Project Write-Up

Ruslan Mukhamedvaleev

Habit Garden

The writing in this PDF is my original work for a habit tracking app, Habit Garden. The project can be found entirely on GitHub (https://github.com/digitalRM/HabitGarden)

Goal

My goal in creating this app was to develop a tool that allows me to track my personal habits in a more engaging way than simply throwing them into the reminders app.

A couple of my friends and I tried this idea out at a hackathon a few years back, but it didn't work out, so I decided to redesign the entire idea from scratch and make it just for me early this year.

The Problem

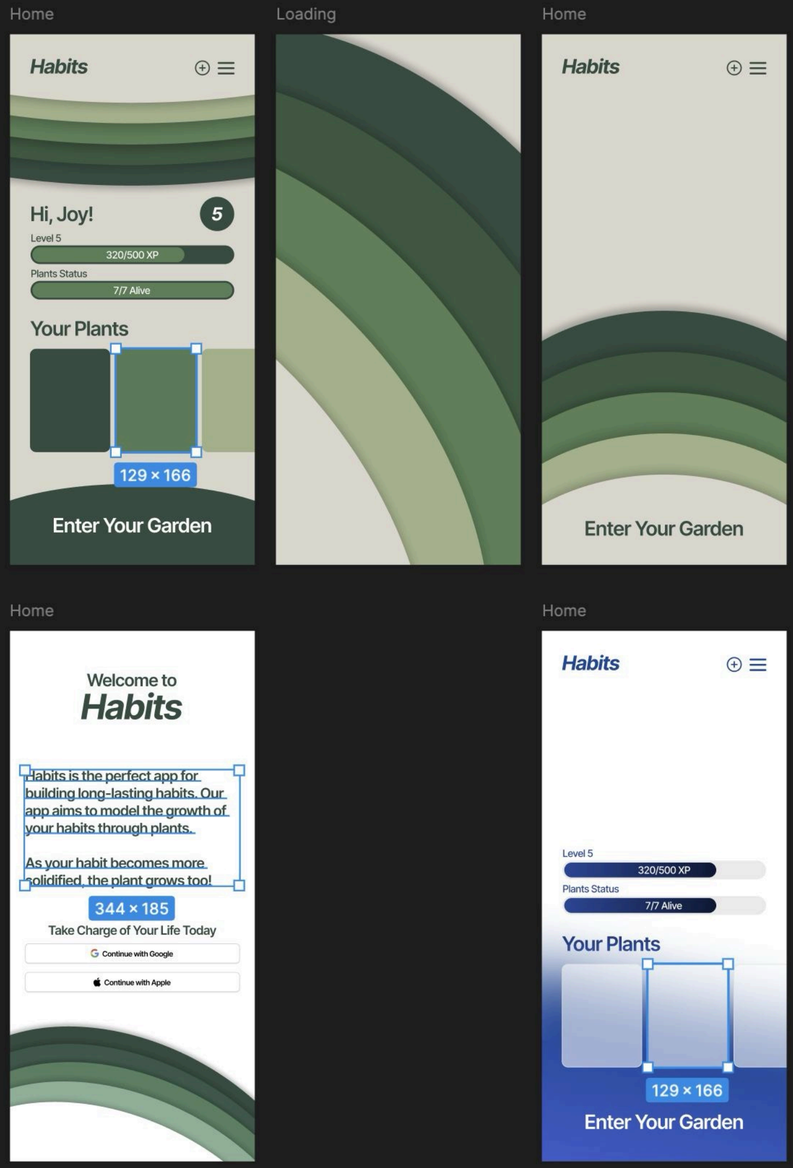
- Many habit tracking apps out there are cluttered with ads or impose arbitrary limits on the number of habits you can track, how many times you can engage with your habits, or have customization options locked behind paywalls.
- So, I decided that building my own app here would be the best solution to all three of these problems. It would allow me to tackle all of these issues, add other features whenever I need them, and have comfort in knowing that all my data is stored on-device.

Ideation

- I considered creating creating a web app, but I immediately dismissed that, as it would have too much friction in its use. My goal with this app is to minimize friction as much as possible, as my current solution for tracking habits/goals is either the reminders app or post-it notes on my bedroom door (haha).
- My second thought was to use React's sister framework, React Native, for building native-like (an important distinction) apps with a coding experience similar to React, at least at first glance. I've worked with react-native before, but it's shockingly different and limited compared to its more developed sister version on the web.

The Solution

- I finally landed on building a Swift app.
- Swift's main drawbacks, it's Apple device only compatibility and my unknowing of the language, were not actually drawbacks in this situation.
    - For the first point, I was going to be the only user of this app so cross compatibility was not a concern.
    - For the second, I've been meaning to try and learn Swift for a few years, and I thought that this would be the perfect first step in learning more about the language. I always like a challenge.



Before jumping into building, I revisited the old designs I made for the hackathon my friends and I competed at.

I knew I wanted it to have a different feel.

Habit Garden Project Write-Up

Mukhamedvaleev 06

Building

- Building the actual app was very interesting, with a lot of my time spent to learn Swift's quirks. I kept the Apple developer docs open at all times. The actual app is pretty simple architecturally. It starts with an AppState at its root with an onboarding gate (not at all needed since I was the only user, but I wanted it to feel like an Apple app).

- Most of the app is built out of various views that interact with the main HabitStore (the data structure responsible for storing habit and visitor information locally). I built on a lot of helper functions (adding/removing/completion/editing/etc) to the HabitStore that are used throughout the app.

Interactions/Architecture

Habit Store

habits (Published)
lastLevelUp (Published)
currentVisitor (Published)

addHabit(habit)
completeHabit(id)
updateHabit(id, ...)
deleteHabit(id)
adjustTime(days)
clearAllHabits()
spawnVisitor(type)
claimVisitorBoostForAllHabits()

HabitCreationSheet

Calls: addHabit() based on user's input.

The is the sheet that appears when you click the "+" button either in the top right corner or on the empty screen.

ContentView (which is like the entire app's wrapper) gets latest state updates whenever HabitStore is updated (for freshest data to display).

ContentView

Reads: habits, currentVistor
Calls: adjustTime(), clearAllHabits(), spawnVisitor()

HabitCardView observes lastLevelUp state to determine if/when the habit levels up to display LevelUpView.

HabitCardView

Calls: completeHabit(id)
If habit levels up, calls LevelUpView.

This is each individual habit card that the user sees. The user can click into the card or mark it as complete.

HabitDetailView

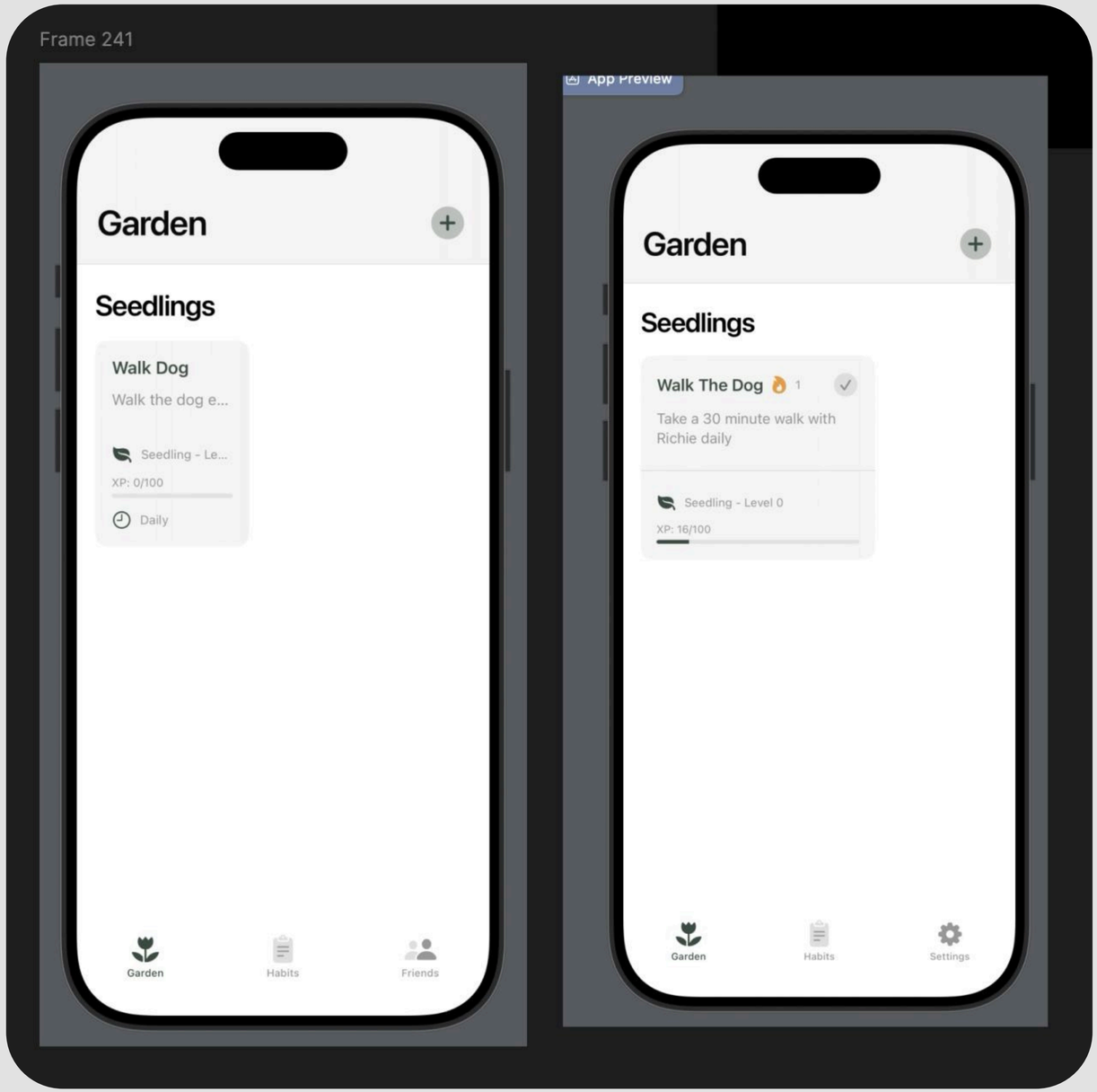
Calls: completeHabit(id), updateHabit(id), deleteHabit(id)

This is the detailed page that users can enter for each habit. From here, they can view/edit information for their habit

LevelUpView and VisitorView are just notifications that a user has leveled up. The LevelUpView view doesn't call anything and is instead triggered by the HabitCardView, which observes the level-up. The VisitorView functions in a similar way, though the user has to "accept" the XP from the visitor for it to be added, which does involve a call. This is why they are not diagramed here.

Early Version

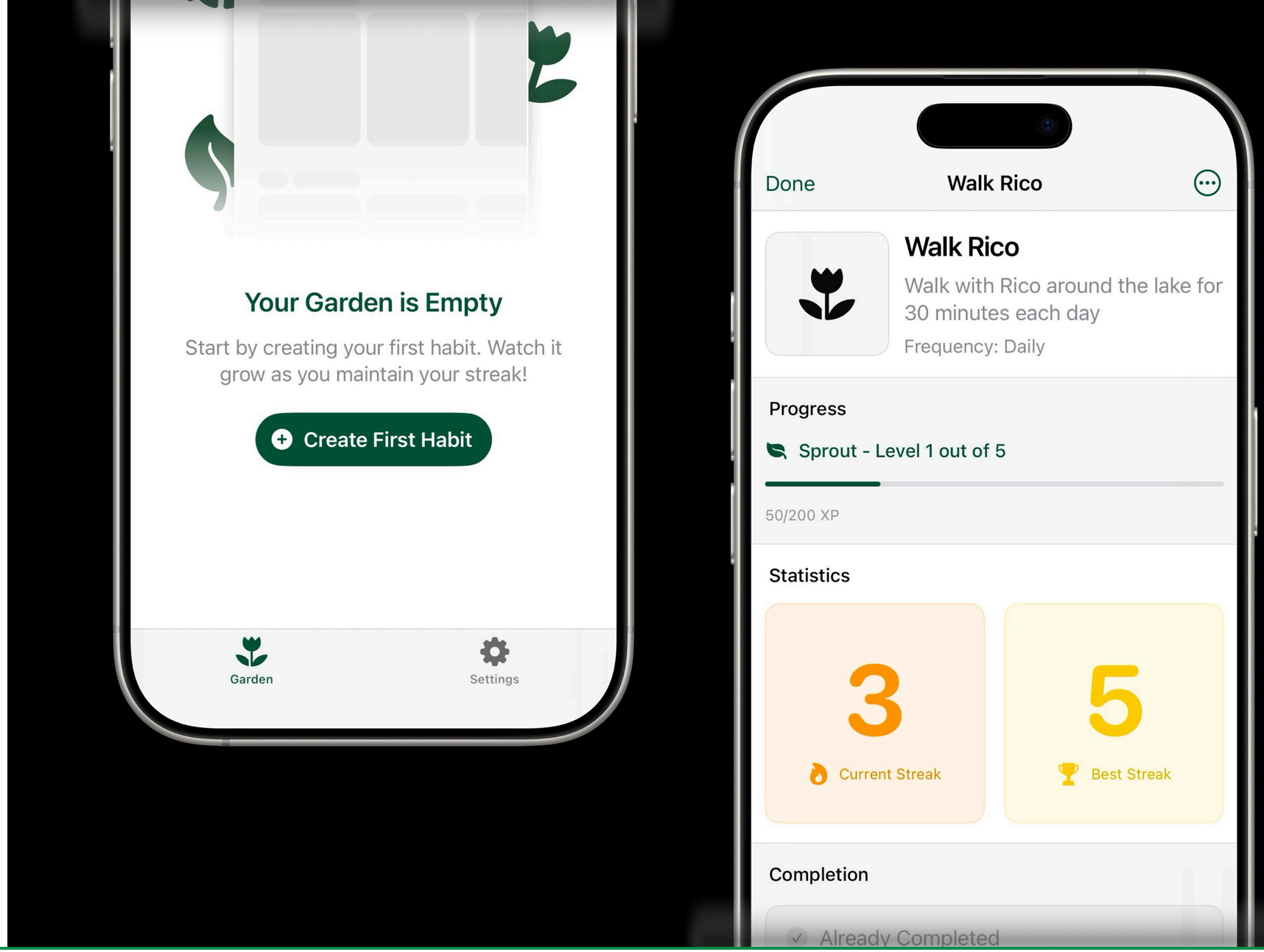
- I focused a lot of my development time, early on, building out the robust backbone because I knew that it would be the most complex part of the app, and so I sort of neglected the UI for a little while.
- I did come back to it, though, creating cleaner-looking habit cards and fun graphic assets in Figma for the empty states!

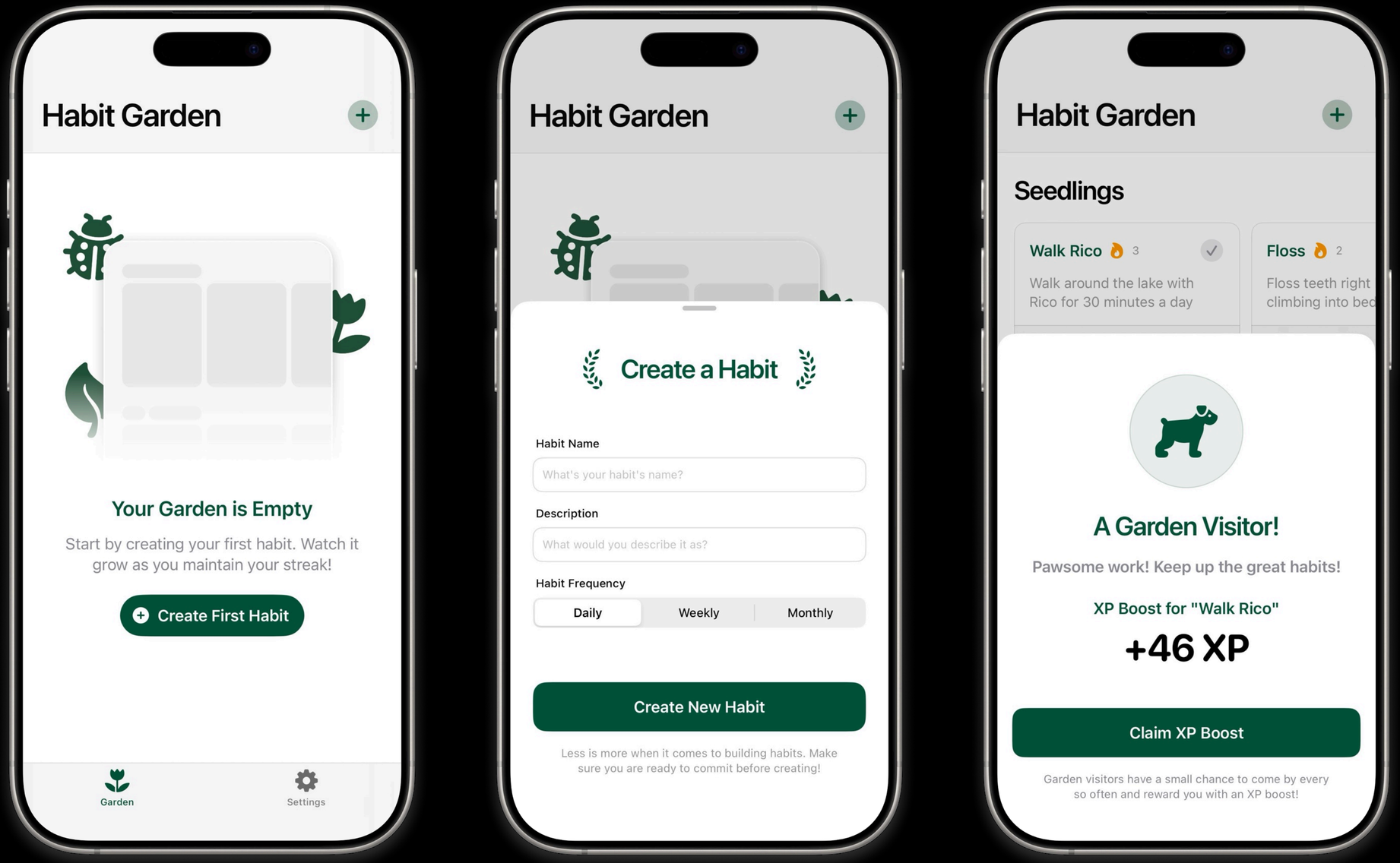


The App

Here is how the final app turned out! I've used it to track my habits, and it seems to work better than any of my prior systems (although I still love using Post-it Notes on my door).

Code on GitHub: https://github.com/digitalRM/HabitGarden




Habit Garden
Your Garden is Empty
Start by creating your first habit. Watch it grow as you maintain your streak!
Create First Habit
Garden
Settings
Habit Garden
Create a Habit
Habit Name
What's your habit's name?
Description
What would you describe it as?
Habit Frequency
Daily
Weekly
Monthly
Create New Habit
Less is more when it comes to building habits. Make sure you are ready to commit before creating!
Habit Garden
Seedlings
Walk Rico 3
Walk around the lake with Rico for 30 minutes a day
Floss 2
Floss teeth right
climbing into bed
A Garden Visitor!
Pawsome work! Keep up the great habits!
XP Boost for "Walk Rico"
+46 XP
Claim XP Boost
Garden visitors have a small chance to come by every so often and reward you with an XP boost!