

From Four Wheels to Two

Launching Lyft Scooters,
Engineering principles for fast paced,
wheel spinning product development



Mobius Moscow 2019
RJ Marsan @rjmarsan₁

I'm RJ

I'm a hacker



Build cool stuff, fast.

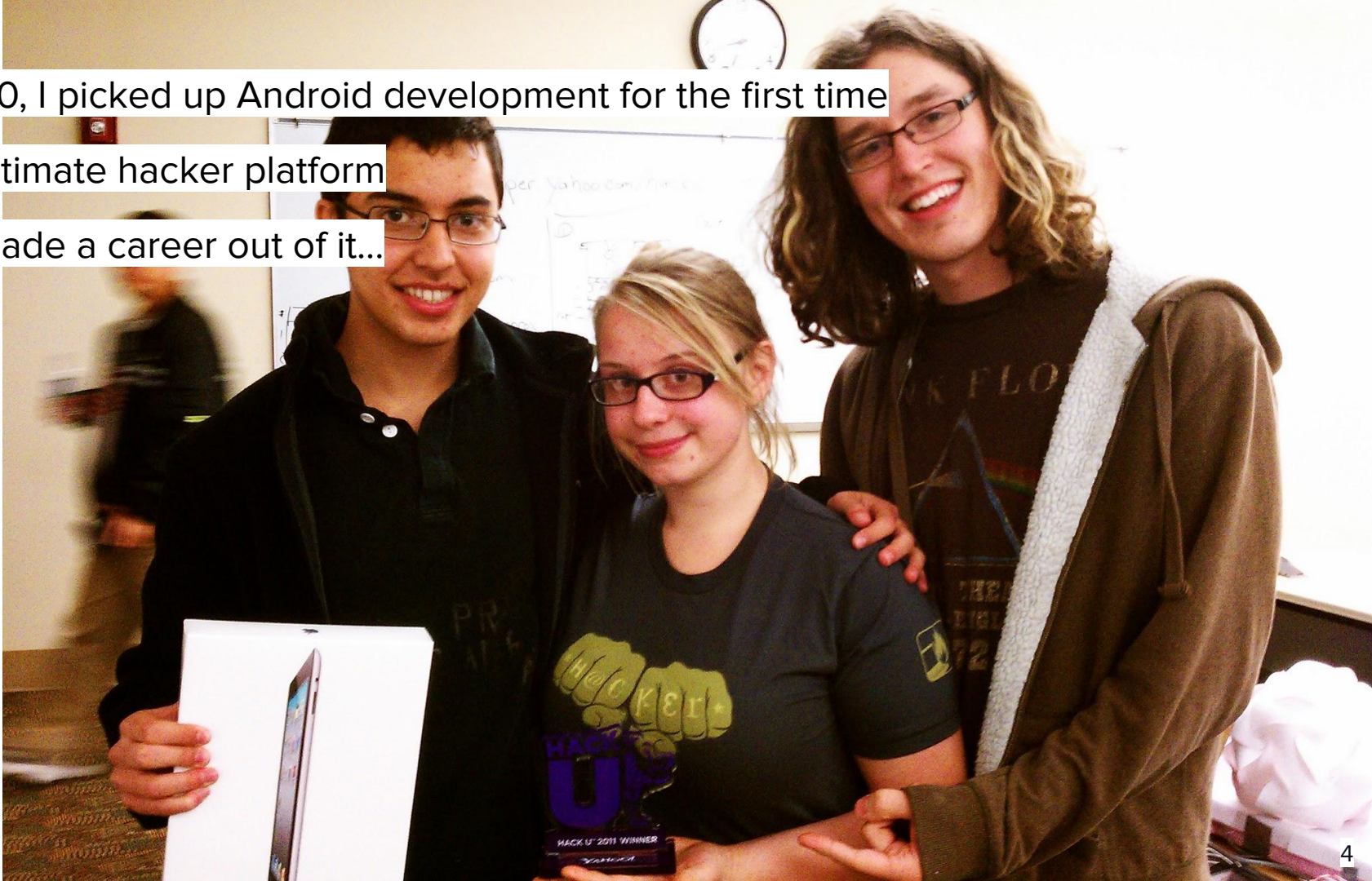
- *A hacker's motto, circa 2010*



In 2010, I picked up Android development for the first time

The ultimate hacker platform

So I made a career out of it...



After years of building stuff that never shipped,
I got burnt out

So I've come up with a new motto

Waste less precious time

Make more meaningful stuff



Product Engineering:

(and this talk's takeaways)

1. Maximize your effort
2. Own what you code
3. Ship meaningful stuff

Yes, there will be code.

Story time

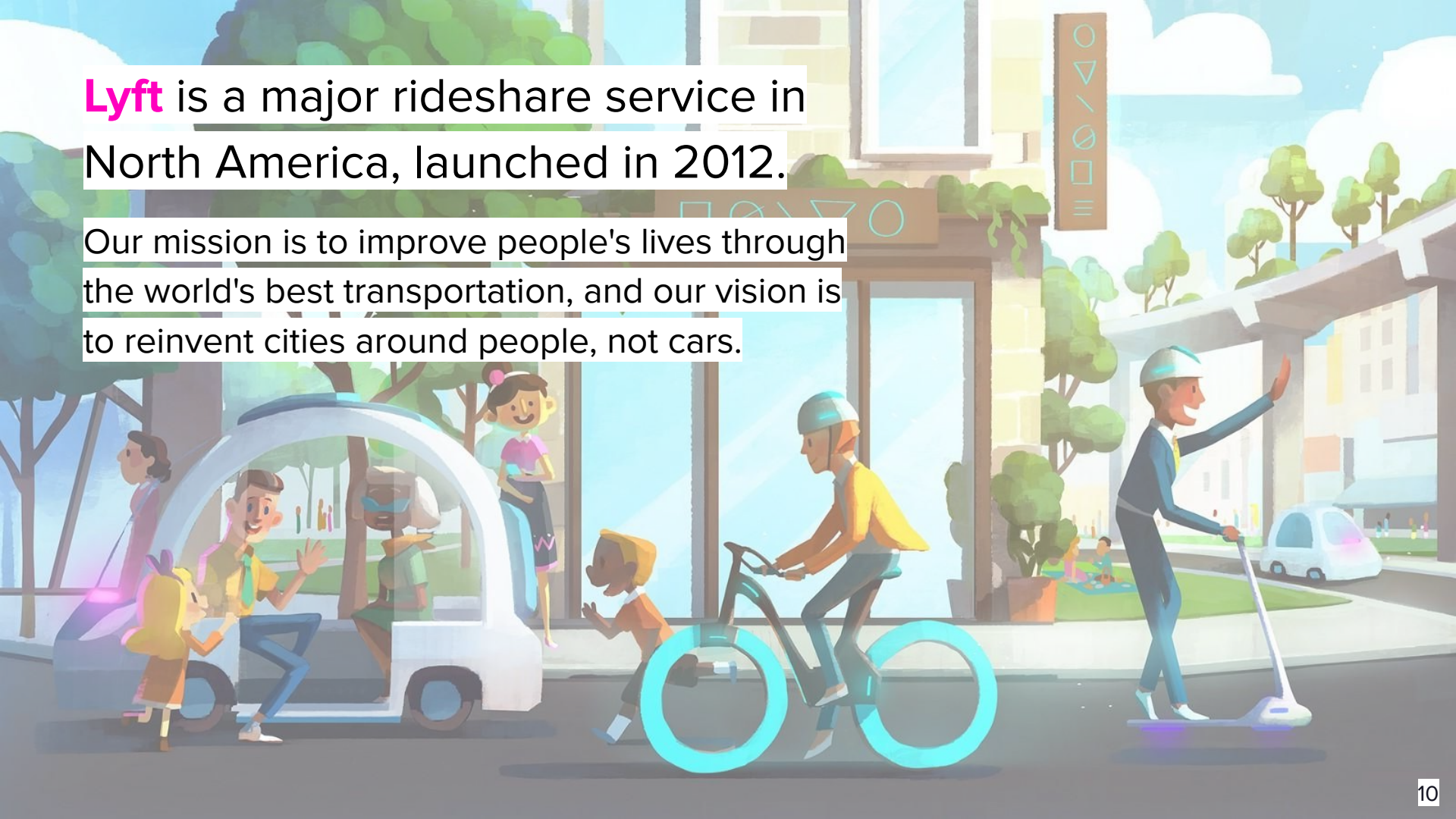
A man and a woman are riding e-scooters on a city street. The man is in the foreground, wearing a dark blue jacket and a black helmet, with a black backpack. The woman is behind him, wearing a red sweater and a white helmet. They are both smiling and looking forward. The street is lined with trees and parked cars, and a white car is driving away in the background. The scene is captured in a cinematic style with soft lighting.

“We want to build a scooter sharing service

... do you want to join?”

Lyft is a major rideshare service in North America, launched in 2012.

Our mission is to improve people's lives through the world's best transportation, and our vision is to reinvent cities around people, not cars.



As scooters became popular around American cities in 2018, we saw it as a natural extension of our company vision and mission.

In June 2018, we had an opportunity to expand Lyft's transportation options in an exciting way.



The proposal

Provide a scooter sharing service for Lyft users

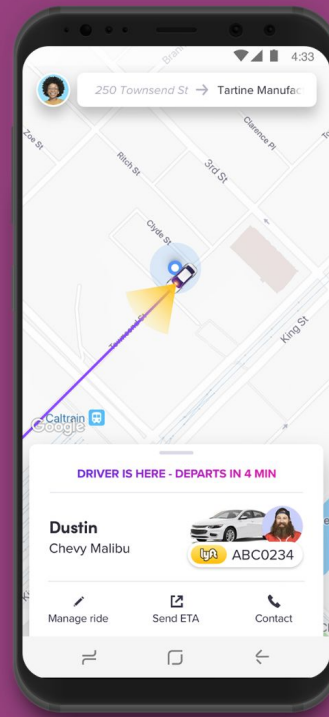
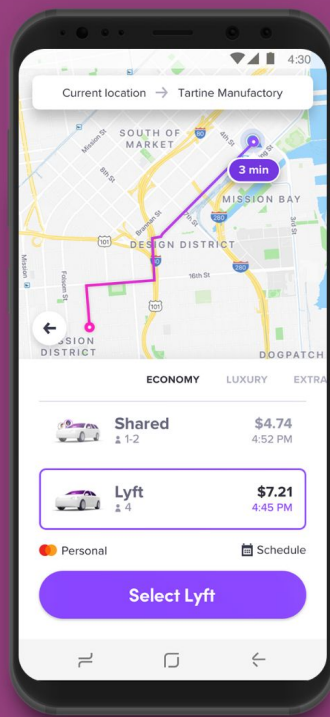
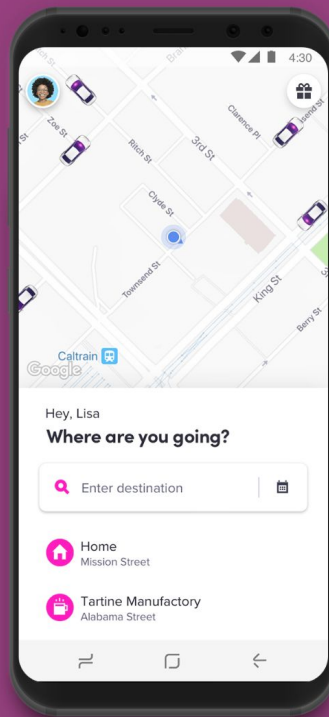
Client, server, firmware, hardware, operations from scratch

Three Two month deadline

The constraint

We are a rideshare company, built for efficiency getting you a car, not a scooter

We can't risk our core experience in pursuit of a new feature



**This could be fun...
or a disaster**



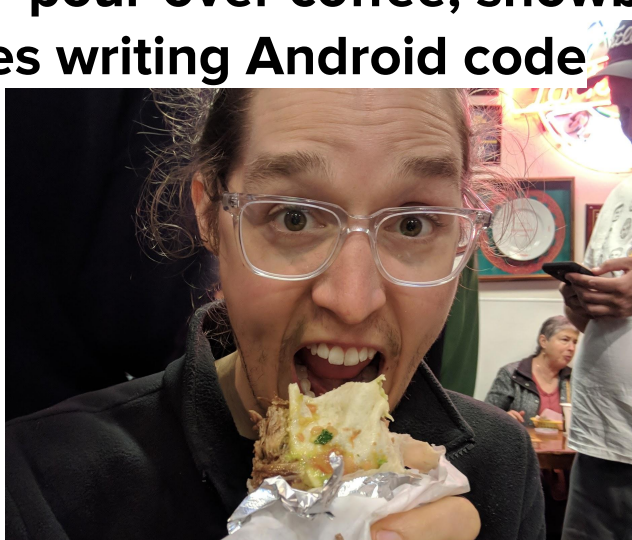
I'm RJ

(@rjmarsan)

SF based, 9 years of Android

@ Lyft, Google & Hulu

You'll find me brewin' pour-over coffee, snowboarding, cooking, hiking, and sometimes writing Android code



**If we want to succeed,
every moment counts.**

Pause and step back

What is success?

What is not-success?

- **Too Slow:** Not building fast enough to meet our deadlines
- **Too Fast:** Rushing to release a non-functional product
- **Too Disruptive:** Interfering with our company's core business or infrastructure
- **Not Useful:** Creating a product our users don't want

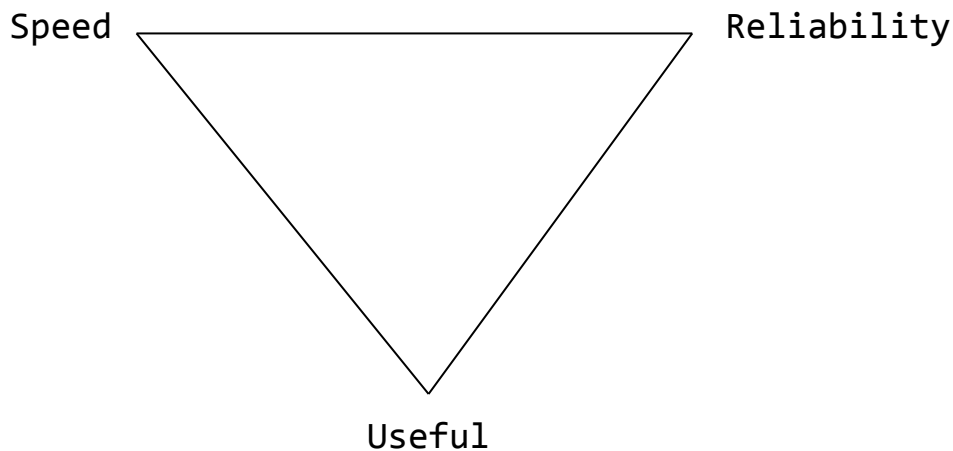
What is success?

- Balancing speed and reliability in our code to meet critical deadlines

Speed _____ Reliability

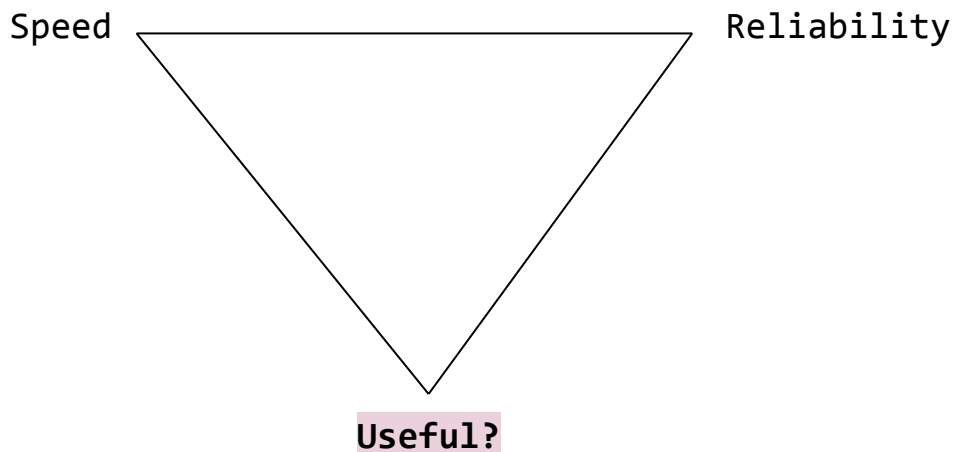
What is success?

- Balancing speed and reliability in our code to meet critical deadlines
- Confidence that what we launch will be engaging and useful



What is success?

- Balancing speed and reliability in our code to meet critical deadlines
- Confidence that what we launch will be engaging and useful



Useful = Solving people problems

Building apps is a human process, intended to solve problems for humans.

People problems are solved with technical solutions.

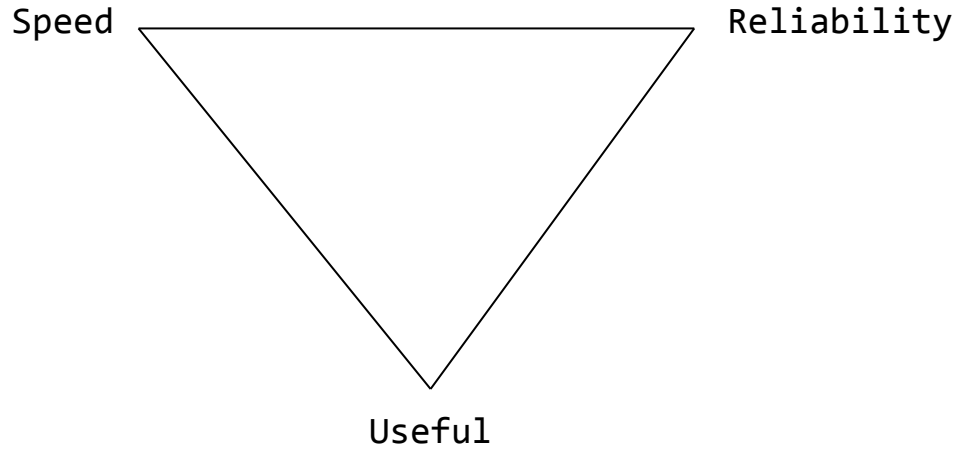
This is the foundation of every engineering decision we make, embracing **ambiguity**, **uncertainty**, and **subjectivity**.



**This is a talk on
product engineering**

**Building and shipping meaningful
products for real humans**

Product engineering



Product engineering principles

For fast-paced mobile product development (and hopefully many other things)

1. Stay simple, stay lean
2. Reimagine over reinventing
3. Listen, learn, and launch what matters

Stay Simple, Stay Lean

Effectively building as little as possible

““

We have an existing
ride-sharing experience we
can't risk

““

We need a  rock solid 
foundation that we can trust
and build from at rocket speed

Stay Simple, Stay Lean

Safe & Solid Foundation

Safeguarding it behind a Feature Flag is a great way to prevent users from seeing your feature.

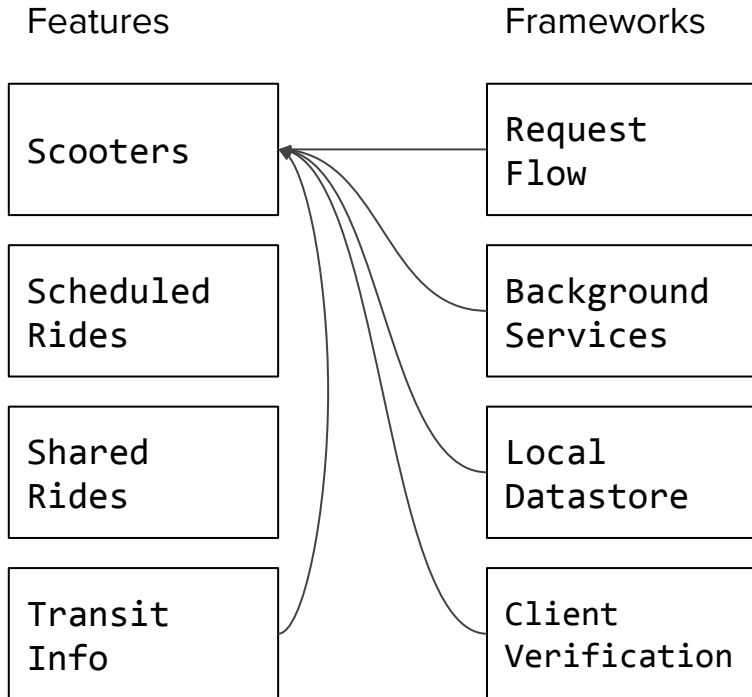
```
if (featuresProvider.isEnabled(Features.LYFT_SCOOTERS)) {  
    return lastMileStepMapper.mapToStep(lastMileRide);  
}
```

But what about all the other places that feature might live?



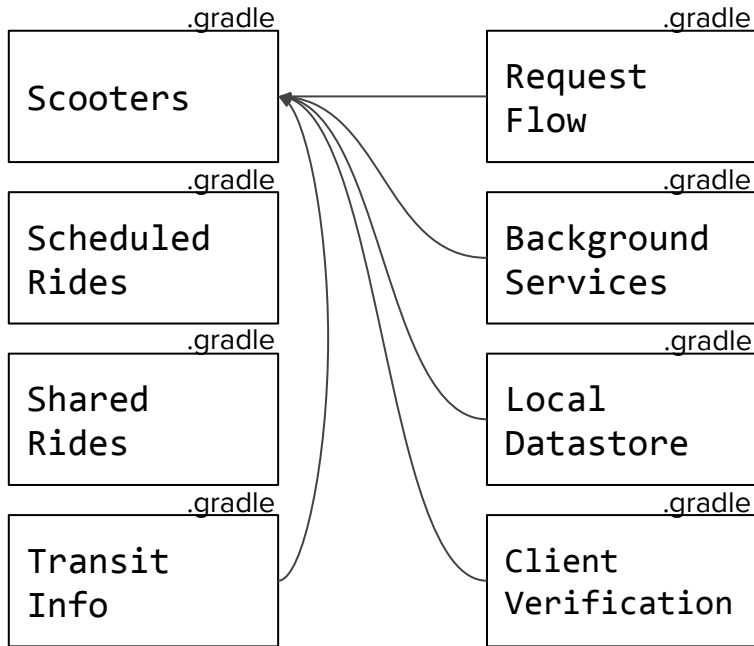
Stay Simple, Stay Lean

Safe & Solid Foundation



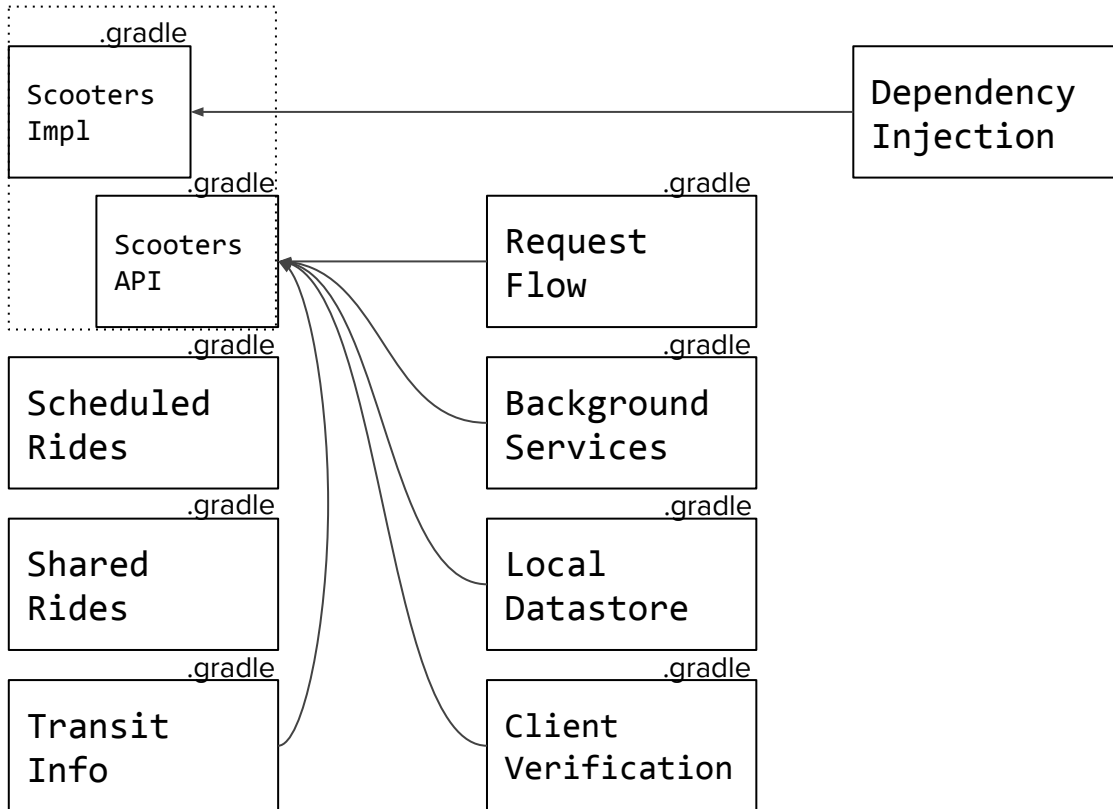
Stay Simple, Stay Lean

Safe & Solid Foundation



Stay Simple, Stay Lean

Safe & Solid Foundation



Stay Simple, Stay Lean

Safe & Solid Foundation

Feature modules minimize surface area and gave us confidence:

in the root scooter module ...

```
package com.lyft.android.passenger.lastmile.core;
```

```
public interface ILastMileRouter {  
    PassengerStep getLastMileStep();  
}
```

... in a separate gradle module ...

```
@Provides  
ILastMileRouter provideLastMileRouter() {  
    return new EnabledLastMileRouter();  
}
```

... and a no-op module



Stay Simple, Stay Lean

Safe & Solid Foundation

FlavorModules let us be confident that prod builds simply didn't have our code:

```
implementation project(':instant-features:passenger-x:last-mile:core:api')
implementation project(':instant-features:passenger-x:last-mile:ride')

// Include scooters in dev and alpha
devImplementation project(':instant-features:passenger-x:last-mile:core:impl')
alphaImplementation project(':instant-features:passenger-x:last-mile:core:impl')

// Do not include in beta and production
betaImplementation project(':instant-features:passenger-x:last-mile:core:no-op')
prodImplementation project(':instant-features:passenger-x:last-mile:core:no-op')
```



Stay Simple, Stay Lean

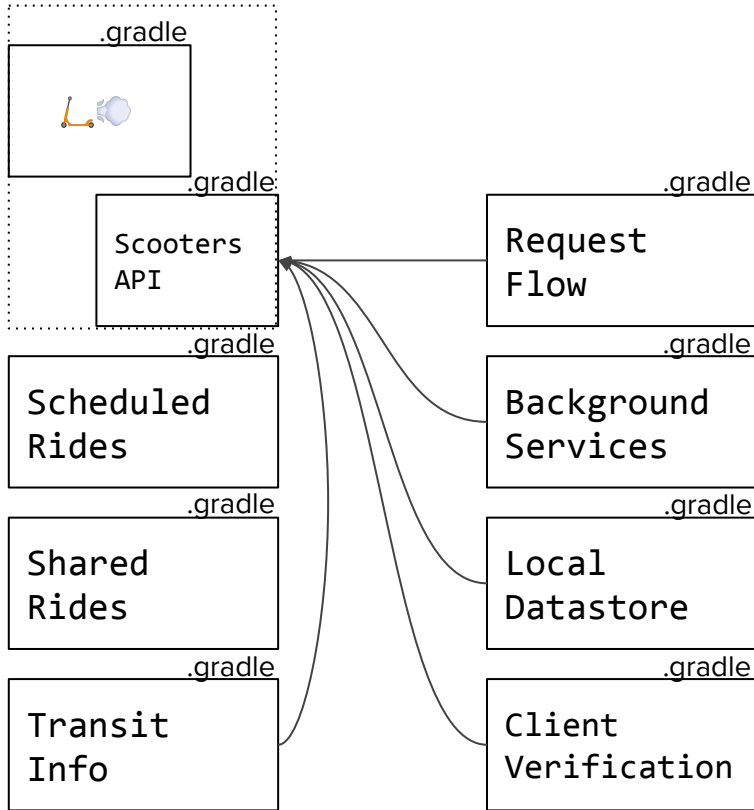
Safe & Solid Foundation

Our original code became:

```
if (featuresProvider.isEnabled(Features.LYFT_SCOOTERS) && !lastMileRide.isNull()) {  
    PassengerStep lastMileStep = lastMileStepMapper.mapToStep(lastMileRide);  
    if (lastMileStep != null) {  
        return lastMileStep;  
    }  
}
```

Stay Simple, Stay Lean

Safe & Solid Foundation



Stay Simple, Stay Lean

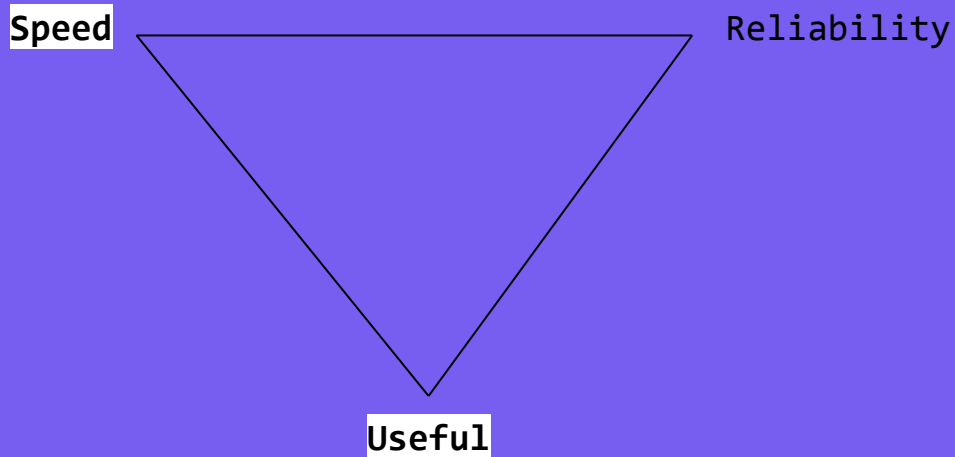
Kill-switch - the big red button

Kill-switches let us remove every interaction with our feature in real-time if something goes dramatically wrong:

```
return killSwitchProvider.observableEnabled(KillSwitches.LYFT_SCOOTERS)
    .switchMap { killSwitchValue ->
        when (killSwitchValue) {
            KillSwitchValue.FEATURE_ENABLED -> {
                authenticationScopeService.doWhenAuthenticated(rideUpdateService.observeAndUpdateLastMileRide())
            }
            KillSwitchValue.FEATURE_DISABLED -> Observable.never()
        }
    }
}
```

**Staying within this foundation,
we focused on building
exactly what we needed to,
and nothing more.**

Balancing usefulness & technical difficulty



Stay Simple, Stay Lean

Define a *Product North Star*

“Empower users with a convenient, easy-to-use and affordable way to get around their city”

1. What people problem are we trying to solve?
2. How will our product help the user through that problem?

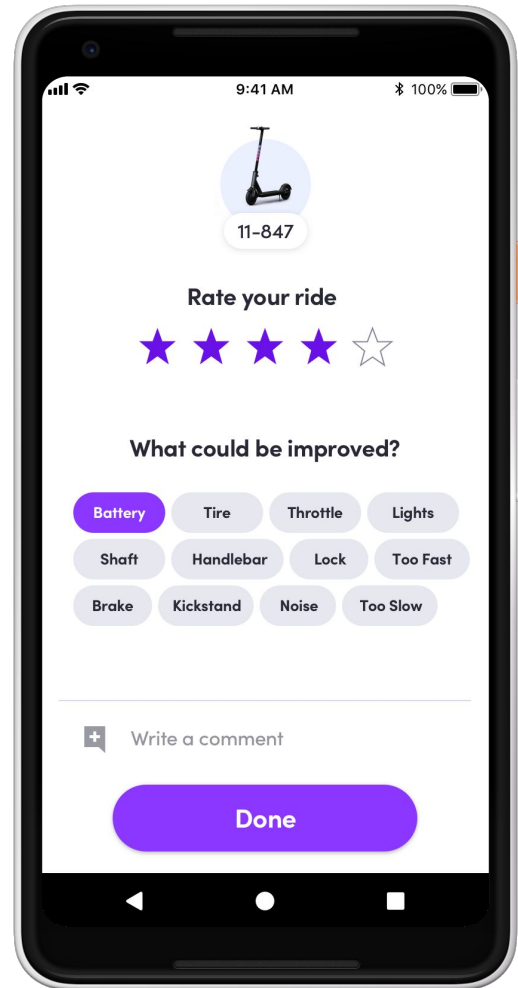
This north star remained constant through all our twists and turns as we decided what was important to build.



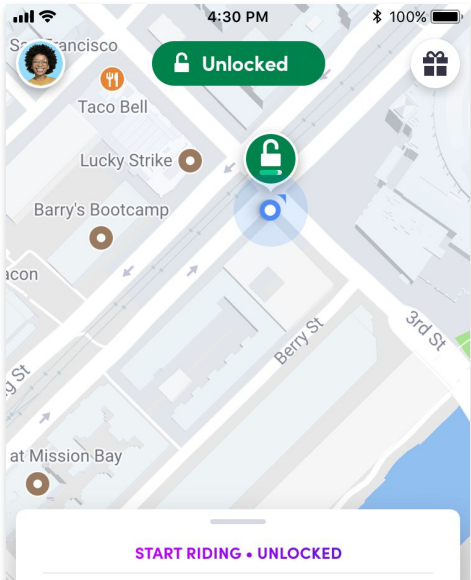
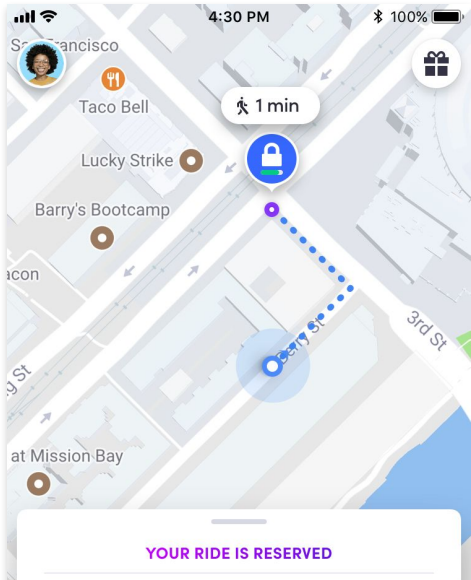
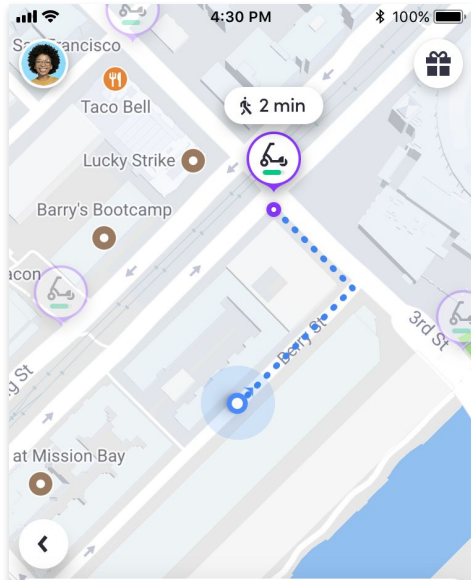
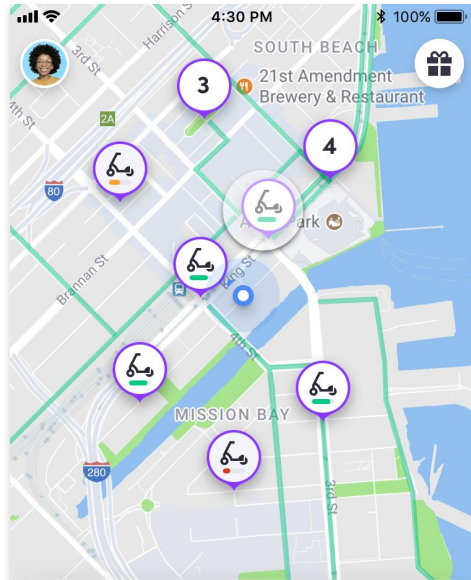
Stay Simple, Stay Lean

Define a *Golden path*


The central user experience to solve the core people problem under optimal conditions



Lyft Scooter's Golden Path




\$1 to unlock, \$0.15 per min
Ride a scooter



Find closest **Scan**

Parked at
234 3rd St
15 mi range




11-847

Scan **Reserve**

YOUR RIDE IS RESERVED

🕒 0:01
📍 0 mi
14 mi range




11-847

Cancel **Unlock**

START RIDING • UNLOCKED

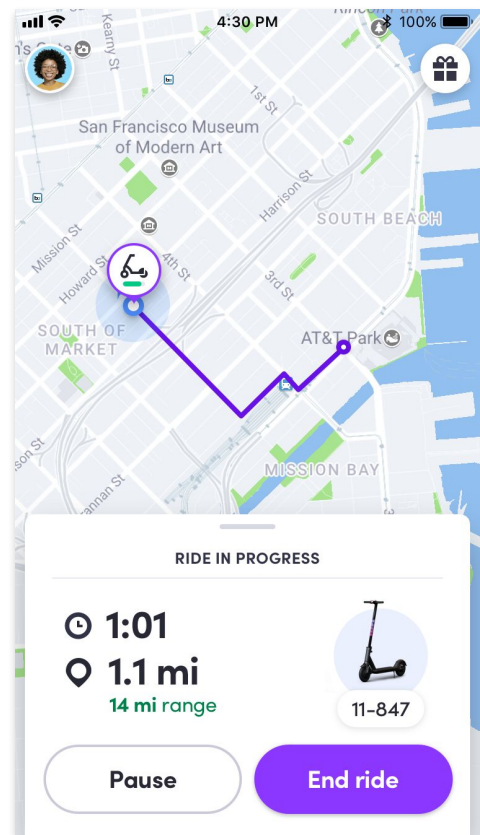
🕒 0:02
📍 0 mi
14 mi range



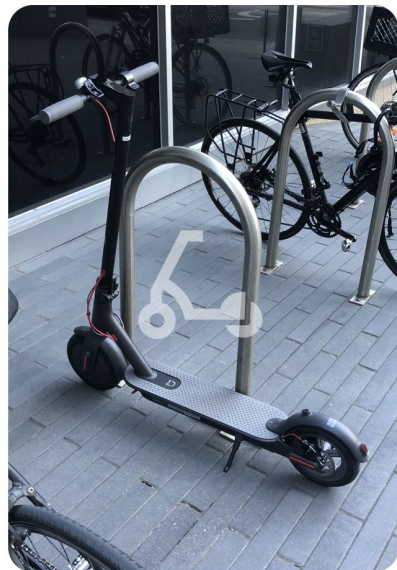
11-847

Pause **End ride**

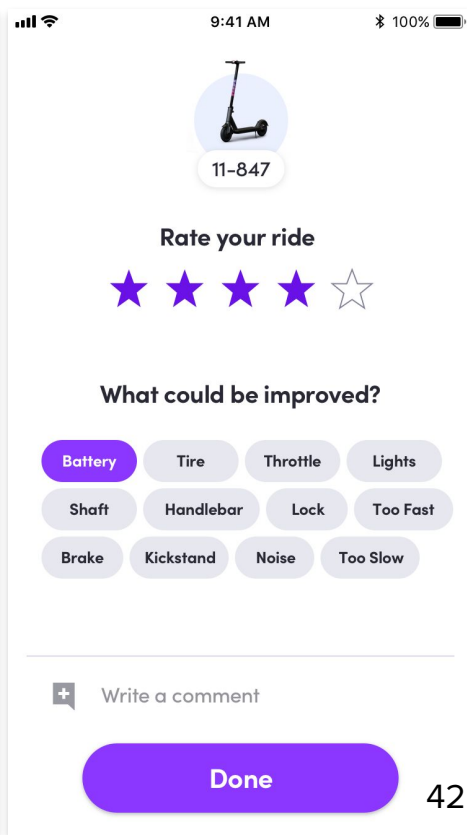
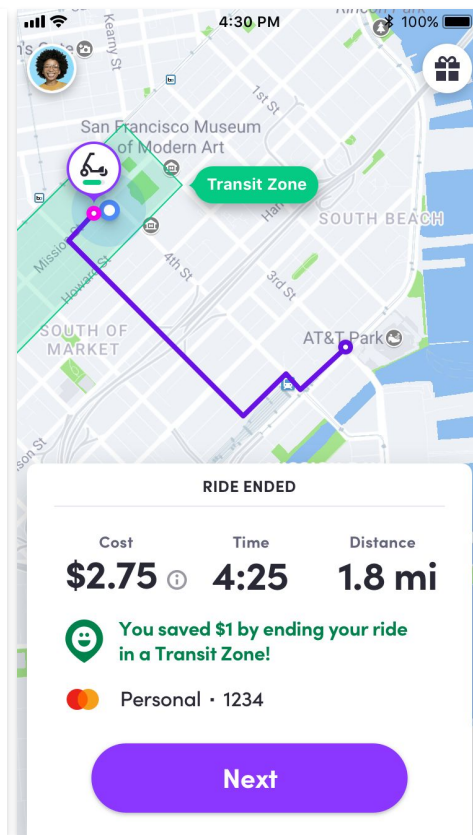
Lyft Scooter's Golden Path



✕ **Park responsibly**
Take a photo to end the ride



📷 End ride

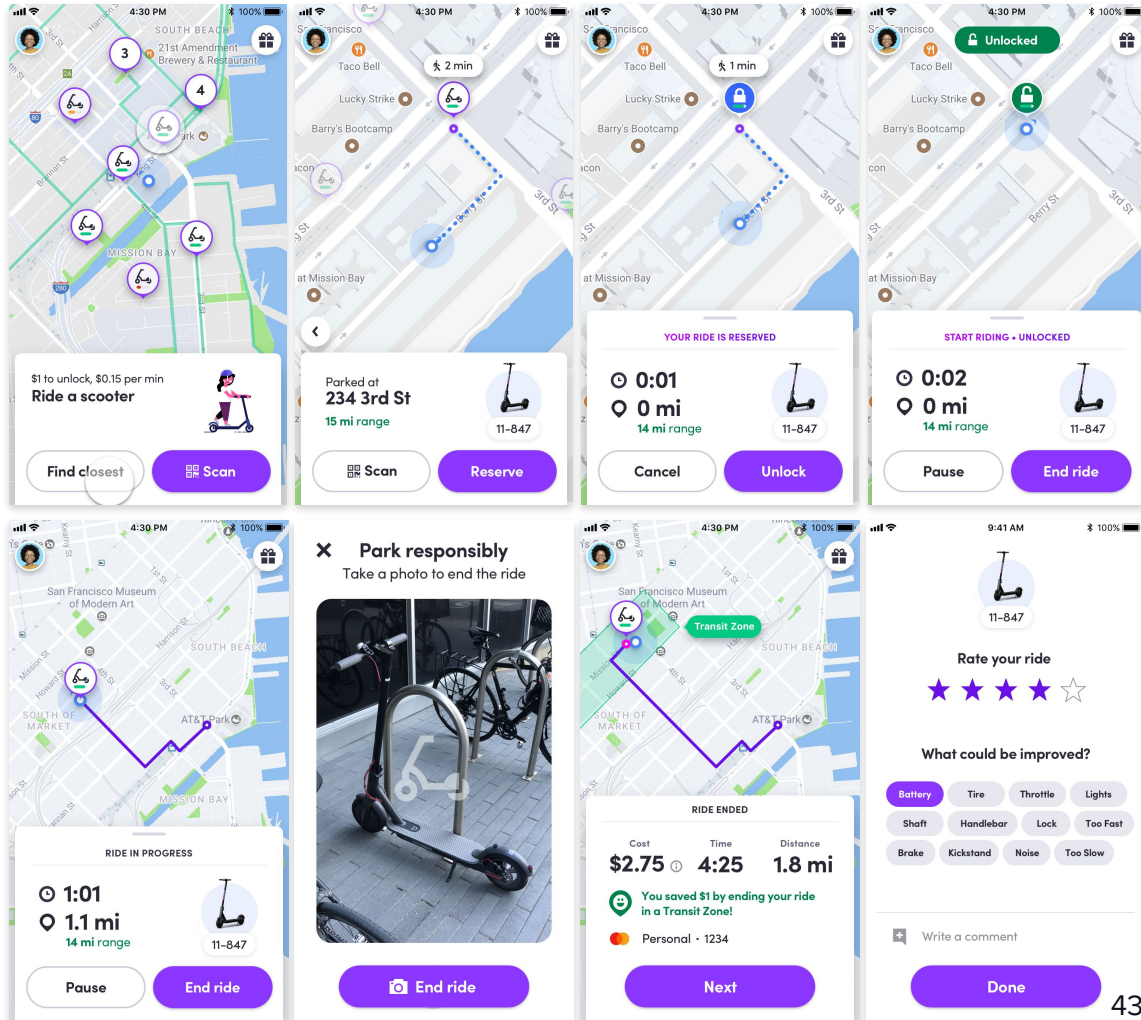


Stay Simple, Stay Lean

Golden Path

Very clearly outlined what our feature would and would not be

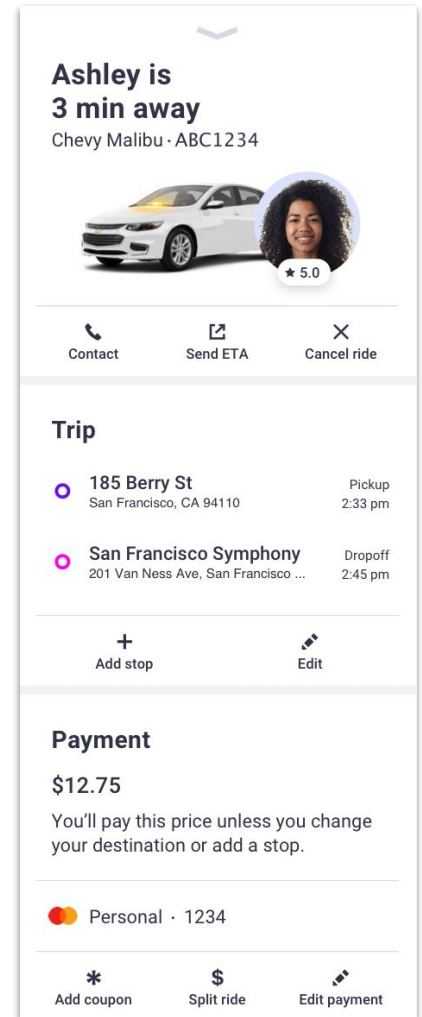
Easily evaluate if we're building towards our north star



Stay Simple, Stay Lean

Limit your features

It was tempting to have all the features of our existing rideshare service, but realistically we couldn't launch in time with all of that.



Stay Simple, Stay Lean

Limit your features

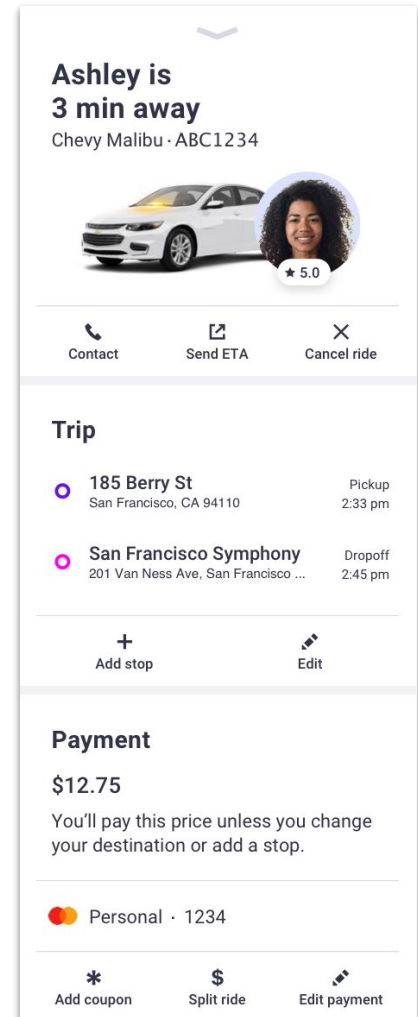
Extra feature ideas:

- Sharing ETA with friends
- Setting a destination
- Coupons & promotions

With our north star and our golden path, we can balance usefulness with technical difficulty.

“Does sharing an ETA get us closer to our product north star?”

We ended up leaving all extra features out of our first release.



Stay Simple, Stay Lean

Design the simplest architecture

Now that we have a north star, a solid foundation and a limited feature set, we need an overall architecture to make the golden path a reality.

What's the most straightforward and easy-to-reason architecture that gives us enough wiggle room to handle changes?

A photograph of a park with a winding dirt path, green grass, and trees. In the background, there are brick buildings and parked cars. The scene is brightly lit, suggesting a sunny day.

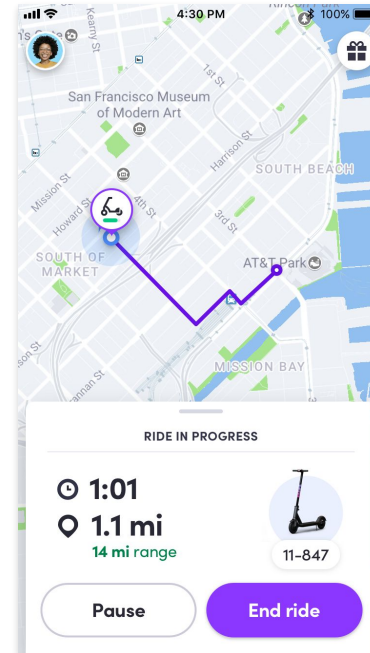
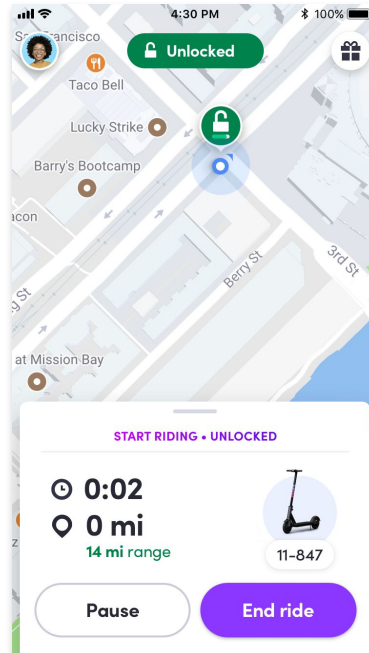
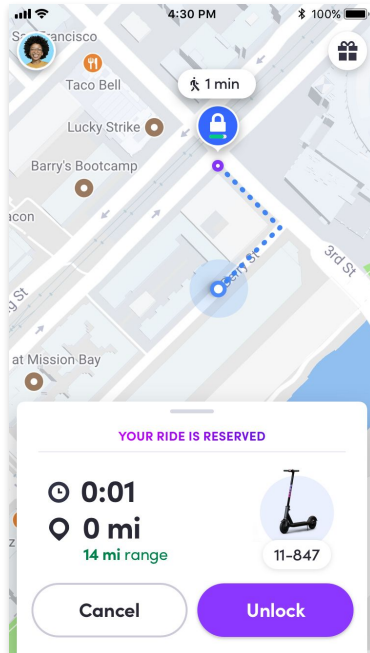
The bigger the feature,

**the more likely it'll take shape
in unexpected ways.**

Stay Simple, Stay Lean

Reviewing the golden path

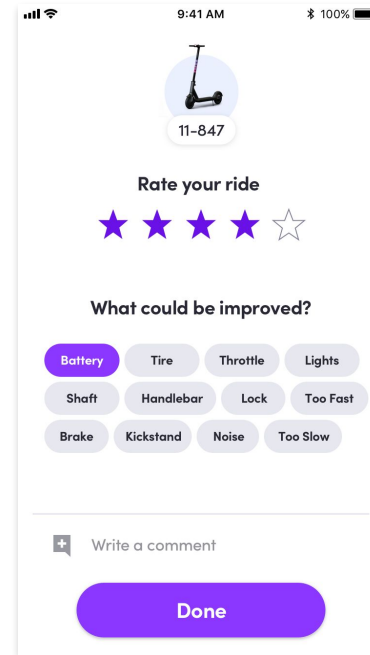
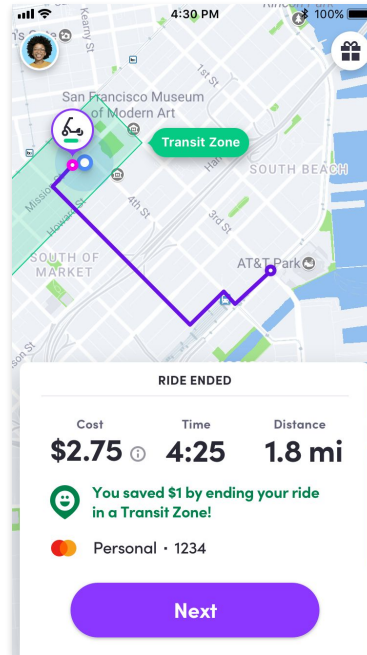
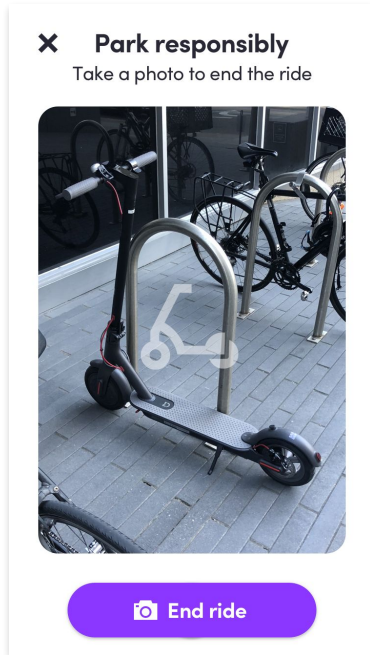
Once the user selected a scooter, it would go from *Reserved*, to *Locked*, to *Unlocked*, to *In Progress*



Stay Simple, Stay Lean

Reviewing the golden path

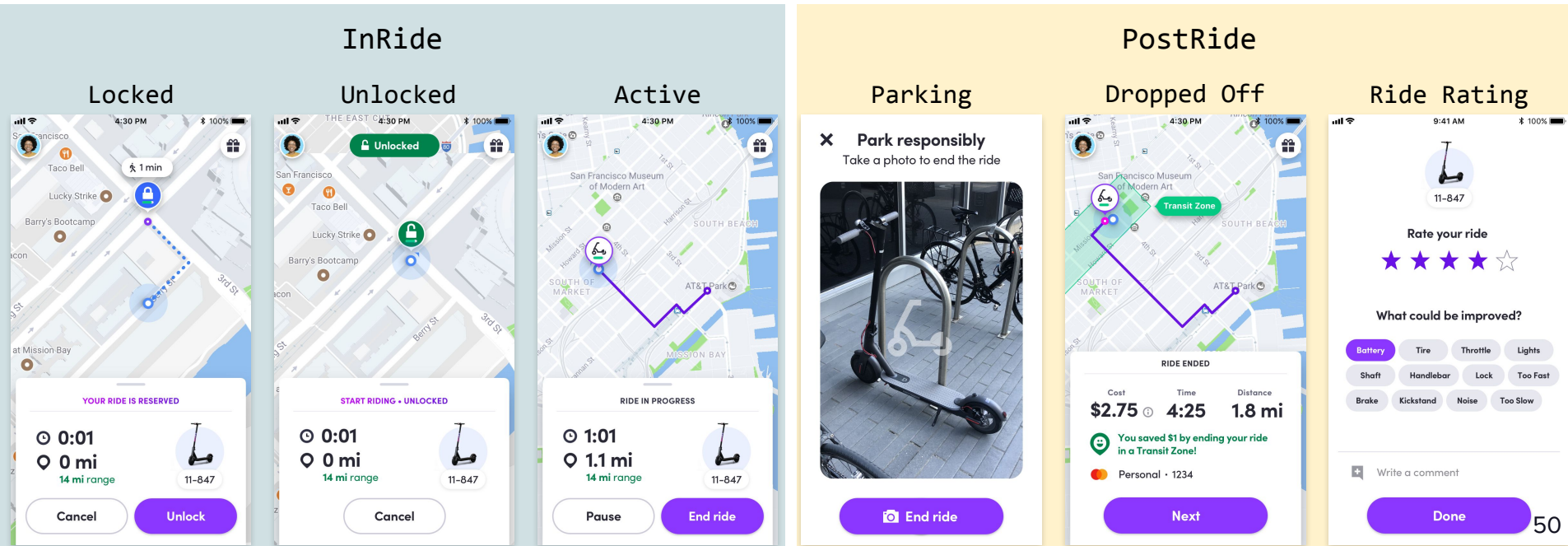
When the user submitted a photo to end their ride, they would arrive at a post-ride screen, then rate their ride



Stay Simple, Stay Lean

A potential architecture

We were tempted to represent the in-ride and post-ride experiences as a separate set of states, building in future-proof flexibility.



Stay Simple, Stay Lean

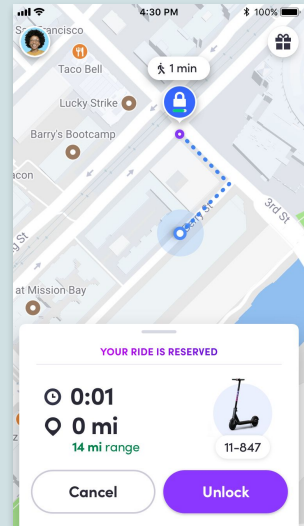
The hidden complexity

What edge cases might be triggered when we transition?

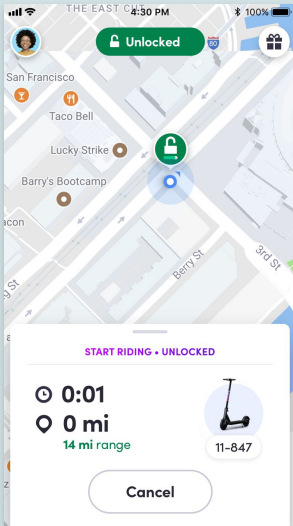
What hidden complexity is this generating?

InRide

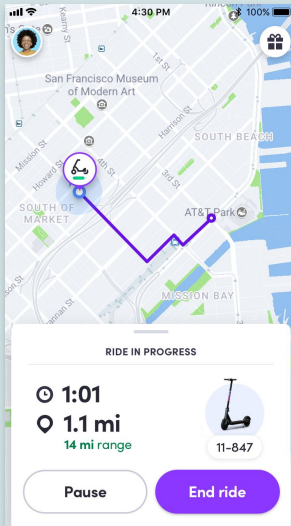
Locked



Unlocked

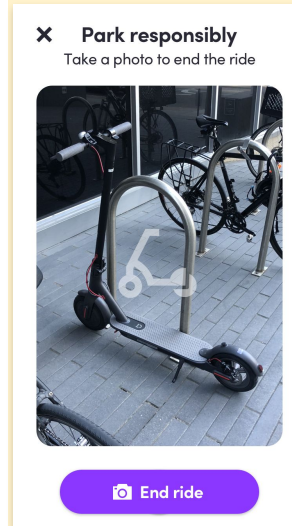


Active

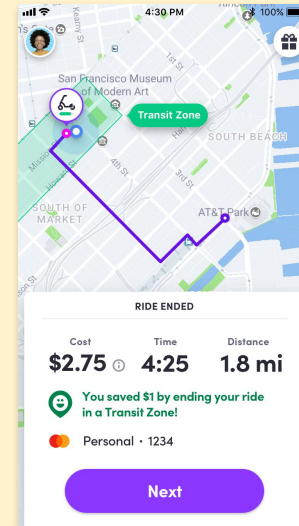


PostRide

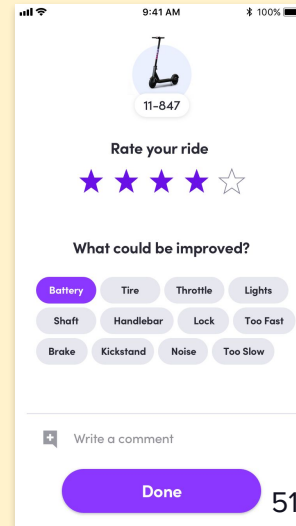
Parking



Dropped Off



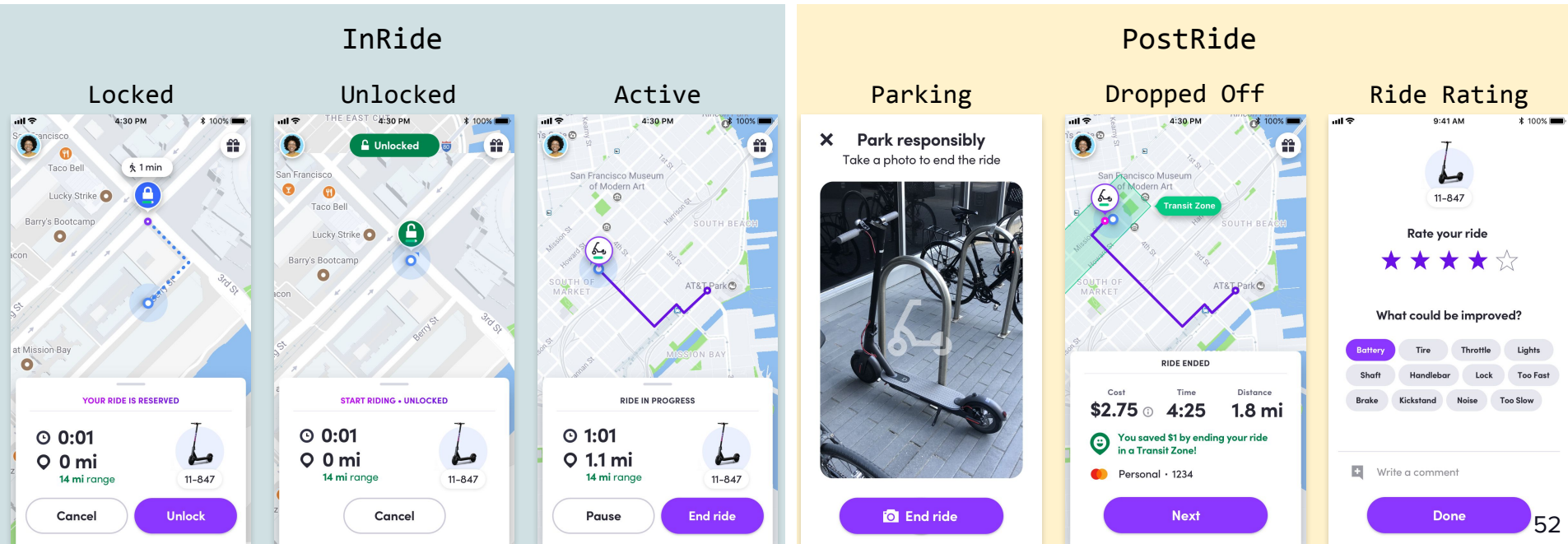
Ride Rating



Stay Simple, Stay Lean

The hidden complexity

Are we optimizing for a path our product might not take?



Stay Simple, Stay Lean

The hidden complexity

We want something we can easily reason about in any circumstance and through any sequence of states

The image displays a sequence of six mobile app screens, each representing a different state in the scooter rental process. The screens are arranged in two rows of three. The top row shows the 'InRide' phase, and the bottom row shows the 'PostRide' phase.

- Locked:** The scooter is reserved. The screen shows a map with a blue dot indicating the scooter's location. The bottom card displays 'YOUR RIDE IS RESERVED', a timer at 0:01, 0 mi, and a '14 mi range'. Buttons for 'Cancel' and 'Unlock' are visible.
- Unlocked:** The scooter is unlocked. The screen shows a map with a green dot. The bottom card displays 'START RIDING • UNLOCKED', a timer at 0:01, 0 mi, and a '14 mi range'. Buttons for 'Cancel' and 'Unlock' are visible.
- Active:** The scooter is in use. The screen shows a map with a purple line indicating the ride path. The bottom card displays 'RIDE IN PROGRESS', a timer at 1:01, 1.1 mi, and a '14 mi range'. Buttons for 'Pause' and 'End ride' are visible.
- Parking:** The user is prompted to park responsibly. The screen shows a photo of a scooter parked in a designated area. A button for 'End ride' is visible.
- Dropped Off:** The ride has ended. The screen shows a map with a purple line indicating the ride path. The bottom card displays 'RIDE ENDED', 'Cost \$2.75', 'Time 4:25', 'Distance 1.8 mi', and a message: 'You saved \$1 by ending your ride in a Transit Zone!'. Buttons for 'Next' and 'End ride' are visible.
- Ride Rating:** The user is prompted to rate their ride. The screen shows a scooter icon with the number 11-847. The bottom card displays 'Rate your ride' with four stars and a placeholder star, and 'What could be improved?' with buttons for 'Battery', 'Tire', 'Throttle', 'Lights', 'Shaft', 'Handlebar', 'Lock', 'Too Fast', 'Brake', 'Kickstand', 'Noise', and 'Too Slow'. A button for 'Done' is visible.

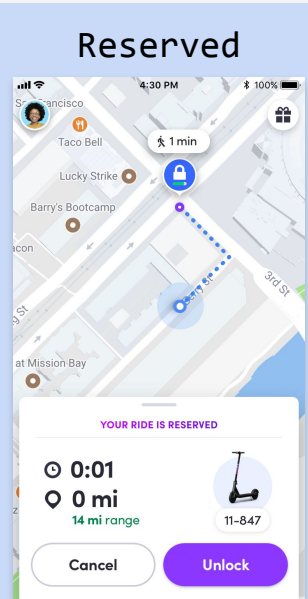
Stay Simple, Stay Lean

Single-state architecture

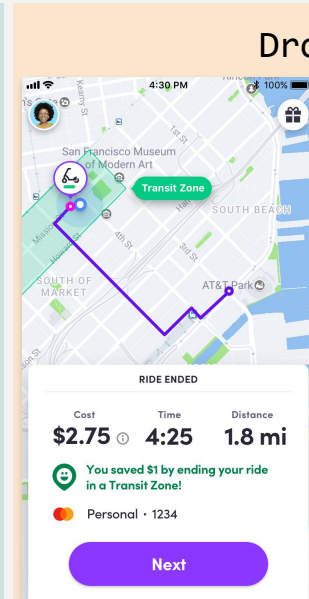
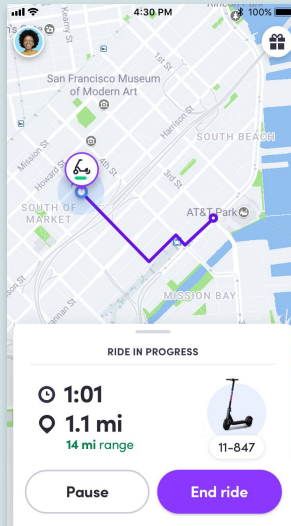
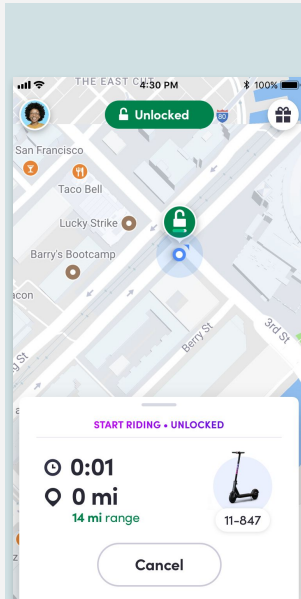
Instead we modeled it as a single state machine with different UI flows corresponding to a single unique server-driven state

RideStatus

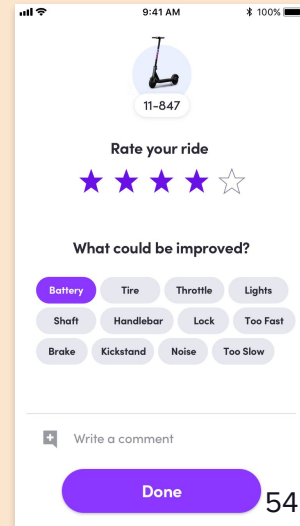
Reserved



Active



Dropoff



Stay Simple, Stay Lean

Single-state architecture

We polled a single endpoint and felt confident that we could reasonably predict what the app would do in any given situation.

```
public class LastMileRideStatus implements INullable {  
  
    public enum Status {  
        IDLE("idle"),  
        RESERVED("reserved"),  
        ACTIVE("active"),  
        DROPOFF("dropoff"),  
        CANCELLED("cancelled"),  
        COMPLETED("completed");  
    }  
}
```

Stay Simple, Stay Lean

Get everyone involved

Everyone from designers to firmware engineers was involved in this architecture.

Making sure everyone knew how it worked was critical to keeping our varying features aligned with our capabilities, and let us easily explain trade offs.

Reimagine Over Reinventing

Treating the codebase like  lego blocks



Should this be a separate app?



We should be ready for 10 million users at launch!

Reimagine Over Reinventing

It's natural when...

You have tight timelines – start from scratch and stay small

You are expecting to scale – reuse everything to leverage existing infrastructure

For us, the optimal path **focus on creative reuse of existing infrastructure while reducing dependence on changing it**



Reimagine Over Reinventing

Leverage everything you can

Build systems, Network stack, Authentication, Databases, Localization

Release process, Testing infrastructure, etc.

Tight timelines aren't possible without building off the great work of your coworkers! 🏋️

Reimagine Over Reinventing

Don't let it slow you down

In our experience, often the biggest roadblocks happen when you depend on another team to change something for you.

Remember: **you have options**, especially if you can explore and communicate them!

Reimagine Over Reinventing

Communicate what you can't change

We were frequently asked if we could push scooter state changes to the client:

```
private Observable<LastMileRideDTO> pollLastMileRide() {  
    return activeRideApi.streamReadLastMileActiveRideAsync(new ReadLMATOBUILDER().build());  
}
```

It was on the roadmap for our networking team, but wouldn't be ready in our timeline.

Reimagine Over Reinventing

Communicate what you can't change

We explained the tradeoffs to our team and found middle ground, restarting our polling at important moments:

```
private Observable<Unit> observeStatusChangesTriggeringRepolling() {  
    return observeRideStatusChangesThatTriggerRepolling()  
        .mergeWith(observeDeviceChangesThatTriggerRepolling())  
        .debounce(200, TimeUnit.MILLISECONDS);  
}
```


Reimagine Over Reinventing

Encapsulate what you can

PM: Can we put a drivers license scanner in the app?

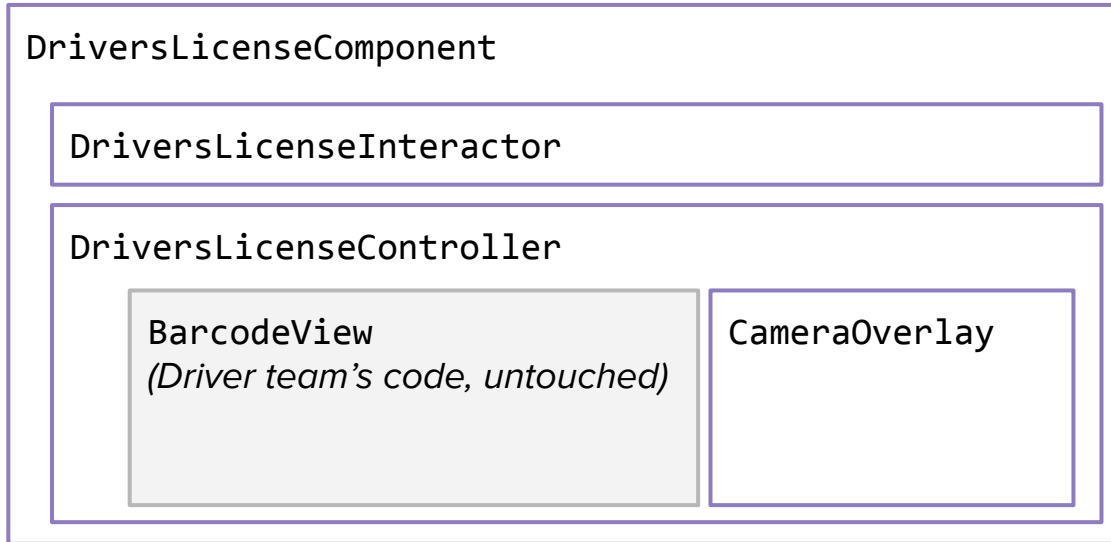
Me: Uhhh, that sounds hard, maybe?

PM: I think we already have it in our driver app

... research, study, find BarcodeView ...

Reimagine Over Reinventing

Composition over Inheritance



Reimagine Over Reinventing

Encapsulate what you can

PM: Can we put a drivers license scanner in the app?

Me: Uhhh, that sounds hard, maybe?

PM: I think we already have it in our driver app

... research, study, make a new DriversLicenseComponent ...

Me: Done!



I've spent many, many hours making minor adjustments to UI.

This is what I want to reuse the most.

```
<!--Image Section-->
<ImageView
    android:id="@+id/ride_mode_image"
    android:layout_width="72dp"
    android:layout_height="wrap_content"
    android:scaleType="fitCenter"
-->
```

```
<com.lyft.android.widgets.shimmer.ShimmerProgressTextView
    android:id="@+id/top_primary_text"
    android:layout_width="@dimen/
    android:layout_marginEnd="8dp"
    android:includeFontPadding="false"
    android:maxLines="1"
    android:ellipsize="end"
    android:gravity="center_vertical"
    android:importantForAccessibility="no"
    app:autoSizeTextType="uniform"
    app:autoSizeMinTextSize="17sp"
    app:autoSizeStepGranularity="1sp"
    app:layout_constraintBottom_toTopOf="@+id/top_secondary_text"
    app:layout_constraintStart_toEndOf="@+id/ride_mode_image"
    app:layout_constraintEnd_toStartOf="@+id/bottom_primary_icon"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_chainStyle="packed"
    tools:text="Lyft"
-->
```

```
<TextView
    android:id="@+id/top_secondary_text"
    style="@style/design_core_Label3_deprecated"
    android:layout_width="wrap_content"
-->
```



Sriracha kitsch franz

Lorem ipsum dolor amet narwhal
truffaut ethical kinfolk

Primary CTA



Headline

Lorem ipsum dolor amet narwhal
truffaut ethical kinfolk

Dismiss

Action

Details Text



Headline

Body copy body copy body copy body

Primary CTA

Secondary CTA

Cancel



12:30

← Title



A



Single Item

Action

Label
Answer



Label

56/100

First line of text you can put until
this line will move

Single Item



Single Item



Single Item



LPL: Lyft's Product Language



Sriracha kitsch franz

Lorem ipsum dolor amet narwhal
truffaut ethical kinfolk

Primary CTA



Headline

Lorem ipsum dolor amet narwhal
truffaut ethical kinfolk

Dismiss

Action

Details Text



Headline

Body copy body copy body copy body

Primary CTA

Secondary CTA

Cancel



A



Single Item

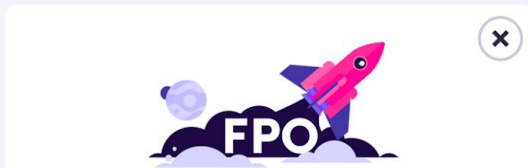
Action

Label Answer

Label 56/100

First line of text you can put until
this line will move

- Single Item
- Single Item
- Single Item



Reimagine Over Reinventing

LPL: Lyft's Product Language

Comprehensive library of UI elements, designed and developed for usability, consistency and accessibility

We could just glance at mocks and know exactly what size, font, and colors we're using

Primary CTA



Headline

Lorem ipsum dolor amet narwhal truffaut ethical kinfolk

Dismiss

Action



Title



A



Single Item

Action

Label

Answer



Label

56/100

First line of text you can put until this line will move



Single Item



Single Item



Single Item



Reimagine Over Reinventing

LPL: Lyft's Product Language

What we got for free:

- Loading states
- Disabled states
- l18n & A11y
- Consistent UX across app
- Pixel-Polished UI
- Well documented APIs

Headline F1

Lyft Pro UI Bold
Size: 26pt

Headline F2

Lyft Pro UI Bold
Size: 22pt

Primary

Primary Elevated

Secondary

Title F1

Proxima Nova Bold
Size: 20pt

Title F2

Proxima Nova Bold
Size: 17pt

TITLE F3

Proxima Nova Bold
Size: 13pt

Subtitle F1

Proxima Nova Medium
Size: 20pt

Subtitle F2

Proxima Nova Medium
Size: 17pt

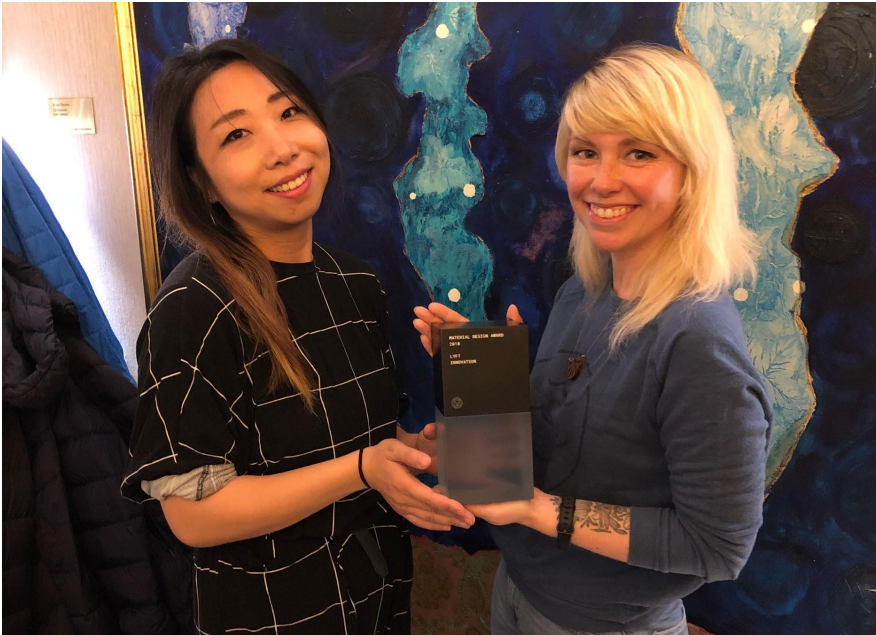
Subtitle F3

Proxima Nova Medium
Size: 15pt

Reimagine Over Reinventing

Possibly the best part of LPL...

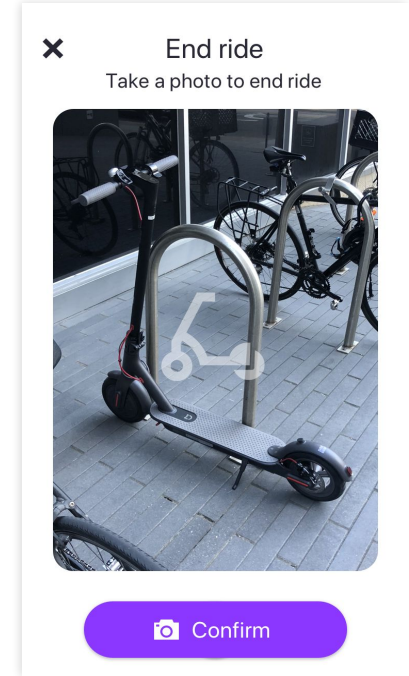
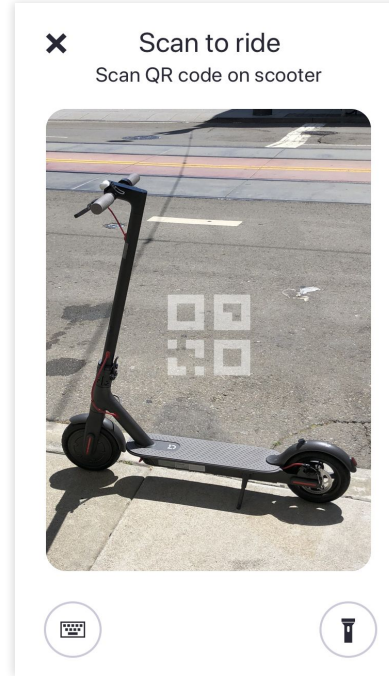
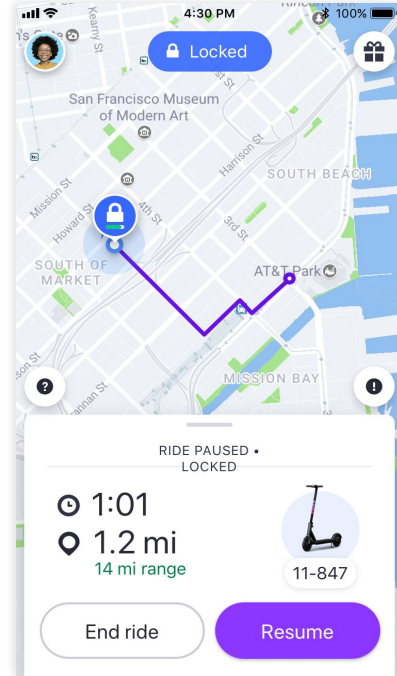
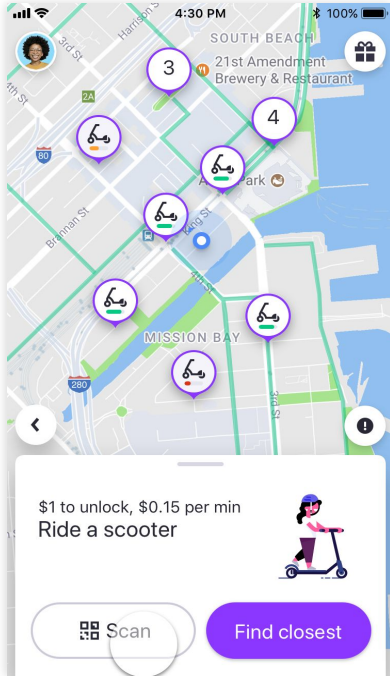
Our designers are passionate about making sure our product language works within and innovates on both Android and iOS.



Reimagine Over Reinventing

Possibly the best part of LPL...

And iOS mocks totally worked for Android!



Reimagine Over Reinventing

Lyft Product Language in action

The common language was key to communicating tradeoffs between design and engineering

Being involved early and often helped avoid the “unimplementable mocks” scenario

Sriracha kitsch franz

Lorem ipsum dolor amet narwhal
truffaut ethical kinfolk

Primary CTA



Headline

Lorem ipsum dolor amet narwhal
truffaut ethical kinfolk

Dismiss

Action

Secondary CTA

Cancel



Title

12:30



A



Single Item

Action

Label
Answer

Label

56/100

First line of text you can put until
this line will move

Single Item

Single Item

Single Item



Reimagine Over Reinventing

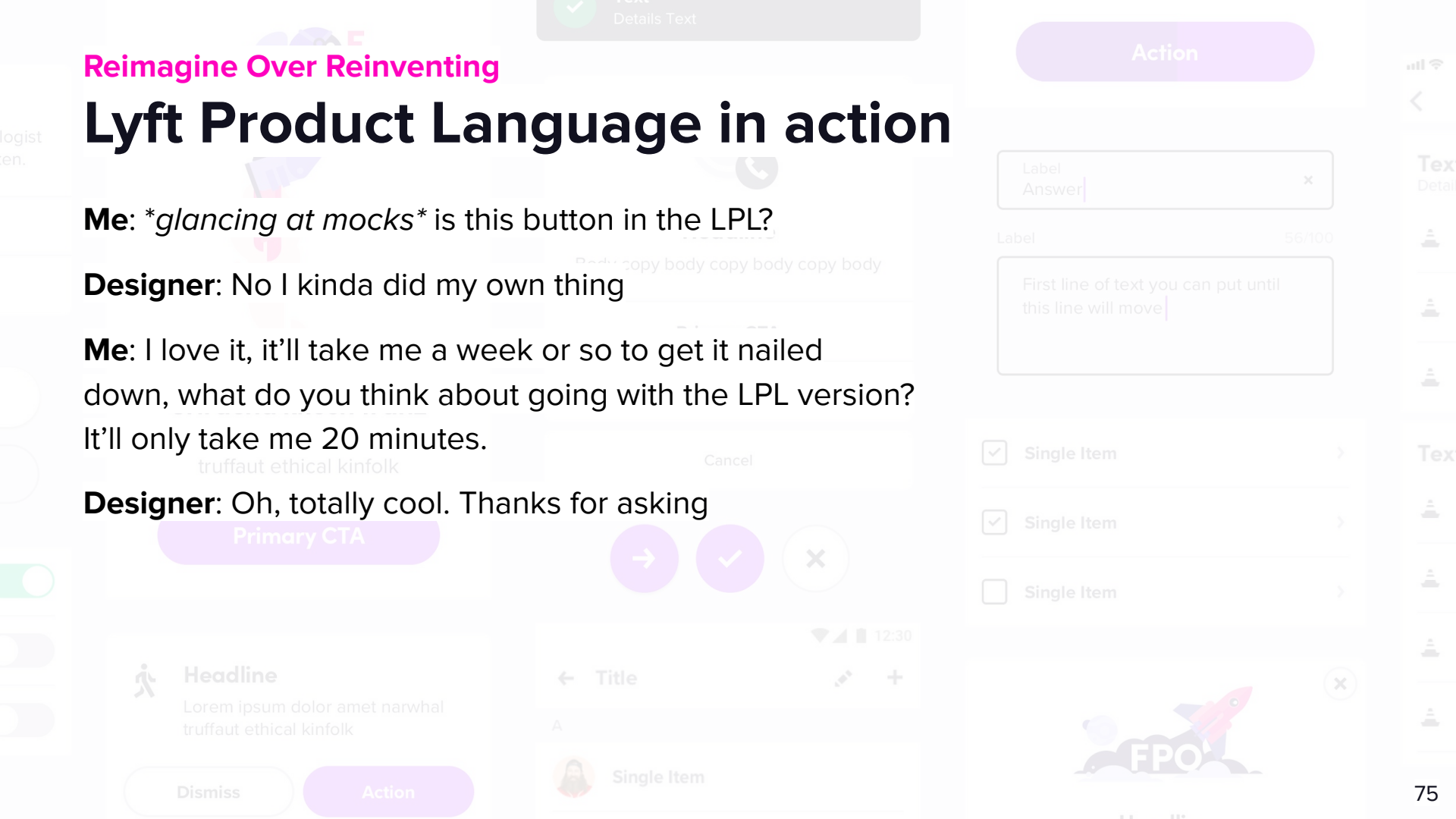
Lyft Product Language in action

Me: **glancing at mocks** is this button in the LPL?

Designer: No I kinda did my own thing

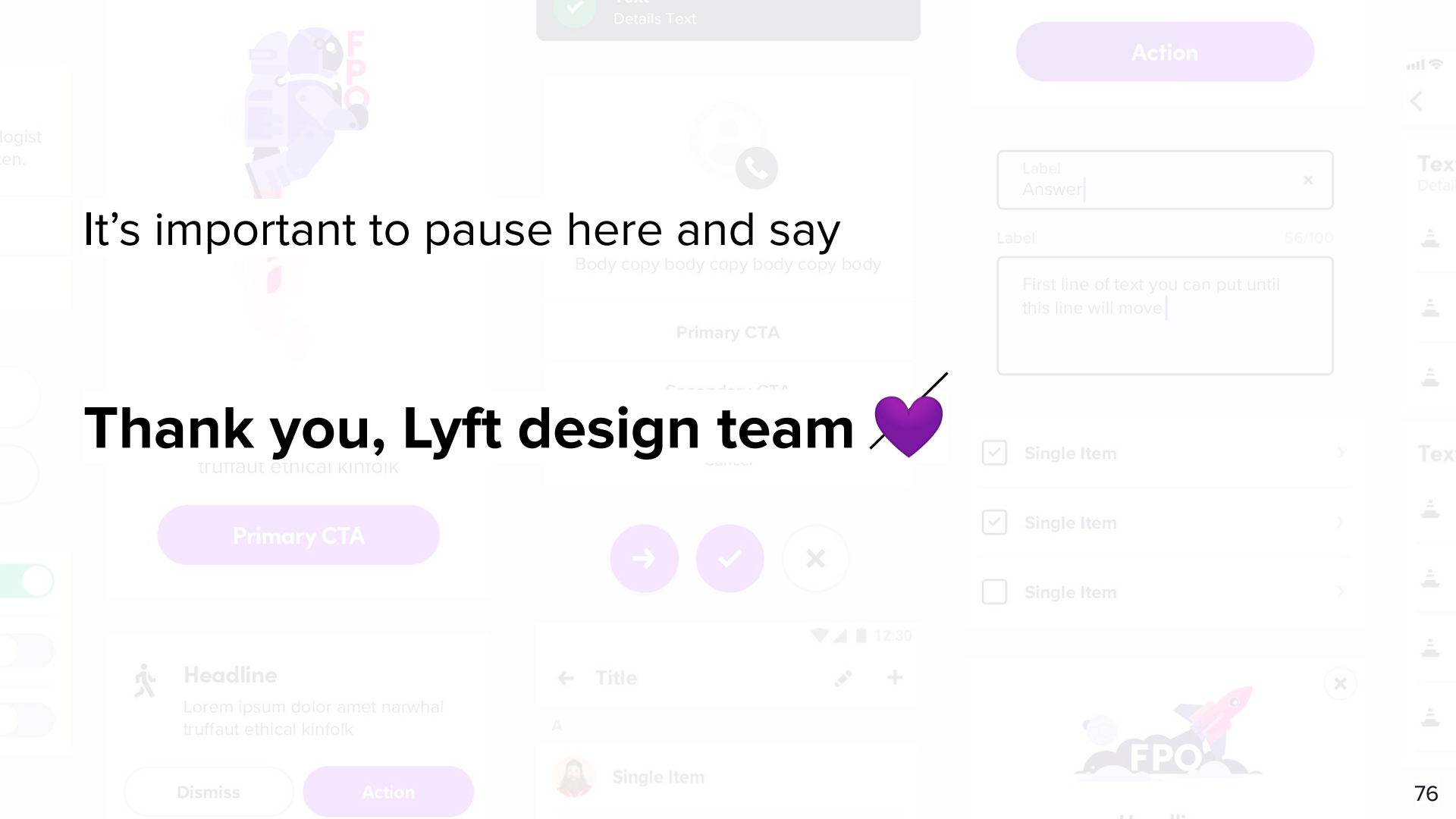
Me: I love it, it'll take me a week or so to get it nailed down, what do you think about going with the LPL version? It'll only take me 20 minutes.

Designer: Oh, totally cool. Thanks for asking



It's important to pause here and say

Thank you, Lyft design team 

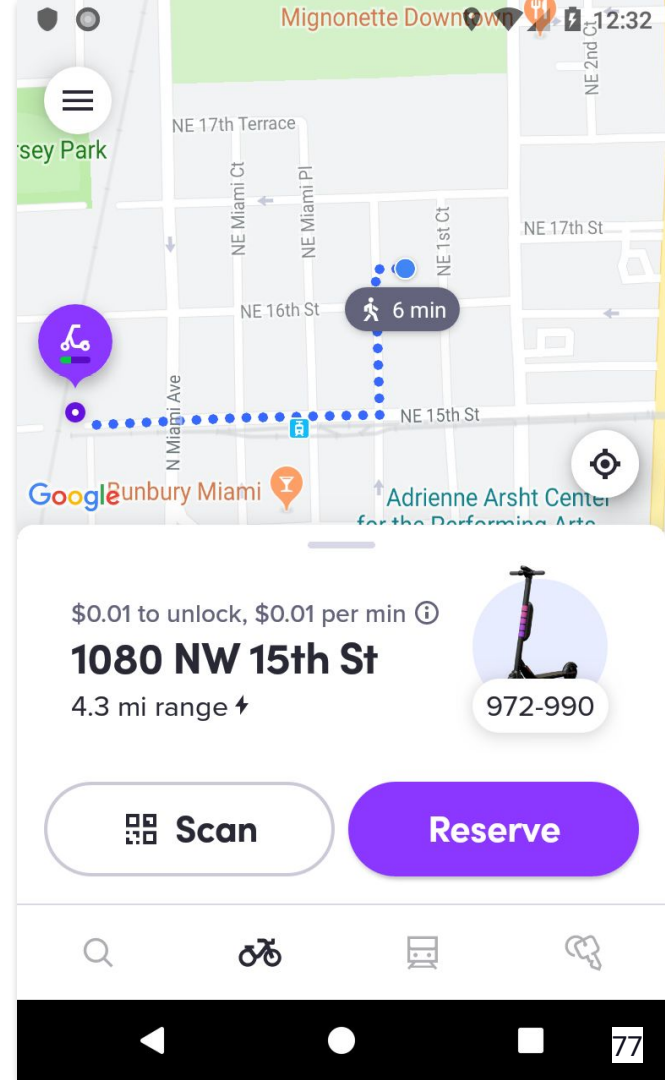


Reimagine Over Reinventing

When you can't reuse...

Eventually we had to write some custom UI components.

For us it was the map bubbles.

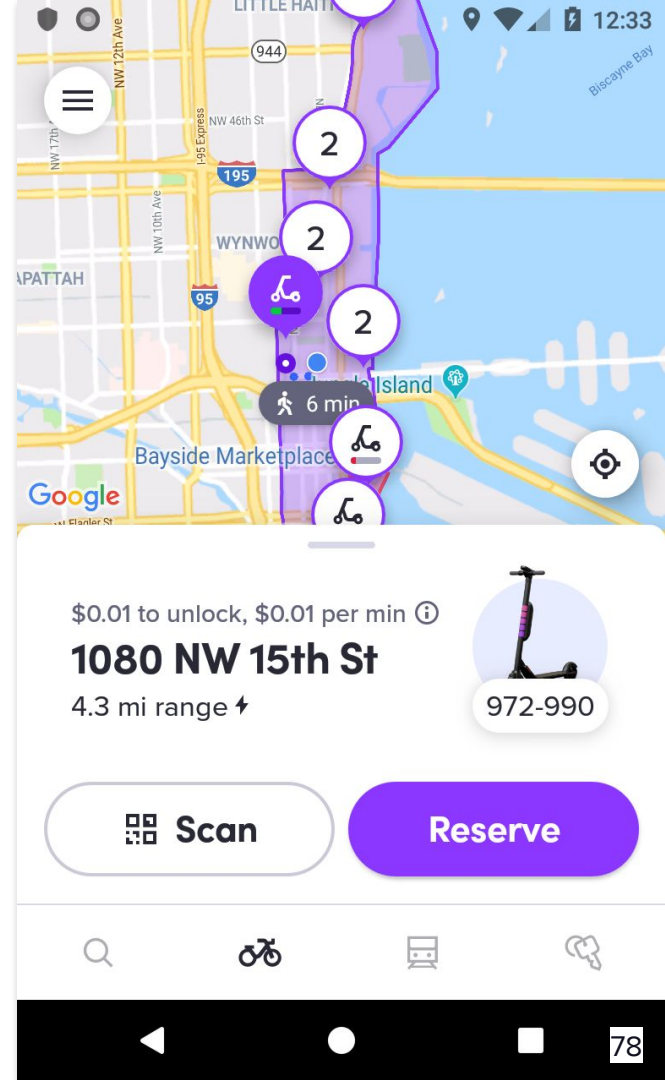


Reimagine Over Reinventing

Bubbles & Clusters

Lyft hadn't extensively used the map enough to include any clustering libraries, and we abstracted away the map implementation so we didn't have access to Google's.

The map clusters were part of our core Golden Path experience, so this was worth the tradeoff.



When I'm in a time crunch, I optimize my code for...

**Readable, predictable,
and works well
enough for our
expected dataset**



**Complex but with
optimal asymptotic
runtime under all
conditions**

Reimagine Over Reinventing

$O(n^2)$ time! 

What do I do with my degree in computer science?

Write a map clustering algorithm in $O(n^2)$ time.

```
public static List<RidableCluster> fromRidables(List<Ridable> ridables, IMapPosition mapPosition) {  
    double metersPerPixel = zoomToMetersPerPixel(mapPosition);  
    double metersGridSize = metersPerPixel * CLUSTER_SIZE_DP;  
    List<ClusterAndAverage> ridableClusters = new ArrayList<>();  
    Iterables.forEach(ridables, ridable -> addToClusterList(ridable, ridableClusters, metersGridSize));  
    //TODO move to google maps ClusterManager  
    return Iterables.map(ridableClusters, ridableList -> makeRidablesCluster(ridableList, selectedRidable));  
}
```

Reimagine Over Reinventing

Why? 🤔 → 😊

1. It's easy to read, and easy to reason that it'll work in all cases
2. Our dataset was reasonably small enough where the added performance wasn't worth it for the time it would take

```
public static List<RidableCluster> fromRidables(List<Ridable> ridables, IMapPosition mapPosition) {  
    double metersPerPixel = zoomToMetersPerPixel(mapPosition);  
    double metersGridSize = metersPerPixel * CLUSTER_SIZE_DP;  
    List<ClusterAndAverage> ridableClusters = new ArrayList<>();  
    Iterables.forEach(ridables, ridable -> addToClusterList(ridable, ridableClusters, metersGridSize));  
    //TODO move to google maps ClusterManager  
    return Iterables.map(ridableClusters, ridableList -> makeRidablesCluster(ridableList, selectedRidable));  
}
```

Listen, Learn & Launch What Matters



Preparing to enter the real world





**Are we sure we're
building the right thing?**



**Are we ready for launch
day?**



**Are we sure we're
building the right thing?**



**Are we ready for launch
day?**

Listen, Learn & Launch What Matters

What matters to our user?

We had a zillion questions about what, when and how our users were going to use Lyft scooters.

Would they...

- Use the “reserve” feature?
- Understand how to lock and unlock it?
- Feel natural to get a scooter within the Lyft app?



Listen, Learn & Launch What Matters

When your feature is already live


For established products, we iteratively release and roll out, A/B testing along the way. This helps us understand user behavior and preferences, and guards against major issues.

Since we had never done something like this before, we couldn't use any of these processes.

Listen, Learn & Launch What Matters

How to learn when you aren't live

We relied on

 foundational research and

 usability testing,

guided by our research team





It's important to pause here and say

Thank you, Lyft research team 



Are we sure we're
building the right thing?



Are we ready for launch
day?

Listen, Learn & Launch What Matters

How we built in parallel

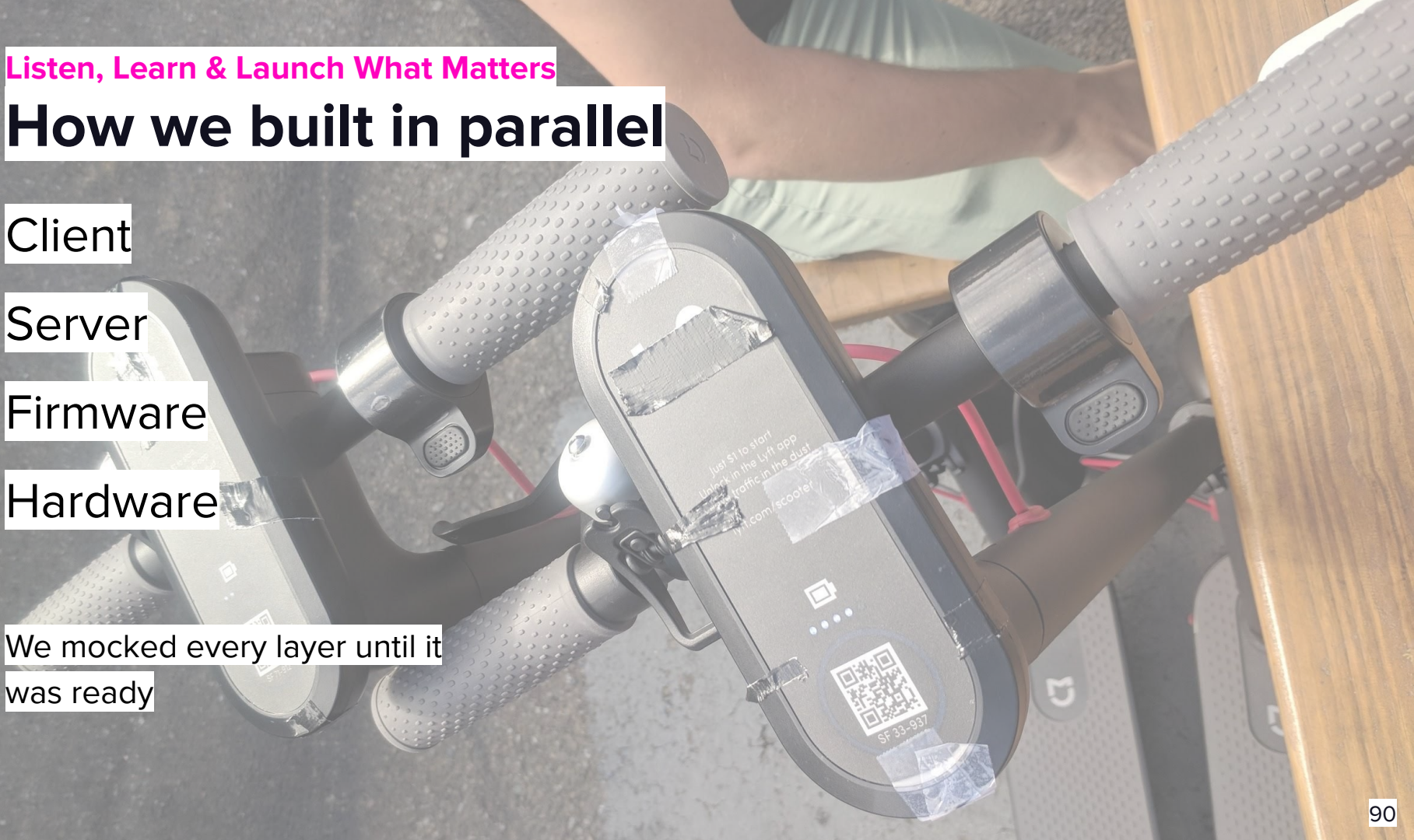
Client

Server

Firmware

Hardware

We mocked every layer until it was ready



Client

Server

Firmware

Hardware



Client-only testability

We buried the mocks down our client stack as far as possible

```
public Single<Result<LastMileRide, IError>> reserve(Ridable rideable) {
    return doReserveApi(rideable).flatMap(result -> {
        if (Results.isSuccess(result)) {
            return lastMileRideProvider.updateRide(this::mapReserve(rideable));
        } else { return handleError(result); }});
}

private Single<Result<Object, IError>> doReserveApi(Ridable rideable) {
    // TODO: Actually do the api call.
    return Single.just(Results.success(rideable));
}
```

Listen, Learn & Launch What Matters

Client-only testability

We buried the mocks down our client stack as far as possible

(don't forget to remove them later!)

```
public Single<Result<LastMileRide, IError>> reserve(Ridable rideable) {
    return doReserveApi(rideable).flatMap(result -> {
        if (Results.isSuccess(result)) {
            return lastMileRideProvider.updateRide(this::mapReserve(rideable));
        } else { return handleError(result); }});
}
```

```
private Single<Result<Object, IError>> doReserveApi(Ridable rideable) {
    // TODO: Actually do the api call.
    return Single.just(Results.success(rideable));
}
```


Client

Server

Firmware

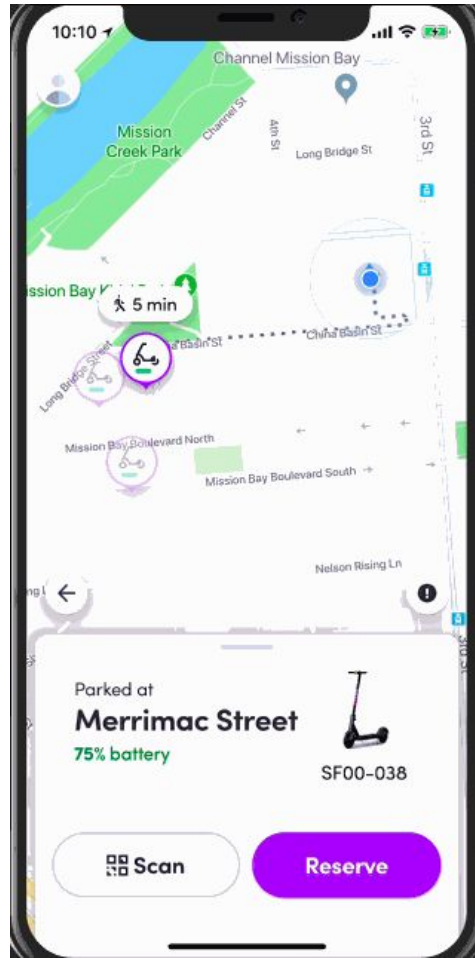
Hardware



Listen, Learn & Launch What Matters

Client<>Server testing

As we got farther along,
scripting and field testing.



```
[22:10:20] narabadjiev:lyft $ python3 ridables-new.py --staging --id 38
```



Listen, Learn & Launch What Matters

Client<>Server testing

My only python contribution at Lyft:

Add janky test_ridables.py script #145

 Merged

lyft-buildnotify-5 merged 3 commits into `master` from `test_ridables`  on Jul 13, 2018

Client

Server

Firmware

Hardware



Listen, Learn & Launch What Matters

Getting the last pieces ready

We also knew no matter how careful we were, not all the pieces would fit on the first try:

- Tweaks to app logic were necessary
- Integration required lots of patience

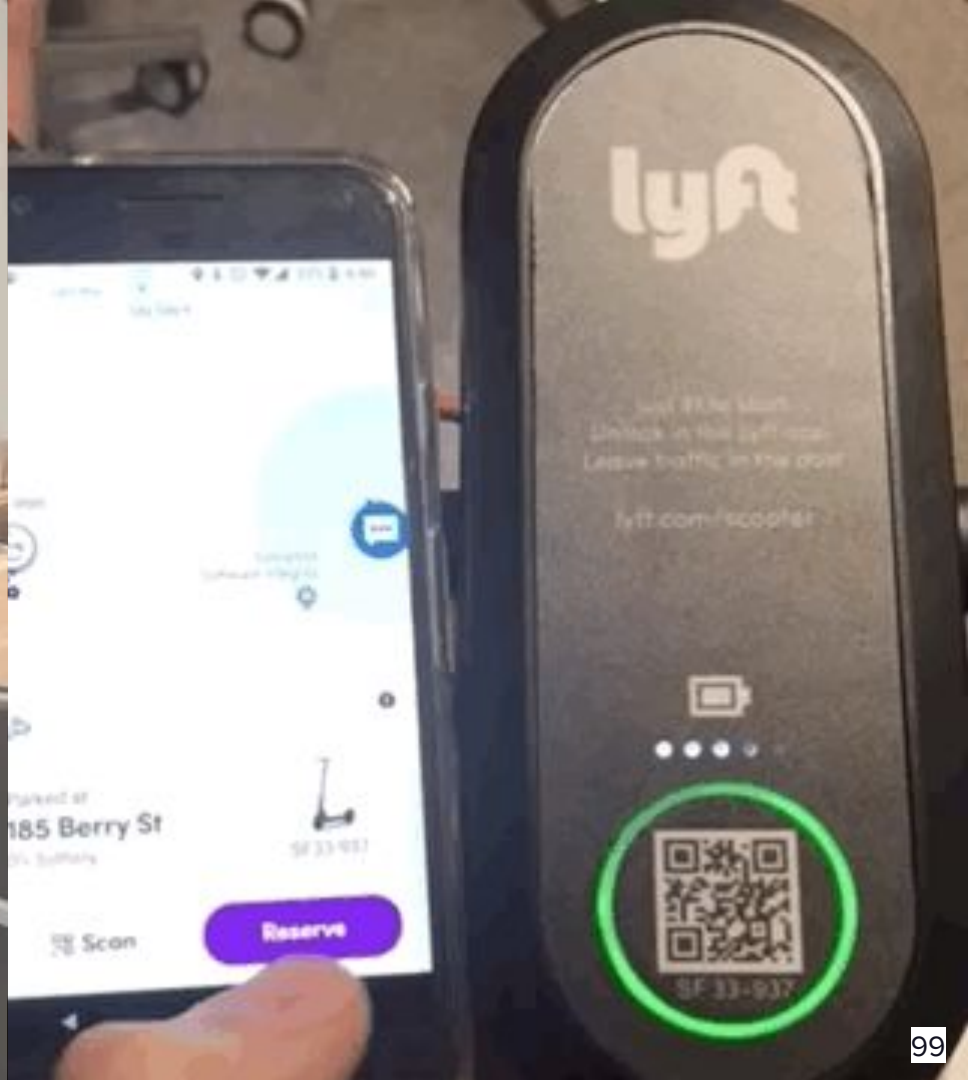


Client

Server

Firmware

Hardware



A person wearing a bright yellow sweater is holding a black and pink Lyft scooter. The scooter has the Lyft logo on the stem and four horizontal pink bands. The person is standing in front of a brick wall. To the right, another person in a green sweater is also holding a similar scooter. A white text box is overlaid on the image.

But are we ready for launch?

Listen, Learn & Launch What Matters

Build for flexibility

Client code is inflexible. Your APK is live. Where do you add flexibility?

Server side!

- Feature flagging different aspects
- Configuration flags
- Server-driven resource overrides

Name	passenger_x_last_mile_service_unavailable_description
Description	Description text displayed when the current time is outside the service hours for the visible map viewport
Type	string
Default Value	Even two-wheelers need their beauty rest.

Build for flexibility

```
public class DynamicResourcesWrapper extends Resources {  
  
    @Override  
    public String getString(int id) throws NotFoundException {  
        final String stringId = getStringKey(id);  
        final String override = constantsProvider.get(stringConstant(stringId))  
        if (override != null) {  
            return override  
        }  
        return originalResources.getString(id);  
    }  
}
```

Listen, Learn & Launch What Matters

Listen & learn on launch day

Seeing real users on launch day is both emotionally rewarding, and important to debug issues. We were able to ask questions and gather feedback.



Listen, Learn & Launch What Matters

Listen & learn on launch day

Over time, these learnings helped us better understand our users, refine our north star vision, prioritize our backlog, and formalize our launch process for future cities and releases.

Key Takeaways





To Recap

Waste less precious time

Build more meaningful stuff

To Recap

Product Engineering:

1. Maximize your effort
2. Own what you code
3. Ship meaningful stuff

To Recap

Engineering Principles:

1. Stay simple and lean
2. Reimagine over reinventing
3. Listen, learn, launch what matters

Thanks!

Try a Lyft bike or scooter! lyft.com/scooters

Read more on eng.lyft.com

Follow me on social: @rjmarsan

Lyft is hiring all sorts of talented engineers around the world!

