.w3.org/>  CSS: <https://jigsaw.w3.org/css-validator/> |  |  | **✔** |

All testing is done using VS Code preview.

After each iteration, I also tested the website on both my phone and laptop (with GitHub deployment) to ensure everything actually works. I consulted my family and random people online for feedback on each iteration.

**Trialling**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component | Alternatives Trialled | Chosen Solution | Why Chosen | How It Improved My Outcome |
| Background System & Visual Theme | -Static CSS gradients  -Single background images  -Animated CSS backgrounds | Layered background images with blur effects | Different backgrounds per page (market\_bg.jpg, inventory\_bg2.jpg) provide contextual visual identity while blur effects maintain text readability. [background-attachment: fixed](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) creates immersion without performance issues. | Users can instantly identify which section they're in, enhancing navigation and creating a more engaging RPG atmosphere that encourages longer usage sessions. |
| Notification System | -Basic JavaScript alerts  -Custom CSS toast notifications  -SweetAlert2 library | Notyf with custom styling | Notyf provided flexibility for different notification types (XP, coins, health, success, error) while being lightweight. Custom CSS classes like .notyf-xp, .notyf-zen-coin allowed RPG-themed styling with proper z-index control. | Immediate feedback for user actions increases engagement and clearly communicates system responses, crucial for gamification effectiveness. |
| Quest Management & Drag-and-Drop | -Simple list with up/down buttons  -Fixed priority ordering | HTML5 Drag & Drop API with custom handlers | Implementation in [app.js](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) with dragstart, dragover, and drop events provides intuitive reordering while maintaining data persistence. Visual feedback (.dragging class with rotation) enhances UX. | Users can prioritize tasks naturally, increasing ownership and task completion rates by allowing personal organization preferences. |
| Progress Bar Animation System | -Simple width transitions  -JavaScript-based counters | CSS animations with pseudo-elements | The [progress-bar-shine](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) animation with ::before pseudo-elements creates engaging visual feedback without JavaScript overhead. The 4-5 second cycling provides subtle movement that doesn't distract. | Visual progress feedback increases motivation and makes stat changes feel more rewarding, core to RPG progression psychology. |
| Icon System | -Unicode emojis  -Font icons (Font Awesome) | Custom PNG icons (star.png, warning.png, calendar.png) | PNG icons ensure consistent appearance across all devices/browsers, maintain the pixelated RPG aesthetic, and allow precise sizing control. The transition from emojis to PNGs eliminated cross-platform inconsistencies. | Consistent visual language strengthens brand identity and ensures all users see identical interface elements regardless of their device. |
| Data Persistence | -Session storage only -Cookie-based storage | localStorage with JSON serialization | localStorage provides persistent storage without server dependencies, JSON allows complex data structures, and the backup/restore system in [utilities.js](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) ensures data safety. | Users never lose progress, encouraging long-term engagement and reducing frustration from data loss. |
| Pomodoro Timer Implementation | -Simple countdown display  -Canvas-based circular progress | SVG circle with stroke-dasharray animation | SVG scales perfectly, stroke-dasharray provides smooth progress animation, and the gradient definitions create appealing visual effects while remaining lightweight. | Visual timer feedback increases focus session completion rates and makes time passage more tangible. |
| Market/Inventory Sorting System | -Fixed sorting options for all items  -Manual category switching | Dynamic category-aware sorting | The updateSortingOptions() functions in [market.js](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) and [inventory.js](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) show relevant options only (level for weapons, quantity for potions), preventing user confusion while maintaining functionality. | Users can efficiently organize large inventories, reducing frustration and improving the shopping/management experience. |
| Gamification XP/Reward System | -Fixed XP values  -Simple linear progression | Variable XP with bonuses and multipliers | The [awardXP()](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") function includes level scaling, potion bonuses, and critical hit chances, creating unpredictable rewards that maintain engagement through variable ratio reinforcement. | Unpredictable rewards increase dopamine response and task completion rates, making productivity feel genuinely rewarding. |
| Feedback Collection System | -Google or Microsoft Forms integration -Mailto links | EmailJS with custom modal | EmailJS sends emails directly from frontend without backend infrastructure, the custom modal provides professional UX, and the template system allows formatted feedback with user context. I’ve been using Microsoft Forms for feedback for my projects the past 2 years, so this would be a good time to try something new and make a feedback form that actually suits the style of my web app. | Direct feedback channel improves user satisfaction and provides valuable insights for iterative development. |
| Branch Decoration System | -CSS border decorations  -Single background images | Multiple overlapping branch.png images with animations | Complex CSS background positioning creates organic, layered vine effects that animate on hover, providing subtle environmental feedback that enhances the nature/growth theme. | Atmospheric details increase immersion and make the interface feel alive, encouraging longer engagement periods. |
| Due Date Priority System | -Color-coded text  -Priority badges | Border color coding with pulsing animations | Left border colors provide immediate visual hierarchy without cluttering the interface, while pulsing animations for overdue items create urgency without being overwhelming. | Clear visual priorities help users focus on urgent tasks first, improving productivity and reducing stress from deadline management. |
| Navigation Architecture | -Tab-based navigation  -Sidebar navigation  -Hamburger menu for mobile | Bottom mobile nav + top desktop nav | Bottom navigation optimizes thumb reach on mobile, while top navigation utilizes desktop screen real estate effectively. The context-sensitive approach serves each platform's strengths. I also believe a hamburger menu doesn’t really suit the style of an RPG game, so I decided against it, and a bottom nav bar also looks nicer. | Platform-appropriate navigation reduces cognitive load and improves user flow, increasing feature discovery and usage. |

Each trialling process involved implementing alternatives, testing usability, measuring performance impact, and gathering user feedback to inform final decisions. The chosen solutions balance functionality, performance, maintainability, and user experience to create a cohesive gamified productivity web app. More in-depth discussion at the end.

**User feedback**

|  |  |  |
| --- | --- | --- |
| Feedback | Improvements | Evidence |
| Create custom icons/sprites to enhance aesthetics | I used PixelLab to generate pixelated icons for my website to replace the emojis. These icons suit the style of my website much more than emojis. |  |
| The pomodoro timer is a bit boring | I added a blue flame animation while the timer is going |  |
| I want to be able to see how many daily quests or habits I have completed | I added quest counters for each daily and habit which allow them to keep track of their productivity  I also added total quests completed for each category in the User page |  |
| Add a way to delete quests/habits without just completing them | I made it so that quests can be removed by removing all content in the textbox (leaving it empty), rather than making an extra delete button (takes up space). |  |
| Add an option to hide the header/HUD. It becomes unnecessary and takes up space when the user is more familiar with the website | Added a button to hide the header and HUD. A mini HUD shows up at the top of the screen instead. |  |
| Add more specific feedback toasts, e.g. successfully bought XXX, instead of just Success! | I made different toasts specific to different situations. |  |
| Ability to sort quests | Users can now drag quests around to rearrange them. I also made the quest items larger to make them easier to drag. |  |
| I should not be able to select an earlier date for my main quest. | I made it so that users can only select a future/current date from the calendar, and if they manually type a previous date, an error message is shown. |  |
| For weapons and equipment, instead of quantity, change it to level. Seems more logical. | I replaced weapon/equipment quantities with levels. After the user buys an item for the first time, the Buy button changes to Upgrade. Sell button in inventory changes to Downgrade. |  |
| Option to randomly generate a side quest | Added a button to randomly generate a side quest (from a preset list) |  |
| I can’t see if my boost potion is working or not | I added indicators which allow users to see what type of boost potion is active currently and its remaining duration |  |
| When I try to type on mobile, the screen keeps jumping around since it forces me to zoom in when typing | I made it so that users don’t get forced to zoom in when typing |  |
| The level up rewards are underwhelming, considering that it is pretty hard to level up | Each time the user levels up, they gain one of each boost potion.  I also made it so that user gains a random potion upon finishing a focus session (pomodoro).  Might add more rewards later. |  |

**3.4**

**Iterative improvement throughout the design, development and testing process to produce a high-quality outcome.**

**Using efficient tools and techniques in the outcome’s production.**

**How I Used Sprint-Based Development**

Throughout developing ZenQuest, I followed an iterative approach using GitHub and Trello to manage my workflow. Instead of trying to build everything at once, I broke the project into smaller sprints that focused on specific features. My Trello Kanban board had columns like To Do, WIP, Testing and Complete, and each card had a label indicating the week it has to be done, which helped me track what I was working on each week. Using this allowed me to allocate my time more efficiently and stick to a deadline as I could see how much work I had remaining. After each sprint, I asked my end-users for feedback to find areas of improvement for the next sprint, ensuring that my website was tailored to my end-users.

For example, I started with basic quest functionality, which is just adding and completing tasks. Then in the next sprint, I added the XP and levelling system. After that came the market system, then the inventory management, the responsive design for desktop, and more advanced features. Each sprint built on the previous one, but I could test and refine each feature before moving on to the next.

For the HUD, I initially created a simple mobile layout with health, XP, and coins stacked vertically. When I moved to desktop, I completely redesigned it using CSS Grid with three columns -level badge on the left, progress bars in the middle, and coins on the right. Later, I went back and added visual enhancements like shadows, gradients, and subtle animations. Each iteration improved the previous version rather than trying to build the perfect interface from the start.

**Using GitHub for Version Control**

GitHub was essential for tracking my changes and experimenting safely. I organized my code into separate files (app.js for main functionality, [market.js](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) for the shop, [inventory.js](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) for items, etc.) which made it easier to keep track and work on different features without breaking existing code. When I was working on complex features like the HUD, market and inventory system and user stat calculation, I could try different approaches knowing I could always revert to a working version if something went wrong.

I made frequent commits with descriptive messages, which helped me track exactly what I changed and when. There were several times when I accidentally broke something (when the all the daily quest functionality stopped working, when the market and inventory suddenly became empty, etc.), and being able to look back at recent commits helped me identify and fix the problem quickly.

The modular file structure also meant I could work on the market system without worrying about breaking the quest functionality, since they were in separate JavaScript files. This separation made debugging much easier when issues arose. E.g. when the inventory system had issues, I knew to look in inventory.js. When the HUD wasn't displaying correctly, the problem was likely in [utilities.js](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) where the updateHUD function is.

**VS Code and Development Tools**

I relied heavily on VS Code's features to write cleaner, more consistent code. The auto-completion helped me avoid typos in long CSS class names and JS function names. The integrated terminal let me test changes quickly, and the built-in Git integration made it easy to commit changes without switching between different apps.

The browser developer tools were crucial for debugging responsive design issues. When the HUD elements weren't aligning properly on different screen sizes, I could inspect the CSS grid properties in real-time and adjust the values until they looked right. The device emulation feature helped me test the mobile experience without having to constantly resize my browser window.

VS Code's live preview extension was a game-changer for rapid iteration. I could see changes immediately as I coded, which was especially helpful when fine-tuning animations and responsive layouts. This instant feedback loop let me experiment more freely with different design approaches.

I configured GitHub Pages to automatically deploy your website directly from my main branch. Every time I pushed commits to my repository, GitHub Pages automatically built and deployed the updated version to my live website. Since ZenQuest is a client-side website using HTML, CSS, and JavaScript (no server-side processing needed), it was perfect for GitHub Pages deployment. This automated deployment system allowed me to focus on development rather than deployment logistics.

**Choosing and Integrating External Libraries**

Rather than building everything from scratch, I strategically chose external libraries that would save development time while adding professional polish. GSAP handles the complex animations you see when the stat categories fade in on the user page -creating those smooth, staggered animations with only CSS and vanilla JS would have been much more complicated and less performant.

For the notification system, I used Notyf instead of building custom toast messages. This saved me hours of development time and gave me a reliable, well-tested foundation that I could customize with my pixel art styling. I was able to create different notification types for XP gains, critical hits, level ups, and errors, etc. all using the same underlying system.

Particles.js creates the animated background effects that make the interface feel more dynamic and game-like. Implementing particle systems from scratch would have been a massive undertaking, but integrating the library only took a few lines of code while adding significant visual appeal.

**Responsive Design Through Testing and Iteration**

Creating a responsive design that worked well on both mobile and desktop required multiple rounds of testing and refinement. I started with a mobile-first approach, making sure the core functionality worked perfectly on small screens. The bottom navigation bar on mobile is positioned for easy thumb access, while the top navigation on desktop is more appropriate for mouse interaction.

The HUD transformation from mobile to desktop was particularly challenging. On mobile, everything stacks vertically which works well with touch scrolling. On desktop, I wanted to take advantage of the wider screen with a horizontal grid layout. This required creating two different layout systems that share the same underlying data but present it differently based on screen size.

I tested extensively on different devices and screen sizes, using both browser developer tools and actual devices after each iteration. Each test revealed small issues, e.g. the HUD components’ display being misaligned, or quests not behaving correctly, which required going back and adjusting the CSS and JS.

**Performance Optimization Strategies**

I focused on several performance optimization techniques throughout development. The pixel art aesthetic wasn't just a design choice, it also naturally results in smaller image file sizes compared to photorealistic graphics. I compressed all images and used appropriate formats to minimize loading times.

For JavaScript optimization, I split functionality across multiple files so each page only loads what it needs. The market page loads market.js, the inventory page loads inventory.js, but the main quest page doesn't need to load those heavier scripts. This keeps the initial page load fast and responsive.

Using localStorage for data persistence means the app works offline and loads quickly since it doesn't need to wait for server requests. I implemented error handling for corrupted localStorage data so the app gracefully falls back to default values if something goes wrong.

**Debugging and Quality Assurance**

Debugging was an ongoing process throughout development. Complex features like the equipment system (with its owned vs equipped stats and level caps) required extensive testing to make sure all the edge cases were handled properly. See testing table.

The pomodoro system integration with rewards was particularly tricky to debug. I had to make sure XP calculations were correct, that critical hits applied properly, that the random potion rewards worked, and that everything was saved to localStorage correctly. Each component worked fine individually, but making them work together required careful testing and refinement.

The daily quest penalty system needed thorough testing too. The app checks if you completed fewer than 5 daily quests and applies HP penalties accordingly. I had to test edge cases like what happens if someone completes exactly 5 quests, or if they complete quests after midnight but before the daily reset.

Cross-browser testing revealed some inconsistencies in how different browsers handled CSS grid layouts and animations. I had to add vendor prefixes and fallbacks to ensure the experience was consistent whether someone used Chrome, Firefox, Edge or Safari.

The error prevention system grew as I discovered potential problems. Form validation, confirmation dialogs for destructive actions, and graceful handling of missing data all came from identifying real issues during testing and implementing solutions. This iterative approach to quality assurance resulted in a much more robust and user-friendly application than if I had tried to anticipate every possible issue upfront.

**3.8**

**Synthesised information from the planning, testing and trialling of components.**

**Discussing how the information led to the development of a high-quality digital technologies outcome.**

**Navigation System Development**

My initial planning phase included a single navigation system that would work across all devices, but early trialling revealed the critical importance of context-aware navigation. During the planning stage, I had decided between a hamburger menu or a bottom nav bar for mobile. I decided to use a bottom navigation bar, as most RPG and modern games have a nav bar on the bottom for easy thumb access. However, when I trialled this approach with users on desktop devices, they consistently reached for the top of the screen rather than the bottom, following established desktop UI conventions. This insight from trialling led me to test two distinct approaches: maintaining the bottom navigation universally, versus implementing a hybrid system with bottom navigation on mobile and top navigation on desktop.

My testing phase involved measuring user interaction patterns and task completion times across different screen sizes. Desktop users completed navigation tasks faster with top navigation, while mobile users maintained optimal performance with bottom navigation. This synthesis of planning intentions, trial feedback, and testing data drove my decision to implement the responsive navigation architecture where @media screen and (min-width: 1200px) dynamically switches from bottom to top navigation. The final outcome demonstrates how systematic trialling and testing transformed a single-approach plan into a sophisticated, user-centred solution.

**RPG Progression System**

During my planning phase, I initially designed a straightforward XP system where completing any task would always give the same amount of XP. I thought this would be simple and fair for users. However, when I started trialling this system with end-users, I quickly discovered a major problem. They told me it felt boring and unrewarding after a while. This made me realize that my linear system wasn't engaging enough to keep people motivated long-term.

This user feedback pushed me to trial three different approaches to make the reward system more interesting. First, I tried adding random XP bonuses where sometimes you'd get extra points. Then I experimented with a critical hit system, and a system where equipment could boost your rewards. Finally, I tested a combination of both approaches to see which worked best.

During my testing phase, I had different people try each version for several days and asked them about their experience. The results were really clear, as everyone preferred the system with critical hits and equipment bonuses. People would actually get excited when they completed tasks because they never knew if they'd get a regular reward or a critical hit with bonus XP and coins. This testing showed me that the unpredictable rewards were much more engaging than fixed amounts. I learned that this connects to something called variable ratio reinforcement in psychology, where unpredictable rewards are more motivating than predictable ones.

Based on all this feedback and testing, I developed my final reward system where the gameStats.awardXP() function includes equipment bonuses, critical hit chances, and level-based scaling. Now when users complete tasks, they might get the base XP amount plus item bonuses with a chance of getting coins, or they might get a critical hit with double XP and bonus coins, especially if they have good equipment equipped. This completely transformed the boring task completion into exciting moments where users genuinely anticipate what rewards they might get, which directly solved the engagement problems I discovered during trialling.

**Visual Design**

My initial planning focused primarily on functional clarity with a basic dark theme and simple layouts. However, early trialling sessions revealed that users wanted more immersive visual elements that reinforced the RPG theme. This challenged my planning assumptions and led me to trial various atmospheric enhancements including particle effects, animated backgrounds, and decorative elements.

One component included the border decorations. I tested three approaches: CSS border decorations, single background images, and multiple overlapping branch images with animations. User feedback consistently favoured the organic branch system, with users commenting that the branches make it feel alive, like a real magical world. This synthesis of aesthetic planning, user trial preferences, and engagement testing resulted in the sophisticated visual system which creates layered branches/vine effects that only appear on the main quest page (forest background), providing contextual atmosphere without overwhelming utility pages (market and inventory unrelated to forests).

**Data Architecture**

Planning initially called for basic localStorage implementation for data persistence. However, trialling this approach quickly revealed significant limitations when users reported losing progress during browser updates or when switching devices. This critical feedback prompted me to trial enhanced data architecture solutions including backup systems, data validation, and recovery mechanisms.

My testing phase involved deliberately corrupting data, clearing storage, and simulating various failure scenarios to evaluate system resilience. The results clearly showed that users needed both automatic data persistence and manual backup capabilities. Testing also revealed that users wanted to see their historical progress, not just current state. This synthesis of planning requirements, user trial feedback, and failure testing drove the development of the comprehensive data system including [localStorage](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") with JSON serialization, automatic backup functionality in [utilities.js](vscode-file://vscode-app/c:/Users/Cyrus/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), and detailed progress tracking through user statistics in the User page. The final outcome provides bulletproof data persistence that users trust.

**Feedback Integration**

Throughout my development process, I made sure to constantly check my assumptions against real user experiences, which completely changed how I approached several key features. One of the best examples of this was my journey with the icon system. I had originally planned to use emojis everywhere because I thought they'd be simple and universal. But when I tested this on different devices, I quickly discovered that emojis looked completely different on iOS versus Android versus Windows. This inconsistency made the app feel unprofessional.

This feedback pushed me to trial PNG icons instead. I used PixelLab to create custom pixel-art sprites that matched my RPG theme. Now everyone will see the same icons regardless of their device. The feedback was overwhelmingly positive, as end-users said it looked "much more polished" and "like a real game." This experience taught me that sometimes what seems like the easier solution during planning actually creates bigger problems that only become apparent through real-world testing.

The due date priority system went through a similar process. My initial plan was just to colour-code the due date text - red for overdue, yellow for soon, etc. It seemed logical and straightforward. But during trialling, users kept telling me they weren't noticing the due dates at all. One person specifically said they only see the colours when they are actively looking for them, but they don't grab their attention when they are just scanning their tasks.

This feedback made me realize I needed something more visually prominent, so I moved to coloured borders on the quest items themselves. That was better, as people started noticing the colours more readily. But then during further testing, I discovered that static, coloured borders still weren't creating the sense of urgency I wanted for overdue tasks.

That's when I decided to add pulsing animations to the red and orange priority borders. Through several rounds of adjustments based on user feedback, I found that sweet spot where the pulsing was "impossible to miss but not annoying”.

What's really interesting is that this process taught me how each stage of development builds on the previous one. My planning gave me a foundation to work from, but it was the trialling that revealed the real problems users were facing. Then testing allowed me to measure whether my solutions actually worked in practice. Without going through all three stages systematically, I would have ended up with an app that worked fine in theory but failed to engage users in reality.