

Design of the Belt

Pre-Design Needs

What is the object?

The object is a smart belt buckle. It can be a fitness device that is small, portable, and concealable. If it's covered by a shirt, nobody has to know you're wearing it!

The buckle can be attached to any normal belt, and can easily be moved to something else.

Interviews

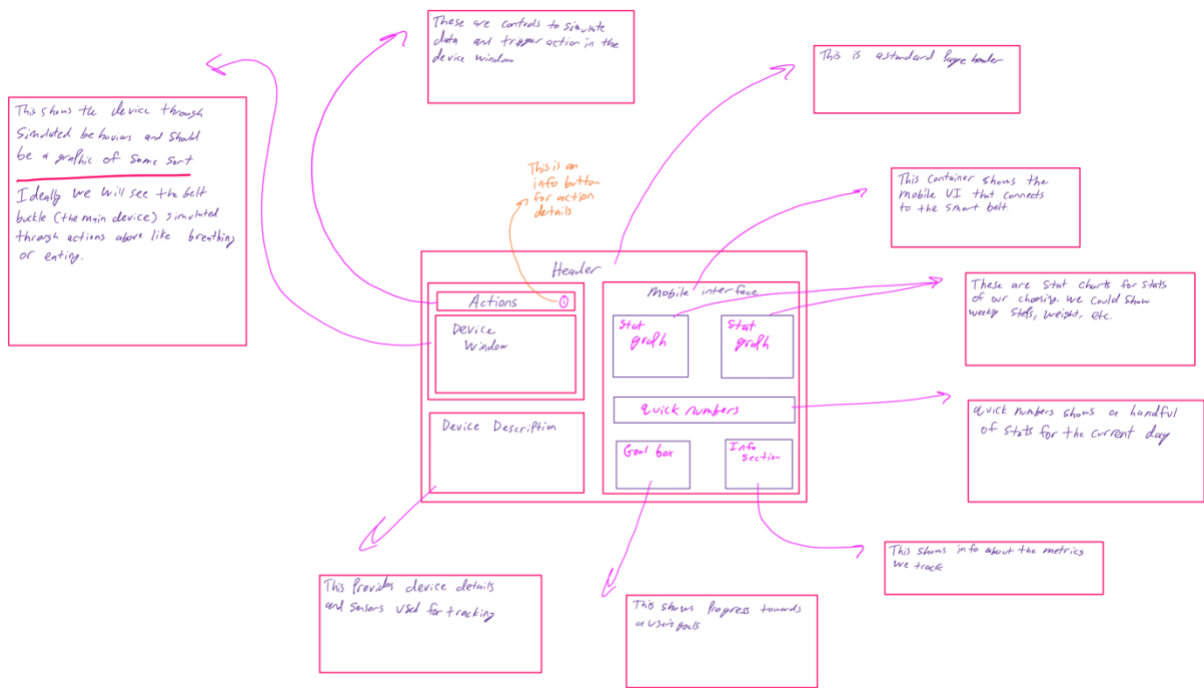
Interview questions:

- How would you describe your style and priorities when choosing accessories?
- How often do you wear a belt?
 - What do you usually wear belts with?
- Tell me about the last time you used a belt. What are your observations about the process?
- Do you adjust your belt during the day?
- If a belt, or belt buckle, had some form of smart capabilities, where would that fit in to your life?
- How important is aesthetic when choosing a belt?
 - Do you typically show it off, or do you hide it under a shirt?
- Given a list of actions, tell me which would be useful to you and why.

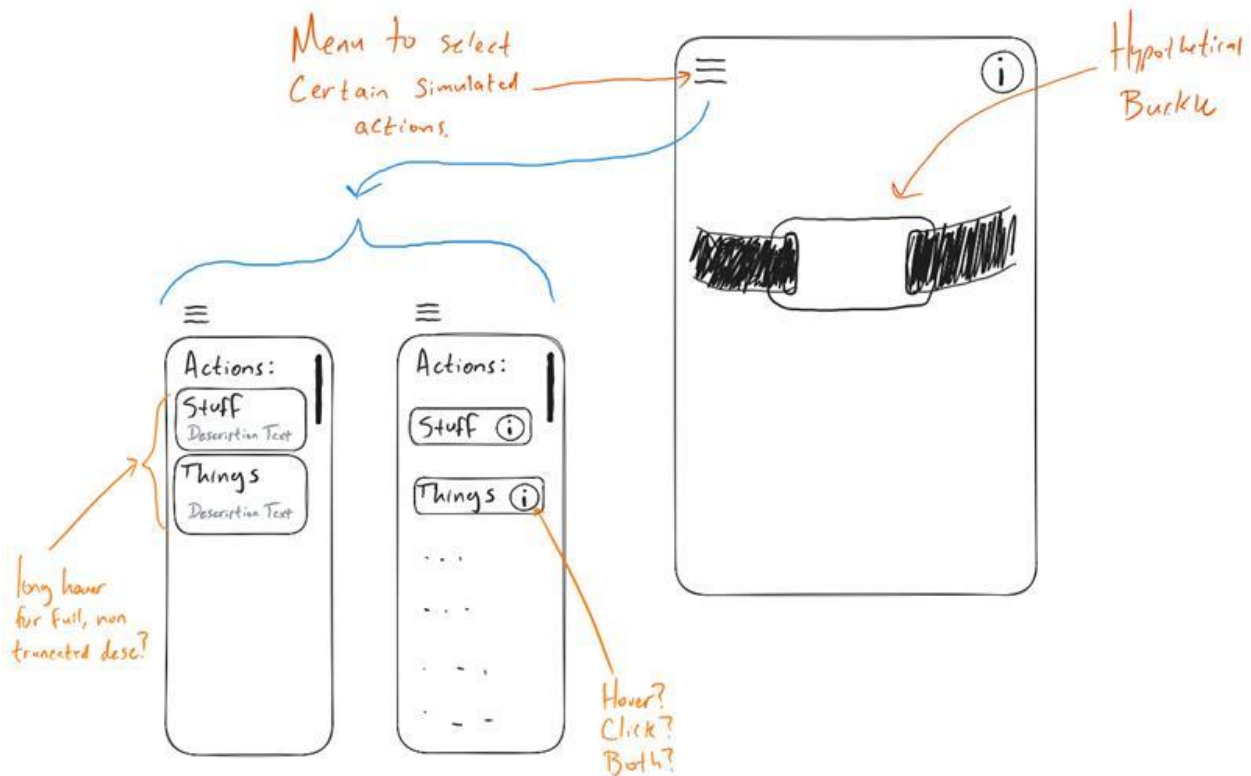
Our interview responses devolved into two distinct user groups:

- Group A: The Active. This group is made up of those who get out and about. These individuals might not want a piece of activewear on the wrist. These users would want to track fitness oriented stats, like breaths in a certain timespan, number of steps in a day, or distance run.
- Group B: The Working Adult. This group consists of adults who have office jobs. Most offices are business casual, or even business formal. Either way, you'll likely be wearing a belt. These users would want something that can track how sedentary they are, giving them that push they need to stand up just a little more often.
- Group C: The Casual. This group would wear a belt casually, with a normal outfit. They would want to show it off, or make it a visual component of their outfit. Fitness capabilities would not be paramount to this group, as they would be much more focused on quality-of-life things. They might want a waist measurement feature, or some way to detect unusual movement.

Sketches



Device View Panel



Evaluation

We presented these images to three separate people. Responses were generally constructive on where to improve elements of the interface.

Notes on their responses follow:

Person A

Person A was much more invested in the idea of the device panel. It seems like most of their feedback was aimed there.

- They were not a fan of the "rough edges." Rounded makes the UI feel more friendly, and more like a card.
- They believe a hamburger icon is a bit unclear for the context of the menu. Since it's actions and a dropdown, how about someone running with an arrow pointing down?
 - Interviewer note: This is a great idea. I'll change this
- Asked what the belt going to do in the device viewer. They feel it doesn't need to be there unless there's some form of animation...

Person B

Person B talked a lot about the mobile interface. The design was very rough at the stage we showed the sketch off, so a lot of these details were already iterated on and reworked by this point.

- Likes the more compact actions dropdown.

- On the note of the actions menu, they felt it should be a pop-in from the side as opposed to a dropdown.
- They asked what the belt is going to do in the UI.
- What are the graphs going to show?
- How "mobile" is the interface?
- What other possible stats could be tracked here?

Person C

Person C asked us more about the device panel. They thought it was the most expanded of our sketches.

- They liked the actions menu with the descriptions (non-compact)
 - They thought the full description on hover was a good idea
- They asked if the belt would be a 3-D model
 - Kyle responded likely not, it's more or less outside the scope of the project.
- We were asked about the info buttons, and what they'd do. Specifically, they asked if the info buttons would expand a field or show a pop up.

Interface Description in Detail

Quick overview



This project is a Svelte + Vite web app that simulates a "Smart Belt" device. The Mobile Interface displays charts, an inline weekly summary (Quick Numbers), goals progress, and metric information. The Device simulator lets you trigger actions (10s Walk, Stairs, Sedentary, Do Not Disturb simulation) that update the in-memory data store.


User-facing interface (what you see and how it behaves)

Mobile Interface (Mobile-Interface.svelte)

The mobile interface is what the user primarily interacts with. It includes power control, connection indicators, data visualizations, and goal tracking.

Header

- Title: Displays "Mobile Interface:".
- Power Button: Toggles device power using `DeviceStatus.changePower()`. Once pressed, it initiates a connection sequence and disables briefly (1.5 seconds) to prevent repeated presses.
- Connection Status Indicator: Shows one of three states depending on `DeviceStatus.getStatus()`:
 -  Connected
 -  Connecting

-  Disconnected
The simulation only functions when the device is powered on and connected. The connection automatically establishes after powering on.

Charts

- Displays StepChart and StairChart side by side.
- Each chart shows the last seven days of data for its metric, where the current day is highlighted on the far right in blue and the goal is shown in green.
- Charts are rendered using Chart.js and include hover effects for detailed values.

Weekly Summary

- Rendered by the SummaryTable component.
- Displays total and average per day for Breath Count, Step Count, and Stair Count.
- Includes a “Metric Info” button that opens a dialog explaining each metric and how it is tracked.
-

Goals

- Rendered by GoalsTable.
- Shows current progress toward daily goals for Steps (default = 6000) and Stand Minutes (default = 50).
- Includes numeric values, progress bars, and percentages.
- Layout dynamically adjusts for small screens using media queries.

Simulation View (Device.svelte)

The simulation provides an interactive visualization of the device’s behavior and tracked activity. It features a context bar, a simulation dropdown, and a live SVG display.

Top Bar / Context Bar

- Contains:
 - Dropdown Menu with simulation actions, each paired with an info dialog opened via InLineDialogOpenButton.
 - Status Box showing Activity (Posture), Calories Burned, and Heart Rate — visible when the device is powered on and connected.
 - Info Button that provides general information about the simulation and overall UI.




Simulation Actions

Each simulation option alters the interface and data visualization in real time.

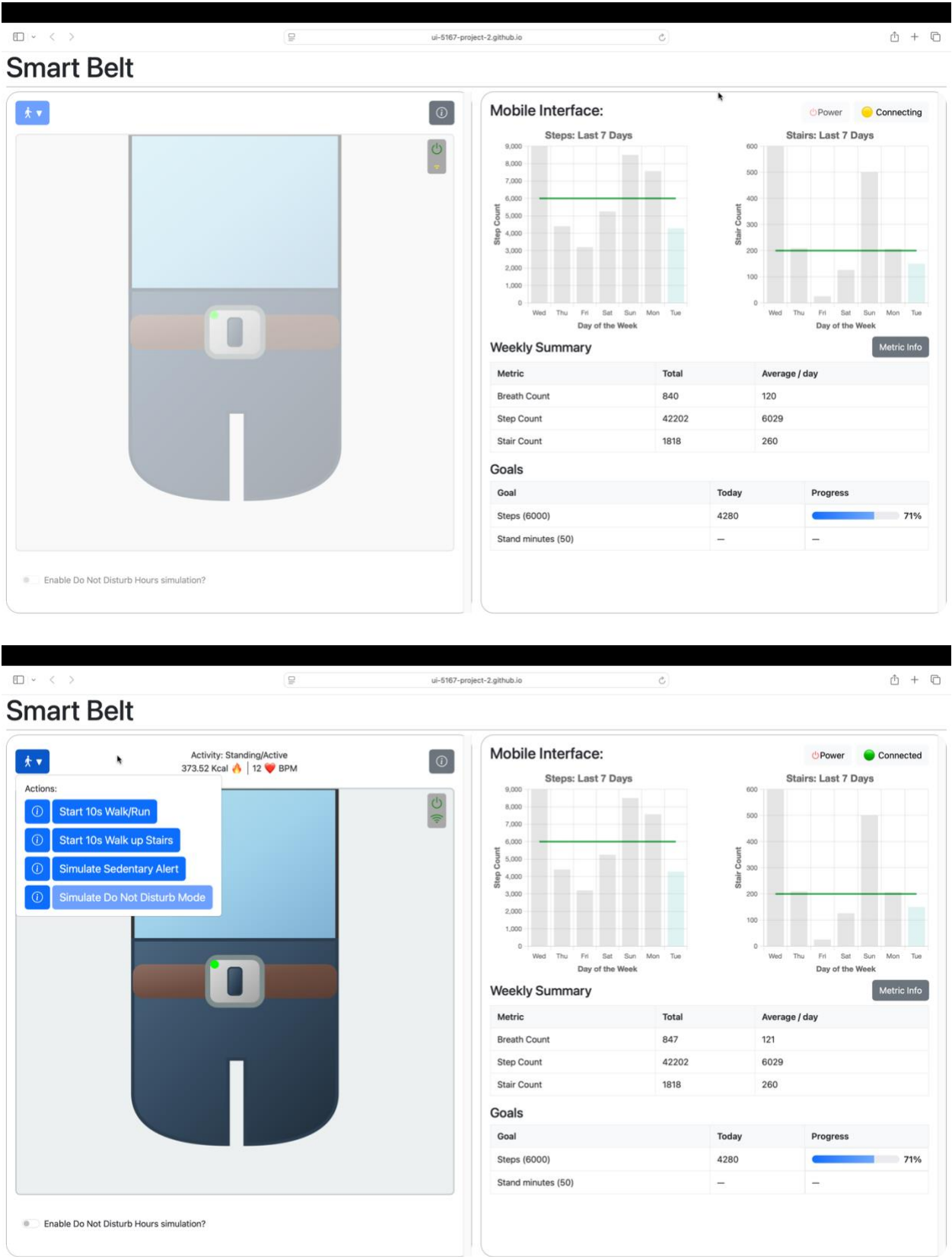
- Start 10 s Walk/Run (`handleWalk()`):
 - Simulates a 10-second walk.
 - Increments Stand Minutes by 10 seconds (≈ 0.1667 minutes).
 - Gradually increases Step Count by 150 steps over 10 seconds and saves the data with `BeltData.updateDay()`.
 - Adjusts breathing rate and posture status, then resets after a short delay.
- Start 10 s Walk Up Stairs (`handleStairs()`):
 - Similar to Walk/Run but also updates Stair Count and increases Stand Minutes using a defined multiplier.
- Simulate Sedentary Alert (`handleSedentary()`):
 - Triggers a visual posture warning after a timeout to indicate inactivity.
 - Alters colors and posture status temporarily before resetting.
- Simulate Do Not Disturb Mode (`handleDoNotDisturb()`):
 - Allows users to set Do Not Disturb (DND) hours for work or sleep via the `NumberLine` component.
 - During DND hours, the belt is inactive and no new data is recorded; outside those ranges, random steps may be added.
 - Time ranges cannot overlap and represent hours the user should not receive notifications.
 - The simulation displays the current time of day and color changes in the belt graphic to indicate DND status.

Simulation actions and dropdown buttons are disabled when the device is off or disconnected. Additionally, they're disabled when another action is currently running.

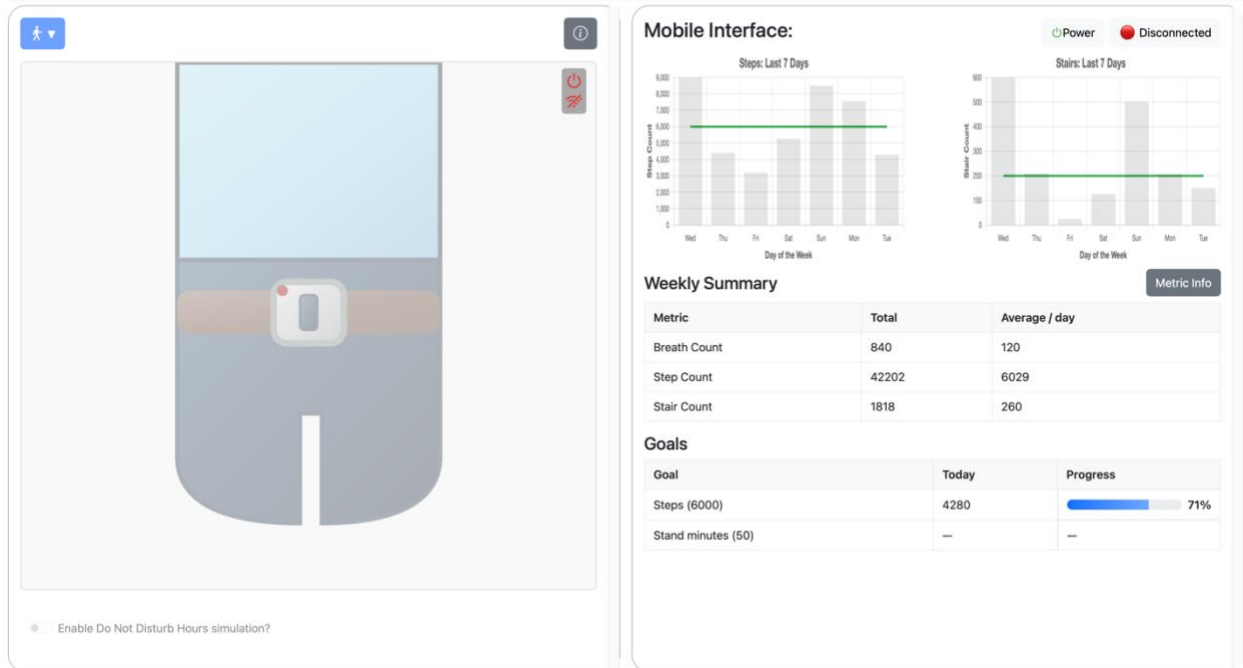
Device Visualization

- The `DeviceDemonstration` component renders an SVG showing the torso and upper legs of a person wearing the belt.
- The power and connection indicators appear both in the top-right corner and on the belt buckle:
 -  On/Connected
 -  Off/Disconnected
 -  Connecting/Disconnecting
- The SVG animates a breathing effect. When powered off, the visualization becomes opaque and no data is collected.
- The parent component passes colors, fills, and element references for animation control via `registerElements`.
- Below the visualization is a space for extra controls to go. Currently, this is used for the do not disturb hours component via `NumberLine`.

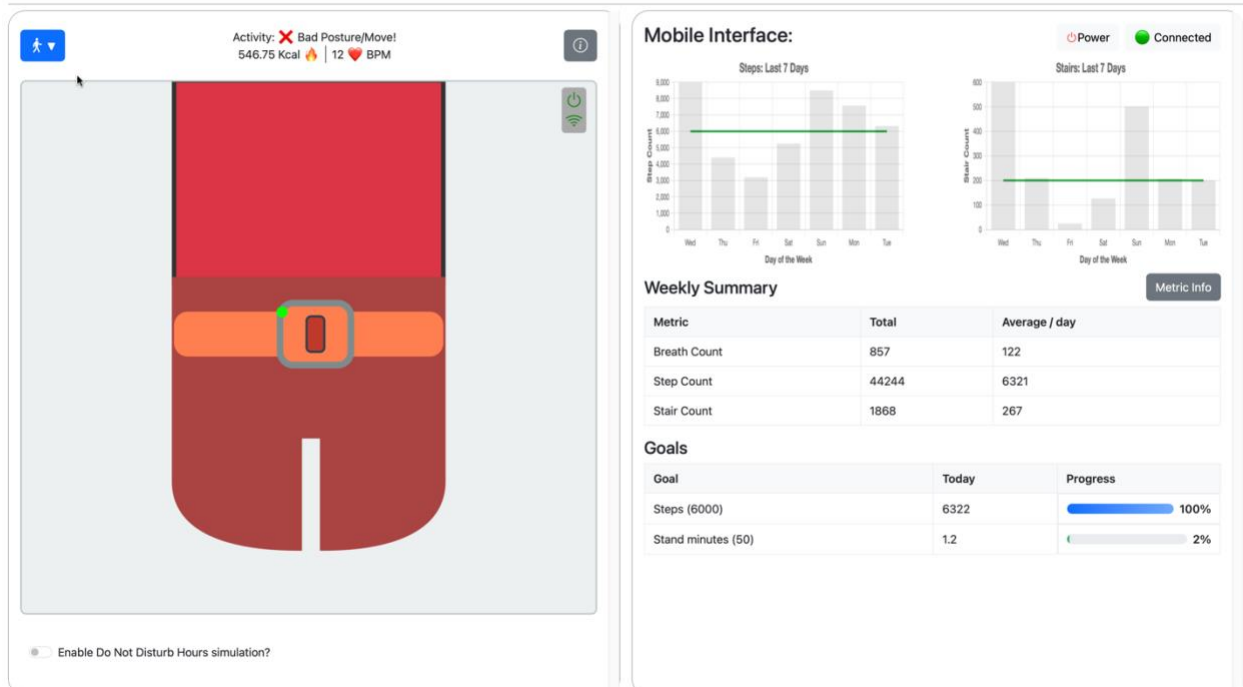
Screenshots



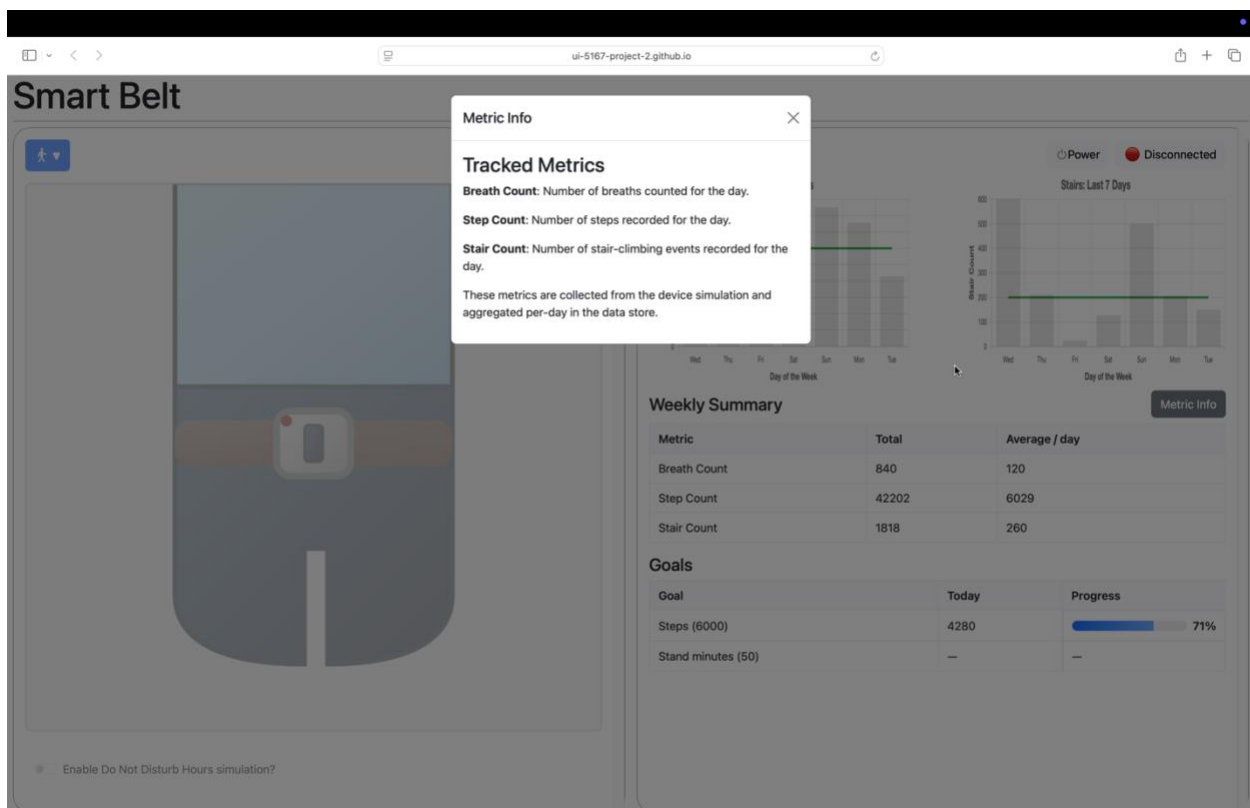
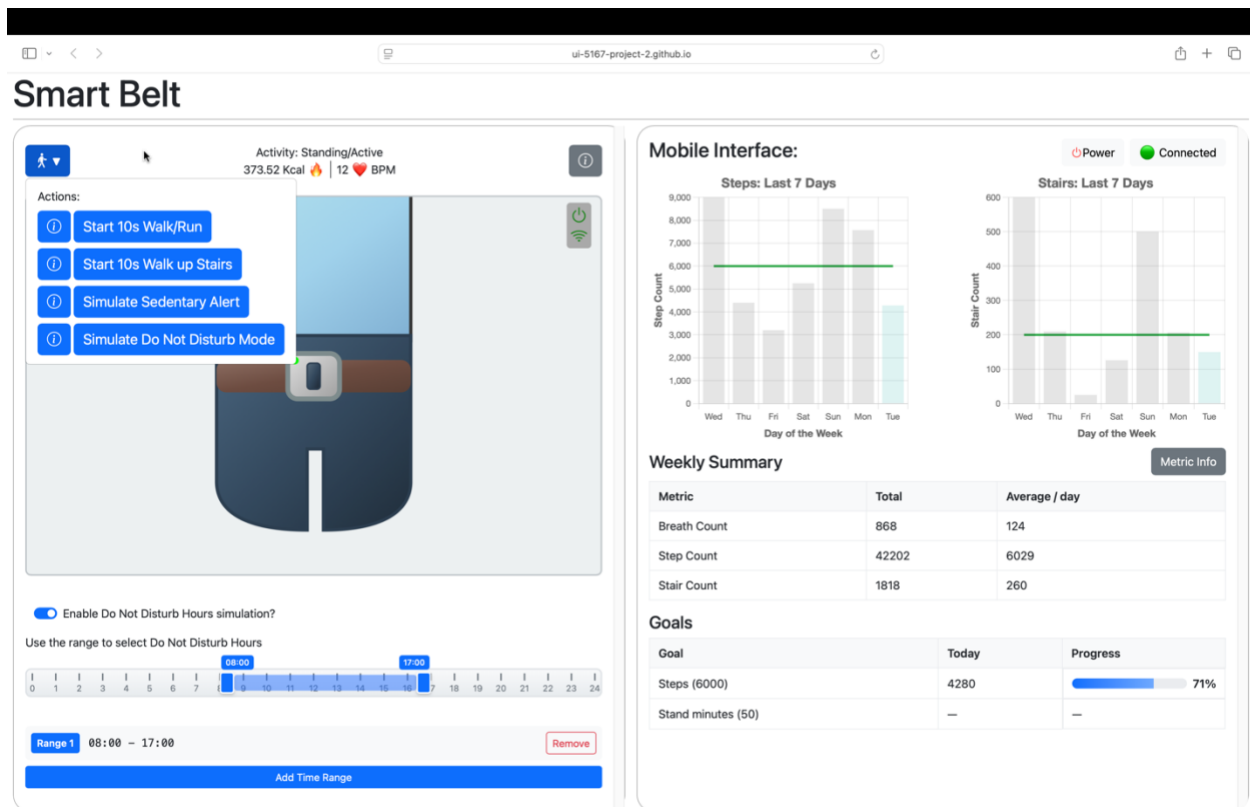
Smart Belt



Smart Belt







Smart Belt

Smart Belt Simulation Guide

The "Smart Belt" is a simulated wearable device designed to track activity and posture. This interface allows you to test the belt's various operational states and data outputs without physical interaction.

Simulation Actions

Start 10s Walk/Run

Simulates continuous motion. This increases your "step count" and triggers "walking animations" on the device visualization.

Start 10s Walk up Stairs

Simulates sustained vertical motion. This increases both your "step count" and "stair count", showing corresponding animations.

Simulate Sedentary Alert

Simulates prolonged inactivity. After a threshold, this triggers a "sedentary alert" and may show a "bad posture shake" on the visualization.

Simulate Do Not Disturb Mode

Simulates intentional periods of inactivity (e.g., sleep). The belt is marked inactive during preset time ranges, and "no data or alerts" will be generated.

Enable Do Not Disturb Hours simulation?

Power

Disconnected

Stairs: Last 7 Days

Day of the Week

Metric Info

Total	Average / day
840	120
42202	6029
1818	260

Today

Progress

4280

71%

Stand minutes (50)

Implementation — libraries, architecture and code structure

- Tech stack & libraries
 - Svelte (v5) for UI components and reactivity.
 - Vite for dev server and build tooling.
 - Chart.js for charts (used by chart components under src/lib/Mobile-Interface/Neel-Charts and Charts).
 - Minimal custom utility components in src/lib/shared-components (dialogs, inputs, Number-Line).
 - No external backend — the app uses an in-memory data store (Data-Store.svelte.js) for demo/persistence during runtime.
- Project layout (important files / folders)
 - Data-Store.svelte.js — central in-memory store (BeltData) and getToday() helper. Seeds several days of example data.
 - src/lib/Mobile-Interface/ — main dashboard view
 - Mobile-Interface.svelte — orchestrates header, charts, summary, and goals (inline).
 - Summary-Table.svelte — "Quick Numbers" / weekly totals & averages.
 - Goals-Table.svelte — goals UI and progress bars (steps & stand minutes).
 - Charts/ and Neel-Charts/ — Step-Chart.svelte, Stair-Chart.svelte, and reusable Chart.js wrappers.
 - Device.svelte — device simulator and all action handlers (walk, stairs, sedentary, DND).
 - src/lib/Device-Viewer/Device-Demonstration.svelte — SVG visualization of the belt/person (animation targets).
 - src/lib/shared-components/ — dialog and input primitives (Dynamic-Dialog, In-Line-Dialog, Number-Line, etc.)
- Data model & store
 - BeltData is implemented with a Map (private) and an exported publicData array for reactivity:
 - API: BeltData.addDay(date, obj), BeltData.updateDay(date, obj), BeltData.getDay(date), and a data getter returning publicData.
 - Day object shape (used across the app): { BreathCount: number, StepCount: number, StairCount: number, StandMinutes?: number }
 - Keys are ISO date strings (midnight Date.toISOString()), and getToday() returns today's midnight Date.
 - Initial example days are seeded in setInitialData().
- Reactivity & state conventions
 - Components use Svelte's reactivity and also project-specific helpers (\$state, \$derived, \$effect) to declare local reactive state, derived values, and side-effects.
 - Summary-Table computes totals and averages directly from BeltData.data.
 - Goals-Table derives stepsToday and standToday from the store and shows percent progress with a pct() helper.
- Device simulation & behavior

- Device.svelte exposes the user actions and animation loops:
 - handleWalk() simulates a 10s walking action: it
 - increments StandMinutes by 10s expressed as minutes ($10/60 \approx 0.1667$) and writes to BeltData.updateDay(...) when connected,
 - animates StepCount increases over the 10s and persists intermediate values so charts / tables update live,
 - updates breathing rate and posture state during the simulation and calls stopAllActivity() when finished.
 - handleStairs() is similar but affects StairCount and step updates.
 - handleSedentary() and handleDoNotDisturb() drive other simulated states (visual cues, DND simulation over a 24-hour tick sequence).
- Animation & writes: the simulation progressively updates the store (frequent writes) to simulate live telemetry. This is intentional for demo/live update UX but can be batched if needed.
- Charts & visualization
 - Chart components import and register Chart.js controllers and use BeltData as the data source so charts update automatically when BeltData.updateDay() is called.
 - Chart components are isolated (Line/Bar/Calendar) and wrapped inside Neel-Charts for reuse.
- UI & dialogs
 - Dialogs use a small Dynamic/Dialog pattern (Dynamic-Dialog.svelte, In-Line-Dialog.svelte) so metric info, action descriptions, and small forms are rendered as reusable modal/inline components.
 - Number-Line.svelte is used to configure Do Not Disturb ranges.

We utilized Bootstrap primarily as our base for styling, theming, and common components and Jordan primarily built a small common components library out of these to connect to Svelte and build upon the consistent styling options. Following an initial effort to display data from Neel with Chart.js, we continued to utilize this package to provide visuals to our users and created several components to separate the charts. We attempted to implement a common structure through the use of the component library and minimize code based on sections to separate components. Additionally we created a couple of data stores to handle shared data across the application and make access to dynamically changed data easier. Lastly we used svgs to leverage the browsers opportunities for simulations which allowed us to create the active visualizations shown.

Use of AI

We utilized AI considerably sparingly throughout the generation of this application. For more complex things such as SVG manipulation, AI tools such as ChatGPT and Google Gemini required significant prompting for relatively simple results and redirections which ultimately added time onto the project. These tools were helpful for smaller tasks such as simple styling inquiries and minor adjustments towards the end of the project. Our simulation was initially generated from these tools and we built out the simulation to include a variety of additional options, changed the look/details of the svg, and modified nearly the entire result to get to our final product. The use of AI in this case was instructive and helpful to our eventually product and helped provide quick answers to understand and gain context on our code.

Future Work

If we could continue working on this project, we would like to engage in more simulations and facets outside of traditional pedometers and smart trackers as well as integrate fall detection and other applications. On the programmatic side of this project, we are currently fully responsive, however there is significant room for growth with the charts to make them more mobile/small-screen friendly and it would be nice to refactor the code from redundancies.

Demo

<https://vimeo.com/1129395376?share=copy&fl=sv&fe=ci>

Links

Repo: <https://github.com/UI-5167-Project-2/Project-2-svelte>

Public site: <https://ui-5167-project-2.github.io/>