I'm going to build a "Life RPG" dashboard, a gamified personal habit tracker. Think of it as turning my daily goals and to-do lists into a role-playing game. I'll create a character, define "quests" (which are really just my habits and tasks), and earn experience points (XP) and gold for completing them. As I gain XP, my character will level up. I can then spend my hard-earned gold on custom, real-life rewards that I define myself. This project is perfect for getting my hands dirty with a wide range of MudBlazor components, managing application state, and handling data persistence, all while creating a genuinely useful and entertaining application.

Project: Life RPG - A Gamified Habit Tracker

Week 1: The Foundation & The Hero

- **Day 1: Project Scaffolding and Layout**
  - **Objective:** Get the project running and establish the basic look and feel.
  - **Tasks:**
    1. In VS Code, create a new Blazor WebAssembly project.
    2. Install the MudBlazor NuGet package.
    3. Follow the MudBlazor setup guide: update Program.cs, _Imports.razor, and wwwroot/index.html to include the necessary CSS and JS files.
    4. Design the primary layout in MainLayout.razor. Use <MudLayout>, <MudAppBar>, a persistent <MudDrawer> for navigation, and <MudMainContent> for your pages.
    5. Create the initial pages and link them in the NavMenu.razor: Dashboard, Quests, Character, and Rewards.
- **Day 2: Character Model and Creation**
  - **Objective:** Define what a "character" is and create a page to view their stats.
  - **Tasks:**
    1. Create a Character.cs class in a Data folder. Include properties like string Name, int Level, int CurrentXP, int XPToNextLevel, and int Gold.
    2. Create a singleton service, CharacterService.cs, to manage the character's state. For now, just create a static instance of the Character within this service.
    3. On the "Character" page, use <MudCard> to display the character's stats. Use <MudTextField> to allow the user to set their character's name.
- **Day 3: The Core Mechanic - Leveling Up**
  - **Objective:** Implement the logic for gaining XP and leveling up.
  - **Tasks:**
    1. In CharacterService, create an AddXP(int amount) method.
    2. This method should increase CurrentXP. It must also check if CurrentXP is greater than or equal to XPToNextLevel.

3. If it is, trigger a LevelUp() method. This method should increment Level, subtract the XPToNextLevel from CurrentXP (carrying over any remainder), and calculate a new, higher XPToNextLevel (e.g., XPToNextLevel *= 1.5).
4. On the "Character" page, add a <MudProgressLinear> to visualize the XP progress.

- **Day 4: Data Persistence with Browser Storage**
  - **Objective:** Ensure the character's data isn't lost when the browser is closed.
  - **Tasks:**
    1. Add the Blazored.LocalStorage NuGet package to your project.
    2. Inject ILocalStorageService into your CharacterService.
    3. Create SaveCharacterStateAsync() and LoadCharacterStateAsync() methods in your service.
    4. Call SaveCharacterStateAsync() whenever the character's data changes (e.g., after AddXP).
    5. In MainLayout.razor, call LoadCharacterStateAsync() within OnInitializedAsync to load the data when the app starts.
- **Day 5: Polishing the Character Sheet**
  - **Objective:** Refine the UI for the character page.
  - **Tasks:**
    1. Use <MudGrid> and <MudPaper> to create a more organized layout.
    2. Use <MudIcon> and <MudChip> to make the stats (Level, Gold) more visually distinct.
    3. Implement a simple avatar system using <MudAvatar>. You could have a few predefined options the user can cycle through.

Week 2: The Quest Log

- **Day 6: Quest System - Models and Services**
  - **Objective:** Define the structure for quests and the service to manage them.
  - **Tasks:**
    1. Create a Quest.cs class. Properties should include Guid Id, string Title, string Description, int XPReward, int GoldReward, bool IsCompleted, and an enum QuestType (e.g., Daily, Weekly, Milestone).
    2. Create a QuestService.cs to manage a List<Quest>.
    3. Implement methods: AddQuest, CompleteQuest, DeleteQuest.
    4. Integrate Blazored.LocalStorage to save and load the quest list.
- **Day 7: Creating Quests with a Dialog**
  - **Objective:** Build a user-friendly way to add new quests.
  - **Tasks:**

1. On the "Quests" page, add a <MudButton> with an icon to open a dialog for creating a new quest.
2. Create a CreateQuestDialog.razor component. Use <MudForm> and various MudBlazor input components (<MudTextField>, <MudNumericField>, <MudSelect> for the QuestType).
3. Implement form validation to ensure required fields are filled.
4. When the form is submitted, the dialog should close and pass the new Quest object back to the "Quests" page, which then calls the QuestService to add it.

- **Day 8: Displaying the Quest List**
  - **Objective:** Show all active quests to the user.
  - **Tasks:**
    1. On the "Quests" page, use a foreach loop to render each quest. A <MudCard> for each quest works well.
    2. Inside each card, display the quest's details. Include a <MudCheckBox> bound to the quest's IsCompleted property.
    3. When the checkbox is toggled, call a method to handle quest completion.
- **Day 9: Completing Quests and Reaping Rewards**
  - **Objective:** Connect quest completion to character progression.
  - **Tasks:**
    1. When a quest is marked as complete, call the QuestService.CompleteQuest method.
    2. This method should then call the CharacterService to AddXP and AddGold using the values from the completed quest.
    3. Use the <MudSnackbar> service to provide immediate, non-blocking feedback to the user (e.g., "Quest Complete! +50 XP, +10 Gold").
    4. Completed quests should be visually distinct (e.g., greyed out) or moved to a separate "Completed" list.
- **Day 10: Filtering and Managing Quests**
  - **Objective:** Allow the user to easily sort through their quests.
  - **Tasks:**
    1. Add a <MudChipSet> to the "Quests" page to allow filtering by QuestType (All, Daily, Weekly, etc.).
    2. Implement the filtering logic that updates the displayed list of quests.
    3. Add a delete button (<MudIconButton>) to each quest card to allow for removal.

Week 3: The Treasury and Beyond

- **Day 11: Building the Dashboard**

    - **Objective:** Create a central hub that gives an at-a-glance view of the game.
    - **Tasks:**
        1. Design the "Dashboard" page using <MudGrid> for a responsive layout.
        2. Create a "Character Summary" component that shows the character's name, level, and XP progress bar. Reuse this on the dashboard.
        3. Add a "Today's Quests" card that displays only incomplete quests of type Daily.
        4. Make the dashboard the default landing page of the application.
- **Day 12: The Reward Store**
    - **Objective:** Create a system for spending gold on real-life rewards.
    - **Tasks:**
        1. Create a Reward.cs model (string Title, string Description, int Cost).
        2. Create a RewardService.cs to manage a list of user-defined rewards, again persisting to local storage.
        3. On the "Rewards" page, build a UI to add/edit/delete rewards. Use the same dialog pattern you used for quests.
- **Day 13: Purchasing Rewards**
    - **Objective:** Implement the logic for buying a reward.
    - **Tasks:**
        1. Display the list of available rewards on the "Rewards" page, each in a <MudCard>.
        2. Each card should have a "Purchase" button. The button should be disabled if the character's Gold is less than the reward's Cost.
        3. When purchased, deduct the gold from the CharacterService and show a <MudSnackbar> confirmation.
- **Day 14: Theming and Polish**
    - **Objective:** Make the application look and feel like a cohesive product.
    - **Tasks:**
        1. Create a custom theme using MudThemeProvider. Define a color palette (Primary, Secondary, etc.) that fits a "game" aesthetic.
        2. Ensure a consistent look across all pages. Check for proper spacing, alignment, and use of elevation with <MudPaper>.
        3. Add transitions and animations where appropriate to make the UI feel more dynamic.
- **Day 15: Final Review and Refactoring**
    - **Objective:** Clean up the codebase and hunt for bugs.
    - **Tasks:**
        1. Review all your services. Is there any duplicate code that could be abstracted?

2. Test the application thoroughly. What happens if you try to complete a quest offline? What happens if local storage fails to load?
3. Add comments to your code and ensure your component structure makes sense for future expansion.