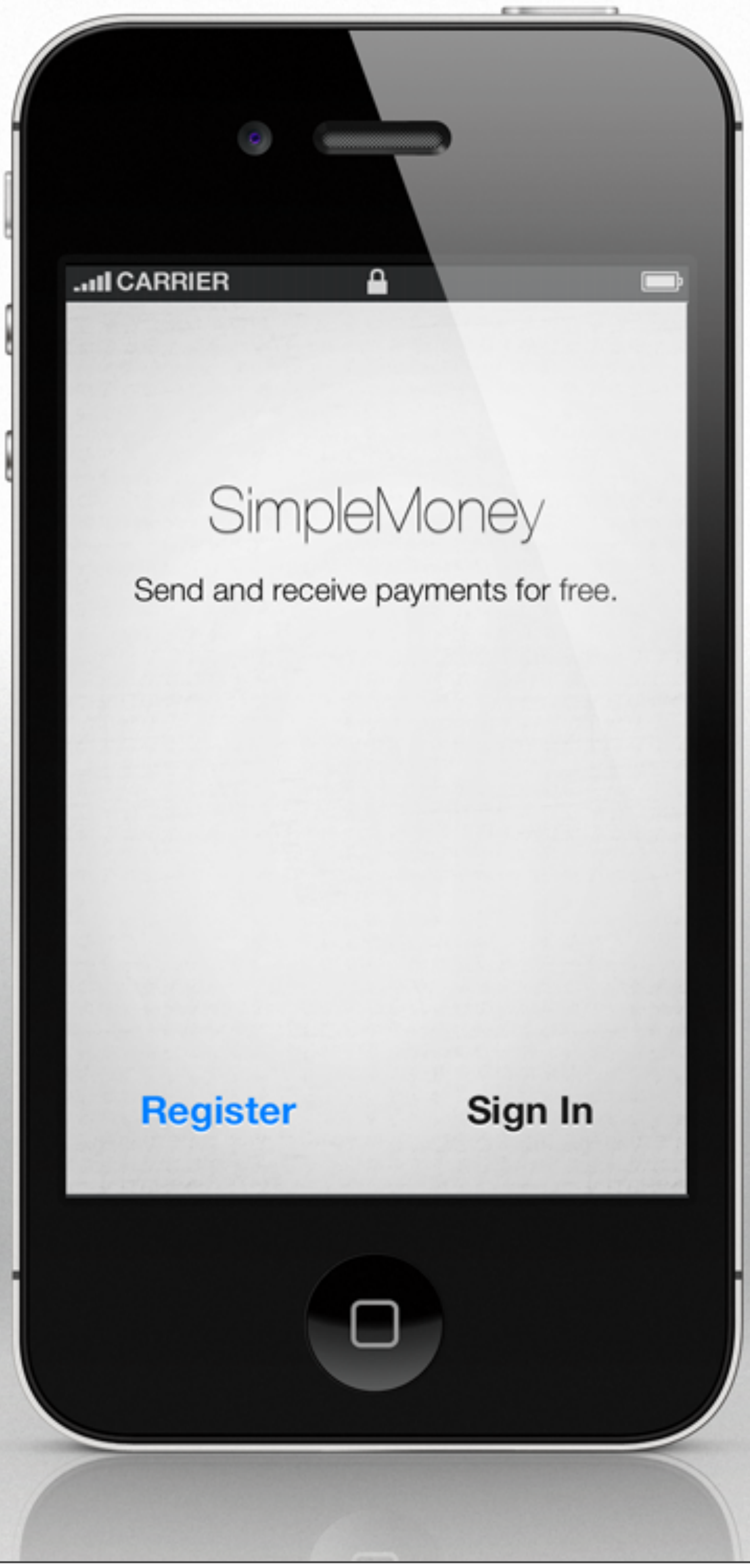
Making Money Simple



Team Saon
Arthur Pang - Joshua Conner - Nicholas Pallares
April 27, 2012

Our Client

Joshua Cross

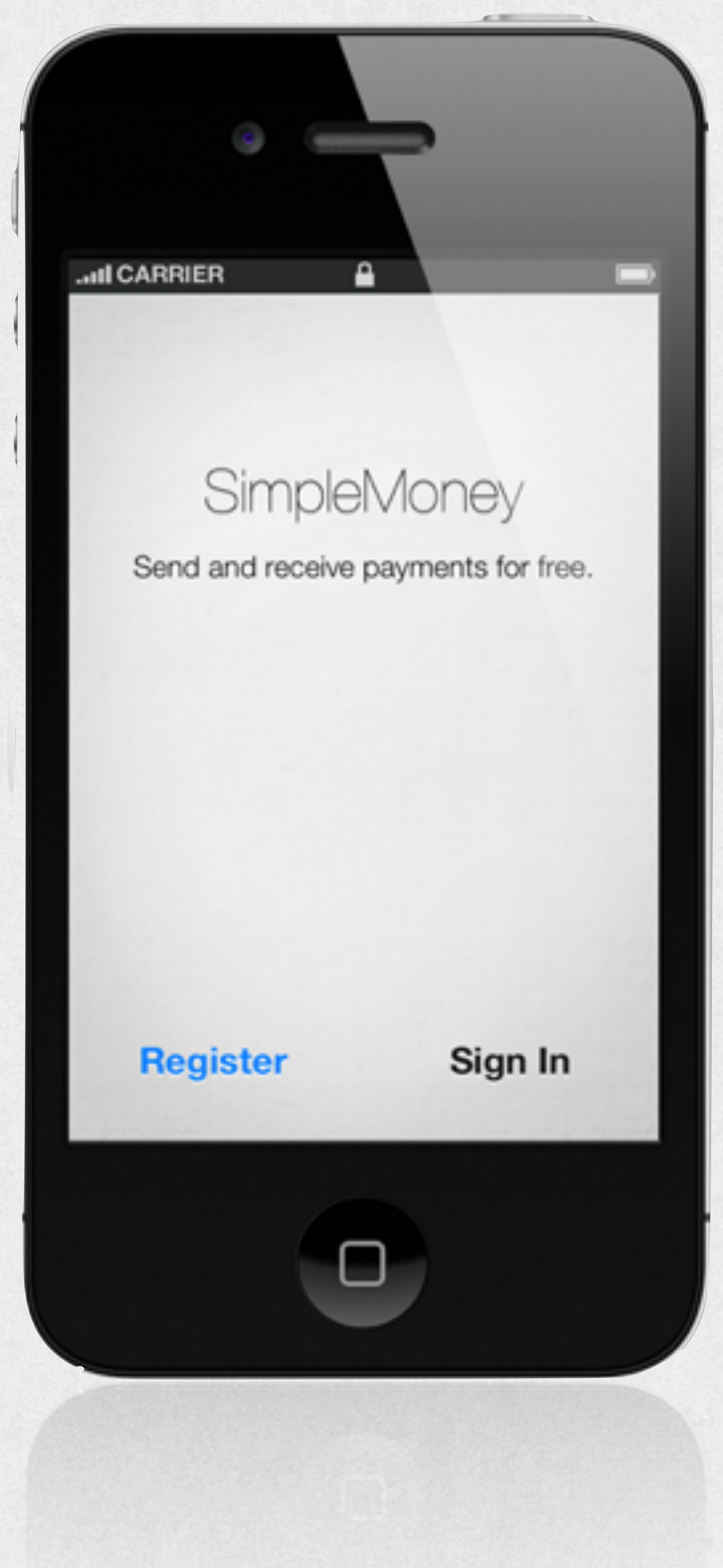
CEO, Hermes Commerce
Ph.D., Applied Physics, Cornell

Received grant from NSF to develop a mobile payments platform.

Dr. Cross recieved a grant from the NSF to develop an easy-to-use, secure and fee-free mobile payments platform.

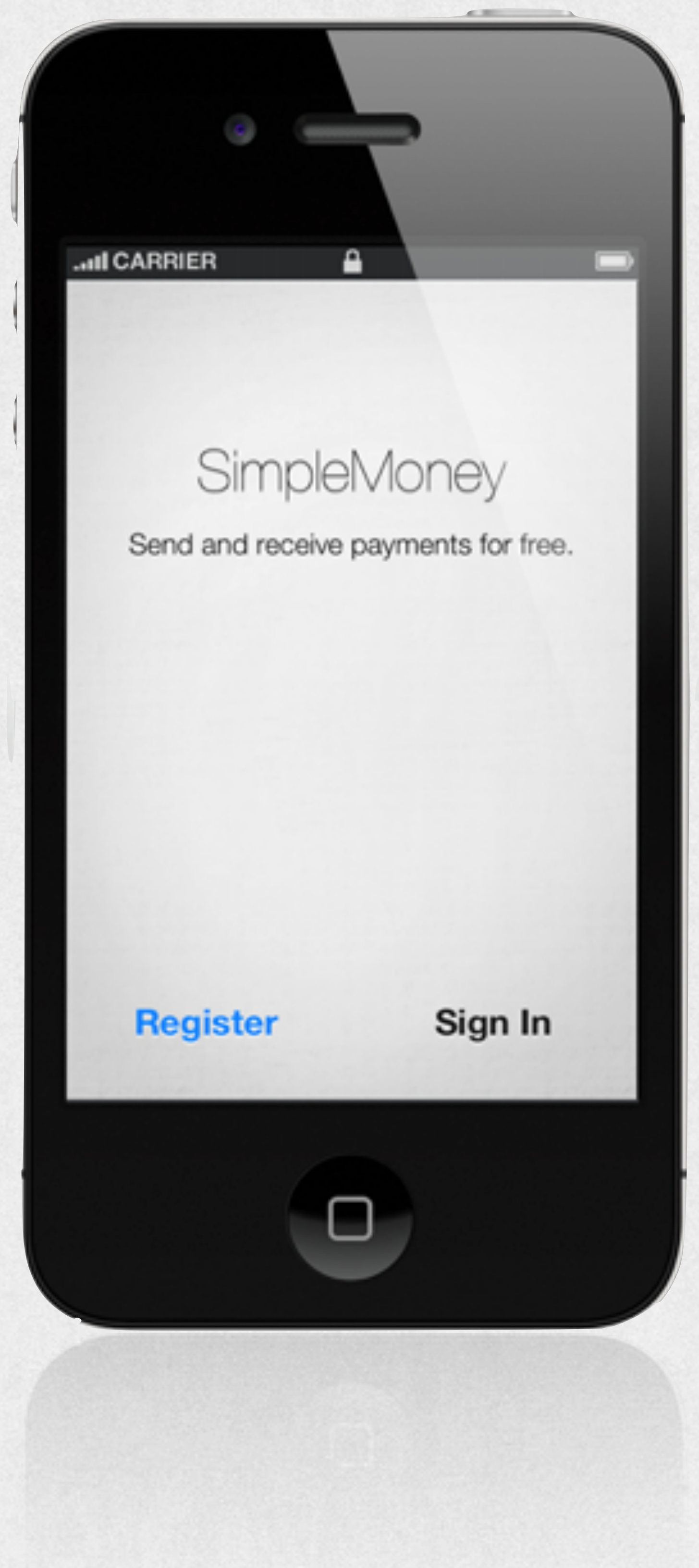
Our Task

- Consumer-facing mobile apps for iPhone and Android



- Came to us to build the keystone of the platform
- In our initial meeting with Dr. Cross, he told us that one of the biggest challenges he thought Hermes faced in developing SimpleMoney was overcoming inertia

So we asked ourselves: what could we do to help him overcome inertia? Build a great app; but what makes a really great, transformative app?

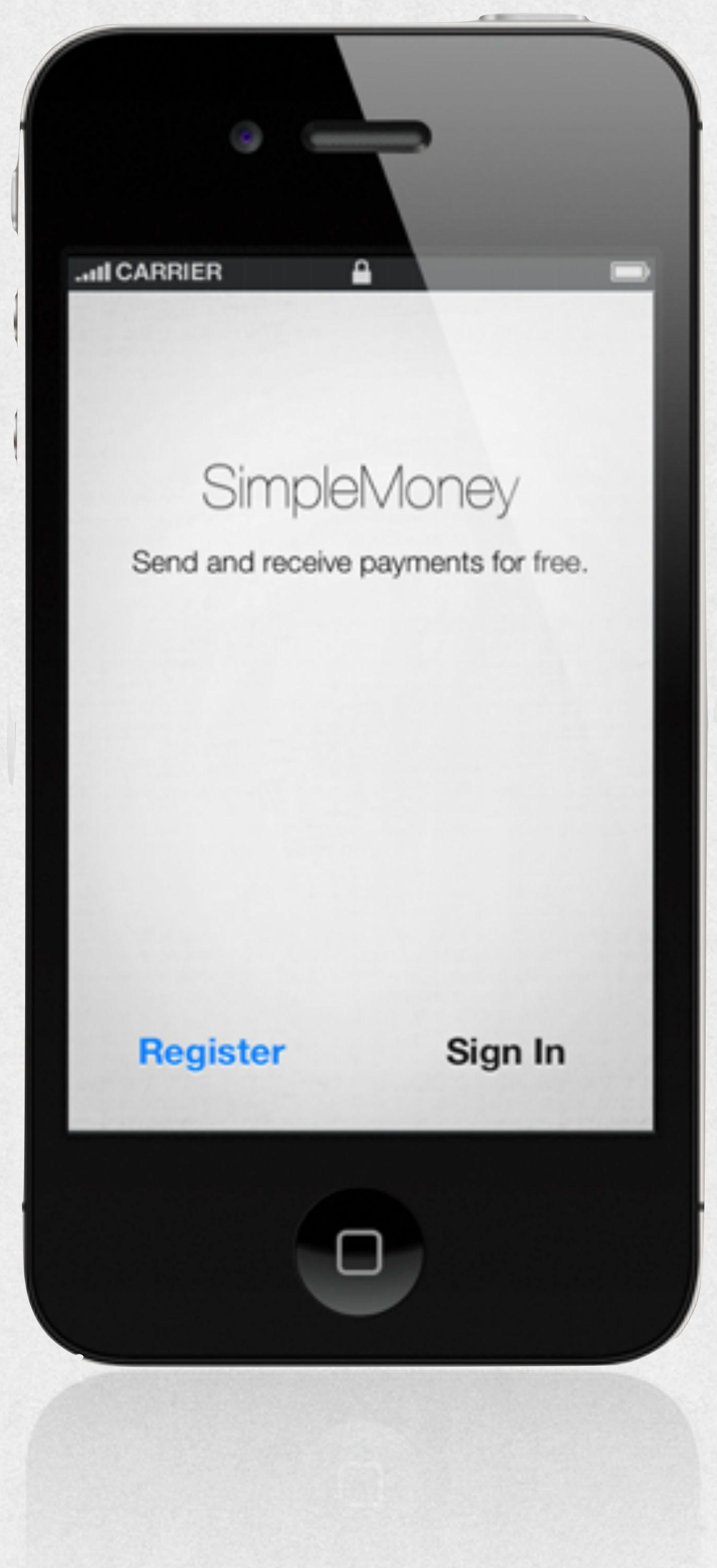


Our Task

- Consumer-facing mobile apps for iPhone and Android
- Overcome inertia

- Came to us to build the keystone of the platform
- In our initial meeting with Dr. Cross, he told us that one of the biggest challenges he thought Hermes faced in developing SimpleMoney was overcoming inertia

So we asked ourselves: what could we do to help him overcome inertia? Build a great app; but what makes a really great, transformative app?

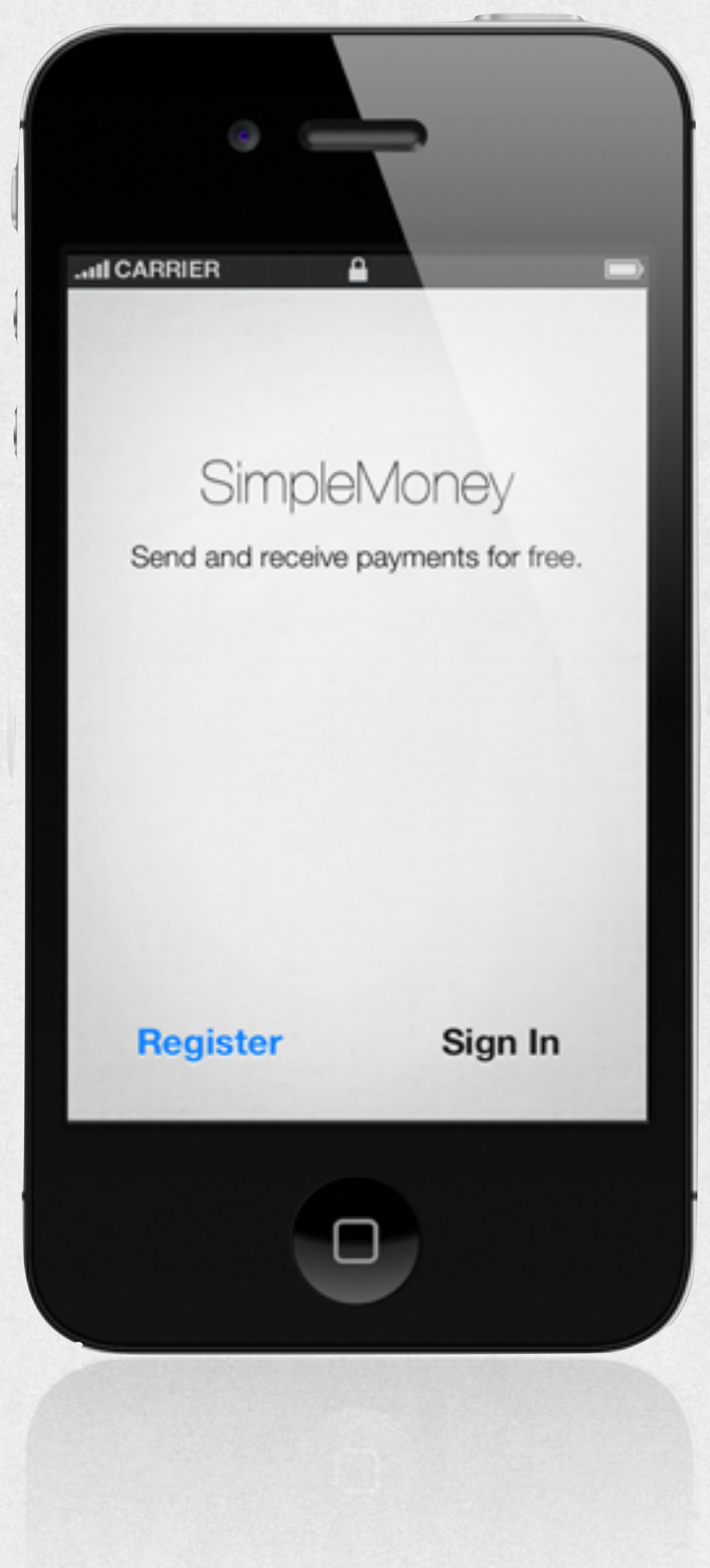


Our Task

- Consumer-facing mobile apps for iPhone and Android
- Overcome inertia
- Be a compelling alternative to credit cards

- Came to us to build the keystone of the platform
- In our initial meeting with Dr. Cross, he told us that one of the biggest challenges he thought Hermes faced in developing SimpleMoney was overcoming inertia

So we asked ourselves: what could we do to help him overcome inertia? Build a great app; but what makes a really great, transformative app?



Our Task

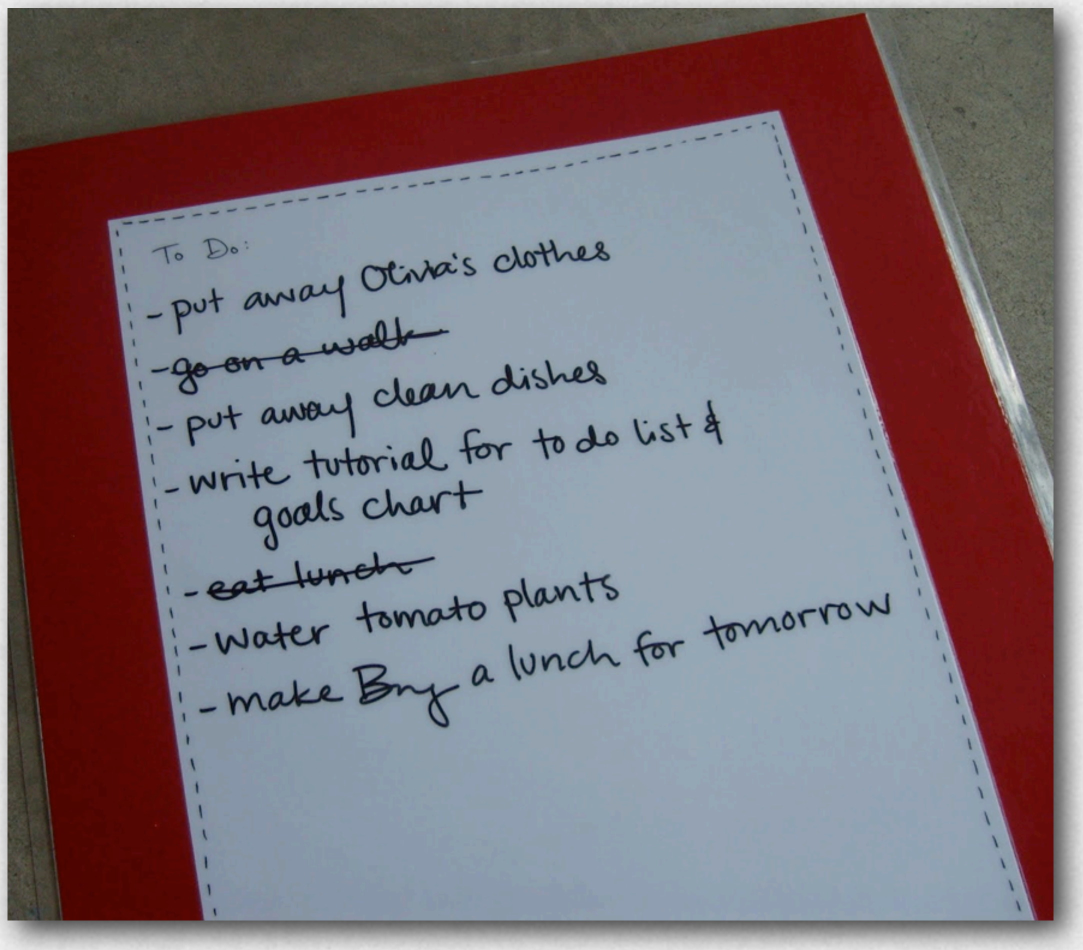
- Consumer-facing mobile apps for iPhone and Android
- Overcome inertia
- Be a compelling alternative to credit cards

...but how?

- Came to us to build the keystone of the platform
- In our initial meeting with Dr. Cross, he told us that one of the biggest challenges he thought Hermes faced in developing SimpleMoney was overcoming inertia

So we asked ourselves: what could we do to help him overcome inertia? Build a great app; but what makes a really great, transformative app?

Smart lists



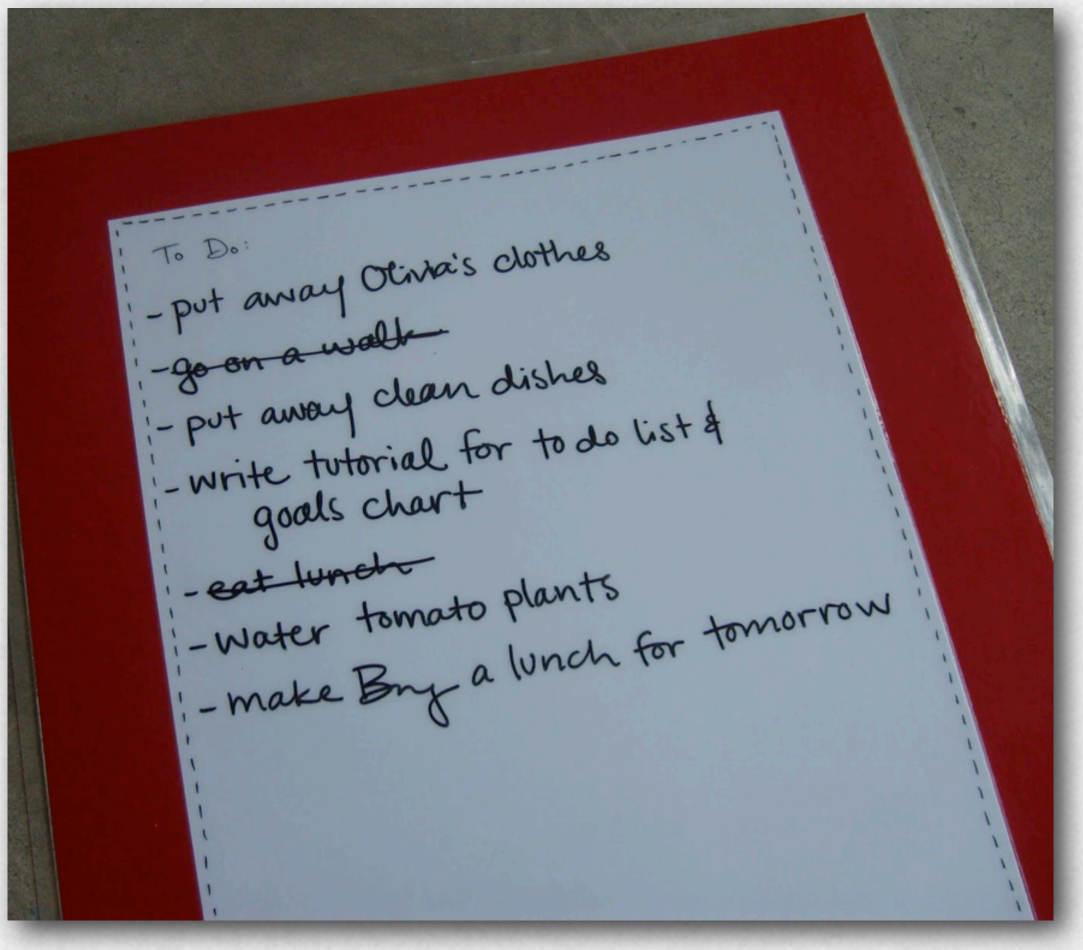
To Do:

- put away Olivia's clothes
- ~~go on a walk~~
- put away clean dishes
- write tutorial for to do list & goals chart
- ~~eat lunch~~
- water tomato plants
- make Bry a lunch for tomorrow

Let's consider some other apps that have done a good job at replacing their tangible counterparts...

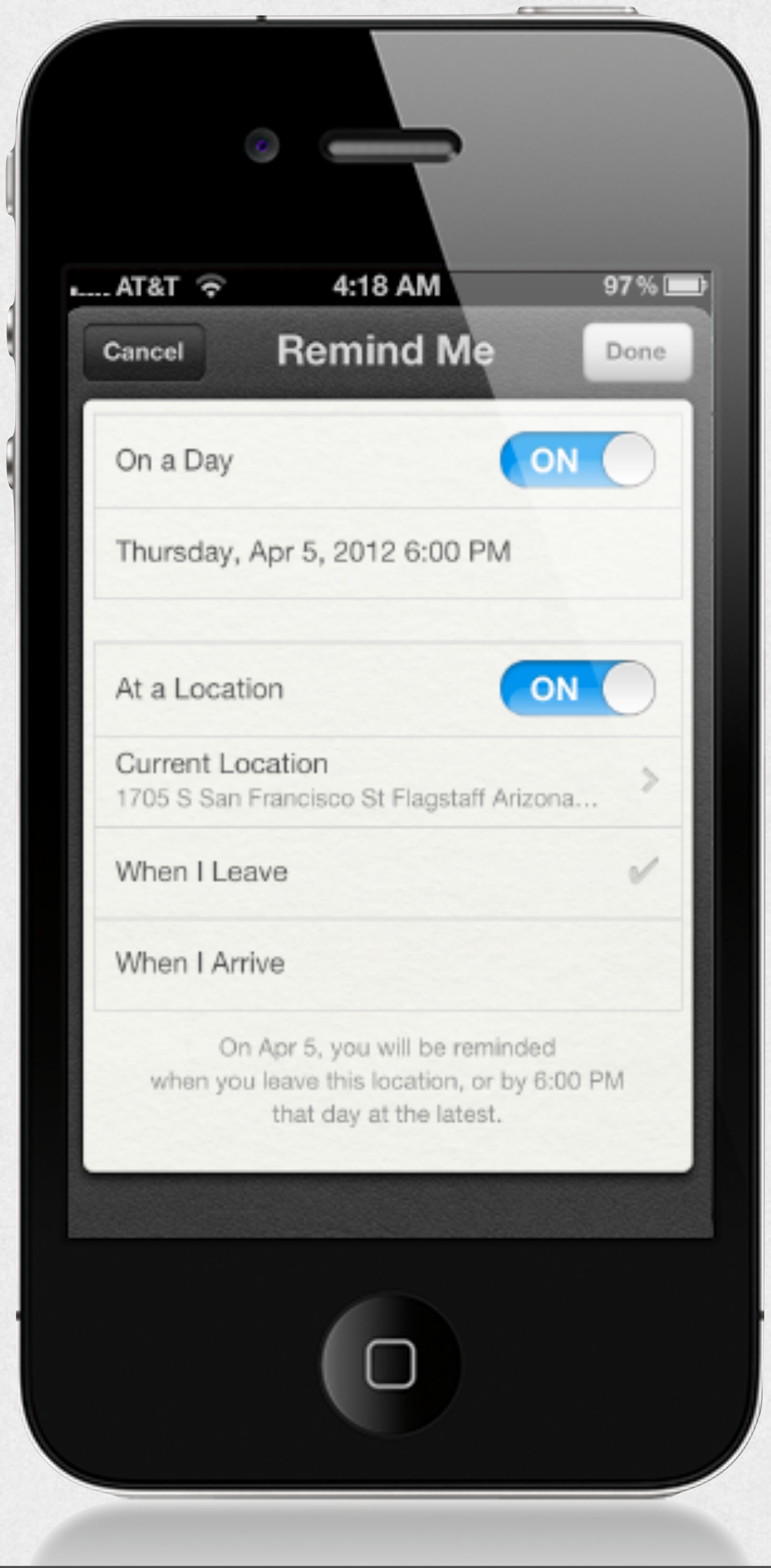
Todo list vs. iPhone "Reminders" app"
- can not only set off reminder at particular time
- but at particular PLACE

Smart lists


To Do:

- put away Olivia's clothes
- ~~go on a walk~~
- put away clean dishes
- write tutorial for to do list & goals chart
- ~~eat lunch~~
- water tomato plants
- make ~~Bry~~ a lunch for tomorrow

vs.


Cancel | Remind Me | Done

On a Day: ON

Thursday, Apr 5, 2012 6:00 PM

At a Location: ON

Current Location
1705 S San Francisco St Flagstaff Arizona...

When I Leave ✓

When I Arrive

On Apr 5, you will be reminded when you leave this location, or by 6:00 PM that day at the latest.

Let’s consider some other apps that have done a good job at replacing their tangible counterparts...

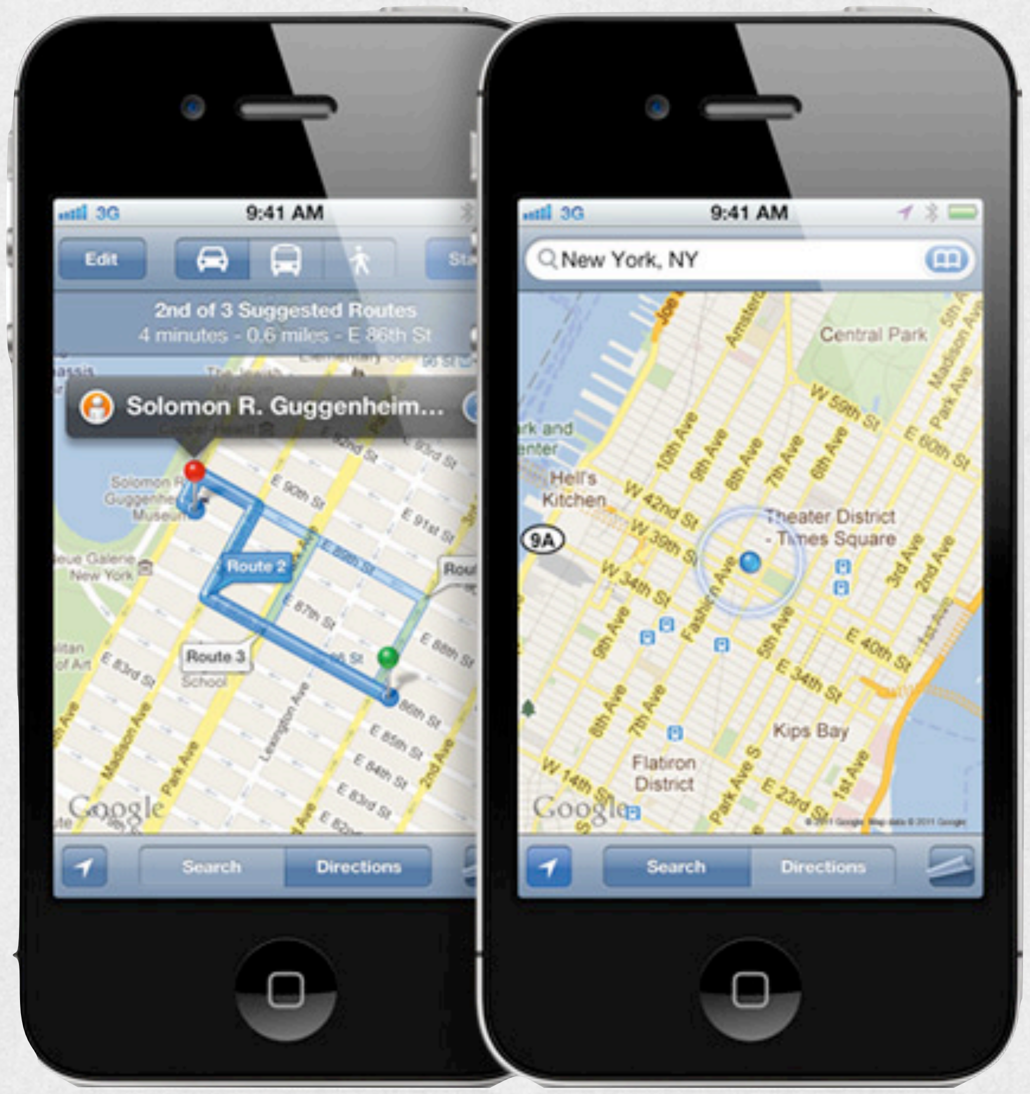
Todo list vs. iPhone “Reminders” app”
- can not only set off reminder at particular time
- but at particular PLACE

Smart maps

Dumb map (they're lost! how do you orient?)
vs.
iPhone or Android "Maps" app
 - uses GPS chip to figure out where you are and give you turn-by-turn directions
- directions take into account the traffic on the roads you'd take to give you the fastest route at that exact moment in time.
- don't need to know an address at all! can type in "target" to get nearest "Target" store

Smart maps

vs.



Dumb map (they're lost! how do you orient?)
vs.
iPhone or Android "Maps" app
- uses GPS chip to figure out where you are and give you turn-by-turn directions
- directions take into account the traffic on the roads you'd take to give you the fastest route at that exact moment in time.
- don't need to know an address at all! can type in "target" to get nearest "Target" store

Smart cards?

- CC’s are “dumb” - can’t even simple things like checking balance from card
- Merchants paid $48 billion in swipe fees in 2011
- Losing CC on vacation -> mega bummer

Smart cards?

- Not smart
- Can't pay peer-to-peer
- Fees and interest
- Tied to hardware
- Poor user experience

- CC's are "dumb" - can't even simple things like checking balance from card
- Merchants paid $48 billion in swipe fees in 2011
- Losing CC on vacation -> mega bummer

There's an app for that?

Credit card-based:

Square, Paypal Here

- No value added for consumers
- Still uses credit cards
- Still pay swipe fees
- No peer-to-peer

Square: Fees! Still CC based
- Great for small merchants who wouldn't otherwise be able to accept CC's
- but little value-added for consumers

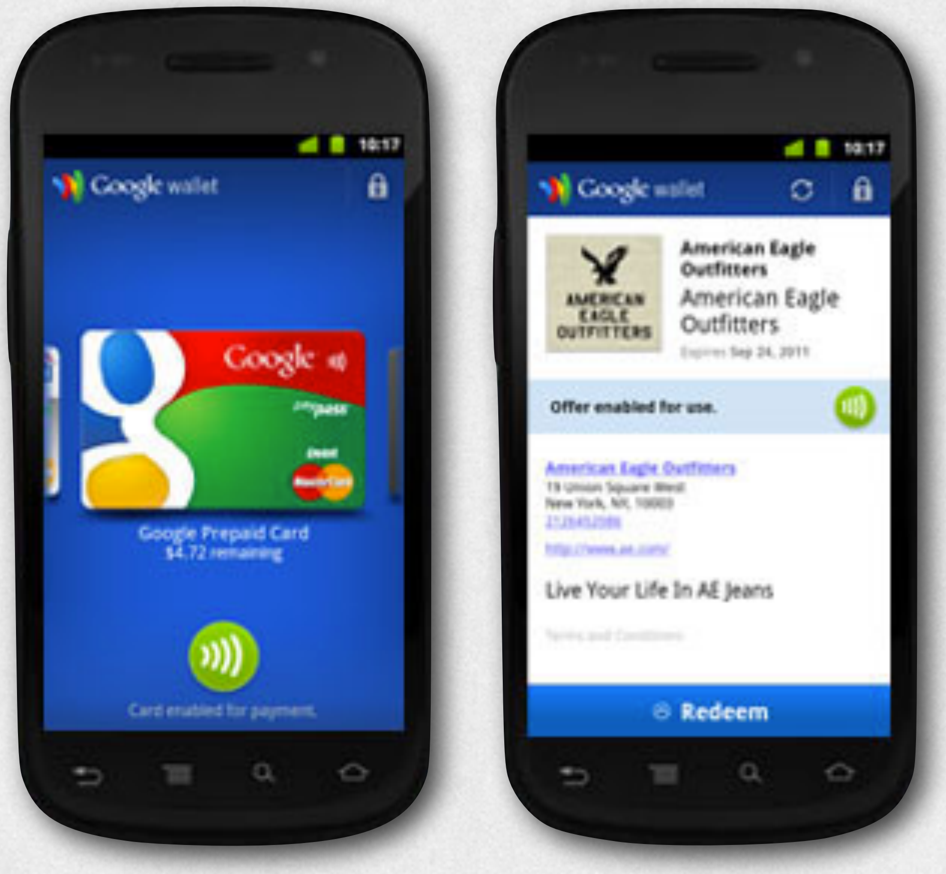
Google Wallet: blocked by Verizon, who is only carrier of only phone that can use GW.

There's an app for that?

Credit card-based:

Google Wallet

- Link from phone to CC account
    - Still hardware-based!
- Still pay swipe fees
- No peer-to-peer
- Android-only: 44% of market



There's an app for that?

Smartphone-based:

"Vanilla" Paypal, Dwolla

- Lower fees if ACH-funded
- No consumer-to-merchant payments

These are closer:
- Less fees if using ACH
- Not hardware-based: can use from any smartphone
- BUT can't do consumer-to-merchant payments

We can do better

We can do better

- No merchant fees

We can do better

- No merchant fees
- Peer-to-peer AND merchant payments

We can do better

- No merchant fees
- Peer-to-peer AND merchant payments
- Fast and easy: scan a QR code, pay in seconds

  (or peer-to-peer pay w/Address Book integration)

We can do better

- No merchant fees
- Peer-to-peer AND merchant payments
- Fast and easy: scan a QR code, pay in seconds

  (or peer-to-peer pay w/Address Book integration)
- View balance and transaction history

We can do better

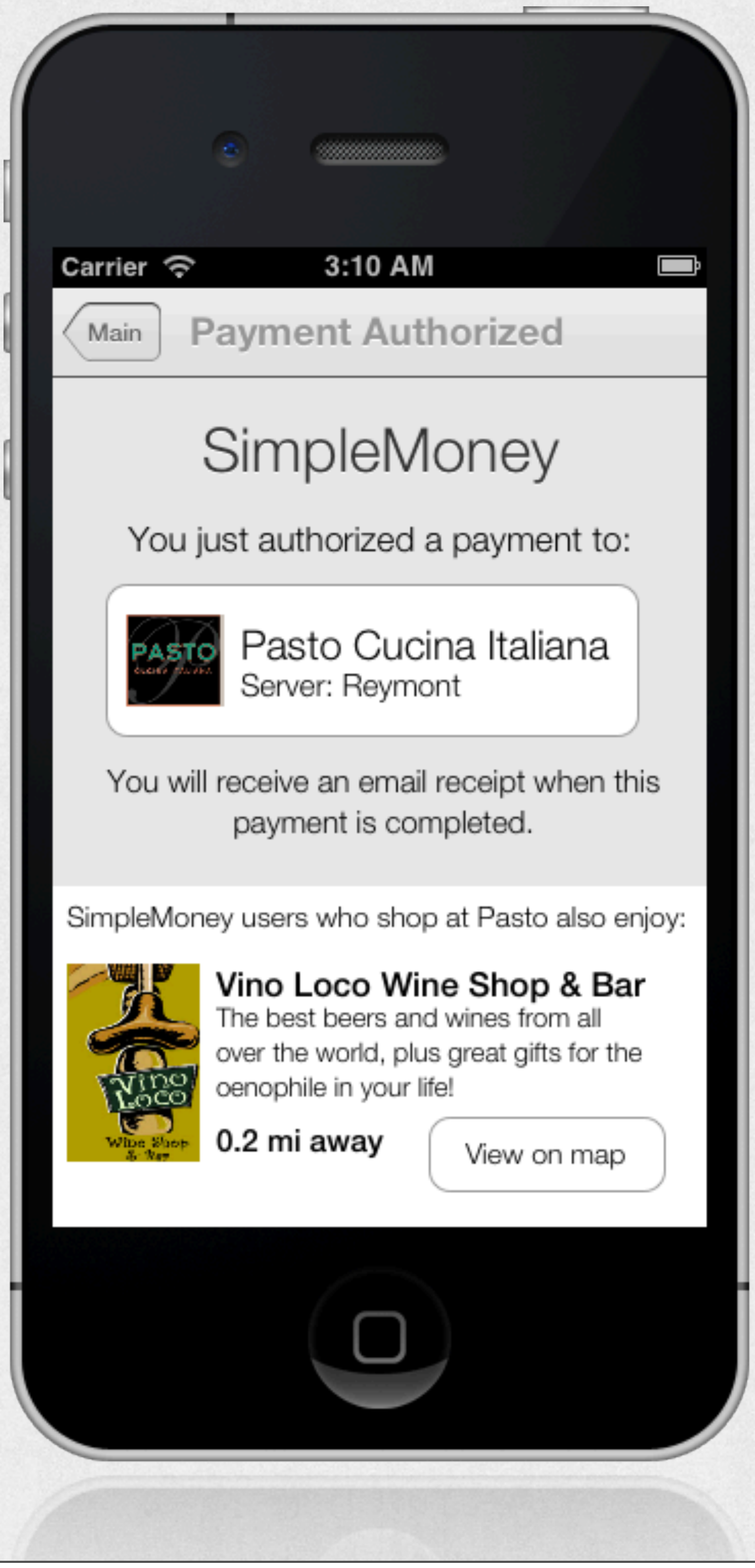
- No merchant fees
- Peer-to-peer AND merchant payments
- Fast and easy: scan a QR code, pay in seconds

  (or peer-to-peer pay w/Address Book integration)
- View balance and transaction history

- Be the “smartest” smart money app

Recommendations

- Great value-add for merchants AND consumers
- Location-aware: encourages users to “shop local”



SimpleMoney

You just authorized a payment to:

Pasto Cucina Italiana
Server: Reymont

You will receive an email receipt when this payment is completed.

SimpleMoney users who shop at Pasto also enjoy:

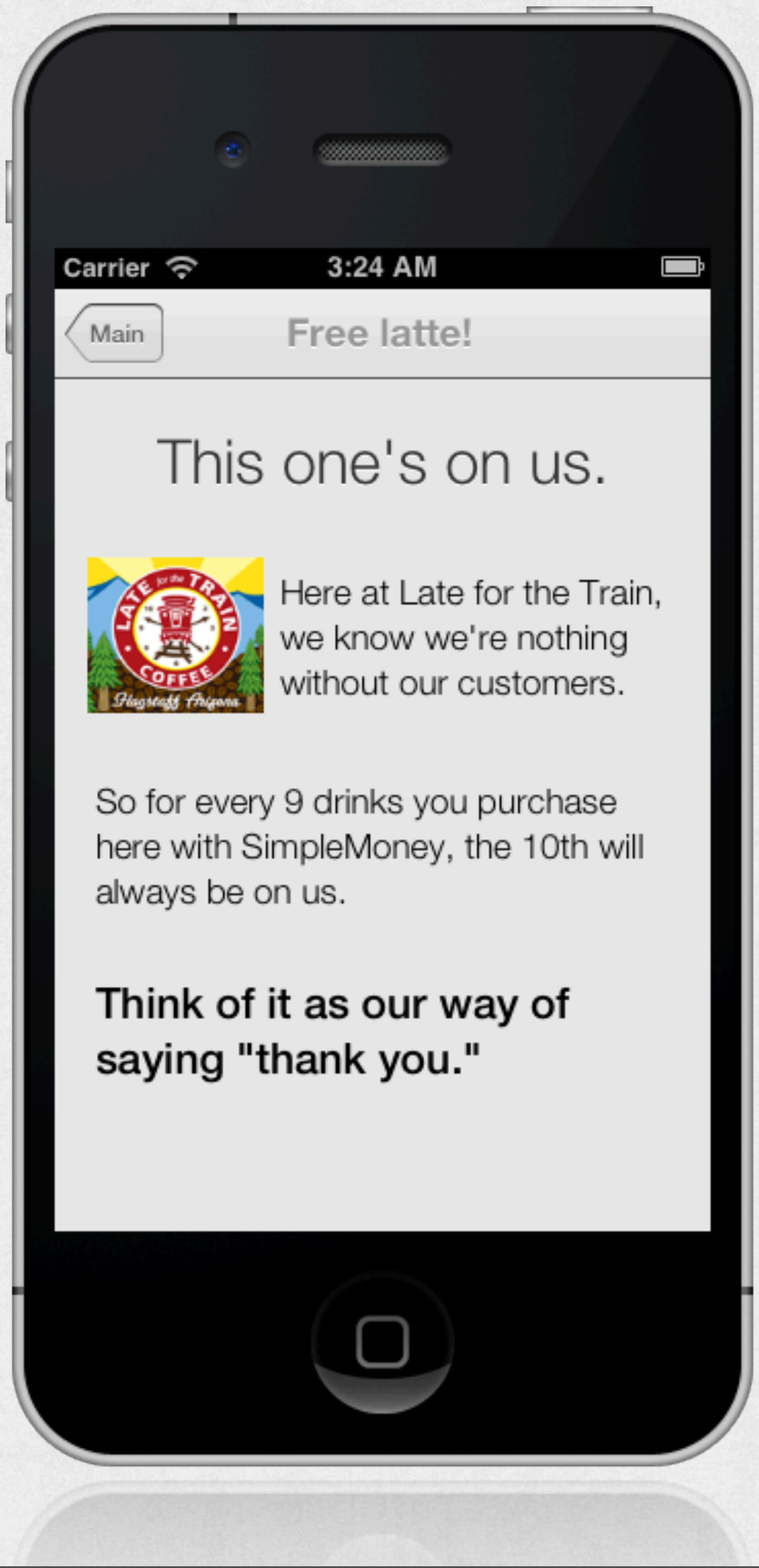
Vino Loco Wine Shop & Bar
The best beers and wines from all over the world, plus great gifts for the oenophile in your life!

0.2 mi away

- big money in online shopping
- location aware: shows distance, and has “view on map” button

Loyalty Programs

- Normally require expensive POS or tracking systems
- Encourages user adoption and customer loyalty



This one's on us.

Here at Late for the Train, we know we're nothing without our customers.

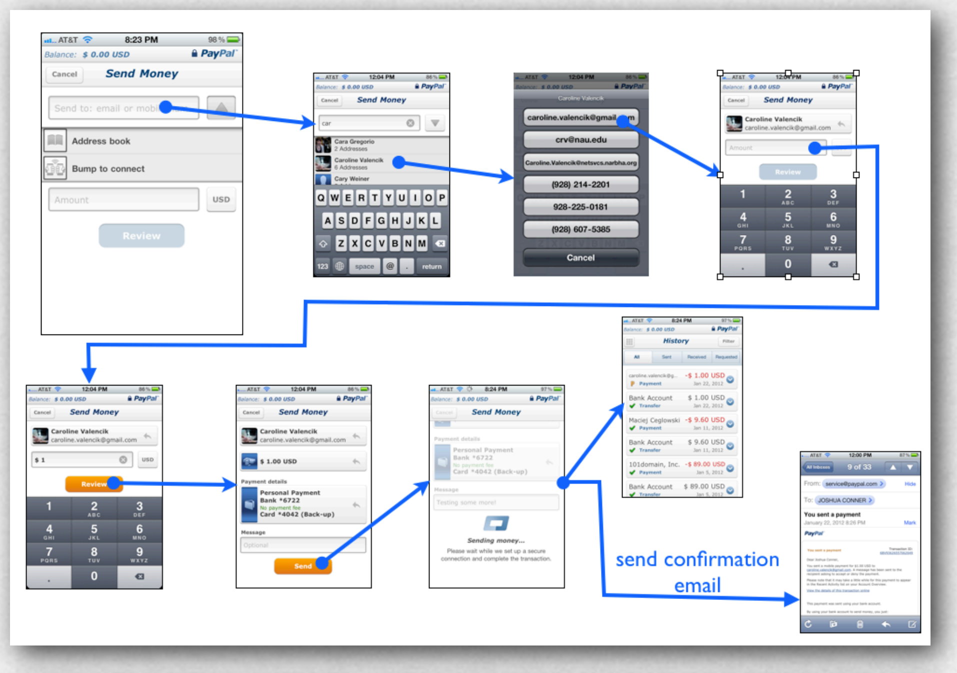
So for every 9 drinks you purchase here with SimpleMoney, the 10th will always be on us.

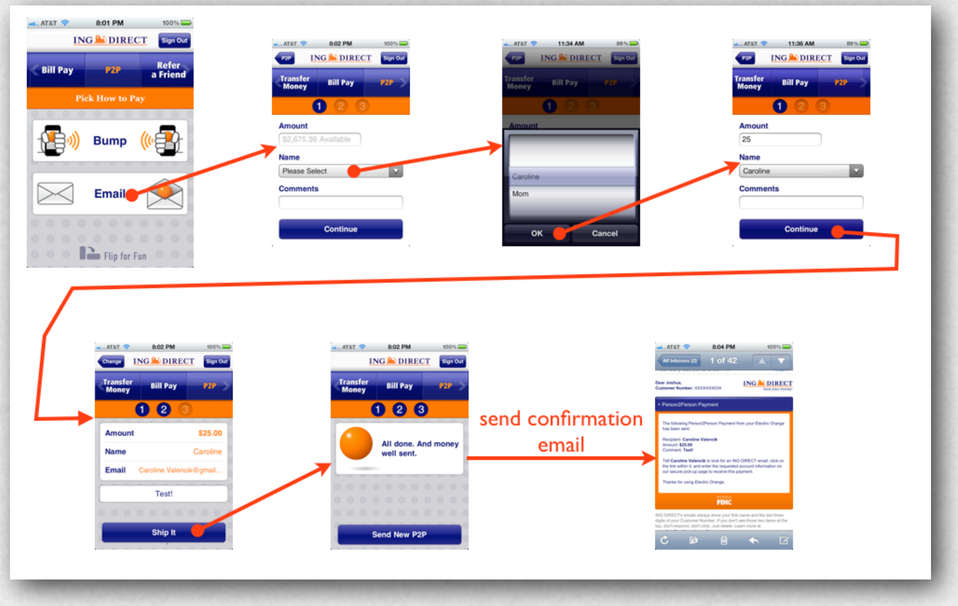
Think of it as our way of saying "thank you."

instead of carrying around punch card, what if it were automatic?

Design Process

1) Analyze competition





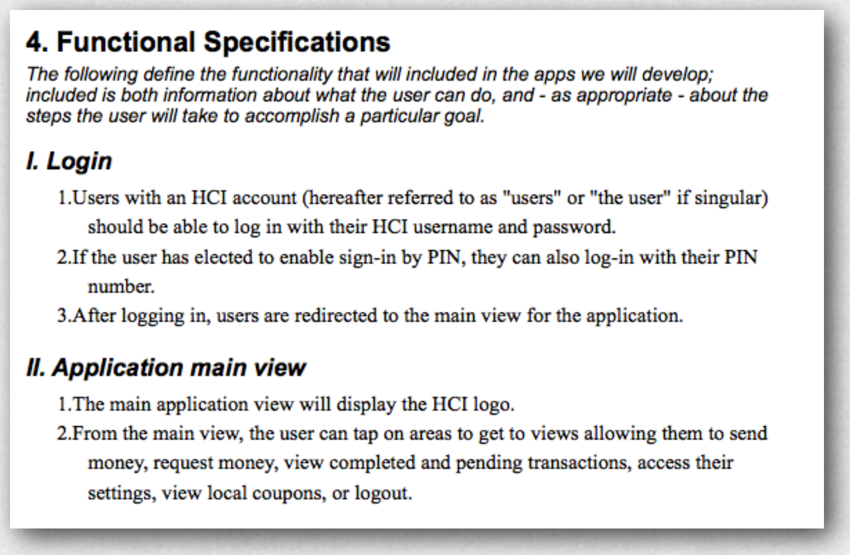
Competitor interaction patterns

1) Gathered and analyzed interaction patterns from competitor apps
2) Clip of formal requirements, some early wireframes we developed
3) Some

Design Process

1) Analyze competition

2) Develop initial spec



4. Functional Specifications

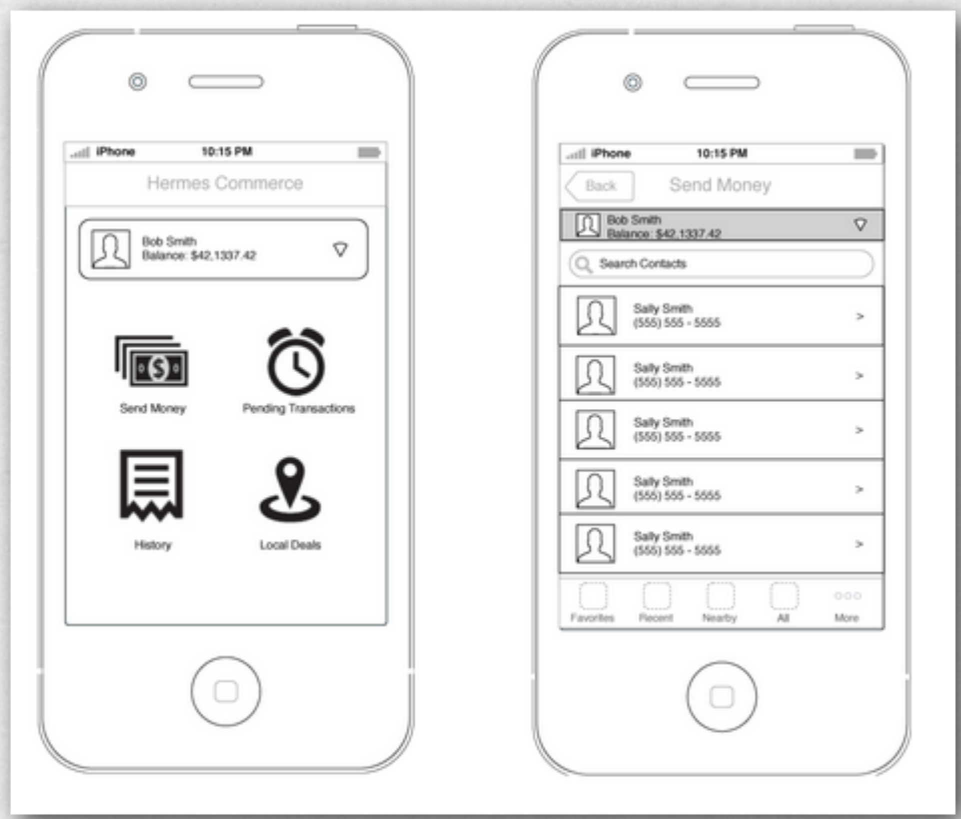
The following define the functionality that will included in the apps we will develop; included is both information about what the user can do, and - as appropriate - about the steps the user will take to accomplish a particular goal.

I. Login

1. Users with an HCI account (hereafter referred to as "users" or "the user" if singular) should be able to log in with their HCI username and password.
2. If the user has elected to enable sign-in by PIN, they can also log-in with their PIN number.
3. After logging in, users are redirected to the main view for the application.

II. Application main view

1. The main application view will display the HCI logo.
2. From the main view, the user can tap on areas to get to views allowing them to send money, request money, view completed and pending transactions, access their settings, view local coupons, or logout.



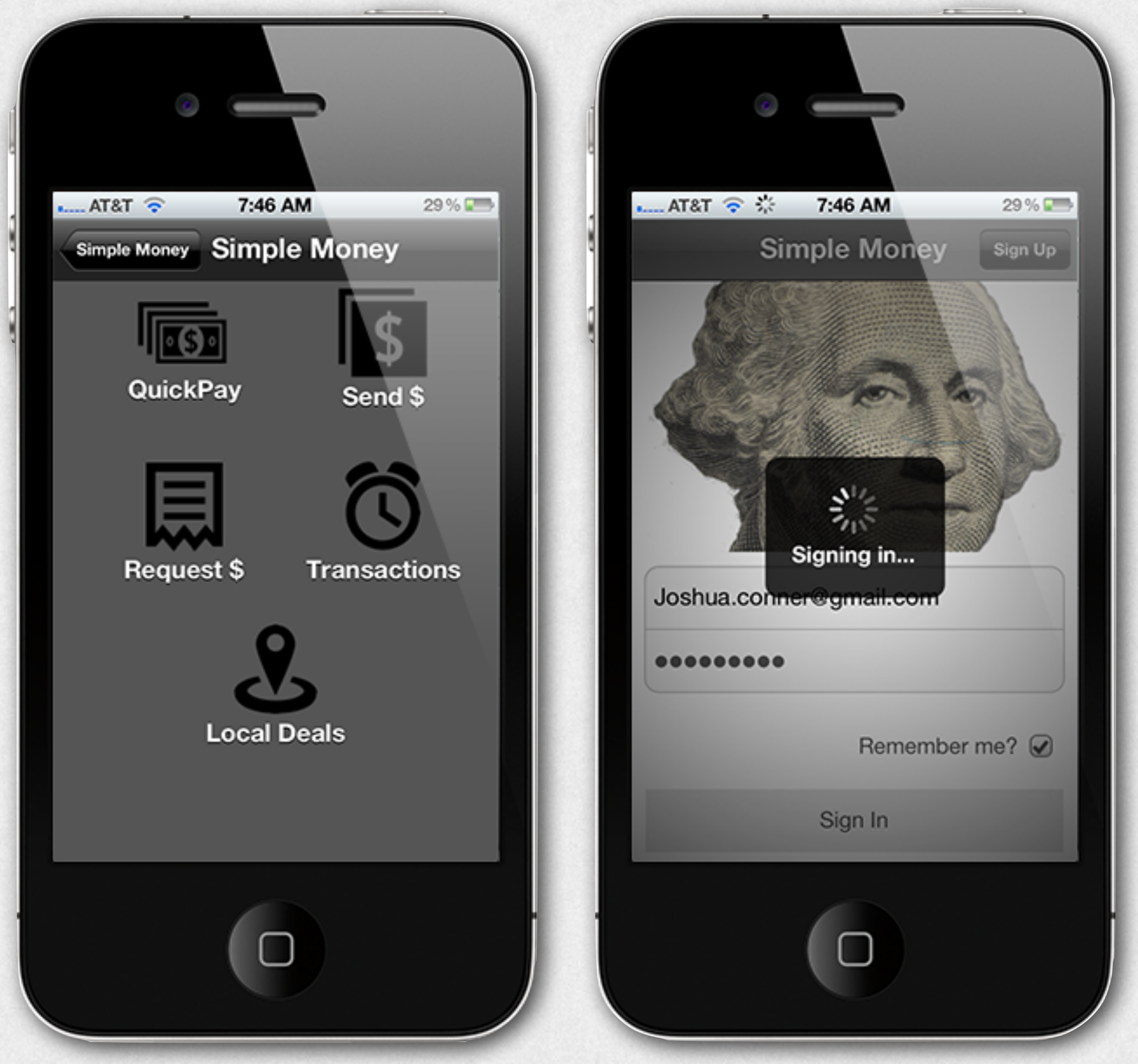
1) Gathered and analyzed interaction patterns from competitor apps
2) Clip of formal requirements, some early wireframes we developed
3) Some

Design Process

1) Analyze competition

2) Develop initial spec

3) Prototype and iterate



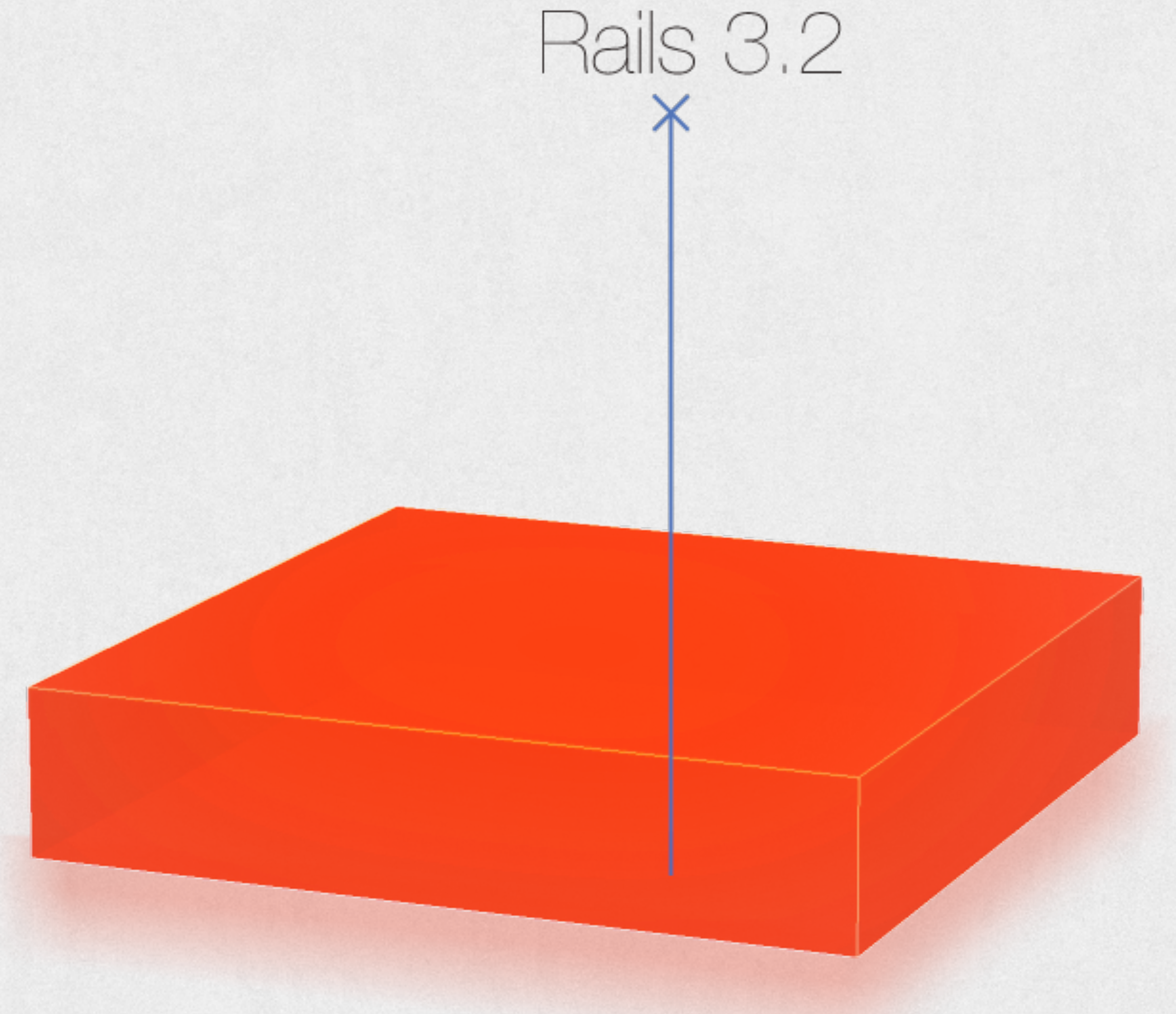
Early SimpleMoney prototypes

1) Gathered and analyzed interaction patterns from competitor apps
2) Clip of formal requirements, some early wireframes we developed
3) Some

Technology Stack

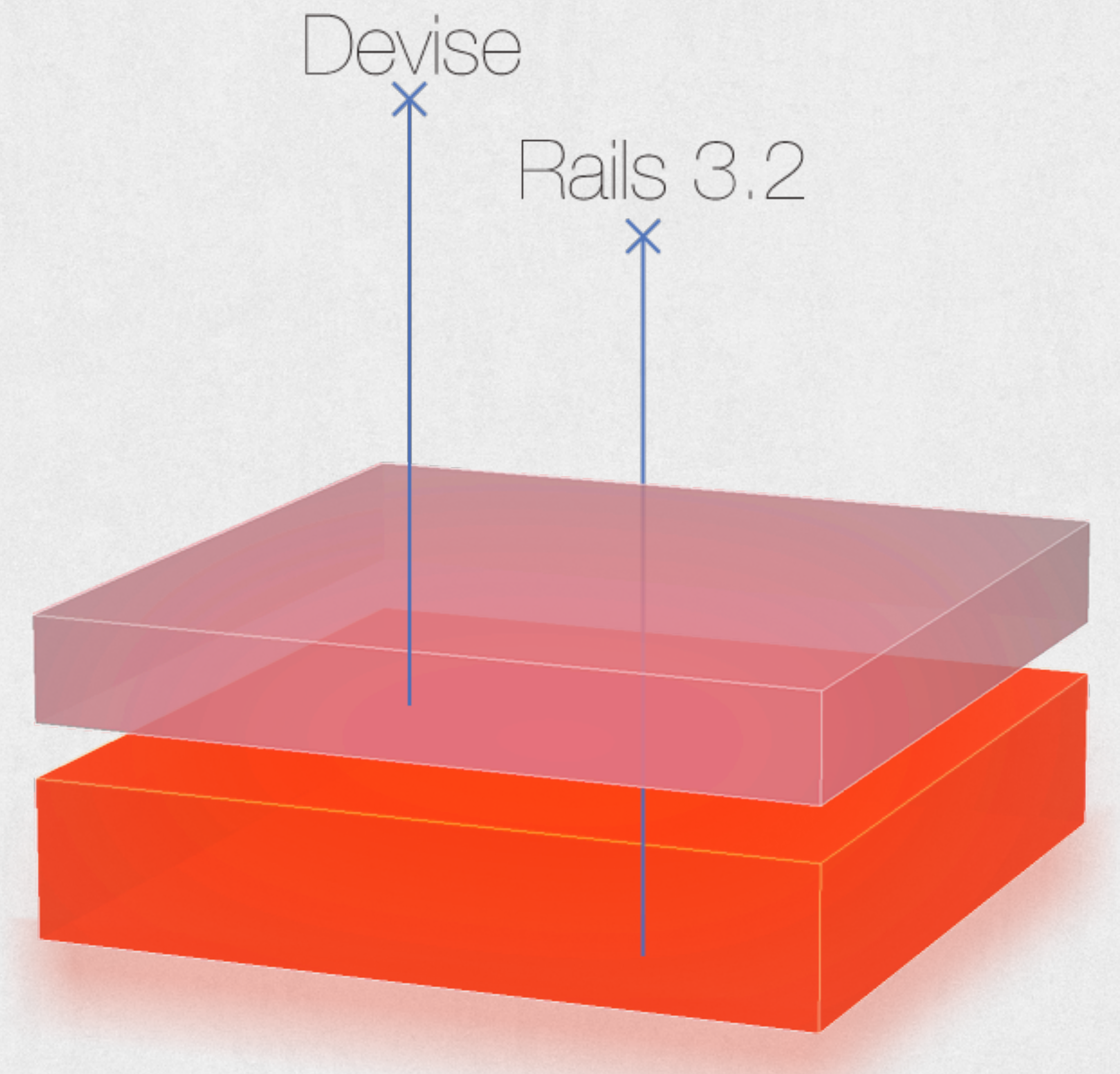
Since we only had one semester to implement the backend architecture, AND the iOS and Android client applications, we had to take advantage of a lot of open source software. Using open source software allowed us to skip the process of reinventing the wheel, and let us to focus on adding features that define our product. At the core of our system is our server, which is built on Ruby on Rails. On top of that, we're using a rock solid authentication system called Devise to handle user authentication and authorization. The core of our client applications are based on the Android and iOS SDKs. They communicate with our server through a REST API, using the help of GSON and RESTKit for object mapping and data serialization. Lastly, we're using Zebra Crossing on Android, and ZBar on the iPhone, to help us read and decode QR codes.

Technology Stack



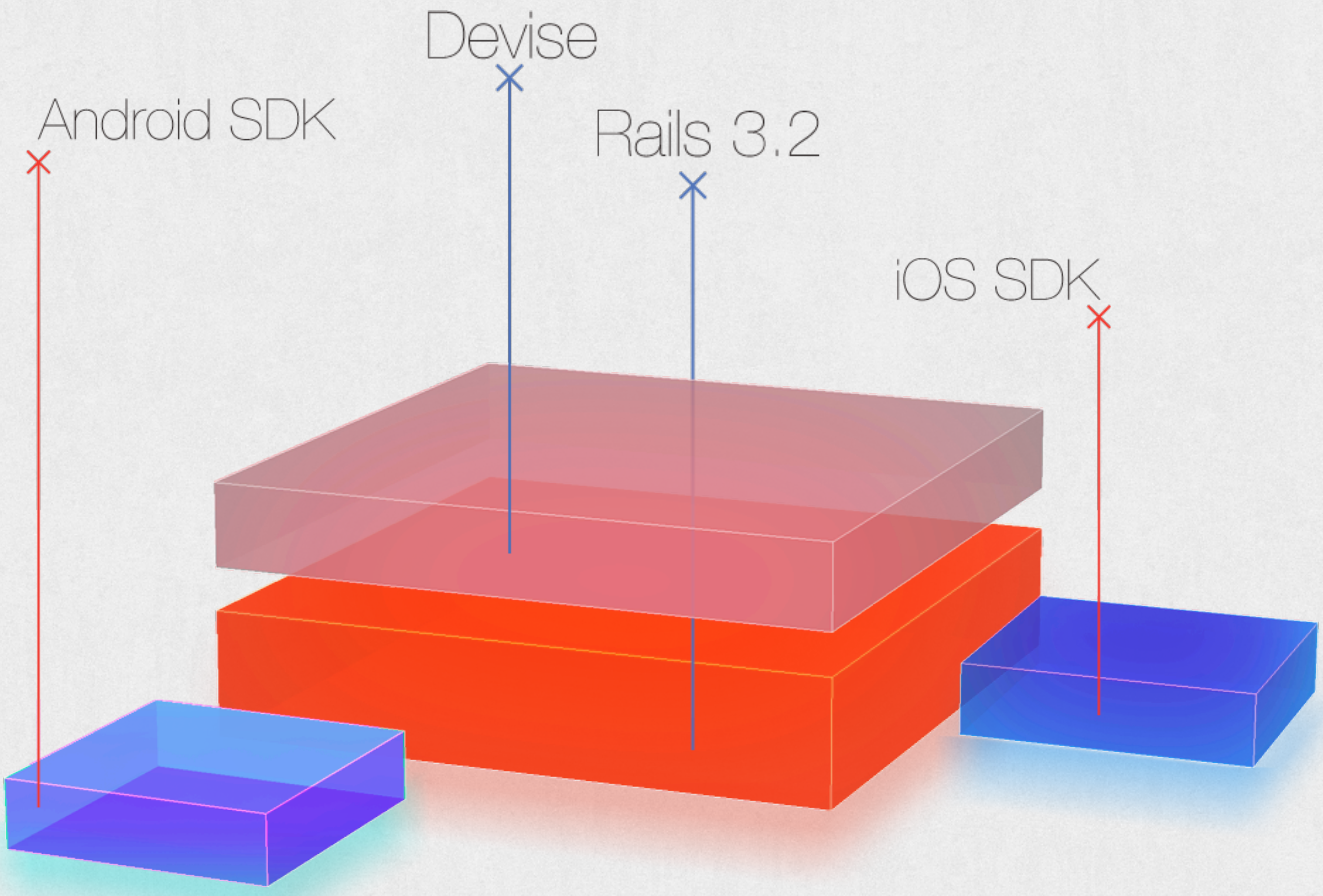
Since we only had one semester to implement the backend architecture, AND the iOS and Android client applications, we had to take advantage of a lot of open source software. Using open source software allowed us to skip the process of reinventing the wheel, and let us to focus on adding features that define our product. At the core of our system is our server, which is built on Ruby on Rails. On top of that, we're using a rock solid authentication system called Devise to handle user authentication and authorization. The core of our client applications are based on the Android and iOS SDKs. They communicate with our server through a REST API, using the help of GSON and RESTKit for object mapping and data serialization. Lastly, we're using Zebra Crossing on Android, and ZBar on the iPhone, to help us read and decode QR codes.

Technology Stack



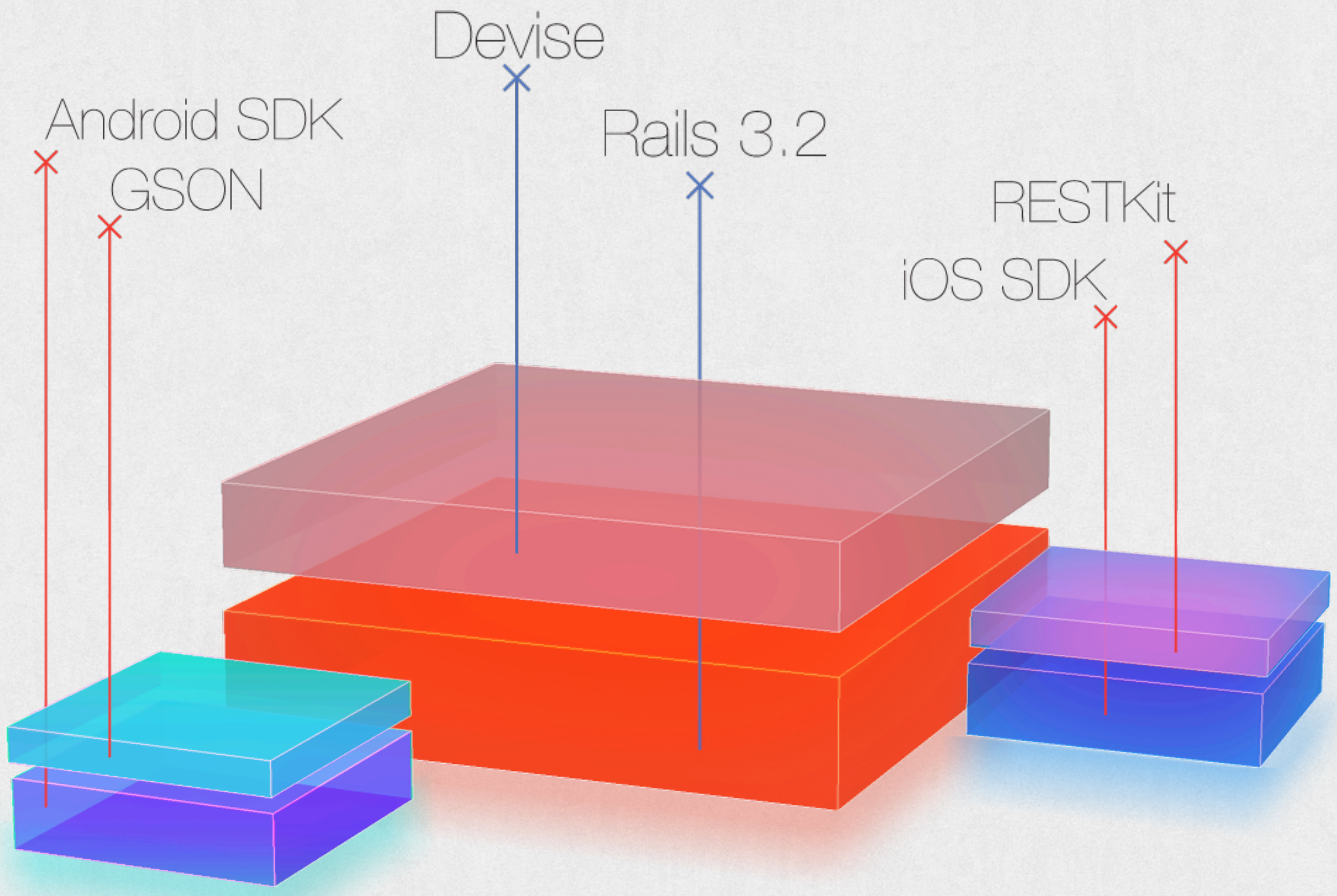
Since we only had one semester to implement the backend architecture, AND the iOS and Android client applications, we had to take advantage of a lot of open source software. Using open source software allowed us to skip the process of reinventing the wheel, and let us to focus on adding features that define our product. At the core of our system is our server, which is built on Ruby on Rails. On top of that, we're using a rock solid authentication system called Devise to handle user authentication and authorization. The core of our client applications are based on the Android and iOS SDKs. They communicate with our server through a REST API, using the help of GSON and RESTKit for object mapping and data serialization. Lastly, we're using Zebra Crossing on Android, and ZBar on the iPhone, to help us read and decode QR codes.

Technology Stack



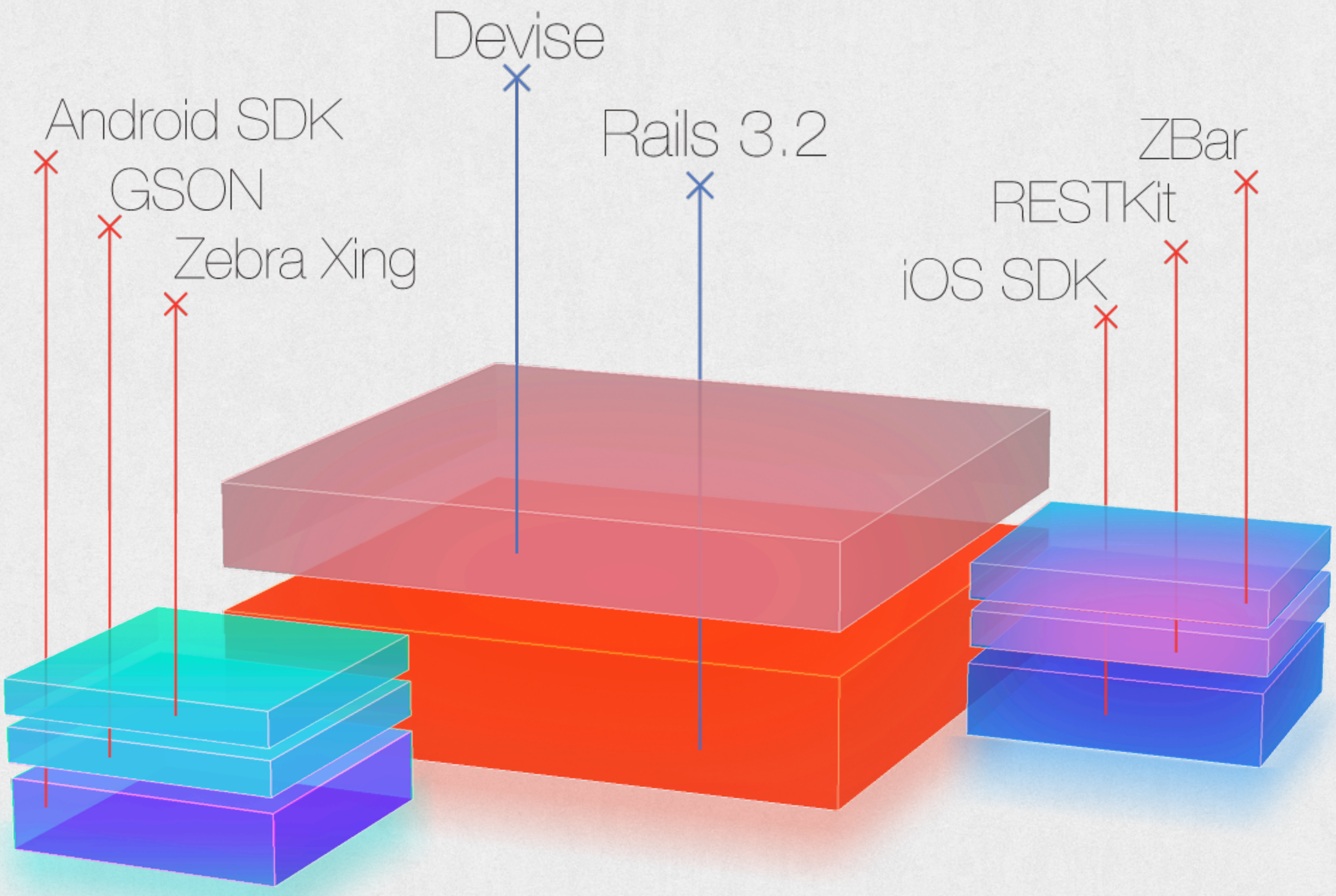
Since we only had one semester to implement the backend architecture, AND the iOS and Android client applications, we had to take advantage of a lot of open source software. Using open source software allowed us to skip the process of reinventing the wheel, and let us to focus on adding features that define our product. At the core of our system is our server, which is built on Ruby on Rails. On top of that, we're using a rock solid authentication system called Devise to handle user authentication and authorization. The core of our client applications are based on the Android and iOS SDKs. They communicate with our server through a REST API, using the help of GSON and RESTKit for object mapping and data serialization. Lastly, we're using Zebra Crossing on Android, and ZBar on the iPhone, to help us read and decode QR codes.

Technology Stack



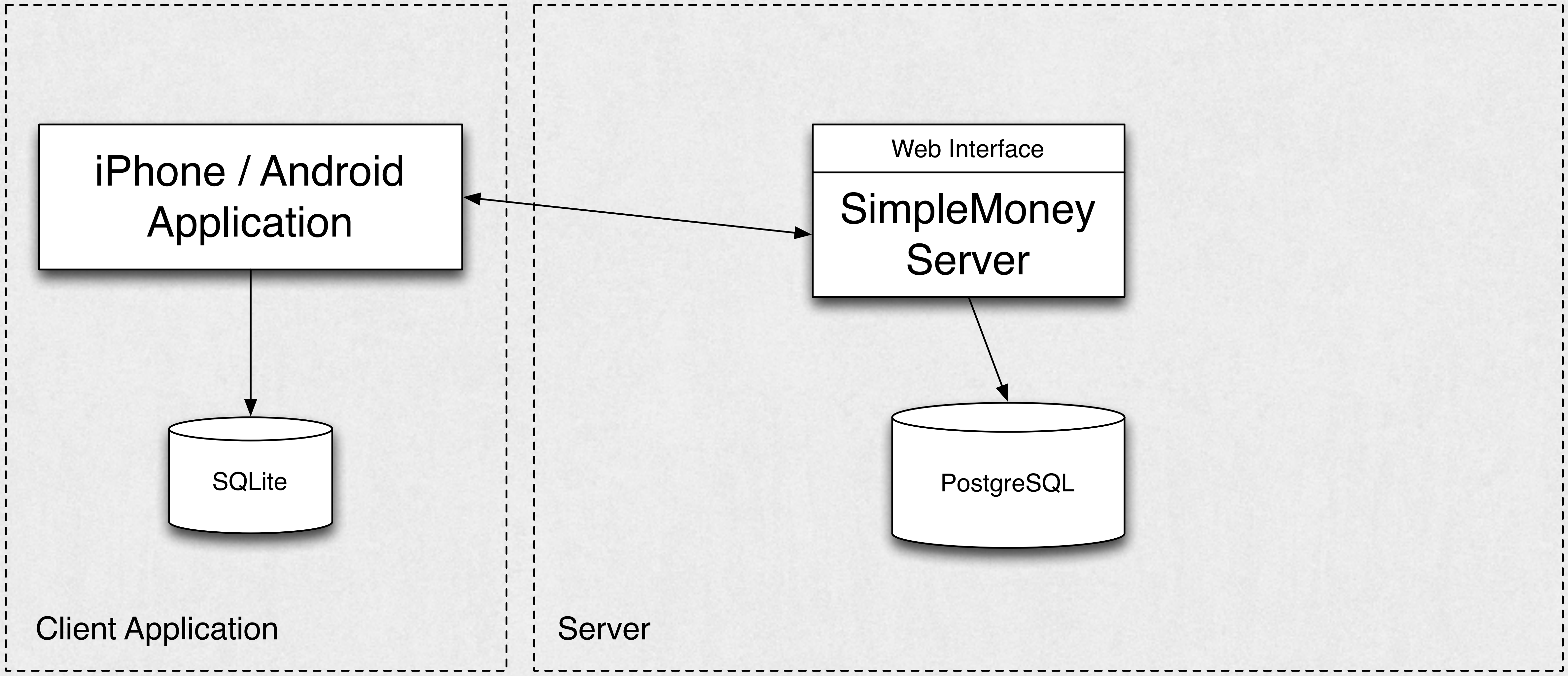
Since we only had one semester to implement the backend architecture, AND the iOS and Android client applications, we had to take advantage of a lot of open source software. Using open source software allowed us to skip the process of reinventing the wheel, and let us to focus on adding features that define our product. At the core of our system is our server, which is built on Ruby on Rails. On top of that, we're using a rock solid authentication system called Devise to handle user authentication and authorization. The core of our client applications are based on the Android and iOS SDKs. They communicate with our server through a REST API, using the help of GSON and RESTKit for object mapping and data serialization. Lastly, we're using Zebra Crossing on Android, and ZBar on the iPhone, to help us read and decode QR codes.

Technology Stack



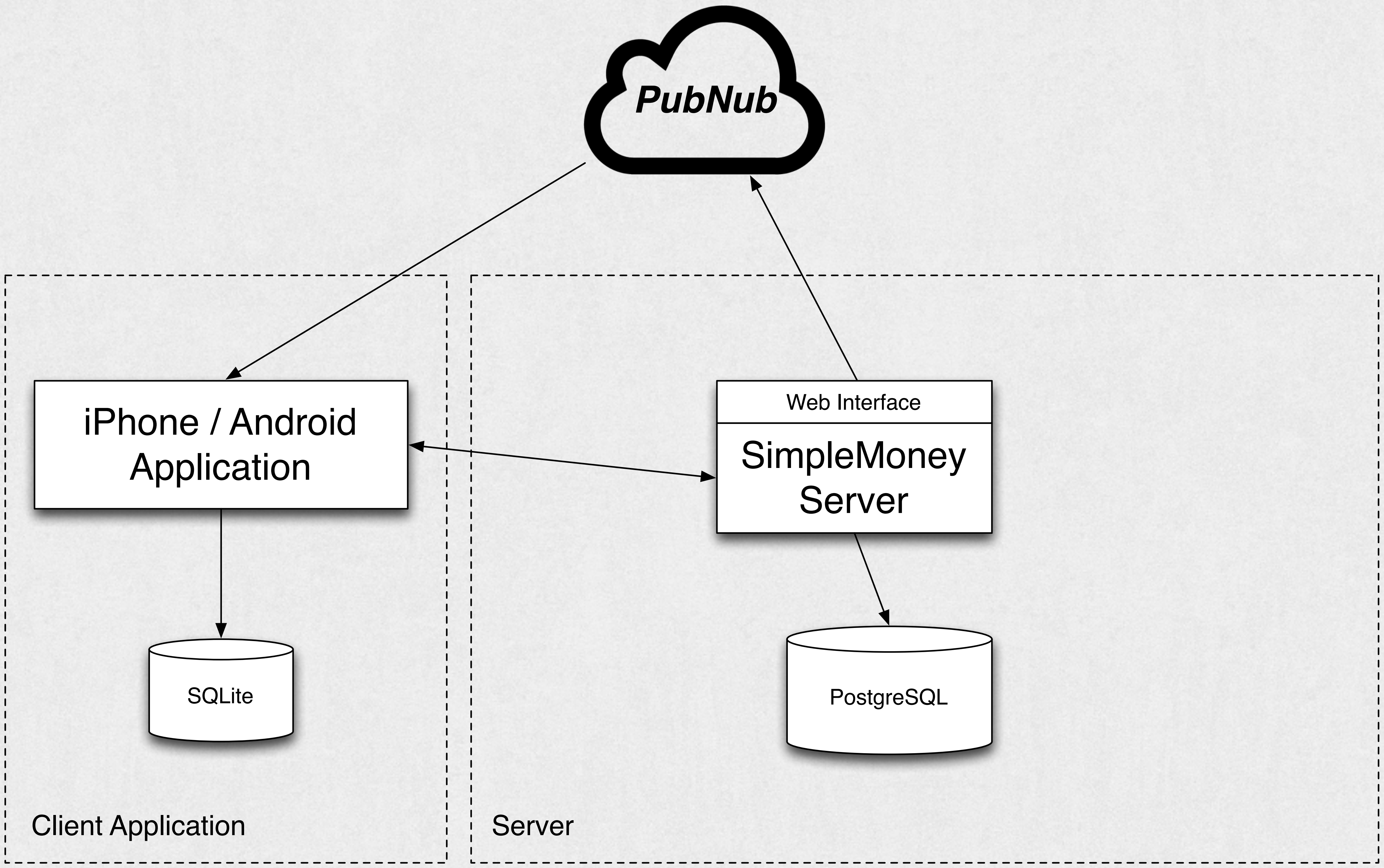
Since we only had one semester to implement the backend architecture, AND the iOS and Android client applications, we had to take advantage of a lot of open source software. Using open source software allowed us to skip the process of reinventing the wheel, and let us to focus on adding features that define our product. At the core of our system is our server, which is built on Ruby on Rails. On top of that, we're using a rock solid authentication system called Devise to handle user authentication and authorization. The core of our client applications are based on the Android and iOS SDKs. They communicate with our server through a REST API, using the help of GSON and RESTKit for object mapping and data serialization. Lastly, we're using Zebra Crossing on Android, and ZBar on the iPhone, to help us read and decode QR codes.

Architecture



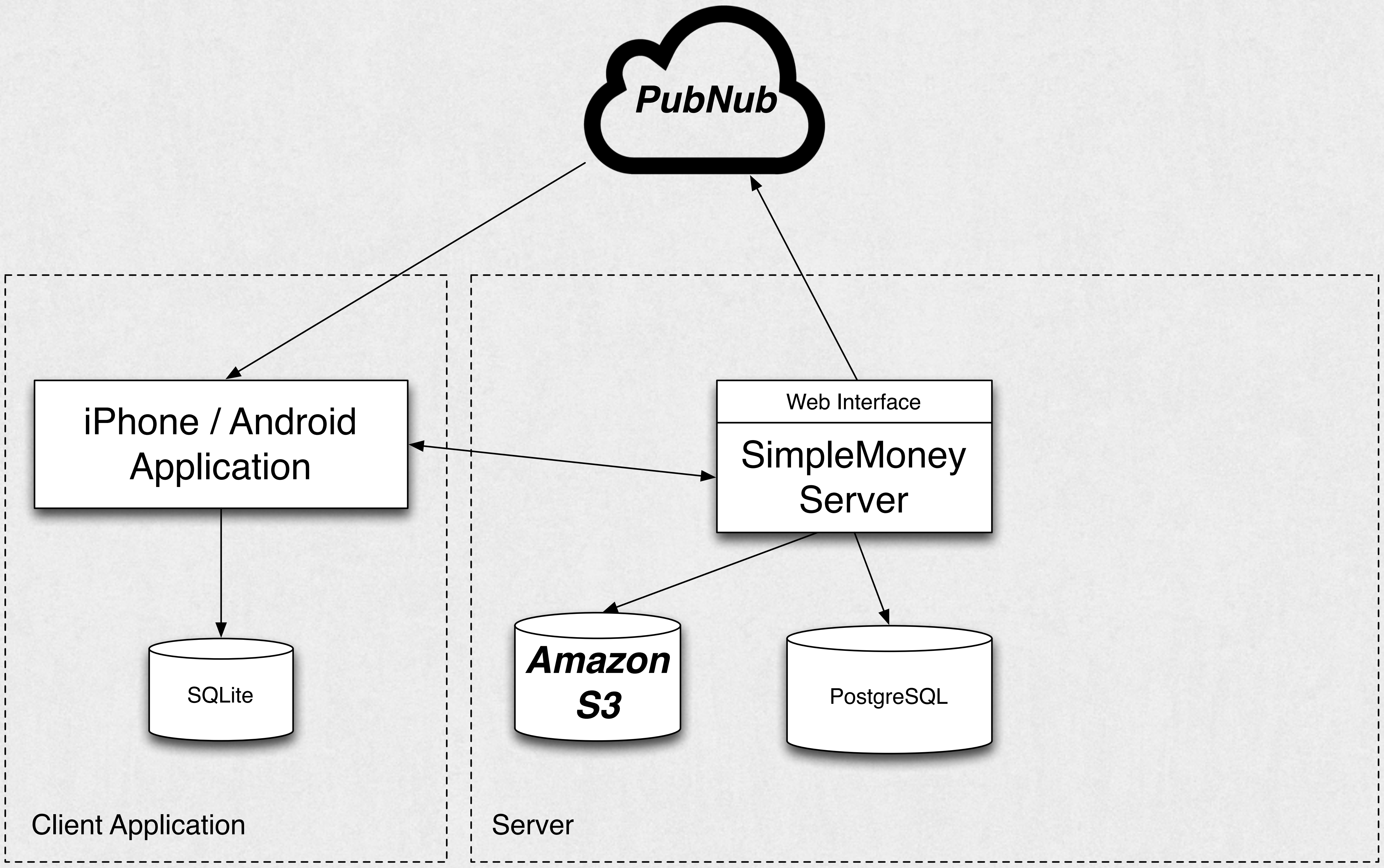
Today, our application is deployed on the web, and interfaces with several cloud services to bring users some powerful features. We're using a push notification service called PubNub to notify our users of transaction events in real time. We're also using Amazon S3 to store our image assets, such as user avatars, and lastly we're using Mailgun to dispatch confirmation emails, password reset tokens, and transaction receipts. I won't walk you through the entire application here, instead I'll show you some of the views we've implemented to illustrate how our application works in practice.

Architecture



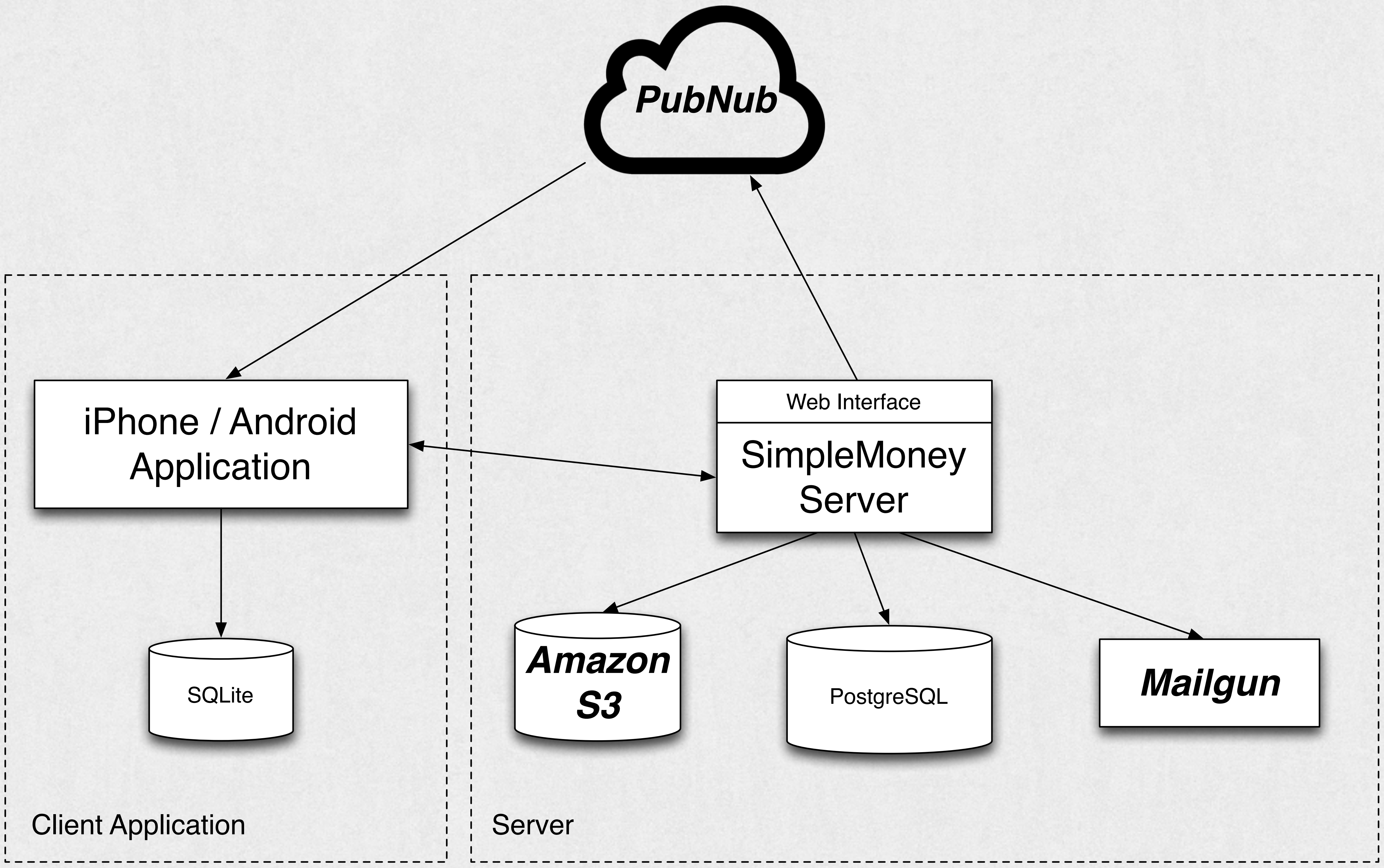
Today, our application is deployed on the web, and interfaces with several cloud services to bring users some powerful features. We're using a push notification service called PubNub to notify our users of transaction events in real time. We're also using Amazon S3 to store our image assets, such as user avatars, and lastly we're using Mailgun to dispatch confirmation emails, password reset tokens, and transaction receipts. I won't walk you through the entire application here, instead I'll show you some of the views we've implemented to illustrate how our application works in practice.

Architecture



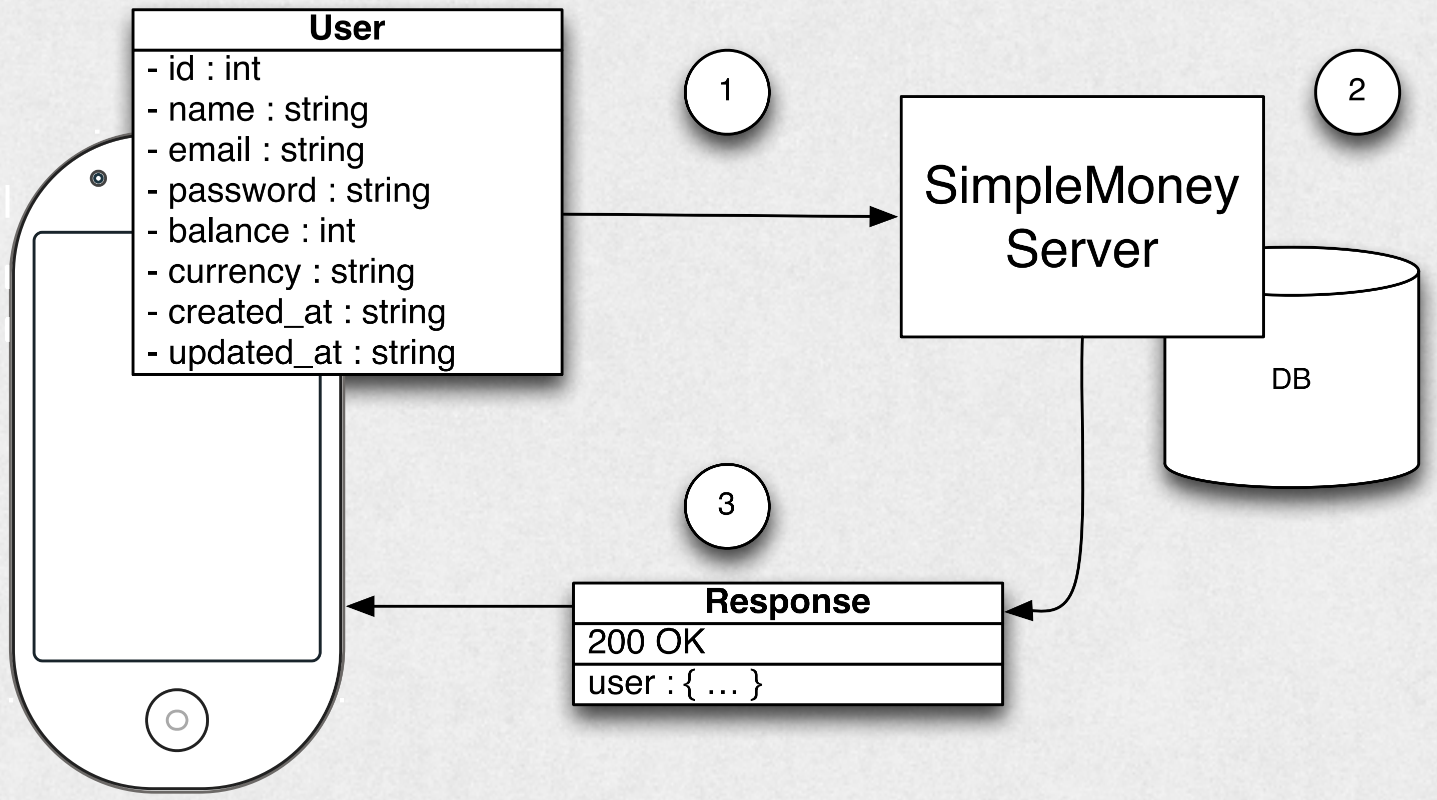
Today, our application is deployed on the web, and interfaces with several cloud services to bring users some powerful features. We're using a push notification service called PubNub to notify our users of transaction events in real time. We're also using Amazon S3 to store our image assets, such as user avatars, and lastly we're using Mailgun to dispatch confirmation emails, password reset tokens, and transaction receipts. I won't walk you through the entire application here, instead I'll show you some of the views we've implemented to illustrate how our application works in practice.

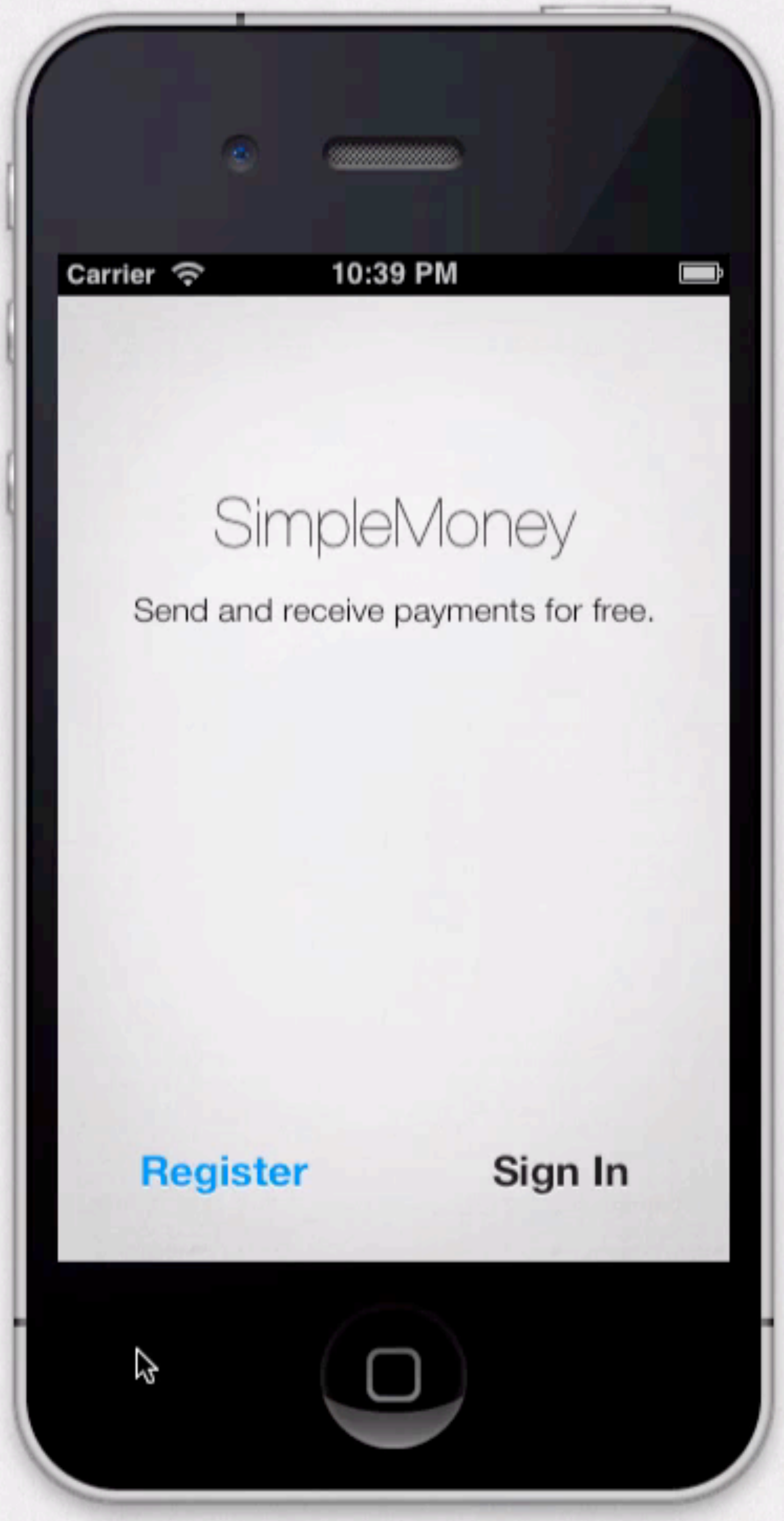
Architecture



Today, our application is deployed on the web, and interfaces with several cloud services to bring users some powerful features. We're using a push notification service called PubNub to notify our users of transaction events in real time. We're also using Amazon S3 to store our image assets, such as user avatars, and lastly we're using Mailgun to dispatch confirmation emails, password reset tokens, and transaction receipts. I won't walk you through the entire application here, instead I'll show you some of the views we've implemented to illustrate how our application works in practice.

Sign Up



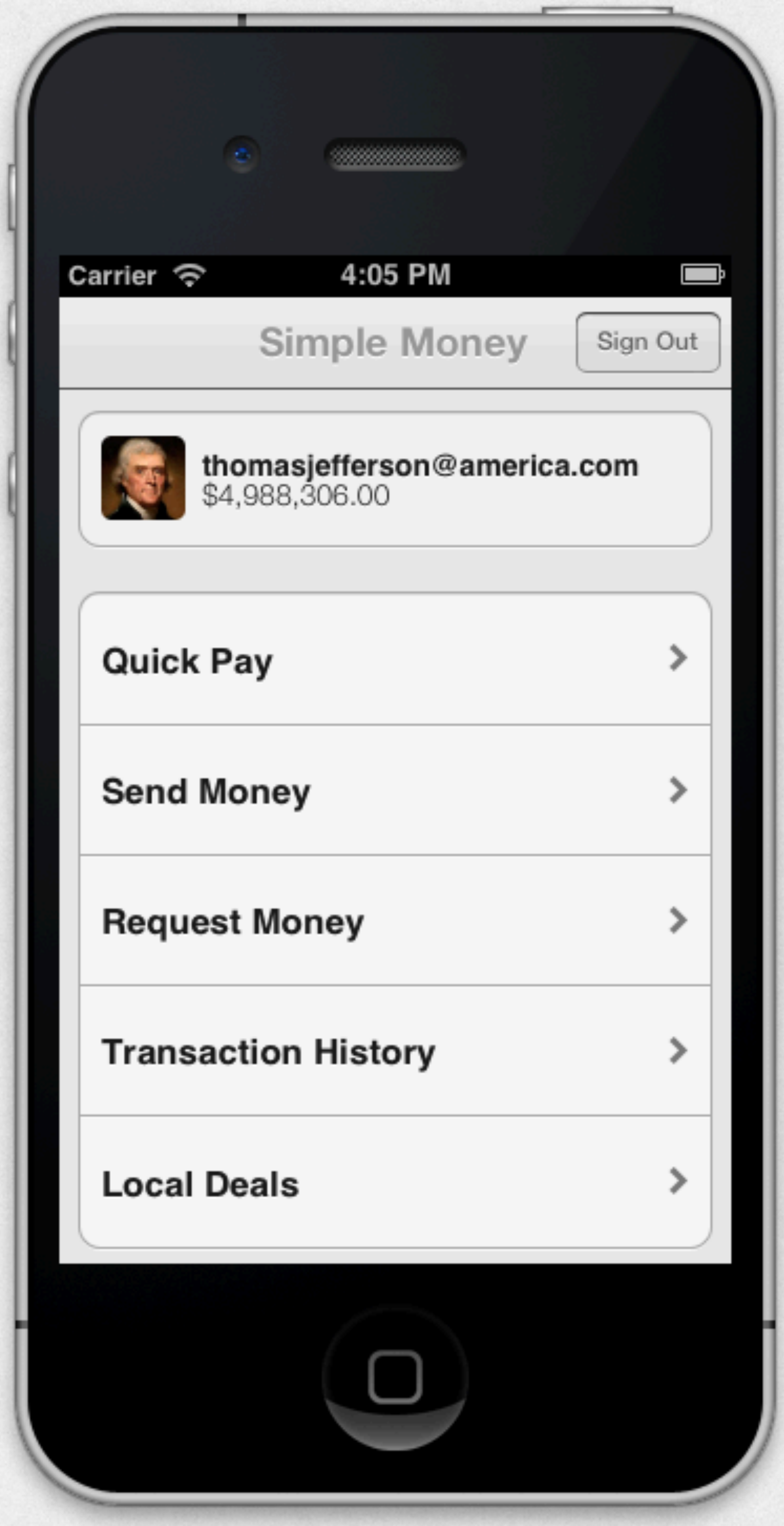


The first thing a user will want to do is sign up. First we populate the required parameters such as the user's name, email address and password, along with an optional user avatar. Users can take a photo with their camera, or choose an existing one from their library. When we're done filling out the form, the application sends a POST request, and the server validates the uniqueness of the email address, along with the length of the name and password. If the user is saved to the database, our server sends back the newly created object so the application can save the user and their transaction data to a local database on the phone.

The first thing a user will want to do is sign up. First we populate the required parameters such as the user's name, email address and password, along with an optional user avatar.

Home Screen

- View account balance
- Pay by scanning a QR code
- Send and request money
- View transactions
- View local deals

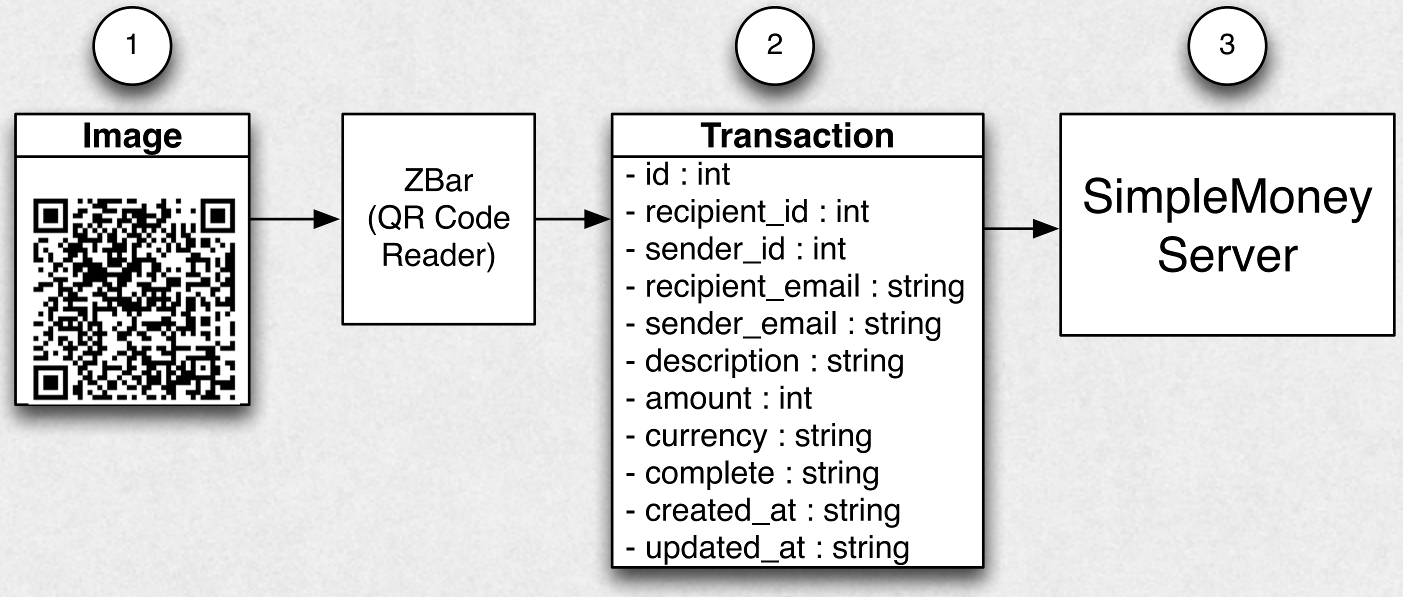


thomasjefferson@america.com
$4,988,306.00

Quick Pay
Send Money
Request Money
Transaction History
Local Deals

After a user signs up or signs in, they're taken to the home screen where they can view their account balance, make a quick payment, send or request money, view their transaction history, or view local deals around them.

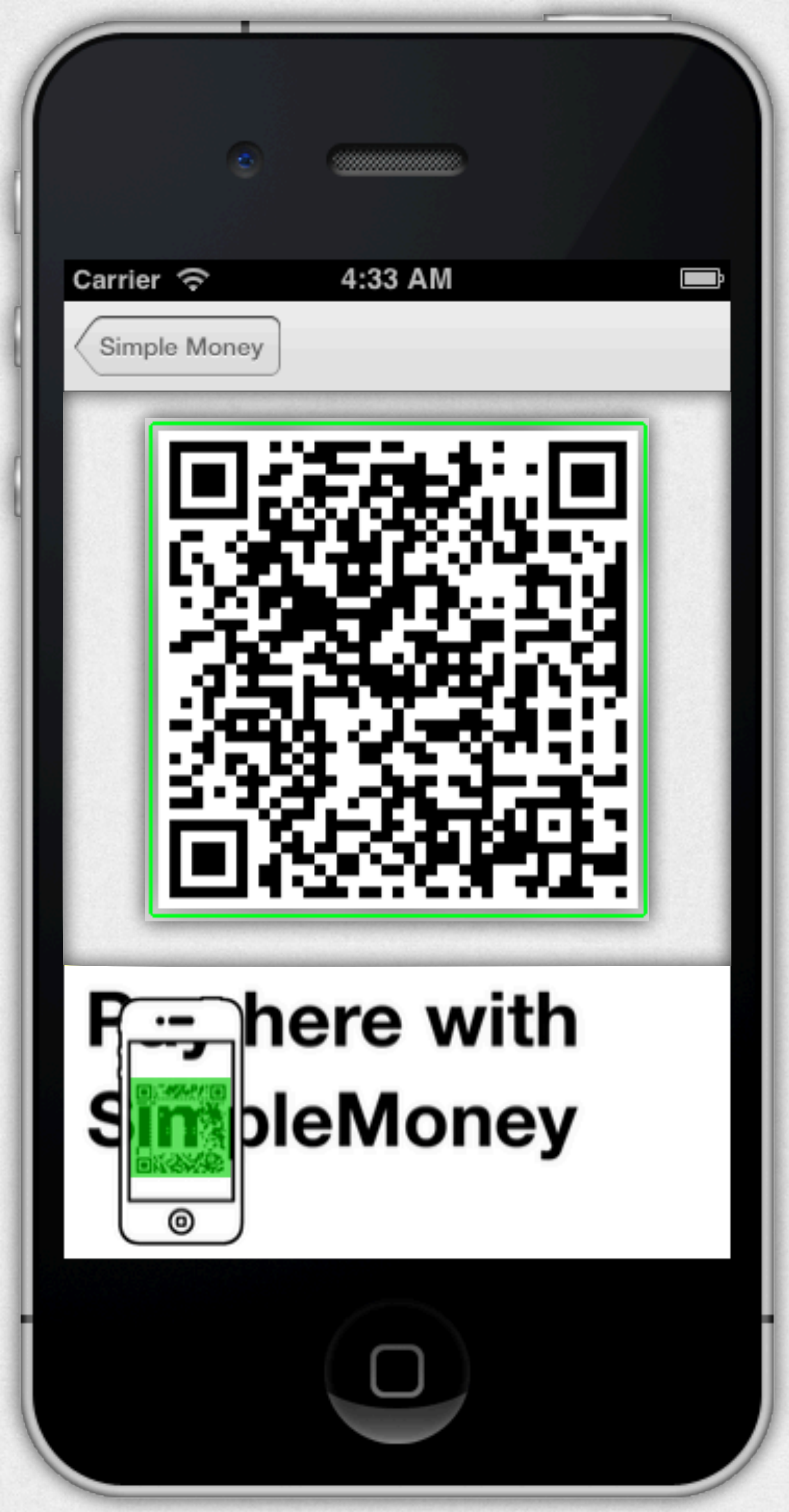
Quick Pay



1. QR Code is scanned

2. App uses info from QR to create a transaction

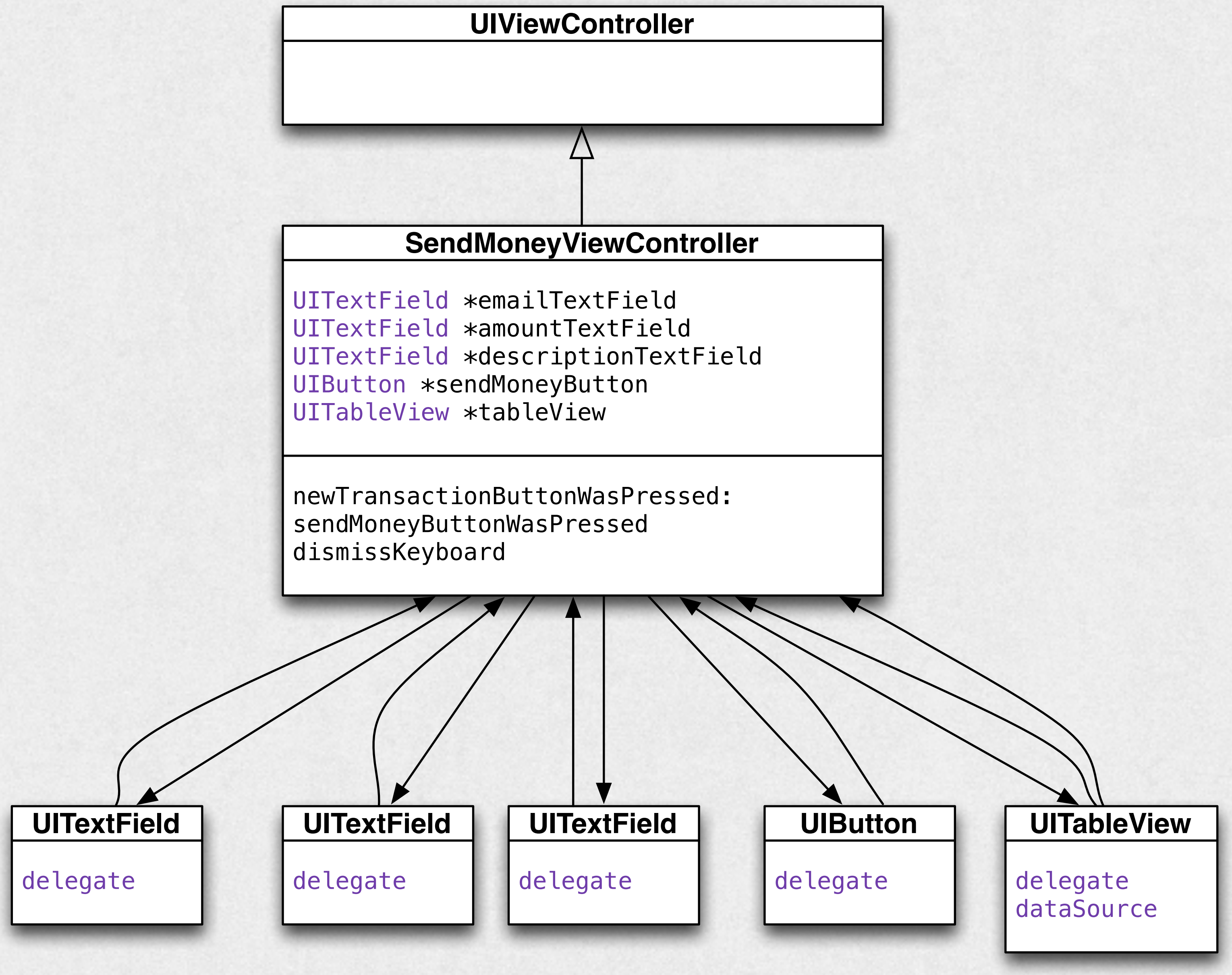
3. Sends a POST request to simplemoney.dev/transactions/

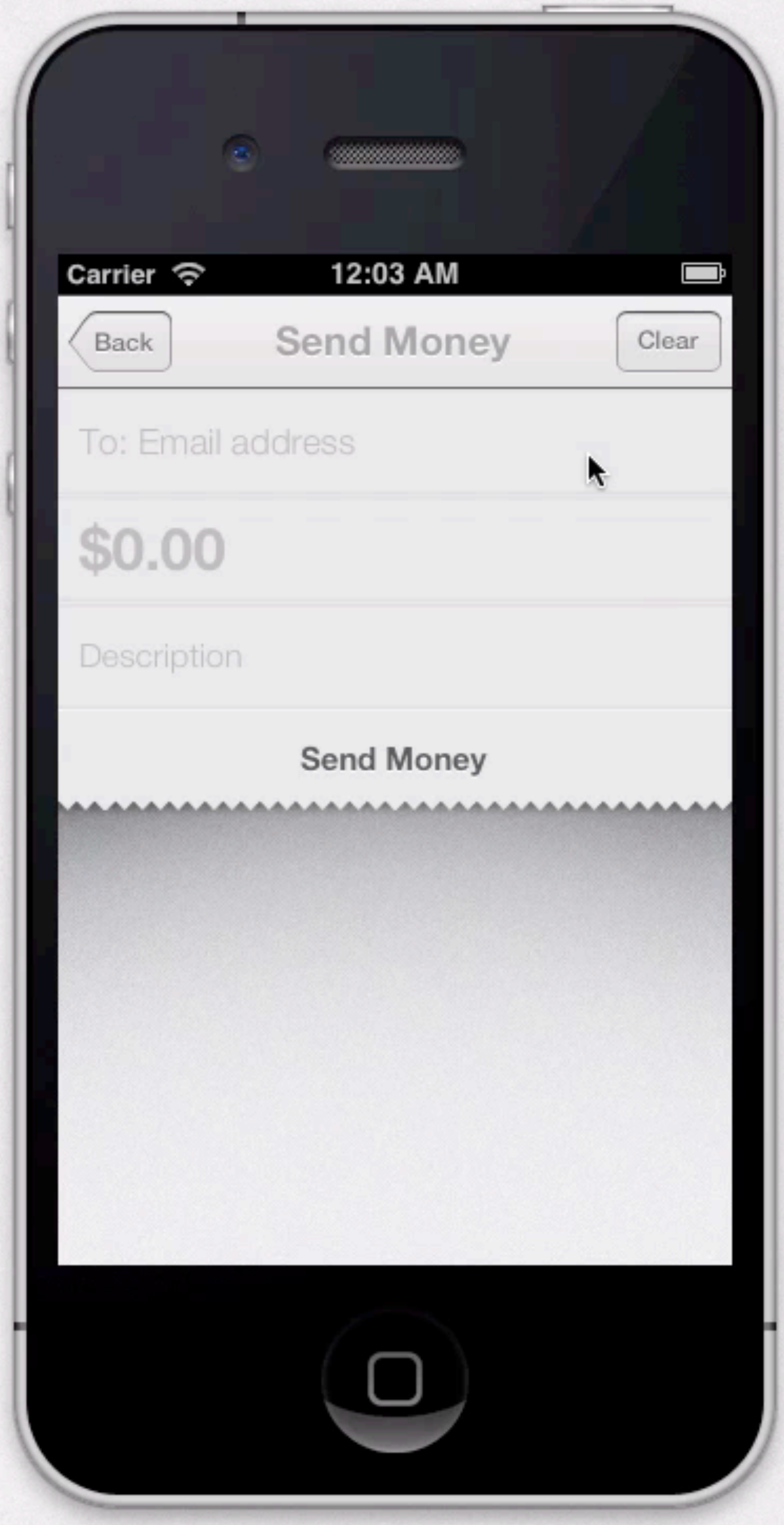


We wanted to make payments as fast and easy as possible. To do that, we're using ZBar's camera controller to recognize a SimpleMoney QR Code. Once a QR Code is recognized, the camera automatically dismisses itself, grabs the merchant ID, builds a new transaction locally on the device, and POSTS it to the server. The transaction model has a boolean complete flag that determines if the transaction is paid for or not. Similar to the process of authorizing a charge on a credit card, a user can scan a QR Code to authorize a merchant to charge their account.

If QuickPay is selected, we use ZBar’s camera controller to scan a QR Code that contains a merchant id. Once a QR Code is recongizned, the camera controller is dismissed and our app

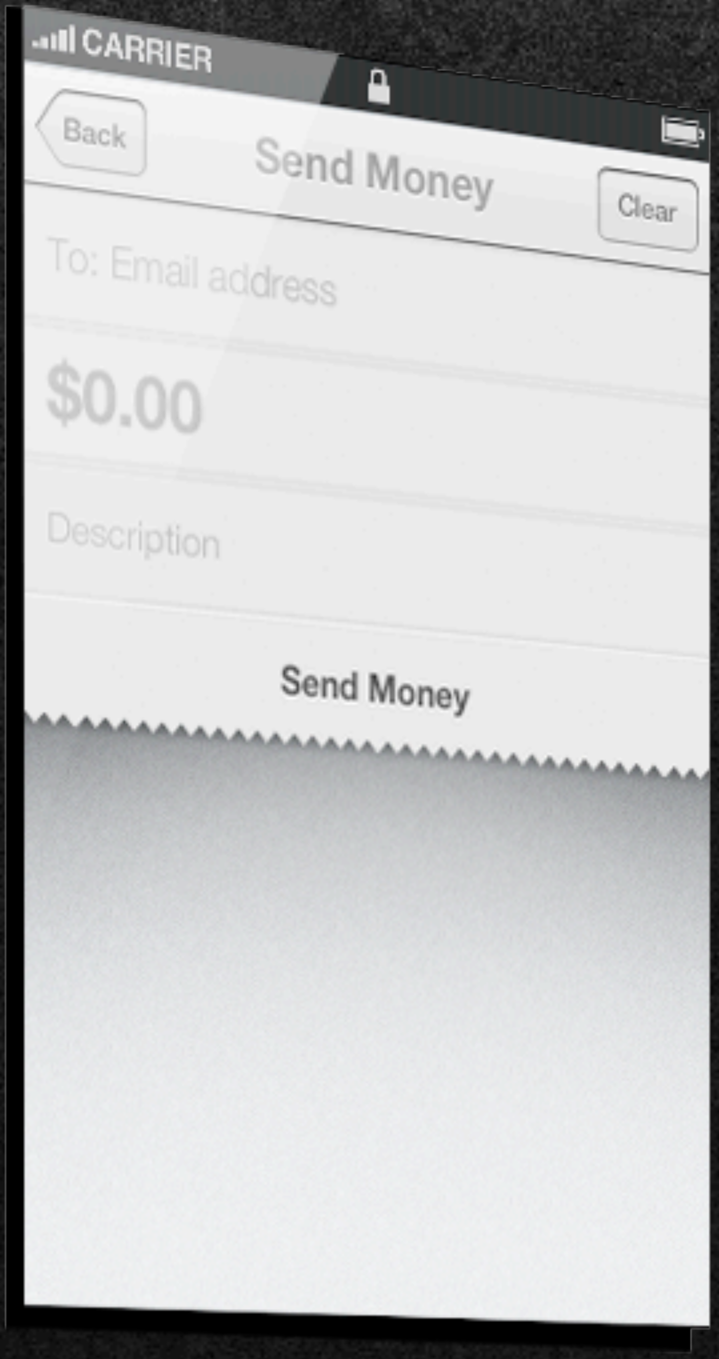
Send & Request Money



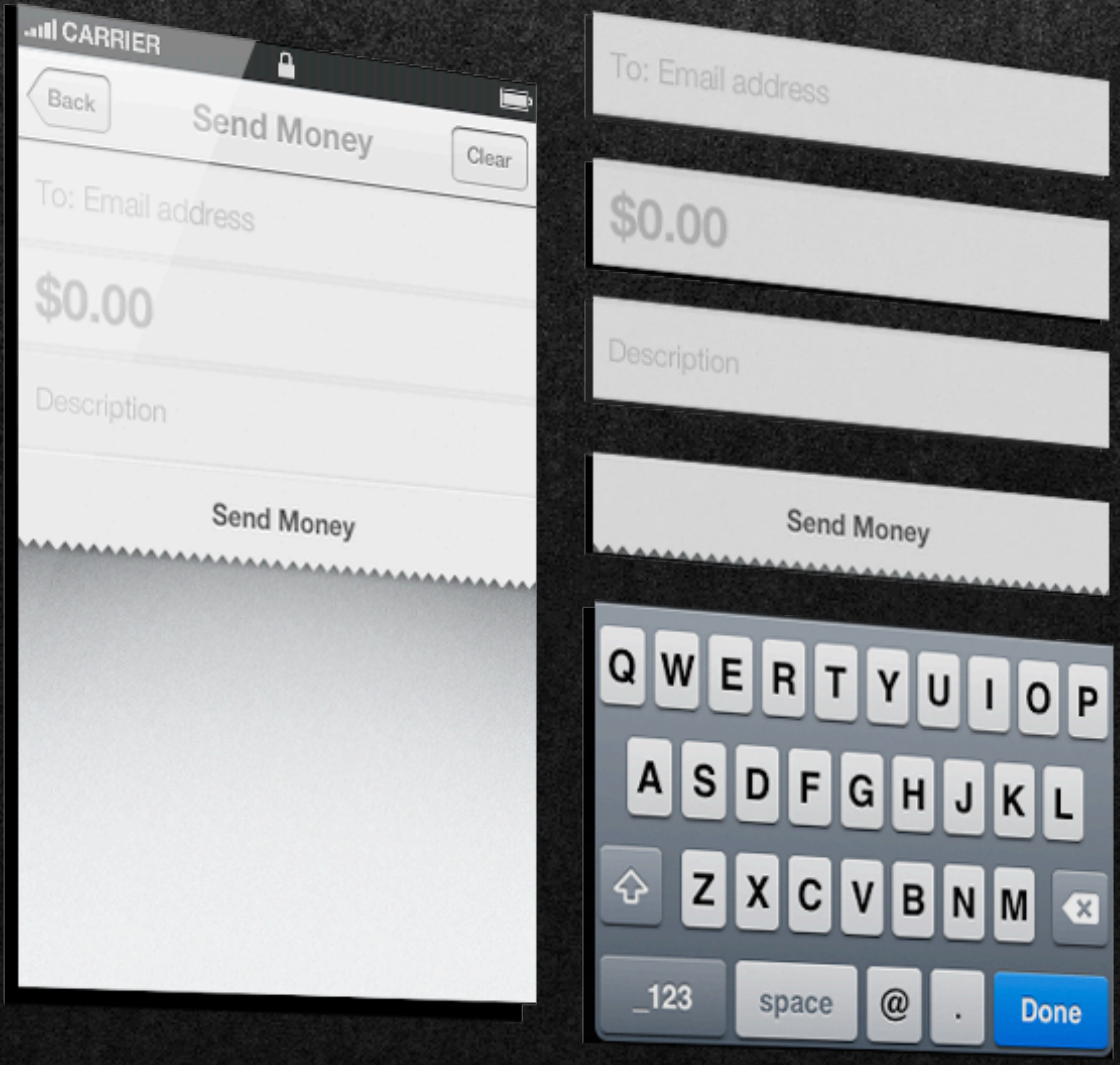


Our app also makes it easy to send and request money from friends by reading from the phone's address book. Tapping on the email text field shows a list of all your contacts that is searchable by name or email address. When you're done selecting a contact, the list gracefully slides up to get out of your way.

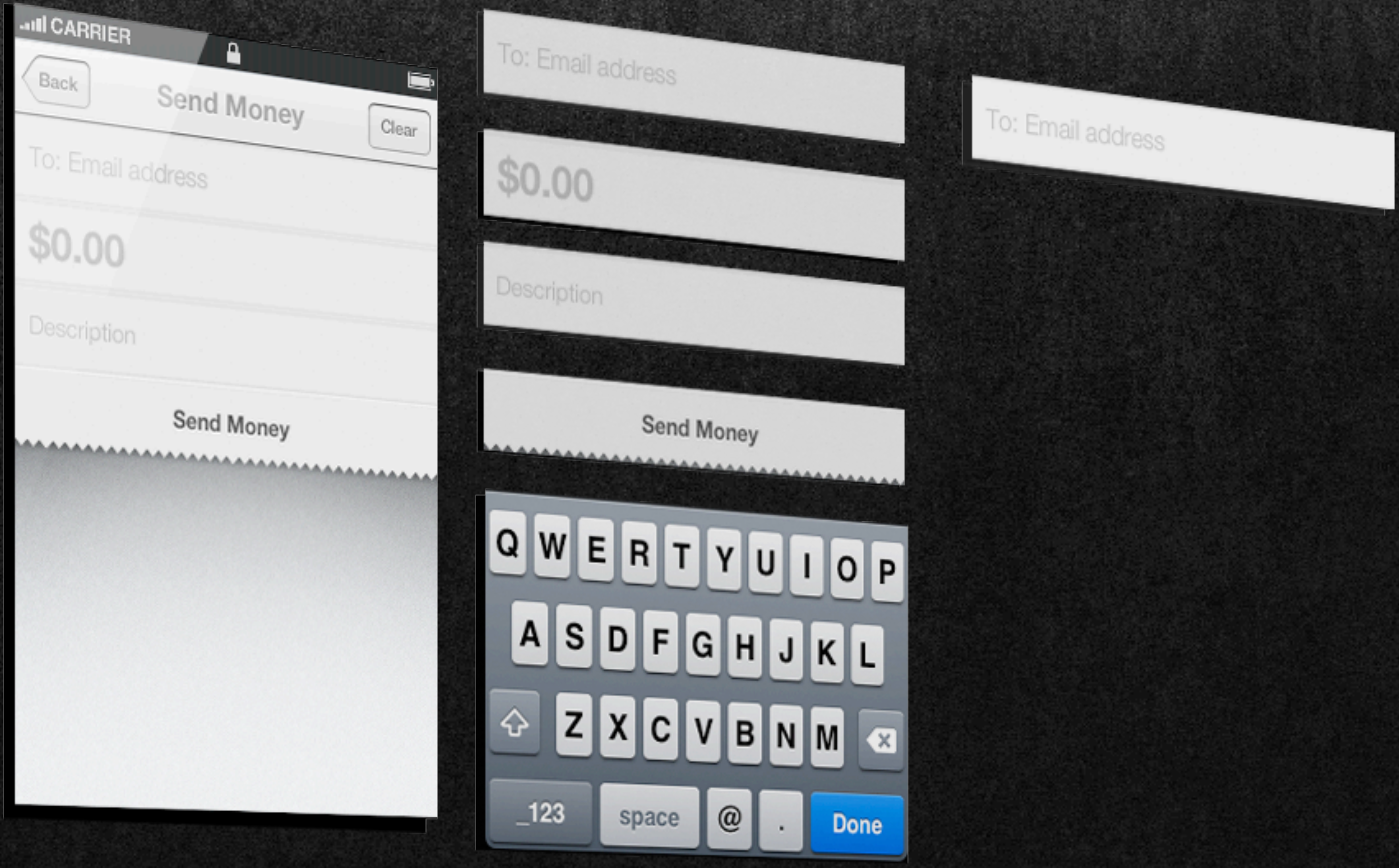
When we were building this view, and other views with complex interactions, we made it a point to design the interface before we started programming. Starting with the interface allowed us to iterate quickly and ask ourselves, "Does this view make sense? Is this easy to use? Does it solve the problem at hand?". We could only truly answer those questions when we were dealing with a real interface.



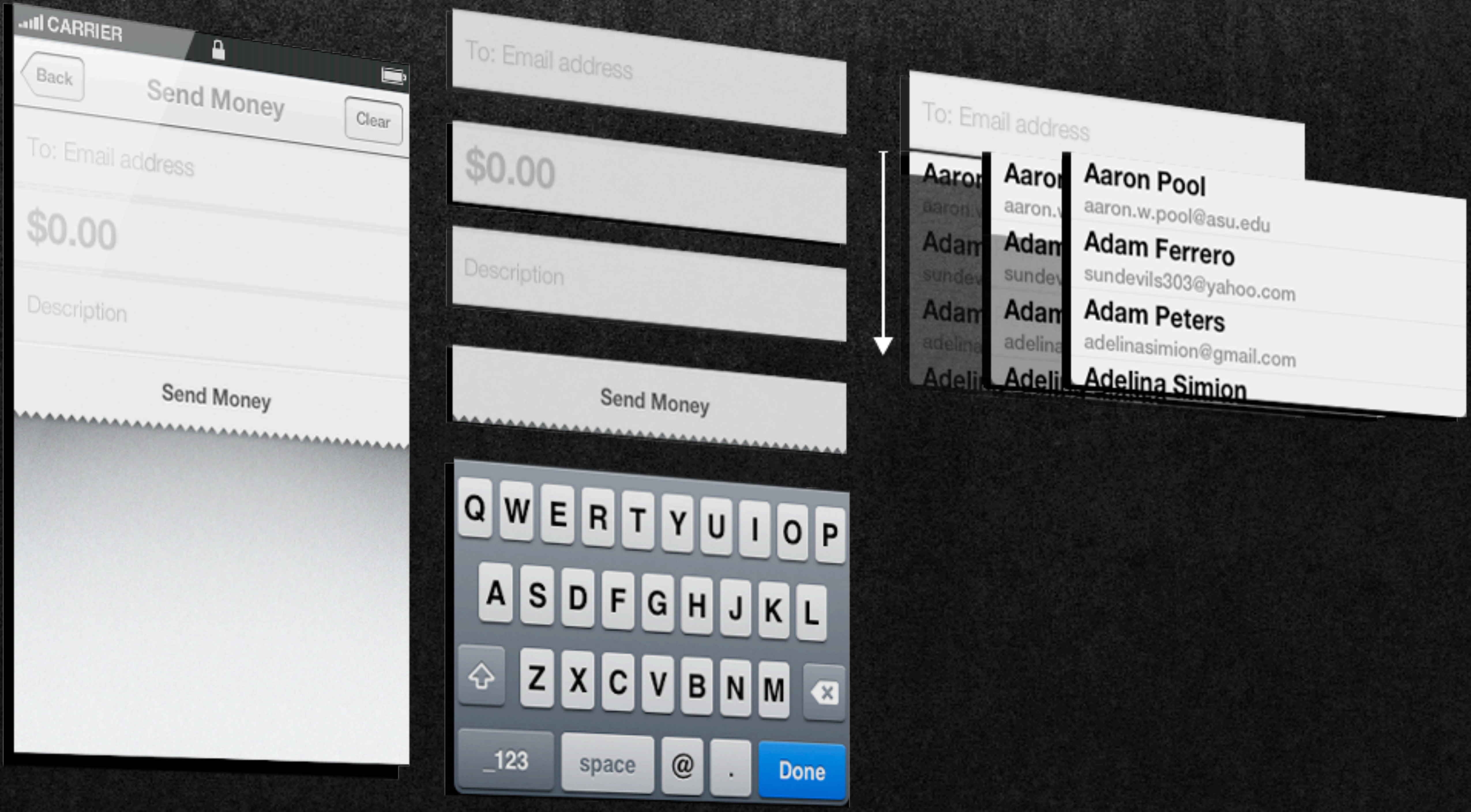
One of the design patterns that we used frequently was the delegate pattern. In the delegate pattern, we have a view controller that references and manages several views. When something interesting happens to one of these views, the view will notify it's view controller, so it can decide what to do next. So for example, when a user taps on the email text field, the view controller slides down a list of contacts, and fades out the views behind the list. After a contact is selected, the view controller slides up the list of contacts and fades in the other views. The delegate pattern is really powerful, and it allowed us to translate our design into real code.



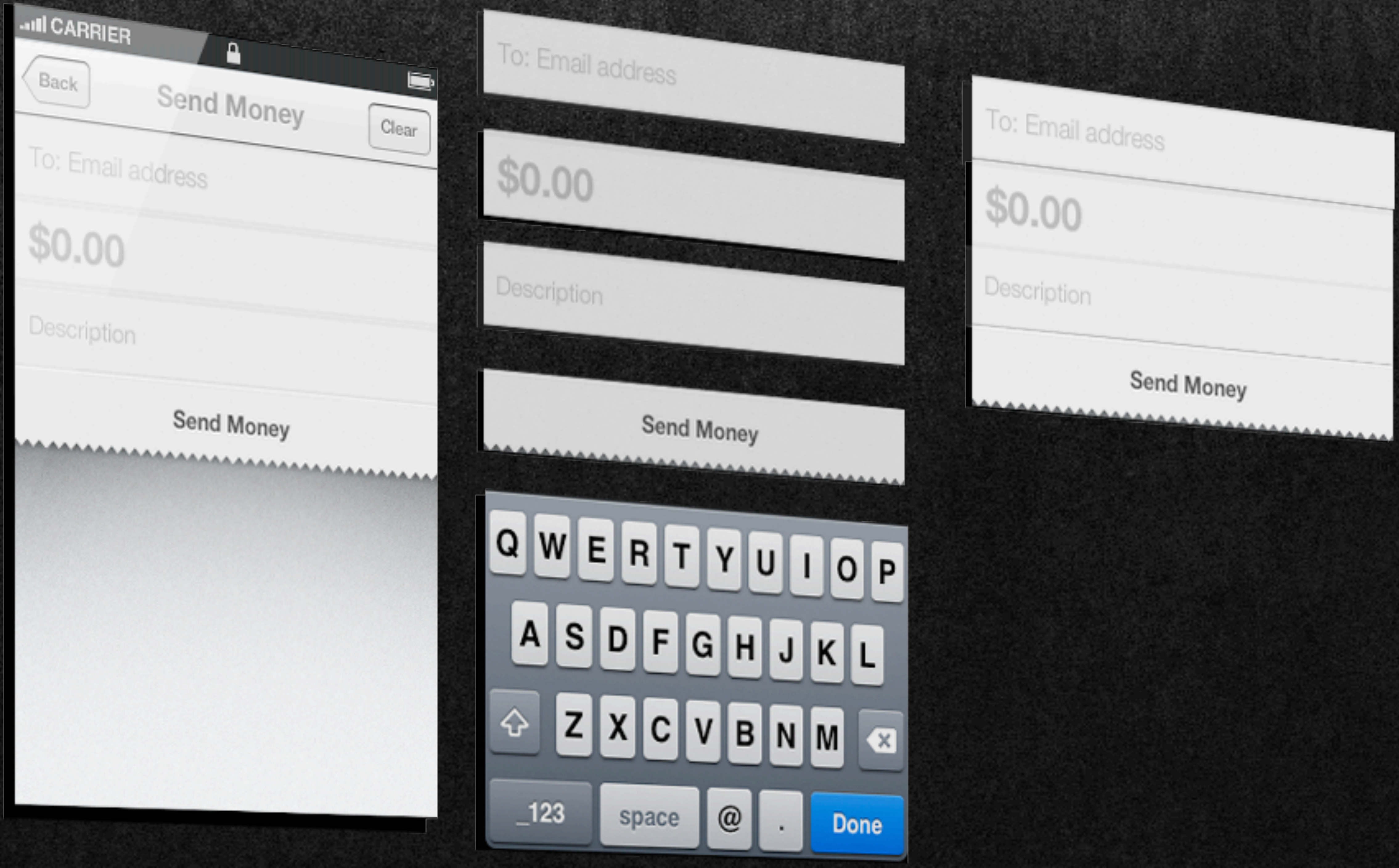
One of the design patterns that we used frequently was the delegate pattern. In the delegate pattern, we have a view controller that references and manages several views. When something interesting happens to one of these views, the view will notify it's view controller, so it can decide what to do next. So for example, when a user taps on the email text field, the view controller slides down a list of contacts, and fades out the views behind the list. After a contact is selected, the view controller slides up the list of contacts and fades in the other views. The delegate pattern is really powerful, and it allowed us to translate our design into real code.



One of the design patterns that we used frequently was the delegate pattern. In the delegate pattern, we have a view controller that references and manages several views. When something interesting happens to one of these views, the view will notify it's view controller, so it can decide what to do next. So for example, when a user taps on the email text field, the view controller slides down a list of contacts, and fades out the views behind the list. After a contact is selected, the view controller slides up the list of contacts and fades in the other views. The delegate pattern is really powerful, and it allowed us to translate our design into real code.

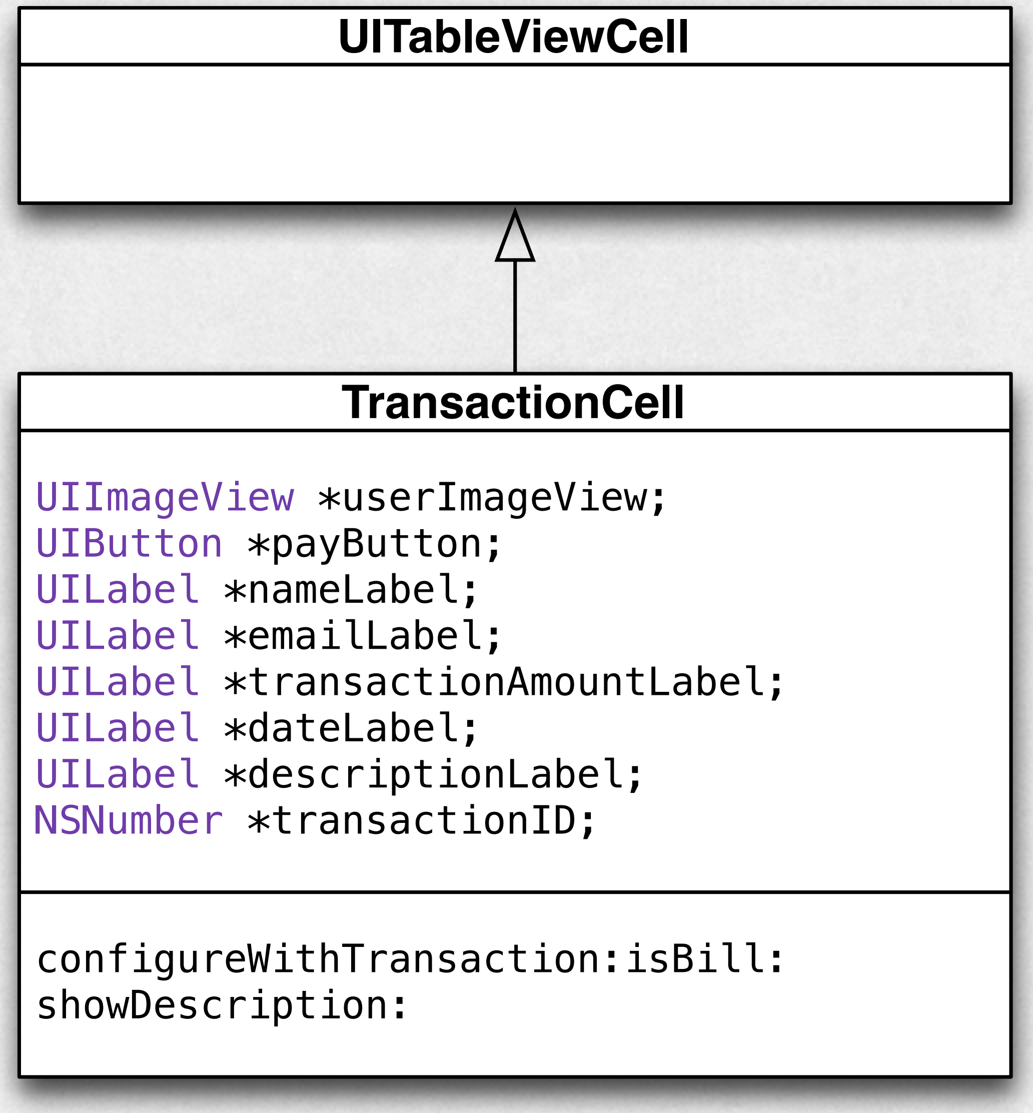


One of the design patterns that we used frequently was the delegate pattern. In the delegate pattern, we have a view controller that references and manages several views. When something interesting happens to one of these views, the view will notify it's view controller, so it can decide what to do next. So for example, when a user taps on the email text field, the view controller slides down a list of contacts, and fades out the views behind the list. After a contact is selected, the view controller slides up the list of contacts and fades in the other views. The delegate pattern is really powerful, and it allowed us to translate our design into real code.



One of the design patterns that we used frequently was the delegate pattern. In the delegate pattern, we have a view controller that references and manages several views. When something interesting happens to one of these views, the view will notify it's view controller, so it can decide what to do next. So for example, when a user taps on the email text field, the view controller slides down a list of contacts, and fades out the views behind the list. After a contact is selected, the view controller slides up the list of contacts and fades in the other views. The delegate pattern is really powerful, and it allowed us to translate our design into real code.

Transaction Cell

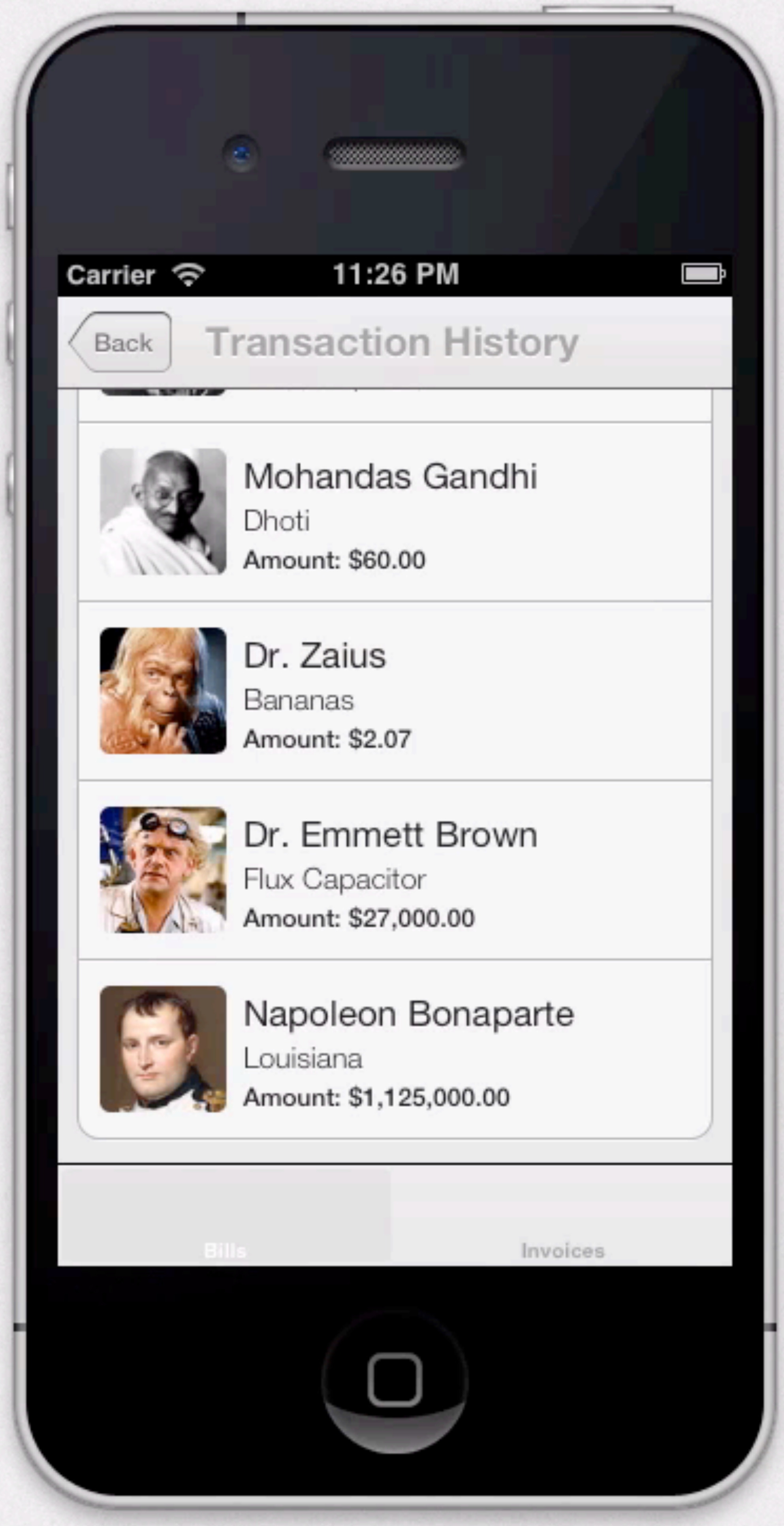


```
UITableViewCell
```

```
TransactionCell

UIImageView *userImageView;
UIButton *payButton;
UILabel *nameLabel;
UILabel *emailLabel;
UILabel *transactionAmountLabel;
UILabel *dateLabel;
UILabel *descriptionLabel;
NSNumber *transactionID;

configureWithTransaction:isBill:
showDescription:
```



Mohandas Gandhi
Dhoti
Amount: $60.00

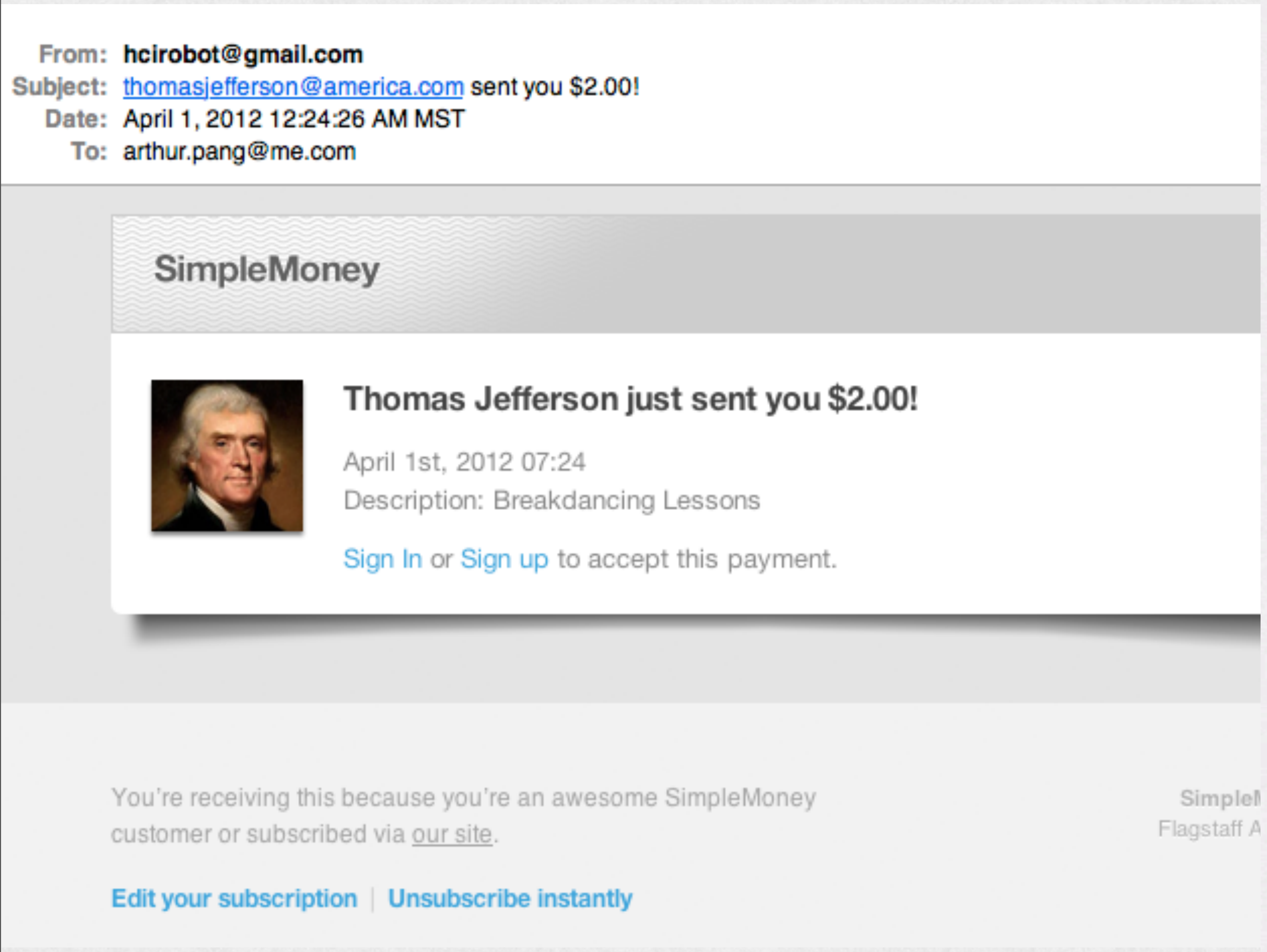
Dr. Zaius
Bananas
Amount: $2.07

Dr. Emmett Brown
Flux Capacitor
Amount: $27,000.00

Napoleon Bonaparte
Louisiana
Amount: $1,125,000.00

Another custom UI component that we built was the transaction cell. We subclassed a tableview cell to accept a transaction object as a parameter so it can display relevant transaction information like the recipient's email address and avatar. Just like the previous view, we used the delegate pattern to enable the cell to expand when it's tapped.

Pay a Bill



From: hcirobot@gmail.com
Subject: thomasjefferson@america.com sent you $2.00!
Date: April 1, 2012 12:24:26 AM MST
To: arthur.pang@me.com

SimpleMoney

Thomas Jefferson just sent you $2.00!

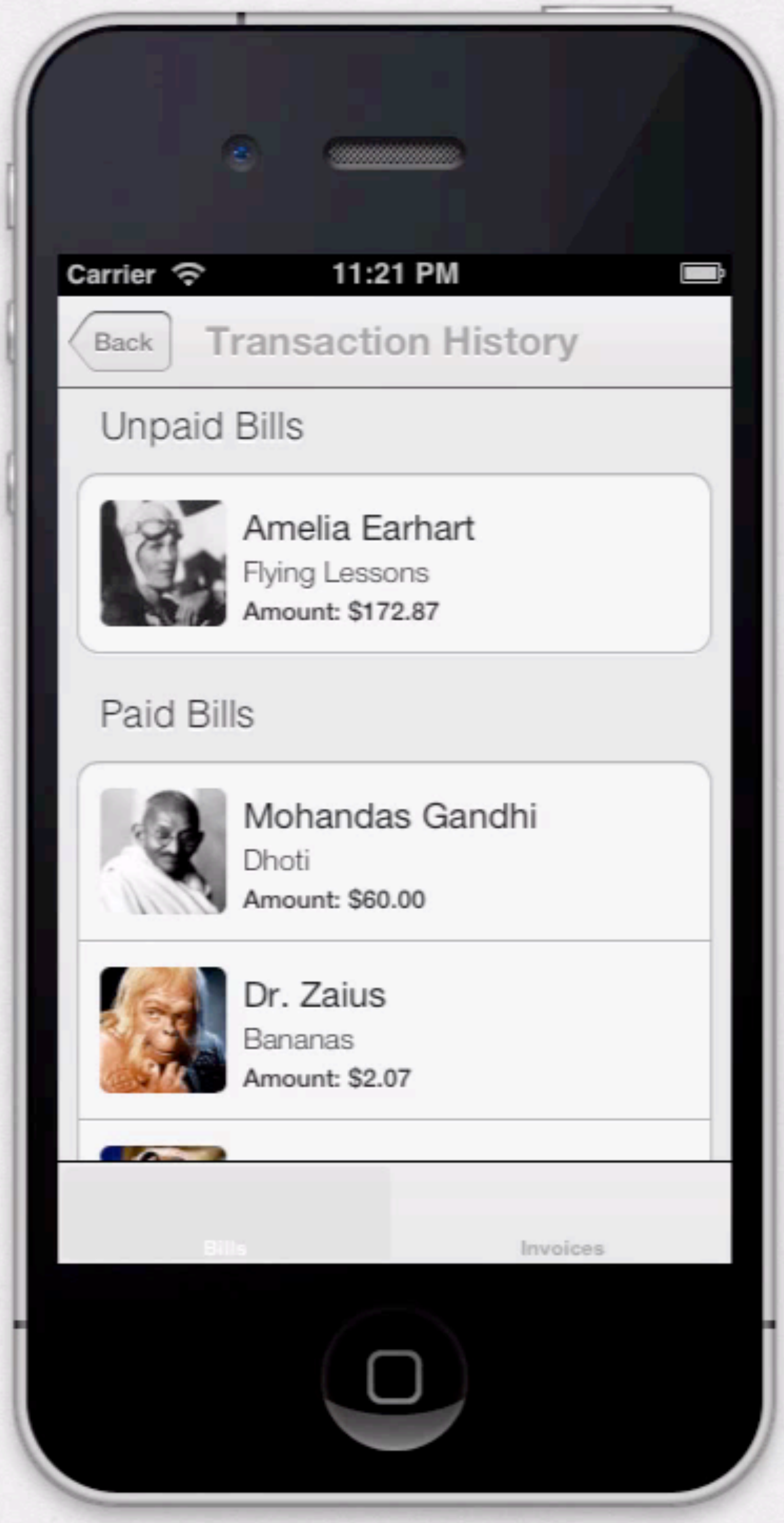
April 1st, 2012 07:24
Description: Breakdancing Lessons

Sign In or Sign up to accept this payment.

You're receiving this because you're an awesome SimpleMoney customer or subscribed via our site.

Edit your subscription | Unsubscribe instantly

SimpleM
Flagstaff A



Carrier 11:21 PM

Back Transaction History

Unpaid Bills

Amelia Earhart
Flying Lessons
Amount: $172.87

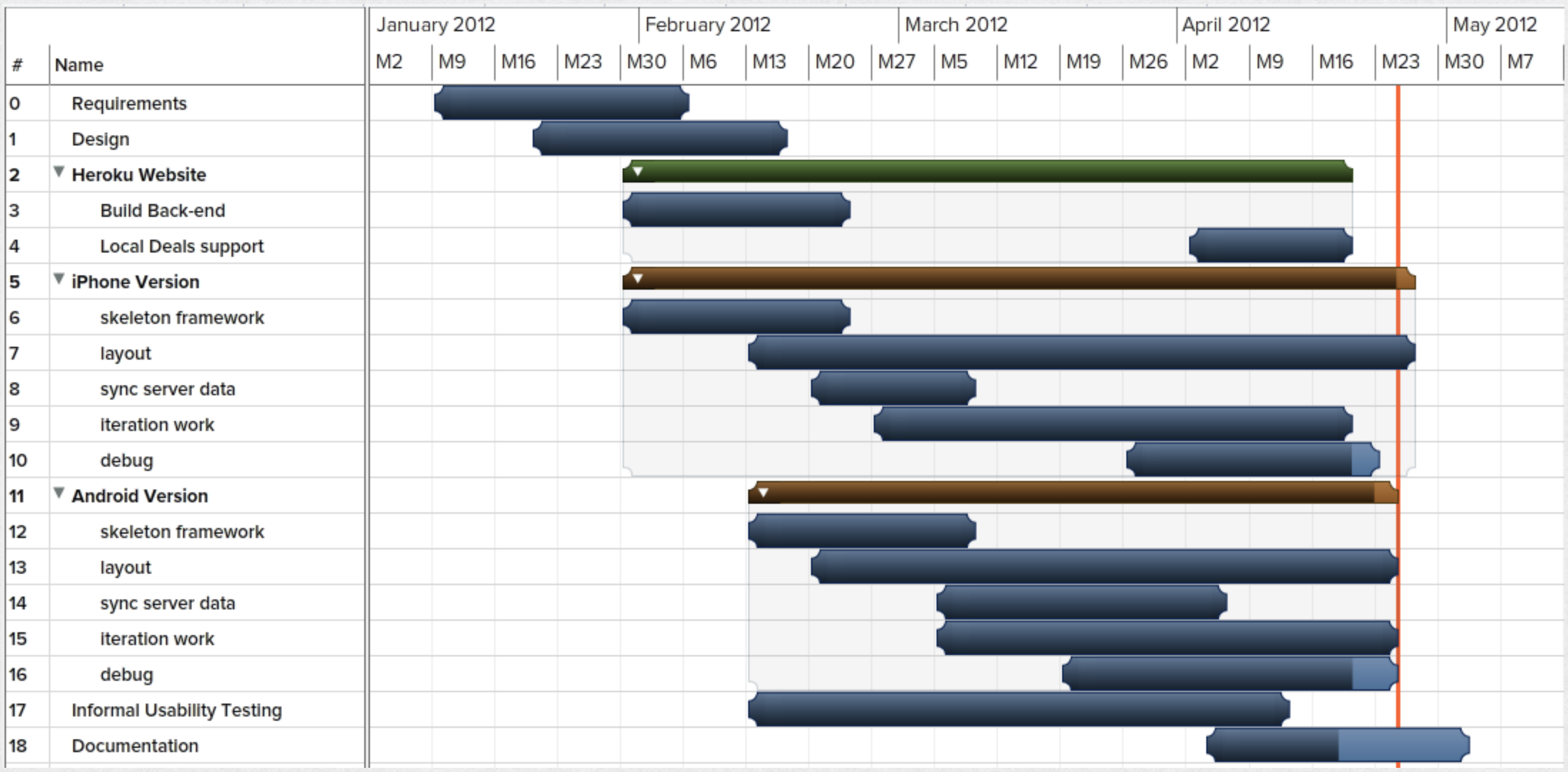
Paid Bills

Mohandas Gandhi
Dhoti
Amount: $60.00

Dr. Zaius
Bananas
Amount: $2.07

Another cool feature of the Transaction cell is that it allows users to pay bills by selecting an unpaid bill and tapping on the pay button. This updates the transaction locally on the device and sends a request to the server. If the transaction is updated successfully, the server transfers money from the sender's account to the recipient's account, and emails both parties a transaction receipt.

Development



#	Name
0	Requirements
1	Design
2	▼ Heroku Website
3	Build Back-end
4	Local Deals support
5	▼ iPhone Version
6	skeleton framework
7	layout
8	sync server data
9	iteration work
10	debug
11	▼ Android Version
12	skeleton framework
13	layout
14	sync server data
15	iteration work
16	debug
17	Informal Usability Testing
18	Documentation

high level challenges
learning android SDK, iOS SDK, ruby on rails

lower level challenges
object mapping, etc working in production

modeling data - stupid mistake: representing money as floats instead of ints
- setting up relatonships between users, transactions, items

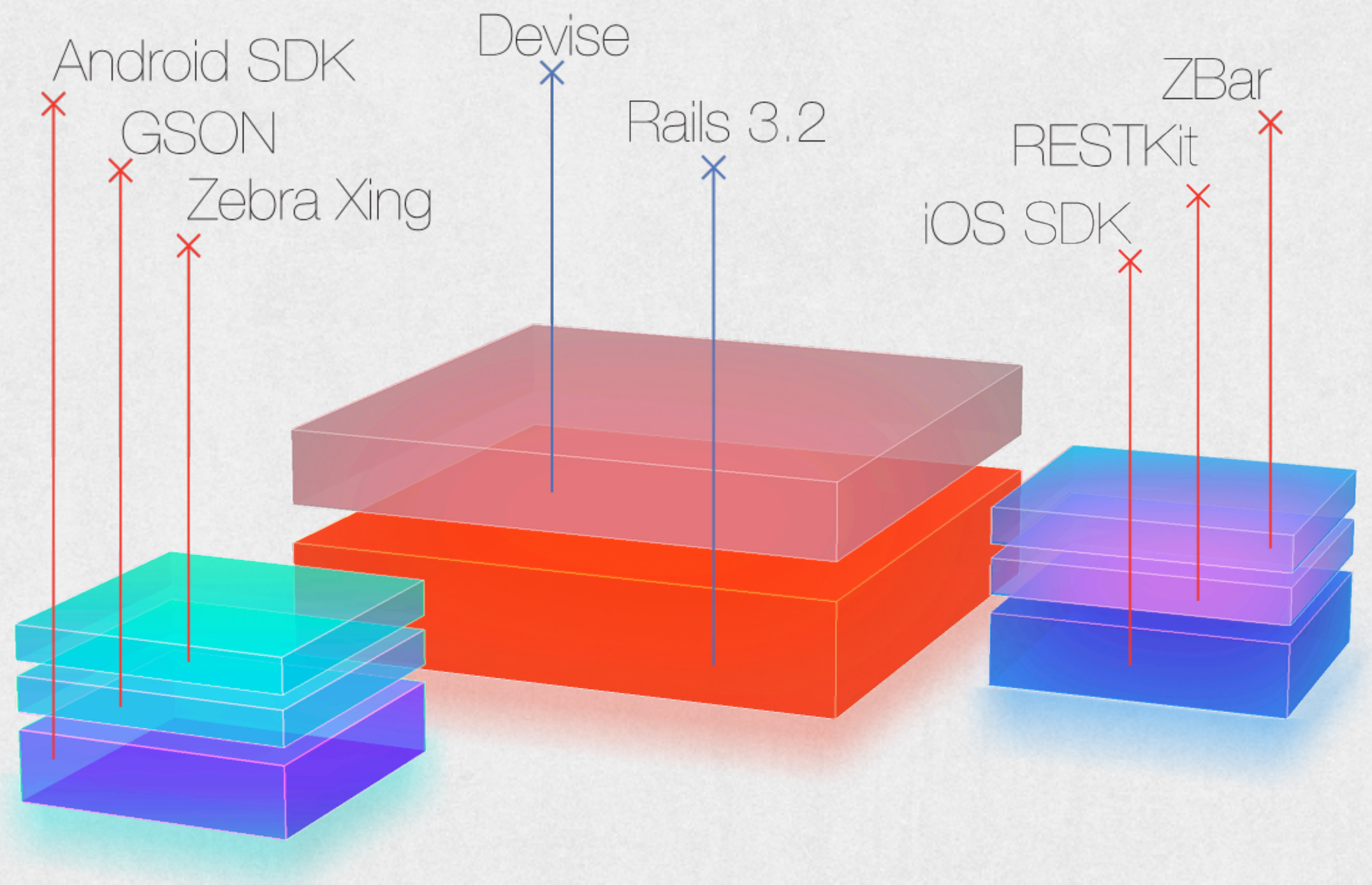
replicating data - user and transaction models

Challenges

1) Gathered and analyzed interaction patterns from competitor apps
2) Clip of formal requirements, some early wireframes we developed
3) Some

Challenges

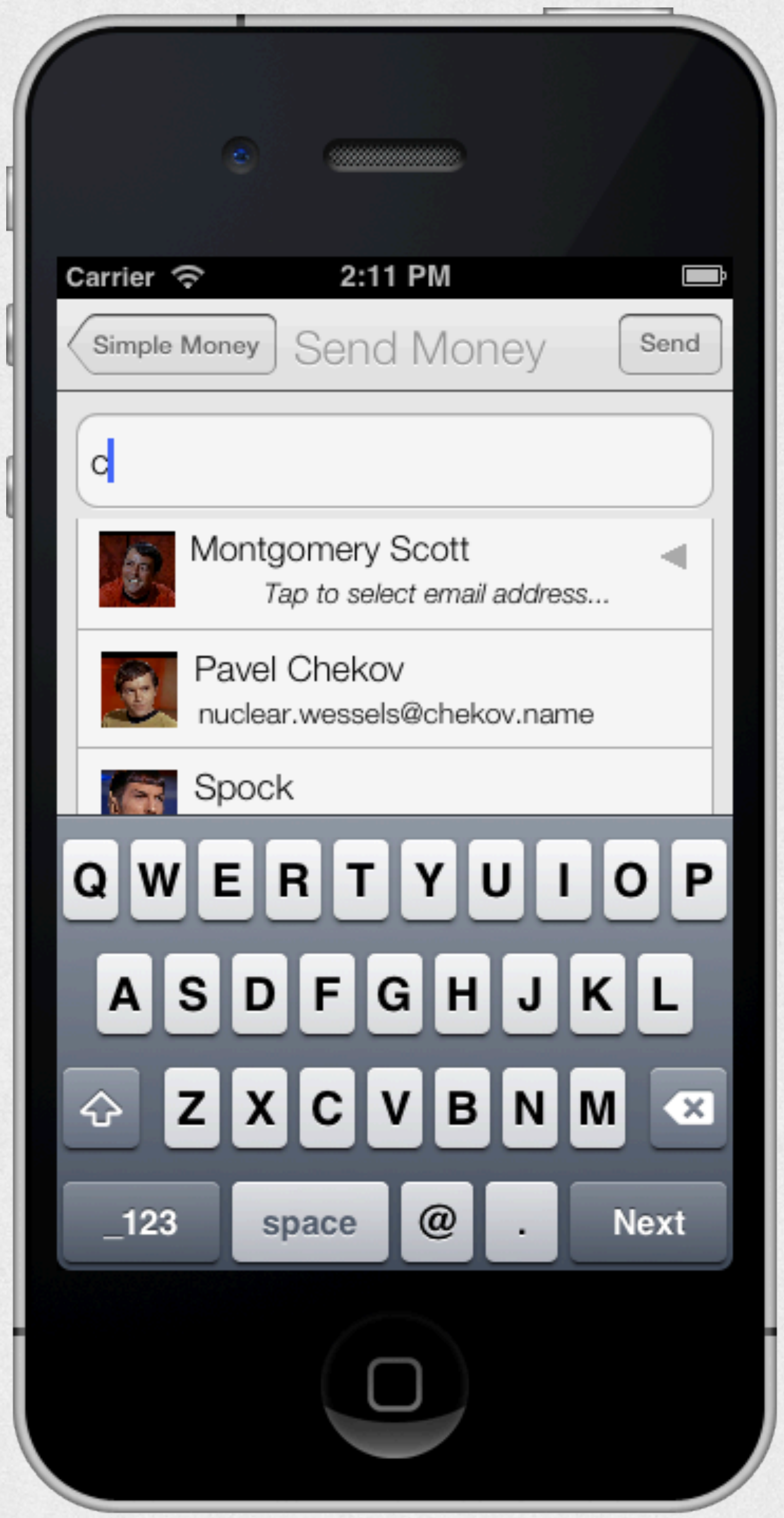
- New technologies



1) Gathered and analyzed interaction patterns from competitor apps
2) Clip of formal requirements, some early wireframes we developed
3) Some

Challenges

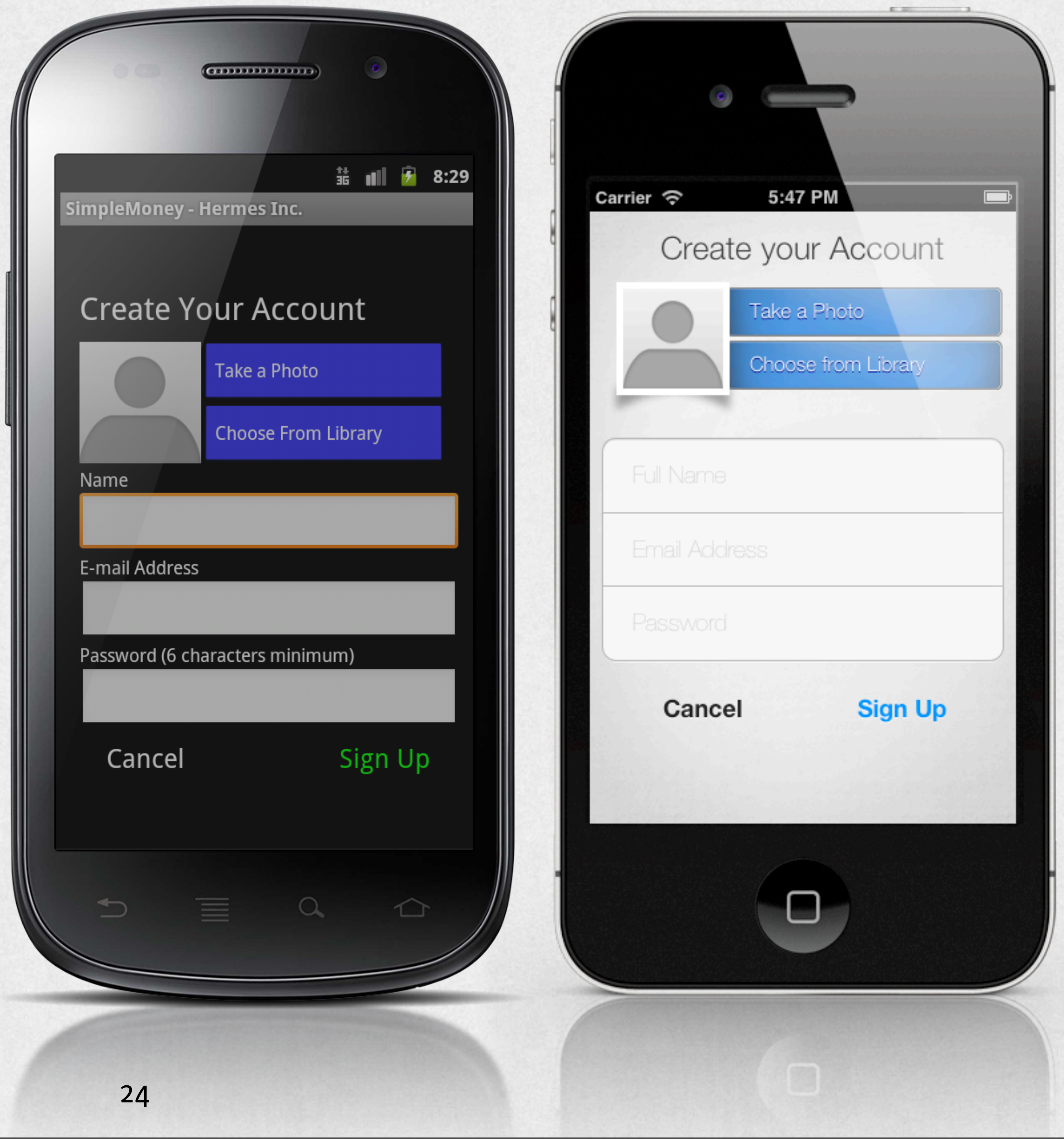
- New technologies
- Building custom UI



1) Gathered and analyzed interaction patterns from competitor apps
2) Clip of formal requirements, some early wireframes we developed
3) Some

Challenges

- New technologies
- Building custom UI
- Matching UI across platforms



1) Gathered and analyzed interaction patterns from competitor apps
2) Clip of formal requirements, some early wireframes we developed
3) Some

Testing and Validation

high level challenges
learning android SDK, iOS SDK, ruby on rails

lower level challenges
object mapping, etc working in production

modeling data - stupid mistake: representing money as floats instead of ints
- setting up relatonships between users, transactions, items

replicating data - user and transaction models

Testing and Validation

- Peer to peer and merchant transactions

high level challenges
learning android SDK, iOS SDK, ruby on rails

lower level challenges
object mapping, etc working in production

modeling data - stupid mistake: representing money as floats instead of ints
- setting up relatonships between users, transactions, items

replicating data - user and transaction models

Testing and Validation

- Peer to peer and merchant transactions
- Pay by scanning a QR code

high level challenges
learning android SDK, iOS SDK, ruby on rails

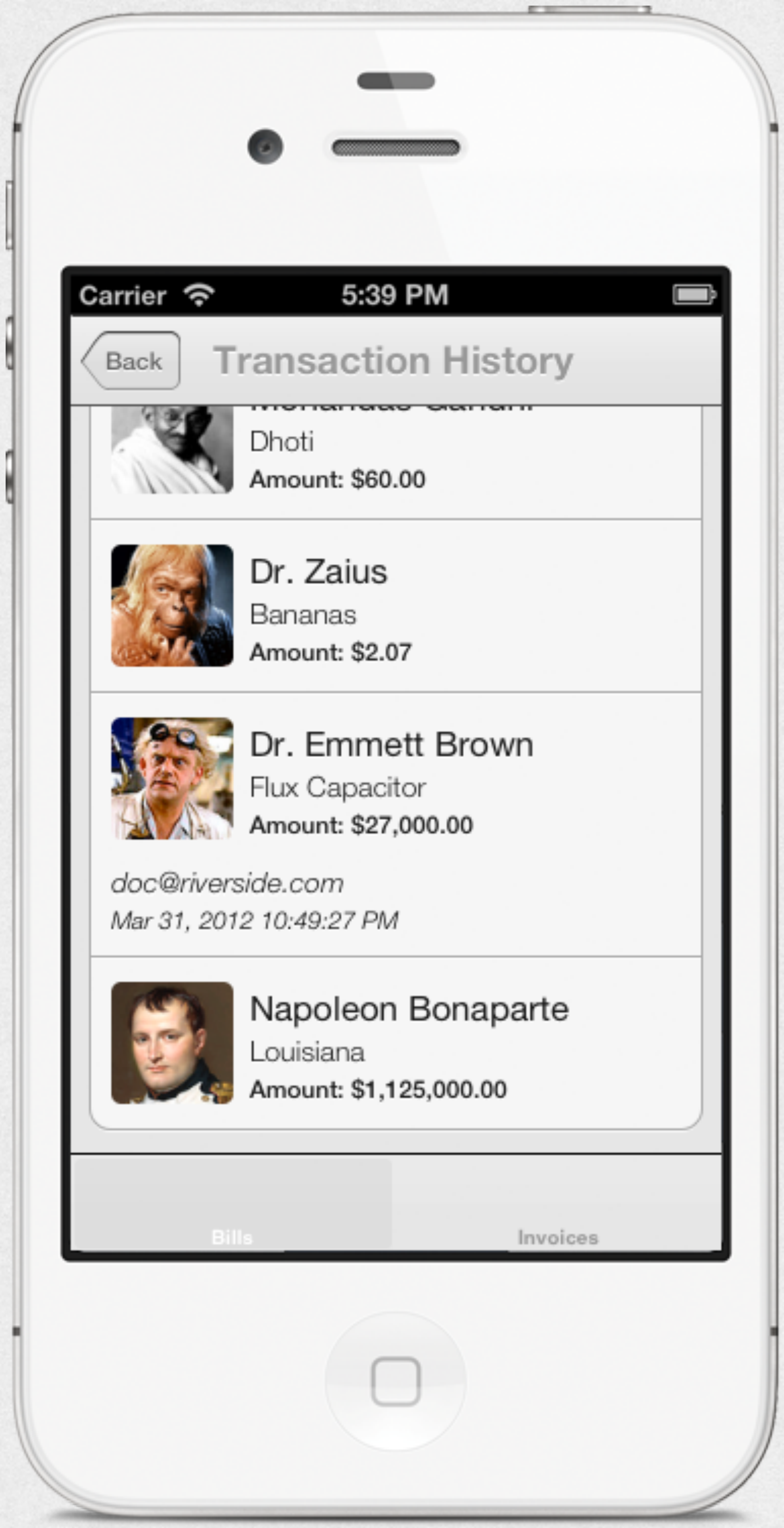
lower level challenges
object mapping, etc working in production

modeling data - stupid mistake: representing money as floats instead of ints
- setting up relatonships between users, transactions, items

replicating data - user and transaction models

Testing and Validation

- Peer to peer and merchant transactions
- Pay by scanning a QR code
- View balance and transaction history



Dhoti
Amount: $60.00

Dr. Zaius
Bananas
Amount: $2.07

Dr. Emmett Brown
Flux Capacitor
Amount: $27,000.00
doc@riverside.com
Mar 31, 2012 10:49:27 PM

Napoleon Bonaparte
Louisiana
Amount: $1,125,000.00

high level challenges
learning android SDK, iOS SDK, ruby on rails

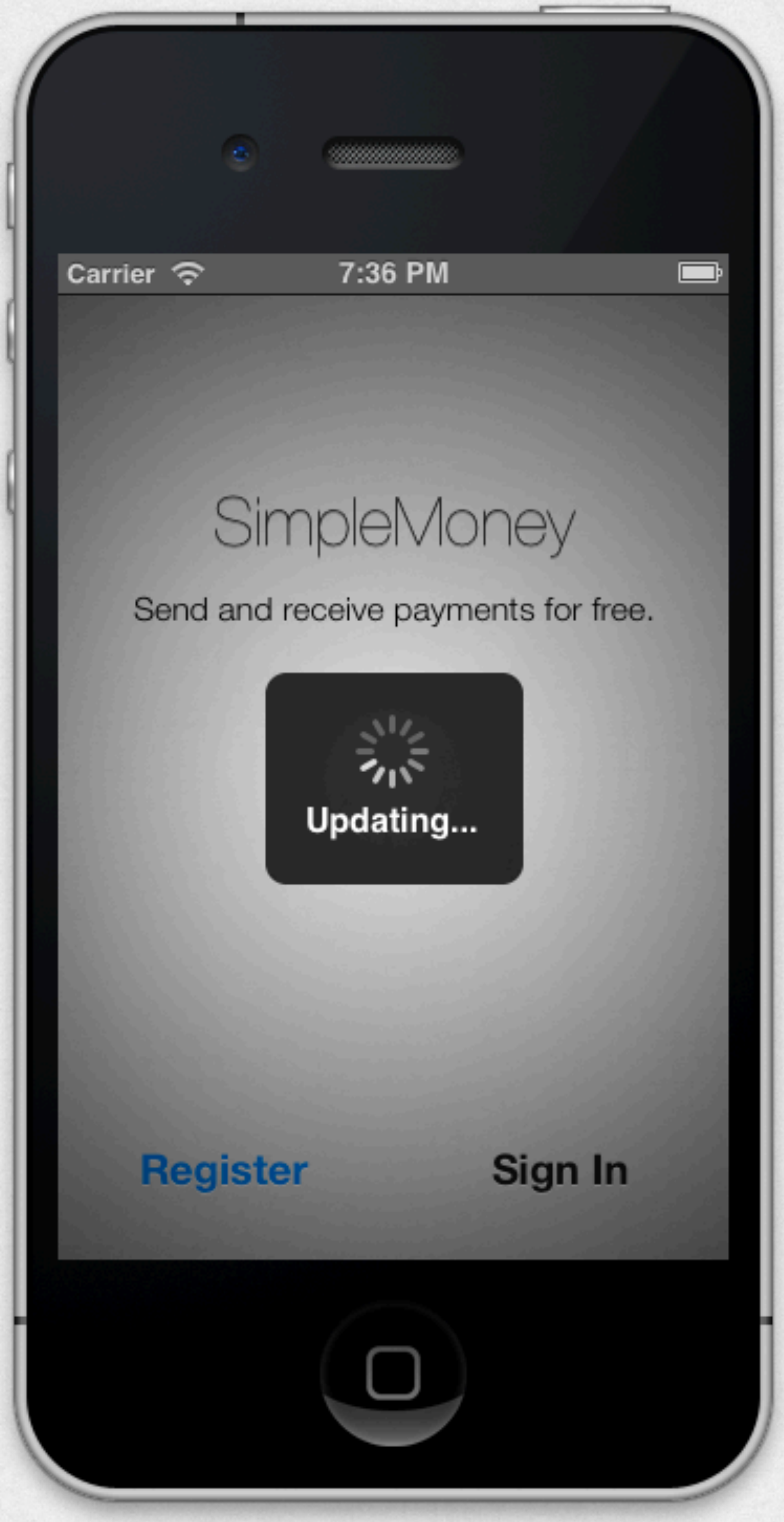
lower level challenges
object mapping, etc working in production

modeling data - stupid mistake: representing money as floats instead of ints
- setting up relatonships between users, transactions, items

replicating data - user and transaction models

Future Work

- Using real money
- Improving recommendations



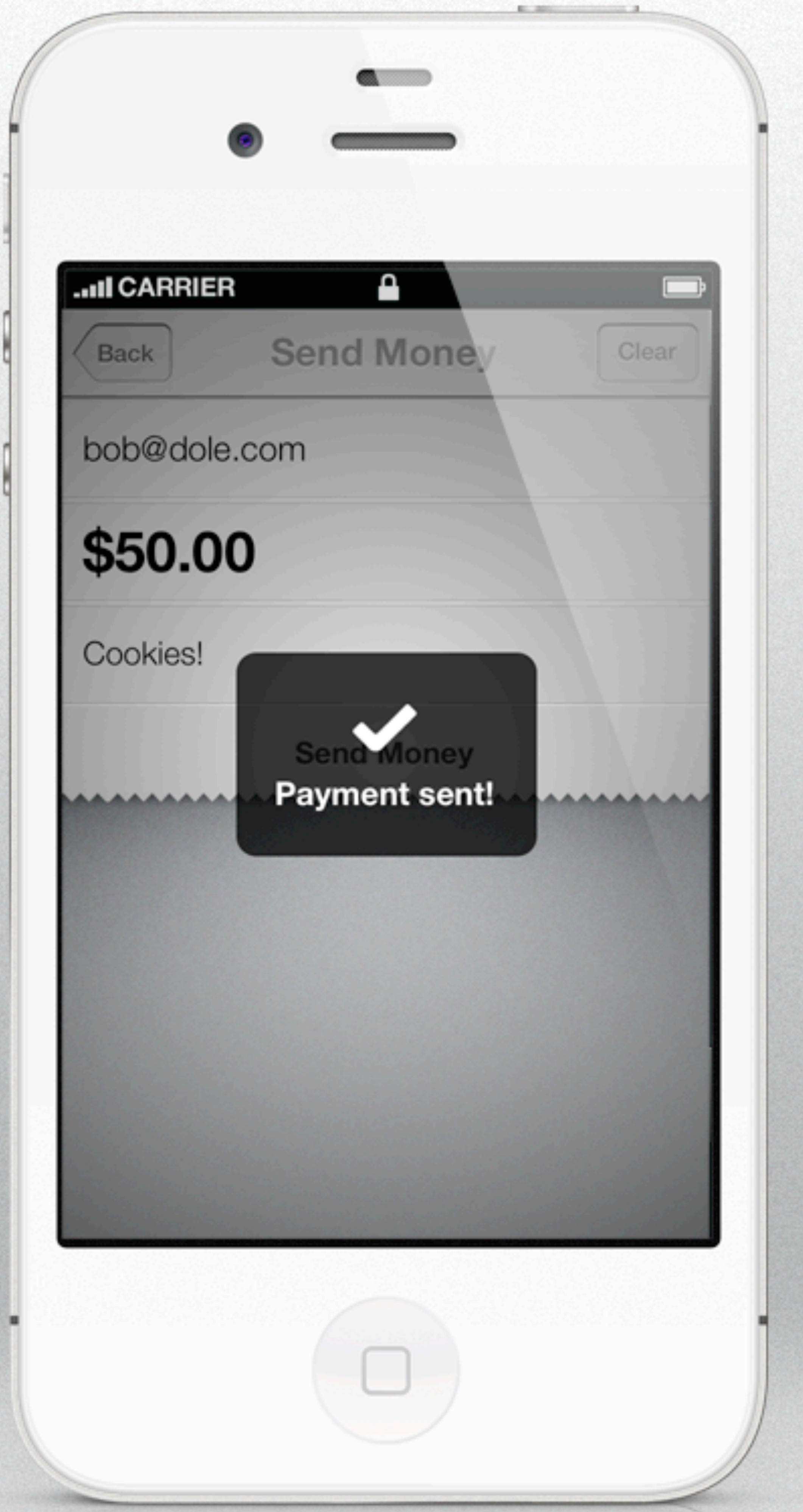
Adding merchant features –
to date, we have backend support for adding purchase items and their associated data, such as images.

Not using ACH – automated clearing house – API to transfer money between accounts
Currently, our app is using play money

Matching Andriod UI – we might have to build A LOT of custom UI to match apple’s ui components
tableviews?

Conclusion

- Simple, flexible and powerful payment solution
- Replace the credit card
- “Smart”: leverages context



Credit cards are stupid - they don’t tell you your balance or transaction history
banks offer apps that let you check your balance, why not take a step further?
A phone knowing a little about you can go a long way.

SimpleMoney

Department of Computer Science
Northern Arizona University
Client: Joshua Cross, Hermes Inc.
Team Saon: Arthur Pang - Joshua Conner - Nicholas Pallares

Background

Credit card usage comes at a cost. Merchants are charged a swipe fee, about 1-3% of the transaction, each time a credit card is used as the method of payment. The swipe fee is ultimately passed to consumers, which amounted to a total of $48 billion in 2008.

Purpose

SimpleMoney is a mobile payment system that allows users to send and receive money for free. Our task was to build a user interface for iOS and Android devices that allows users to send and receive money, view previous transactions, and view local deals and coupons around them.

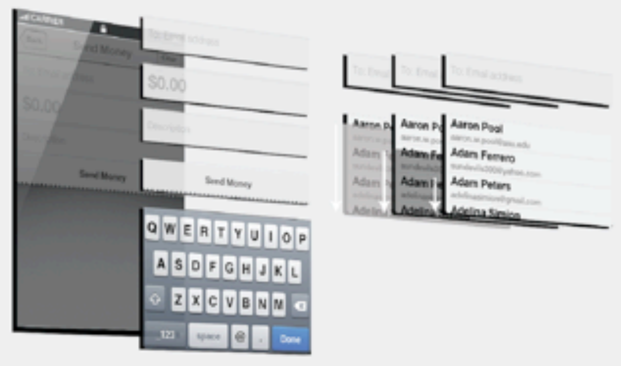
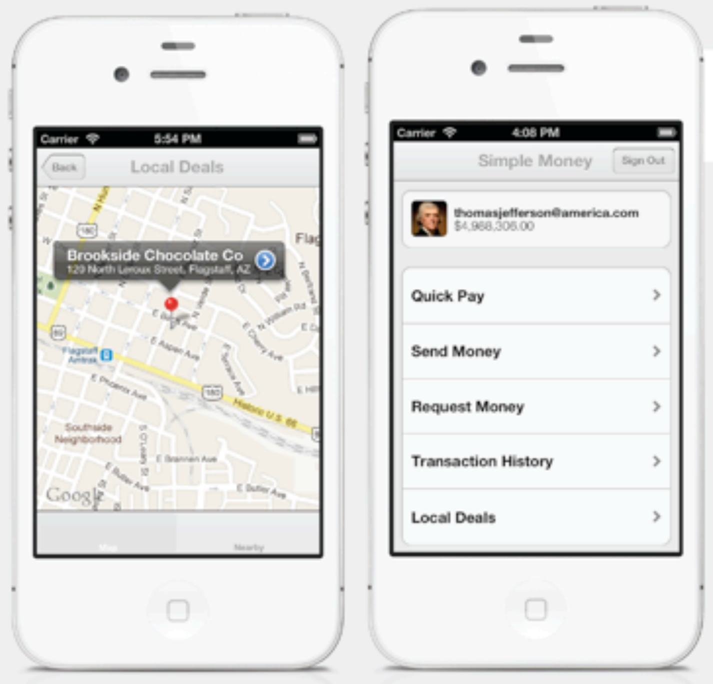


Figure 1. A custom user interface was designed to allow the user to select a transaction recipient from their phone's address book.



Payment has never been easier.

Four powerful features that simplify your life

Blazing fast payments.
Scan a QR Code to pay instantly with your phone.

Send money to anyone.
Anyone with an email address can send or receive a payment.

Keep track of your finances in real-time.
Instant payment notifications sent to your phone and email.

Find the best deals around you.
Save money at nearby businesses with SimpleMoney Coupons.

Scaleable and flexible.

Interfaces with cloud services for push notifications, data storage, and email notifications.

A full stack solution.

Built on open source software to work across the web and iOS / Android devices.

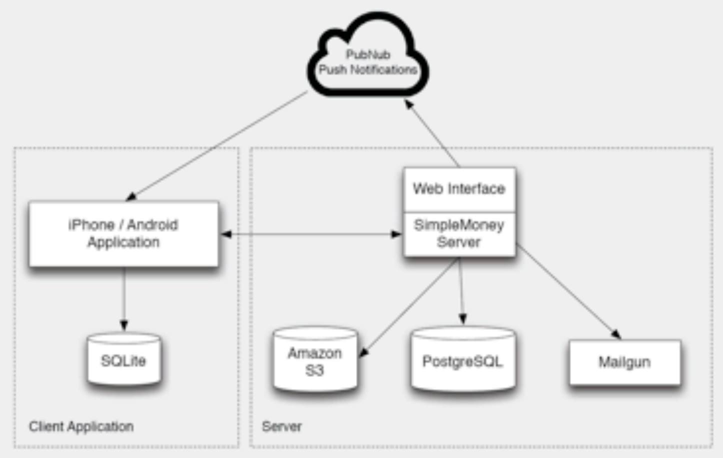


Figure 2. High-level architecture of the SimpleMoney system.
PubNub was used for push notifications, Amazon S3 for image assets, and Mailgun for dispatching emails.

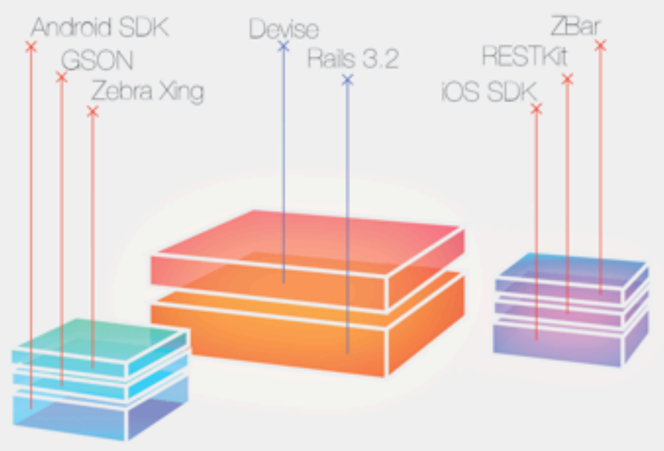


Figure 3. SimpleMoney's technology stack.

Chat with us!

Poster #279

Room B

On display:

1:30-4:30pm