****

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**WINDOWS PROGRAMMING**

**2025 – 2026**

**PROJECT-1 REPORT**

**DELIVERY DATE**

11/11/2025

**PREPARED BY**

05210000296 - Ali Osman Taş

05210000260 – Kutlu Çağan Akın

**Project Overview**

**Project Context**

We developed a prototype for "LifeHub," a personal productivity and wellness concept. This project consists of four separate, single-screen .NET MAUI applications, each addressing a core domain of the concept.

The DashboardApp acts as the main navigation menu for the other modules. The HabitTrackerApp allows users to add daily habits via an Entry, mark them as complete using a CheckBox, and manage the list. The MoodJournalApp lets users record their mood using a picker and add an optional note. Finally, the PlannerApp functions as a to-do list where users can assign tasks to a specific day using a DatePicker and see completed items struck through (using an IValueConverter).

**Project Objectives**

Our primary objectives for this project were aligned with the course requirements :

* To apply modern .NET MAUI layout principles (Grid, VerticalStackLayout, and Styles) to create aesthetic and consistent interfaces.
* To demonstrate strong data binding capabilities between XAML and C# code-behind, specifically using ObservableCollection, BindingContext, and the required ICommand interface .
* To integrate the 10 core UX Laws (Fitts's, Hick's, Miller's, etc.) into our design decisions for each application.
* To establish a consistent visual language and user experience (UX) across all four applications.

**Contribution of Students**

This project was completed by a 2-person team. The workload was distributed as follows:

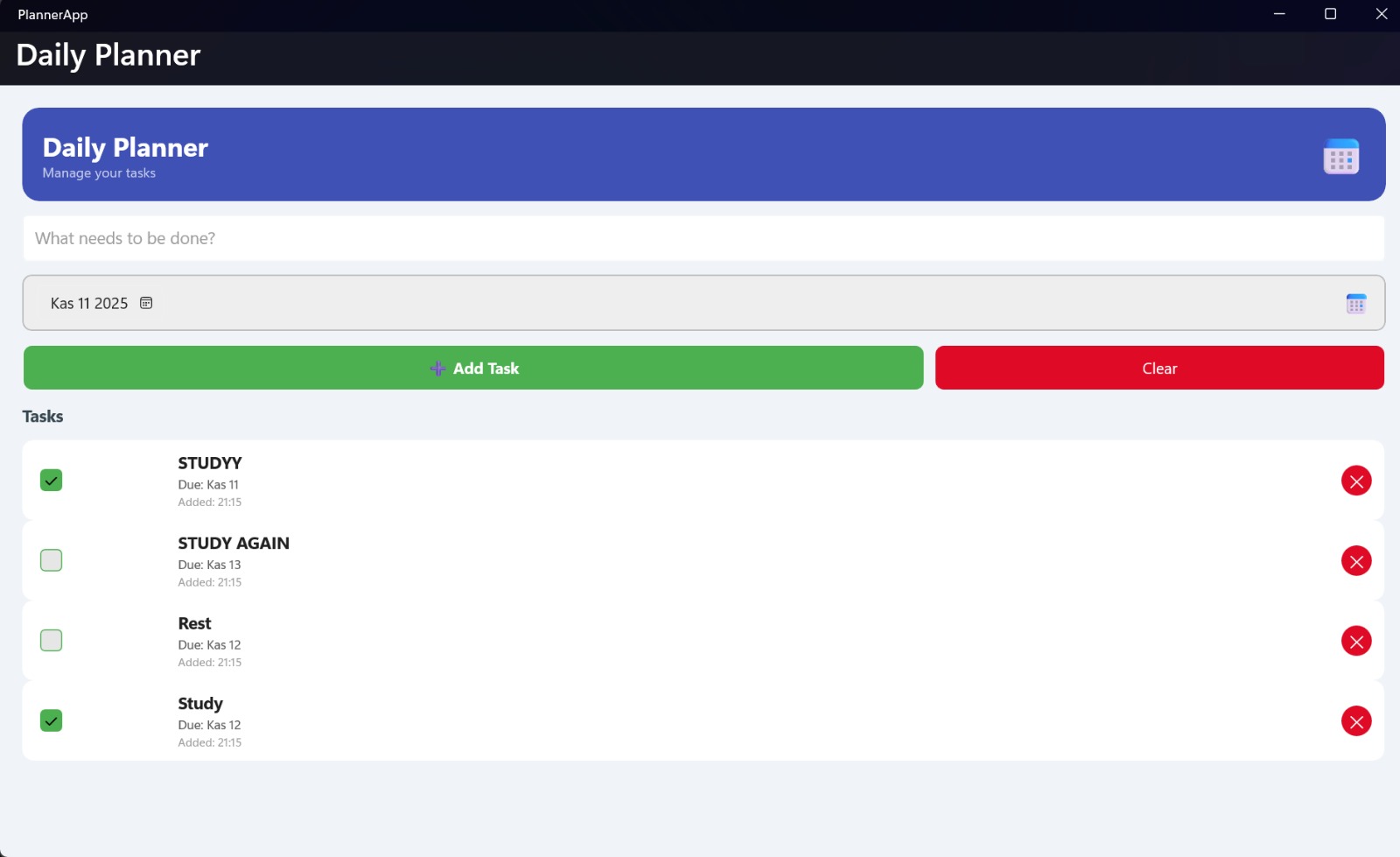
**Ali Osman Taş**: I was responsible for the C# logic and XAML design of the PlannerApp and the MoodJournalApp. This included implementing custom C# logic like the IValueConverter (for the task strikethrough effect), handling the data binding for the DatePicker and Picker controls, and creating the modern XAML interfaces based on UX laws . I also managed the technical setup and troubleshooting for our team on the macOS environment, which involved configuring the .NET 9.0 SDK, correcting .csproj files for Mac/Windows compatibility, and resolving complex Android SDK path errors (like Hata 127 and XA5207) to enable deployment.

**Kutlu Çağan Akın**: I was responsible for the development of the DashboardApp (the main menu) and the HabitTrackerApp. This included implementing the core navigation logic (distinguishing between ICommand and Clicked events) in the dashboard, and writing the data manipulation logic for the ObservableCollection in the HabitTrackerApp (add, delete, and the '7-item limit' feature). I also played a key role in defining the common aesthetic decisions (color palette, style resources) used across all projects.

**UX Law Implementation**

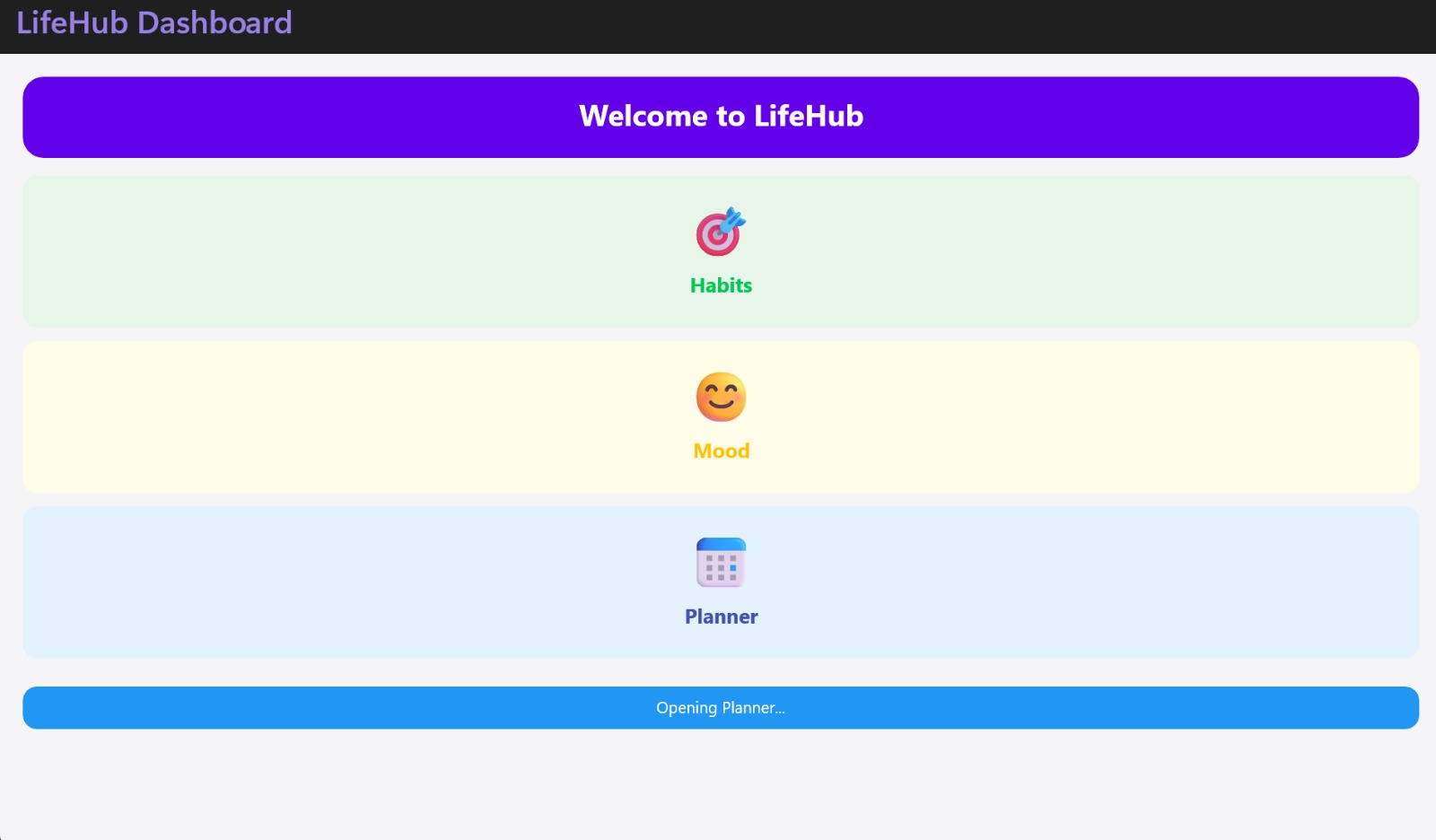
1. **Fitts’s Law:**

All primary action buttons ("Add Task," "Clear All") and dashboard cards were designed as large, easy-to-tap targets.



1. **Hick’s Law:**

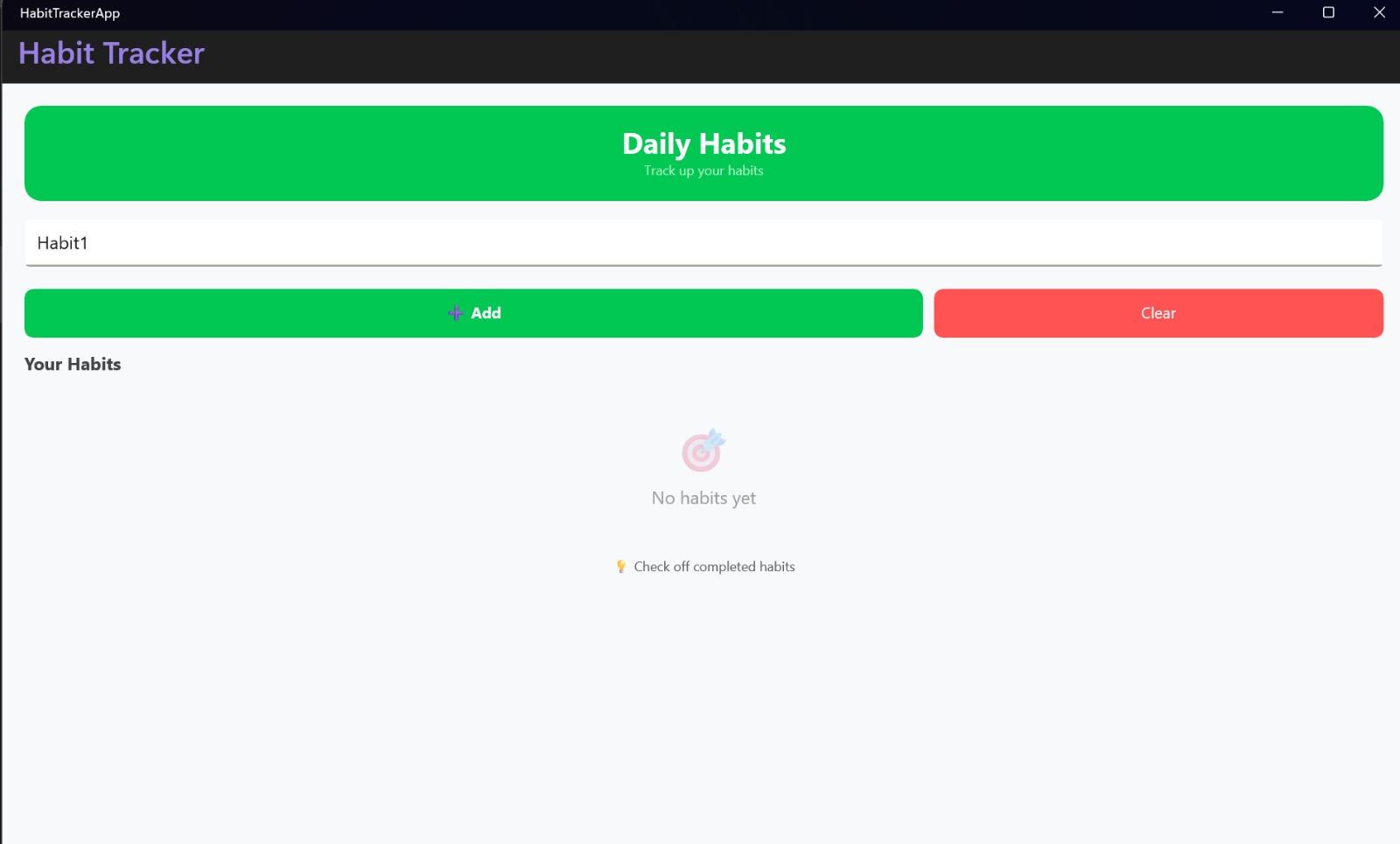
The DashboardApp presents the user with only three clear, distinct choices (Habit, Mood, Planner).

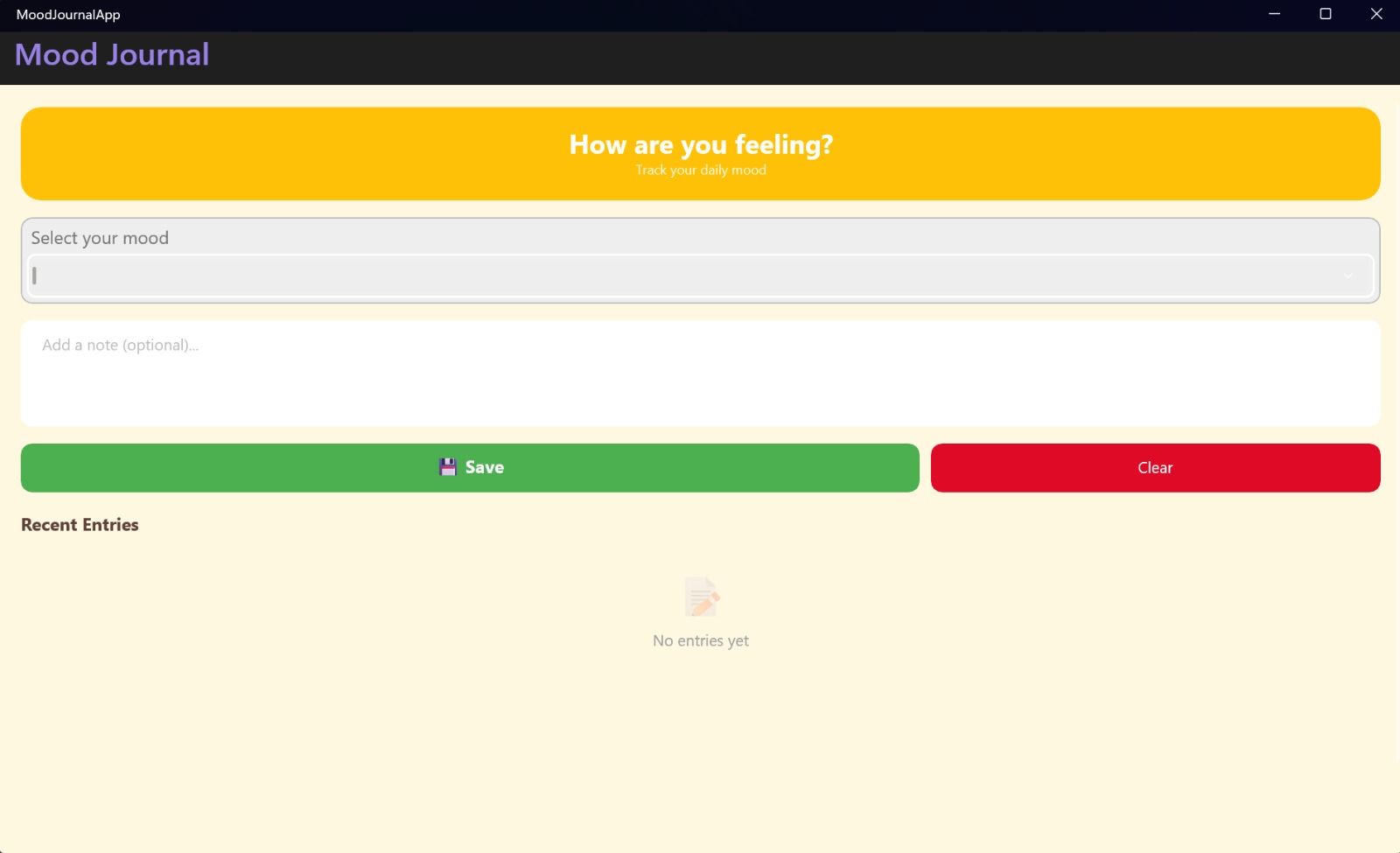


The time it takes to make a decision increases with the number of choices . By limiting the main menu to three options, we minimized cognitive load and prevented "choice paralysis," allowing the user to navigate faster.

1. **Aesthetic-Usability Effect**

We used a consistent design system (Style resources) across all four apps, featuring Border with CornerRadius, consistent Shadow effects, and a unified color palette.

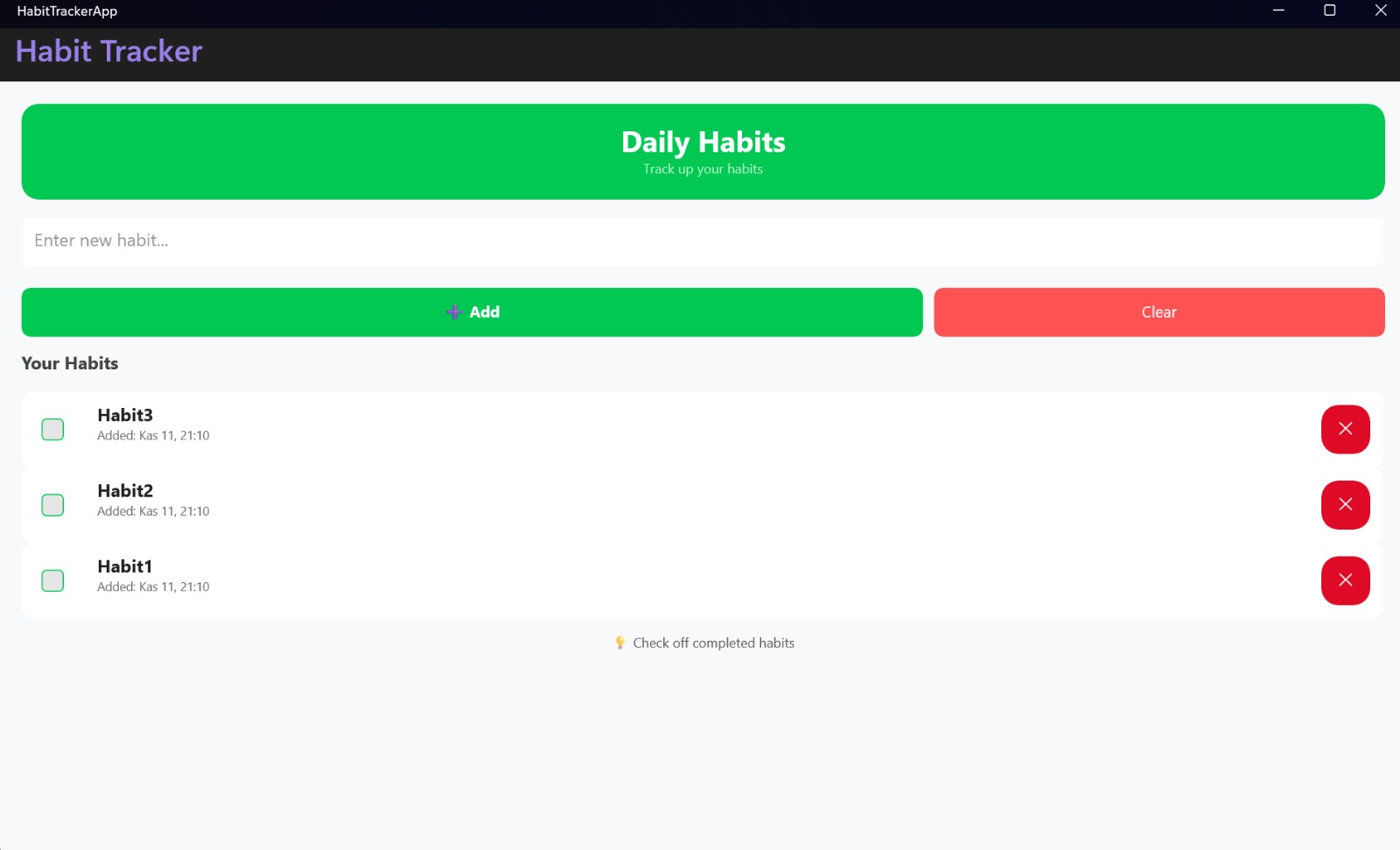




Users perceive aesthetically pleasing designs as more usable . Our clean, consistent, and modern UI builds trust and makes users more tolerant of any minor usability issues

1. **Jakob’s Law:**

Our PlannerApp and HabitTrackerApp lists follow a familiar pattern: CheckBox on the left, task description in the center, and a Delete button on the right.



Users expect your app to work like other apps they already know . This standard "to-do list" layout matches the user's existing mental model, making our apps immediately intuitive and easy to learn

1. **Gestalt Principles:**

In PlannerApp and MoodJournalApp, we grouped all input controls (e.g., Entry, DatePicker, Picker) inside a single, visually distinct <Border> (card).

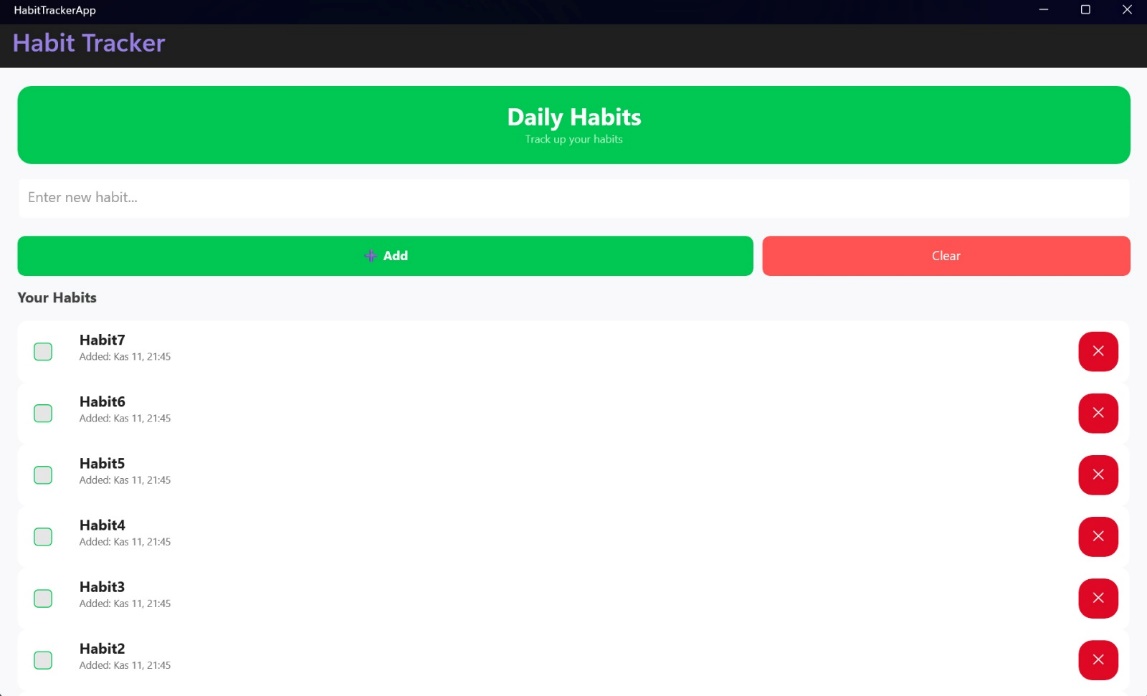
metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

This is the Principle of Common Region. By enclosing related items in a boundary, the user's brain perceives them as a single functional unit ("Add Task"), separating them from the "List" area and making the UI easier to scan

1. **Miller’s Law:**

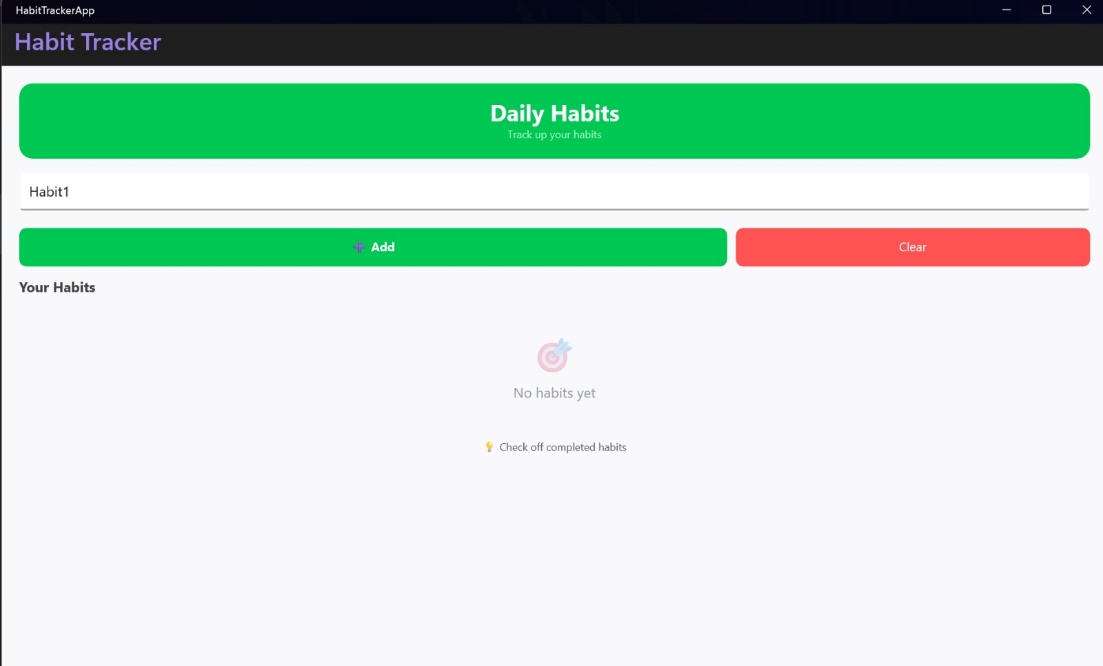
In the C# code for HabitTrackerApp, MoodJournalApp, and PlannerApp, we programmatically limit the list to 7 items (e.g., if (Habits.Count > 7) Habits.RemoveAt(Habits.Count - 1);).



The average person can only keep 6 (±2) items in their working memory. By "chunking" the list to the 6 most recent entries, we prevent cognitive overload and make the list feel manageable

1. **Von Restorff Effect:**

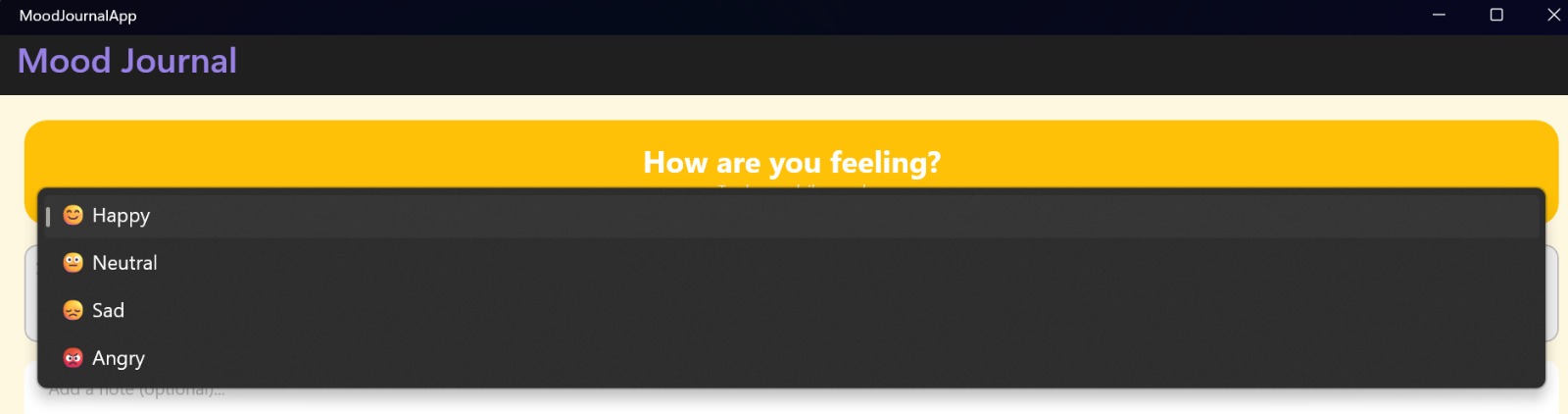
In all apps, the primary action ("Add") is a solid, positive color (e.g., green #27AE60), while the destructive action ("Clear") is a visually distinct, contrasting style (e.g., red transparent #E74C3C).



The item that stands out is the one most easily remembered . This color contrast highlights the primary call-to-action ("Add") while also making the dangerous "Clear" button distinct, which helps prevent user error.

1. **Tesler’s Law:**

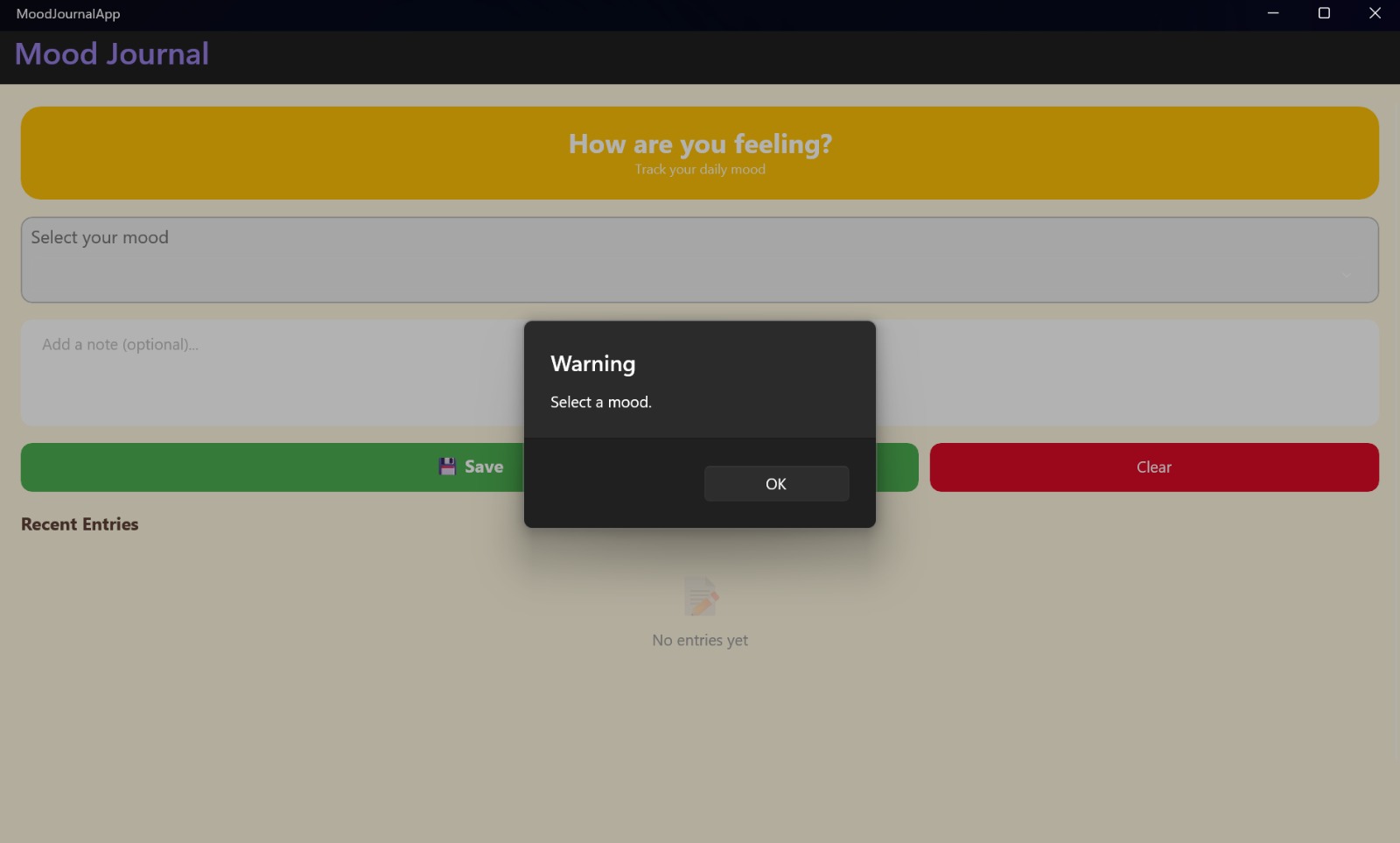
In MoodJournalApp, the user can leave the note field blank. The system handles this by auto-populating "No note" instead of showing an error



All processes have a core complexity . We moved the complexity of handling an "empty" (but optional) note from the user (forcing them to enter something) to the system (which provides a sensible default)

1. **Postel’s Law:**

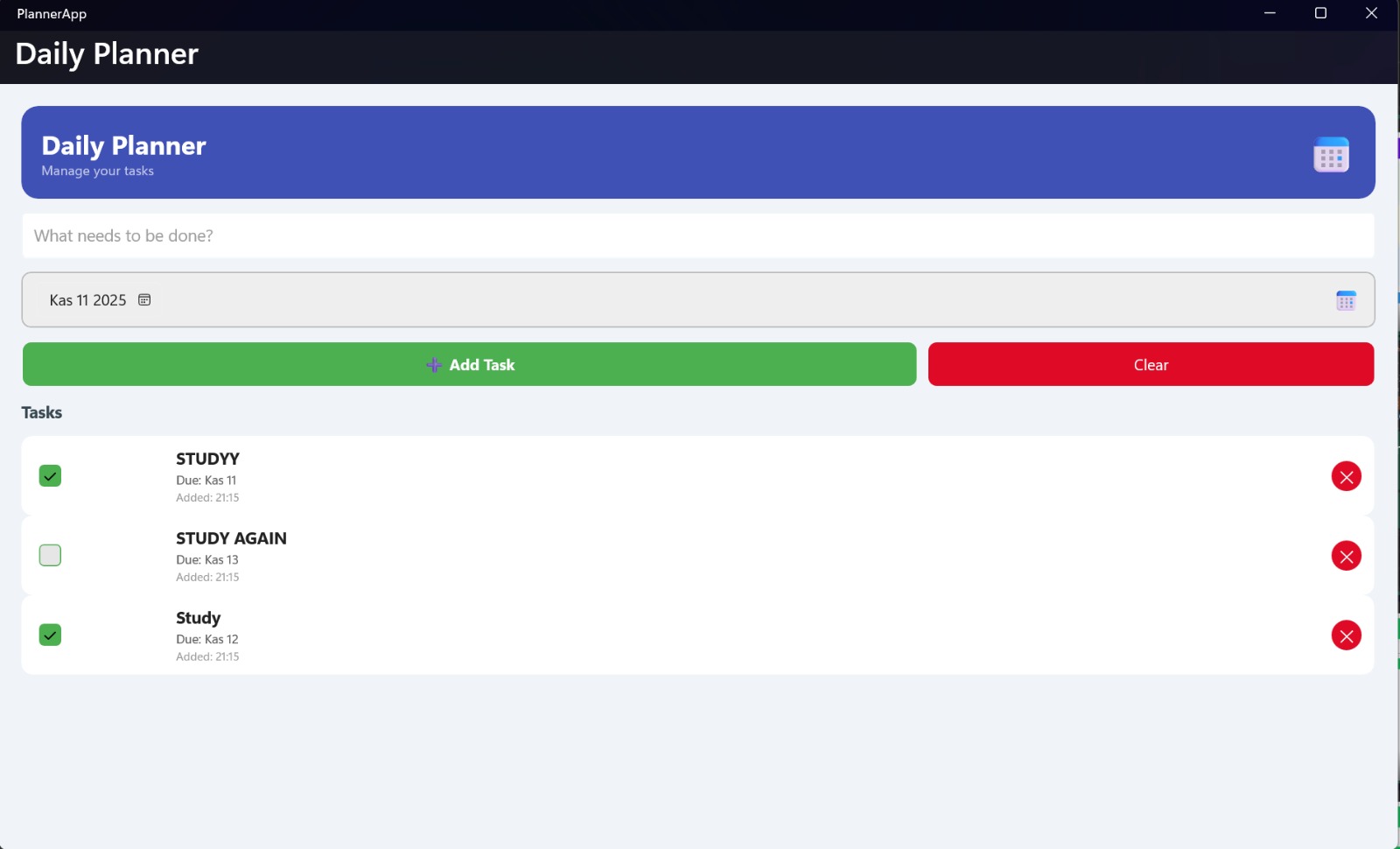
In PlannerApp, if the user tries to add a task with an empty title (string.IsNullOrEmpty(title)), the app does not crash. It gracefully handles this "bad" input by showing a DisplayAlert warning.



Be liberal in what you accept". We accept the user's "imperfect input" (clicking "Save" with no selection) without crashing, and we are "conservative in what we send" (preventing the invalid data from being added) by showing a helpful alert.

1. **Doherty’s Treshold:**

In HabitTrackerApp, MoodJournalApp, and PlannerApp, we used ObservableCollection for all lists. When a user adds an item, it appears in the CollectionView instantly.



Productivity soars when the system responds in <400ms . Using ObservableCollection provides this immediate feedback, assuring the user their action was successful and keeping them "in the flow."

**Layout and Design Decisions**

Our design philosophy was centered on creating a clean, consistent, and predictable user experience across all four applications, adhering to the core principles of cognitive psychology discussed in our course materials Statistics Area: Labels that display summary information, such as completion rates.

Our primary layout structure for all four applications (DashboardApp, HabitTrackerApp, MoodJournalApp, PlannerApp ) was the VerticalStackLayout. Given that each app is a single screen with a clear top-to-bottom information flow (Header - Input -List), this layout provided the most straightforward and maintainable structure.

However, we did not place controls directly into the main VerticalStackLayout. To apply the Gestalt Principles (specifically *Common Region*), we "chunked" related UI elements by grouping them inside <Frame> elements, which function as visual cards . For example, in PlannerApp and MoodJournalApp , all input controls (Entry, DatePicker, Picker) are grouped in a single "Input" <Frame> , visually separating them from the CollectionView list . Where horizontal alignment was needed for multiple buttons (e.g., "Add" and "Clear" in HabitTrackerApp ), we utilized a <Grid> with proportional column definitions (ColumnDefinitions="2\*,\*" ) to ensure balanced and responsive spacing.

To support the Aesthetic-Usability Effect , we ensured that while each app has a unique theme, the underlying typography is consistent. All four projects are configured in MauiProgram.cs to register and use the "OpenSans-Regular" and "OpenSans-Semibold" fonts. We also used distinct, high-contrast colors for primary and secondary actions. This serves the Von Restorff Effect : in each app, the "safe" action (e.g., "Add") and the "destructive" action (e.g., "Clear") , are given sharply different colors (e.g., Green vs. Red ) to immediately signal their different functions to the user.

Our layout was also driven by cognitive limits. The MoodJournalApp is a prime example of Hick's Law . Instead of an open Entry field for mood, we provide a Picker with only four pre-defined options ("Happy," "Sad," etc.). This simplifies the decision-making process, reducing cognitive load . Similarly, in the C# code for HabitTrackerApp, MoodJournalApp, and PlannerApp, we explicitly limit the ObservableCollection to 7 items (e.g., if (Habits.Count > 7) Habits.RemoveAt(Habits.Count - 1); ). This is a direct implementation of Miller's Law , which states that working memory is limited (to roughly 4-7 items), preventing cognitive overload.

Finally, we adhered to Jakob's Law by using familiar patterns. The list structure in all data-driven apps (HabitTracker , Planner ) follows a standard "to-do list" pattern: a CheckBox on the left, primary text in the center, and a Delete button on the right . This matches the user's existing mental model , making our apps instantly intuitive and usable.

# **Functional Features**

Our project implements all core functional requirements specified in the project definition . A summary of features across the four applications is provided below.

**Feature: Dynamic List Display**

**Description**: Displays user-created tasks, habits, or moods in a scrollable list . Implementation Details: We used a CollectionView in HabitTrackerApp, MoodJournalApp , and PlannerApp. The ItemsSource property was bound to an ObservableCollection<T> (e.g., Habits, Entries, Tasks) defined in the code-behind.

**Feature: Data Input**

**Description**: We provided users with a variety of controls to enter data, as required by the "User Input Variety" objective . Implementation Details: We used a simple Entry (x:Name="habitEntry") for HabitTrackerApp. For MoodJournalApp, we used a Picker (x:Name="moodPicker") bound to a string collection . For PlannerApp, we used both an Entry (x:Name="taskEntry") and a DatePicker (x:Name="datePicker") .

**Feature: Data Manipulation (Add, Delete, Clear) Description: Users can add**

**new items,** delete individual items, and clear the entire list in all three data-driven apps. Implementation Details: All data manipulation is handled by ICommand bindings. For example, PlannerApp uses AddTaskCommand, DeleteTaskCommand, and ClearTasksCommand. The DeleteTaskCommand is an ICommand<T> that receives the specific item to be deleted via CommandParameter="{Binding .}".

**Feature: Navigation (Event vs. ICommand)**

**Description:** The DashboardApp provides a main menu to access the other modules . Implementation Details: We implemented this using TapGestureRecognizer. This app demonstrates both interaction models: the "Habits" card is bound to an ICommand (OpenHabitCommand), while the "Mood" and "Planner" cards use traditional event handlers (OnMoodClicked, OnPlannerClicked).

**Feature: Input Validation (Postel's Law)**

**Description:** The system gracefully handles invalid or missing user input without crashing, fulfilling Postel's Law . Implementation Details: In PlannerApp, if the user tries to add an empty task, the AddTask method checks if (string.IsNullOrEmpty(title)) and shows a DisplayAlert. In MoodJournalApp, an empty note is handled by the system saving a default value of "No note".

**Feature**: Timestamping Description: All habits, tasks, and moods are timestamped upon creation. Implementation Details: The AddHabit, AddTask, and AddEntry methods assign DateTime.Now to the model. This is then displayed in the CollectionView using XAML's StringFormat (e.g., StringFormat='Added: {0:MMM dd, HH:mm}' ).

**Feature:** Data Conversion Description: In PlannerApp, completed tasks (IsCompleted = true) are visually struck through. Implementation Details: We implemented a custom IValueConverter named BoolToTextDecorationConverter. This converter is registered as a resource in App.xaml and bound to the Label.TextDecorations property in the DataTemplate.

**Binding and Command Demonstration Summary**

This section demonstrates how we fulfilled the project's core technical requirements: implementing data binding and using at least one ICommand bound from XAML . We implemented these patterns across all four applications. We present one advanced ICommand example from PlannerApp and one foundational data binding example from HabitTrackerApp.

**1. ICommand Example (with CommandParameter)**

The ICommand interface was used to separate UI interaction from business logic, making our code cleaner and fulfilling **Tesler's Law** . Our most comprehensive implementation is the "delete task" feature in PlannerApp , which uses a generic ICommand<T> to pass data.

**C# Code-Behind (PlannerApp/MainPage.xaml.cs):** First, we defined a generic ICommand property that expects a PlannerTask object. In the constructor, we initialized this command, pointing it to our DeleteTask method. The BindingContext was set to this to allow XAML to find the command.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**C# Logic (DeleteTask method):** The DeleteTask method accepts the PlannerTask object passed from the XAML binding and removes it from the main ObservableCollection.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**XAML View (PlannerApp/MainPage.xaml):** Inside the CollectionView's DataTemplate, the Button binds its Command property to the DeleteTaskCommand defined on the page. Crucially, it uses CommandParameter="{Binding .}" to pass the specific PlannerTask object of that row directly to the command.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**2. XAML Binding Example (ItemsSource)**

Data binding was used in all data-driven apps to automatically synchronize the UI with our code-behind data. A clear example is the HabitTrackerApp .

**C# Code-Behind (HabitTrackerApp/MainPage.xaml.cs):** We defined a public ObservableCollection<HabitModel> named Habits. This specific collection type is essential because it automatically notifies the UI of any changes (adds or removes), fulfilling the **Doherty Threshold** by providing instant feedback.

metin, ekran görüntüsü, yazılım içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**XAML View (HabitTrackerApp/MainPage.xaml):** In the XAML file, we bound the CollectionView's ItemsSource property directly to the Habits collection in our code-behind. We did not need to set the ItemsSource manually in C#; setting the BindingContext = this; was enough for the XAML binding to find its source.

metin, ekran görüntüsü, yazı tipi, menü içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.

**Reflection on Design and UX**

The LifeHub application suite successfully integrates all ten required UX laws while maintaining design simplicity and a consistent aesthetic. Our layout decisions, based on a primary VerticalStackLayout scaffold and a <Frame>-based card system , were designed to avoid clutter, present users with minimal choices (Hick's Law) , and group related information logically (Gestalt Principles) .

Implementing command-based interactions (ICommand) for all primary data actions (Add, Delete, Clear) provided a robust and maintainable alternative to procedural event handlers. This separation of concerns was a core objective . The DashboardApp explicitly demonstrates this contrast by using both an ICommand (for Habits) and standard Tapped events (for Mood/Planner).

The result is a set of four functional, visually harmonious prototypes that align with modern mobile design philosophy and successfully demonstrate all key technical deliverables of the LifeHub specification

**UI/UX Notes**

**Applied UX Laws and Principles:**

**Ali Osman Taş - PlannerApp & MoodJournalApp**

**1. Hick's Law (in MoodJournalApp):** To streamline the core function of the app—logging a mood—we intentionally limited the user's choice. Instead of an open text field that could cause "choice paralysis," we implemented a Picker control . This Picker offers only four distinct, pre-defined options ("Happy," "Neutral," "Sad," "Angry"). This application of Hick's Law transforms the task into a quick, two-tap process, reducing cognitive load and encouraging daily use.

**2. Postel's Law (in PlannerApp):** We designed the system to be "liberal in what it accepts" from the user. The AddTask method in PlannerApp does not crash if the user tries to add a task with an empty title. Instead, it checks the input (string.IsNullOrEmpty(title)) , prevents the invalid data from being added, and shows a helpful DisplayAlert warning ("Enter a task."). This robust error handling prevents crashes and guides the user.

**3. Tesler's Law (in MoodJournalApp):** We moved inherent complexity away from the user and into the system. The "notes" field for a mood entry is optional. If the user leaves it blank (string.IsNullOrWhiteSpace(noteEntry.Text)), the system automatically substitutes a default value ("No note") upon saving. This fulfills Tesler's Law by allowing the user to complete the core task (logging a mood) with minimum effort, without being forced to deal with optional complexity.

**4. Jakob's Law (in PlannerApp):** We utilized a universally familiar interaction pattern for our task list . When a task's CheckBox is ticked, the IsCompleted property is updated, which is passed to our custom BoolToTextDecorationConverter. This converter applies a TextDecorations.Strikethrough (üstü çizili) style. This "strikethrough on complete" visual feedback is a standard convention in almost all to-do list apps, making our app's behavior immediately intuitive.

**5. Gestalt Principles (in PlannerApp & MoodJournalApp):** We applied the **Principle of Common Region** to organize our inputs. In both apps, all related controls (Entry , DatePicker , Picker ) are grouped inside a single, visually distinct <Frame> element. This "card" visually separates the "input" area from the "list" area, allowing the user's brain to perceive all inputs as one single functional unit.

**Kutlu Çağan Akın - DashboardApp & HabitTrackerApp**

**1. Miller's Law (in HabitTrackerApp):** The average person can only keep about 7 (±2) items in their working memory. We applied this law directly in our C# logic. The AddHabit method checks the list count after adding a new item and enforces a limit (if (Habits.Count > 7) Habits.RemoveAt(Habits.Count - 1);). This "chunking" ensures the list never becomes an overwhelming, infinitely scrolling burden, keeping the user focused.

**2. Doherty Threshold (in HabitTrackerApp):** This law states that productivity soars when a system responds in <400ms. We achieved this by using an ObservableCollection<HabitModel> for our list. When the user hits the "Add" button, the CollectionView (which is bound to this collection ) updates instantly, without any lag or need for a manual refresh. This immediate feedback makes the app feel responsive and fast.

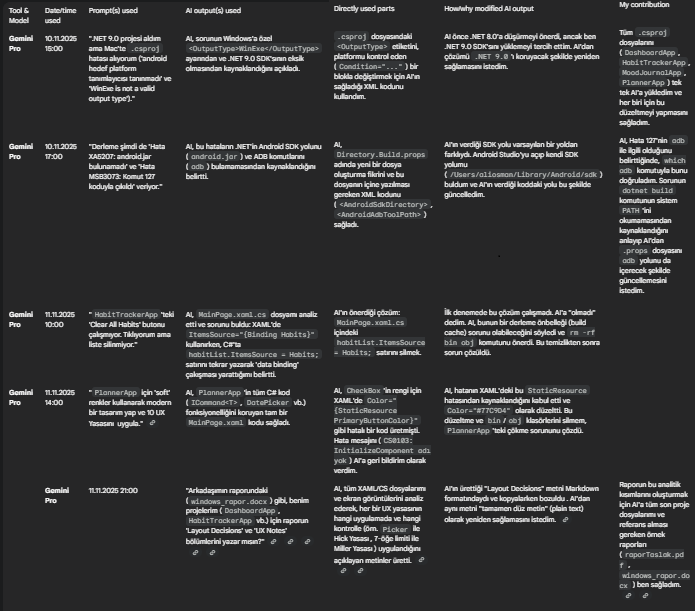
**3. Von Restorff Effect (in HabitTrackerApp):** We used color and visual distinction to guide the user. The primary action ("Add") is a solid, positive green (#00C853), while the destructive action ("Clear") is a contrasting red (#FF5252) . This makes the "Clear" button visually unique, fulfilling the law that distinct items are more easily remembered. This contrast highlights the main action and prevents accidental deletion.

**4. Fitts's Law (in DashboardApp & HabitTrackerApp):** We made targets large and easy to tap. In DashboardApp, the navigation links are not small text but large, full-width <Frame> elements (cards) . In HabitTrackerApp, the "Add" and "Clear" buttons have a large HeightRequest="44" to be easily reachable, reducing miss-taps and improving efficiency, as Fitts's Law dictates.

**5. Aesthetic-Usability Effect (in DashboardApp):** The DashboardApp serves as the user's first impression. We used polished <Frame> elements with CornerRadius , consistent spacing, and large, clear icons (🎯, 😊, 📅). This aesthetically pleasing design builds user trust and creates a positive emotional response, which makes the entire application suite *feel* more usable and professional.

**AI Usage Logs**

**Ali Osman Taş**



**Kutlu Çağan Akın**metin, menü, ekran görüntüsü, yazı tipi içeren bir resim

Yapay zeka tarafından oluşturulmuş içerik yanlış olabilir.