Cromi

Major Studio 1
Development Cycle

Radu Dutzan • December 2017

Cromi: Major Studio 1 Development Cycle

Cromi is an iOS app that aims to support users of Santiago de Chile's public transit system through the delivery of real-time information in an intuitive, clear, and hopefully beautiful way. It started life in December of 2016 as an experimental app to test the new, freely available APIs that Santiago's public transit authority had made available. It was publicly launched in April 2017, with a narrow feature set—it could only display bus time arrival estimations, plus fare reloading locations and subway stops. Cromi is the spiritual successor of Cuánto Falta, an earlier project with similar goals, which had amassed tens of thousands of devoted users.

Cuánto Falta was started in April 2010. It focused mainly on providing bus stop estimations, but could also display service routes. It had a server component which consumed data from GTFS feeds—a standard that enables transit operators to publish data about their systems—and delivered it through a private API. This server component was built by a friend who, after late 2014, no longer had time to maintain it. The data update process was very involved, and beyond my abilities. In February 2016, after over a year of serving outdated information, I decided to pull Cuánto Falta from the App Store due to lack of maintenance and my own shift in focus. Users who already had the app could keep using it with outdated data, but they could no longer download it if they deleted it or switched to a new device.

Soon after launching Cromi with the officially maintained data sources, those data sources began malfunctioning. After contacting the transit authority, they made it clear that, even though the APIs somehow ended up in use by several open-source, third-party projects, they were never intended to be publicly accessible. They made it clear that they would not guarantee their future operation. Since Cromi was dependent on these APIs, after learning this, I put the project on hold, and chronicled the story on Medium. In September 2017, Ignacio Hermosilla, a former colleague at a startup, emailed me saying he'd read my articles, and that he'd be interested in working on an open API that could provide data to Cromi and any other transit app in a frictionless and standardized way. This, paired with the outpouring of melancholy from Cuánto Falta users who missed the app, reignited my enthusiasm in the project. There are other apps that provide this sort of data, but none are created with the care for user experience that Cuánto Falta had and that Cromi aims for.

The challenge is: considering the landscape of available transit tools, how might Cromi become a more useful and complete tool to assist users of Santiago's transit system while retaining a high efficiency of interaction in a simple, fluid, content-focused interface that's easy to understand and navigate?

I first need to define what it means to be a “complete” tool for public transportation. Through surveying users of the transit system, personally discussing the subject with users, and informed by my own experiences building the previous product, I’ve defined that the three key areas that need to be covered in such a product are: data visualization, which means enabling users to access and visualize the available transit system data, whether real time or static; real time notifications, meaning the active monitoring of transit events relevant to the user (such as the arrival of a bus or the approaching of a destination stop) and the delivery of that information without the user needing to have the app on-screen; and route finding and navigation, which means calculating the optimal routes to get from point A to B using the transit system, and guiding the user in real time through every step necessary to arrive.

To aid with the thought process of the user interface, I have laid out what I believe are the 3 principal stages of the public transportation process: plan, wait, and ride. All of them are dependent on the previous one.

On the planning stage, the user needs to know a few things that will enable them to move on to the next stage: the exact location of their destination, what their options are to get there, and which one of them they will choose. For someone making a regular commute, this step is completely solved, but for someone making an unfamiliar trip, it’s probably the biggest issue. Next, during the waiting stage, the user already knows what to wait for and where. The issue of waiting is tightly related to anxiety and discomfort. In an ideal situation, the wait time would be short and waiting conditions would be favorable—good weather, with a shaded place to sit. But in the not-uncommon event of long wait times and hostile waiting conditions, the app could assist the user further. Waiting is the main context where the delivery of real-time data is relevant. Lastly, during the riding stage, the user needs to know where to get off the bus, whether they need to start a new waiting process in case they need to transfer, and how to get to their actual destination after descending from the bus. This stage is tightly related to the planning stage, in the sense that a user making a familiar route needs less assistance than one making an unfamiliar one.

To this project, there are no distinctions between experienced or inexperienced users of the product. The interface should be as understandable to someone who is new to the app as it is to someone who has used it for a long time. The main distinction is the nature of each trip: is it a familiar trip, is it a completely unfamiliar trip, or is it somewhere in between? The app needs to be useful for any of these cases, which means that a user who needs a lot of guidance will use many of the app’s features as part of a step-by-step navigation sequence, whereas someone making a routine commute will only use the specific features they need to cover their specific areas of uncertainty. The interface needs to allow for this kind of modularity of interaction.

The app aims to keep the map as a key anchor to all interactions in order to empower users to randomly access any data that they might need. Other apps, such as Google Maps, lock users into screen-based modes that have proven to be very rigid, particularly in situations where accessing different pieces of data on-demand might be useful, such as the trip planning or riding stages. An on-screen map is a freely scrollable and zoomable interface where the user might bring up different pieces of data as they please, so the traditional screen-based interface design methodology falls short. This reality, coupled with the previously described requirement of feature modularity, requires a component-based approach to the user interface. This means that sets of controls and data displays need to be designed somewhat independently from each other, and rules for their on-screen composition and interrelation need to be laid out.

Having established these three core definitions (ultimate product goals, key stages of interaction, and requirements for the user interface), it's useful to break them down into concrete features in order to assess their viability and development cost, and from those conclusions, develop a roadmap. The following lists describe these features, and are separated into what I believe can be developed in the near-term, versus what I'm envisioning as a plan for a longer-term development. Their order roughly corresponds to the order in which they will be approached.

Near-Term Features

- **Line View.** Provides a visual representation on the map of the route and stops a bus service makes, with controls to toggle between different directions.
- **Live Bus Tracking.** Provides a visual representation on the map of the location of the buses in a specific service/direction pair, in connection with Line View.
- **Bus Arrival Notifications.** Provides system-level notifications to alert the user when their observed buses are approaching the stop where they plan to ride them from.
- **Fare Card Balance.** Provides an interface to add fare cards to the app in order to visualize their balance.
- **Search.** Provides an interface to search for places and bus services in order to visualize them on the map.

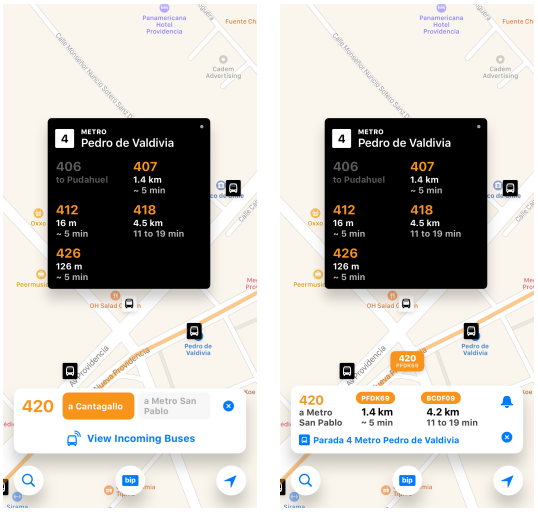
Longer-Term Features

- **Places.** Provides interfaces to store and recall user-defined locations or stops. Selecting a place pans the map towards it.
- **Destination Stop Notifications.** Provides system-level notifications to alert the user when the bus they are currently riding is approaching their destination stop.

- **Route Finding.** Provides interfaces to calculate routing options between two points, and to visualize and choose the option the user deems as optimal.
- **Step-By-Step Navigation.** Provides interfaces to guide the user throughout the entire process of traveling to their destination.
- **Subway Data.** Provides visualization of subway lines and service updates that might assist the user in their election of an optimal routing option.

The longer-term features involve a significant investment in research, design, and engineering, and even though they are key to complete the product's coverage of the entire process of using public transportation, even without them, the near-term improvements to the product will still have a substantial impact on its utility.

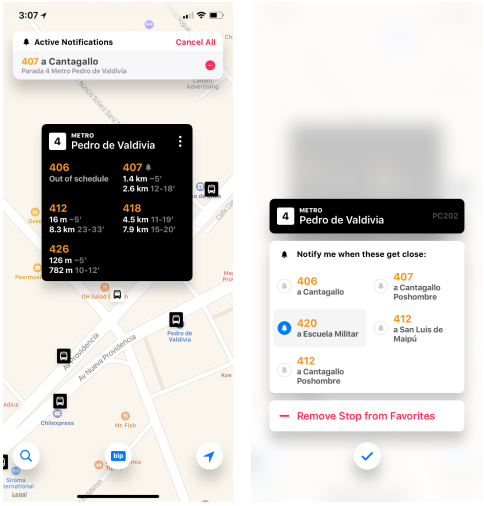
On the span of time allowed by this exercise, I aimed to design and iterate over most, if not all, of the features described in the near-term roadmap. I decided to first focus on Line View, Live Bus Tracking and Bus Arrival Notifications, since they seemed tightly related. I first thought of these features as connected very sequentially in the interaction. For example: a user would first trigger Line View, and from there, they could choose to track the upcoming buses, and only from that state could they ask the app for bus arrival notifications. This was very cumbersome, and led to some very cluttered iterations as I tried to add previously unavailable data to new and convoluted components, instead of rethinking what was already there. The original Stop Sign, which displays the stop name and the services pertaining to it, could only display estimations for one upcoming bus. The API provides estimations for two buses, and I was trying to add that data as a part of a bar on the bus tracking state, which could only be arrived at after enabling Line View. This was clearly not working for the users I interviewed, so I decided to make bus tracking an automatic part of Line View, and to modify the Stop Sign to accommodate two estimations instead of one.



Original designs for Line View/Bus Tracking

Line View/Bus Tracking as implemented

For Bus Arrival Notifications, I decided to provide two separate entry points: through Line View and through a Stop Sign. A button to enable notifications for the current service would appear on the right of the Line View Bar (visible on the previous page, without said button), and its effects would be instantaneous if a stop containing the current service's direction is currently selected. If not, the app will prompt the user to select a stop from which to observe the buses in order to trigger notifications. The Sign entry point relies on a new Options button on the right side of the Sign's header. That button triggers a modal that allows the user to select one or several services to be notified about, and also to add the stop to Favorite Places. Once a notification has been requested, a bar appears at the top of the app reflecting it and allowing for its cancellation. The bar also has a minimized state, automatically triggered if more than one notification is active in order to save precious vertical space on the screen.



Left: Active Notifications bar and Stop Sign showing Options button

Right: Stop Options modal screen

Implementation of Line View was very straightforward after locking the key design decisions. The simplification of the feature triggered by user interviews performed in Chile during Thanksgiving week cleared a path to a simple and evident user interface. After iterating over the API endpoints for over a month with Ignacio, I felt confident about their adequacy, and though some issues with them cropped up during their implementation, Ignacio was quick to address them. Line View was implemented, seeded to beta testers, fixed, improved, and released over the span of one week, after weeks of design iteration. The feature was very well-received by users, and provided a key anchoring tool for a marketing boost of the app, resulting in 461 downloads during the week of December 4, which is especially remarkable considering the version only went live on the App Store on Saturday morning of that week.

It wasn't so straightforward with Live Bus Tracking. The realities of the data gave way to issues unforeseen during the design process. The bus position data can be up to 2 minutes old by the time it reaches the user, so contradictions can occur between the Stop Sign estimates and the buses on screen. In many cases, buses are not close enough to a stop to be visible at the default zoom level, and zooming out provided a bad visualization experience due to the overload of annotations on the map, which obscured the buses and stops meant to be highlighted. The visual issues have been iterated over in code, and the time delays will be informed to users through in-app notices. At the time of this writing, Live Bus Tracking is on its second beta iteration, and on track to be released on the App Store by December 18.