Repay –
Revamping the reimbursement process

Esha Joshi
Department of Computer Science and Software Engineering
College of Engineering
California Polytechnic State University
2016

Acknowledgements

Repay would not have been made possible without the help of my incredible co-founder, Vivian Fong. Thank you for listening to all of my outlandish ideas and agreeing to work on this project with me. I hope you enjoyed all of our crazy entrepreneurship experiences and pivoting back and forth between the requirements and design. The web application you developed looks fantastic!

I would like to express my sincere gratitude to Dr. Janzen for being a wonderful professor and senior project adviser. Your feedback and support through this entire year has been invaluable and I have definitely enjoyed all of our chats about start-ups, software engineering, and life in general. Thank you for not giving up on Vivian and I when we came to your office each week with a new idea.

Thank you to Lori Jordan and Greg Gibson from the Cal Poly Center of Innovation and Entrepreneurship (CIE). Your endless support for our team dynamic and project made us feel really special and pushed us to work harder on making this project a reality. Thank you to Jared Hirata for providing us with amazing UX designs. Developing this application would have been difficult without your fast feedback! To Jared, Tobias, Kevin, Lucas, Jasper, and Isabella (our hackathon teammates), thank you for deciding to work with Vivian and me during the hackathon! All of you were integral in moving our project forward.

To my family, thank you for always supporting me in whatever crazy endeavors I have. I hope these last four years have made you proud!

At the Evening of Green and Gold 2016!

Abstract

Repay is a streamlined service for interviewees to be repaid by companies for transactions occurring throughout an interview. It plays off the idea that many graduating college seniors endure when looking for a job: there is not a hassle-free process for interviewees to be repaid for expenses when traveling to and from an interview. The goal of this senior project was to develop a cross-platform application (iOS and web) with the hopes of improving the reimbursement process for both future employees and company HR representatives. Research suggests that a more simplistic process will allow candidates to receive their money back quickly and enhance the overall reputation of companies.

Repay is both an iOS application developed in Swift and web application developed in AngularJS, leveraging some of the latest database technologies and design patterns. The project was completed over a period of four months and is currently being branded to appeal to companies that interview thousands of candidates each year.

Contents

1 Introduction

In the fall of 2015, many other enterprising college students and I eagerly began our job-search endeavors with the hopes of landing our "dream job" or any gig that would help pay off treacherous student loans. This grueling process required us to hone in on their skills, passions, and the market realities in order to discern for ourselves the types of companies and roles that would best fit our interests and personalities.

In doing so, my co-founder and I conducted extensive customer development interviews with Cal Poly students and company HR representatives to get a better understanding holistically of how this process works. We learned that there are two major pitfalls associated with the job-search process: time and money. For college students, devoting time to search for jobs with the expected time sync of academics is enough of a deterrent to procrastinate this process. Additionally, the expected travel costs associated with each interview compel students to have to sacrifice interviewing opportunities in order to maintain their financial stability.

In my personal experience, I took a couple of weekends off in September and October of 2015 from my Cal Poly schooling and travelled to a series of interviews and conferences in Seattle, San Francisco, and New York City. I paid an arm and a leg to ensure that my interviewing experiences were positive ones and expected that the reimbursement process after the interview would be seamless and smooth. However it was a harsh realization when I realized that I was at an extreme deficit of $1,400 after all my interviews.

1.1 The Problem

Every interaction with a company is integral when competing for top talent. Inefficient and complicated reimbursement processes can cast a negative impression on the company. Processing expenses is cumbersome and can lead to errors in the money-handling and transferring processes. From the interviewee's standpoint, an interview can be exciting – but costly. We found it incredulous that there were many college students, including me, who were hundreds of dollars in debt from waiting weeks or months to get their money back in their pockets.

From our customer development with Cal Poly students and enterprise companies such as MINDBODY, Intuit, Workday, and Google, my co-founder and I concluded

that there are three main pain points associated with existing reimbursement processes: expense handling after the interview is cumbersome, paperwork filing is inconsistent from company to company, and the reimbursement waiting game is frustrating. The number of varying responses of exasperation towards the entire reimbursement process proved to us that a new approach towards expense management/handling was needed.

1.2 The Solution

With the creation of Repay, we have taken a new approach to the reimbursement process between companies and interviewees. The underlying concept is simple: Repay improves the reimbursement process without casting a sour impression on companies or leaving interviewees several hundred dollars in debt.

Our final solution, which pivoted and was enhanced a number of times in the last four months, is a cross-platform solution that saves time by removing the hassle of physical receipt collection and submission, mitigates confusion by digitizing a run of the mill process, and tracks/receives money sooner by supporting in-app reimbursements. Our stretch goal was to integrate this software with sophisticated and existing money transfer solutions: Venmo, PayPal, and Square Cash.

1.3 Project Evolution

My co-founder and I started out in the fall 2015 with a multitude of different ideas: a news aggregation app, software to create a local marketplace for community needs, and finally reimbursement software. We encountered several pivots before developing our final product – known as Repay today. Our senior project journey was also marked by great strides in entrepreneurship and involvement with Cal Poly's CIE.

1.3.1 Fall 2015

Prior to working on Repay, our preliminary idea was to develop Aucto, a P2P mobile commerce application that could provide students with a reliable new way to hire other skilled students in their college communities. My team entered the on-campus Hatchery program, one of Cal Poly's entrepreneurial initiatives to teach students about startups, and facilitated extensive market research.

1.3.2 Winter 2016

Although we were able to validate the need for a campus marketplace for artistry needs on Cal Polys campus, my co-founder and I were intrigued by how much of a financial burden the reimbursement process was creating for fellow students and I. We quickly pivoted from Aucto and started brainstorming. With this new insight in mind, we participated at Startup Weekend in February 2016 and earned the 'Honorable Mention' award with our new idea, *Moola*. *Moola*, which eventually re-branded to *Mula*, became our focus for the next several months.

Mula was marketed as the first direct payment system in which interviewees could pay for their transactions directly using their mobile devices. Consequently companies could program their expenses for different budgets (i.e. food, lodging, and transportation) into our system to be used a guideline for interviewees. Companies using this product would load money into their *Mula* account which would then be distributed to a predetermined list of interviewees. They would also specify and add constraints to what their recipients could purchase and how much they could be spent in total. The "direct payment" aspect of *Mula* would be enforced by use of the near-field-communication (NFC) technology. Using these technologies in various vendors supporting Apple Pay, this product would mitigate the need for physical receipts and digital invoices. Pictures of receipts would only be necessary if the vendor did not support any advanced communication payment technologies. With *Mula*, there would no longer be a need for current employees at companies to manually validate each individual payment transaction. All of the data associated with the interviewees using the *Mula* application would be stored on our servers and provided back to the companies for statistical analyses.

1.3.2.1 The Harsh Truth

The idea was cool, but was simply not a market reality. First, we found that there is high reluctance for individuals to move to a credit-less society or one that relies heavily on NFC/RFID technologies. Second, mobile payments using these technologies is considered elitist and not available in most parts of the United States. Research indicates that while 52% of North Americans are "extremely aware" of mobile payment technology, only 18% use it on a regular basis [7]. CEO Tim Cook acknowledged during one of Apple's announcements that, as of December 2015, about 3.5% of the roughly 7 million to 9 millions merchants in the U.S. work with Apple Pay [8]. The remaining 96.5% of businesses do not have point-of-sale systems that work with the NFC technology Apple Pay relies on.

In order to understand this better, we spoke to employees from different companies.

Reed Morse from Google advised us against NFC technology: "Reduce friction when building a company. Apple Pay is going to increase friction and [this product] will have to rely on merchants upgrading their checkout process – an endeavor that many businesses are reluctant to do so yet." Justin Hogen-Esch, also from Google, told us, "Data from the payment kiosks is not obtainable at the moment – unsupervised machine learning opportunities would come at a steep price."

Figure 1 (below) shows a high-level timeline (everything previously discussed) of our senior project during the winter 2016 quarter.

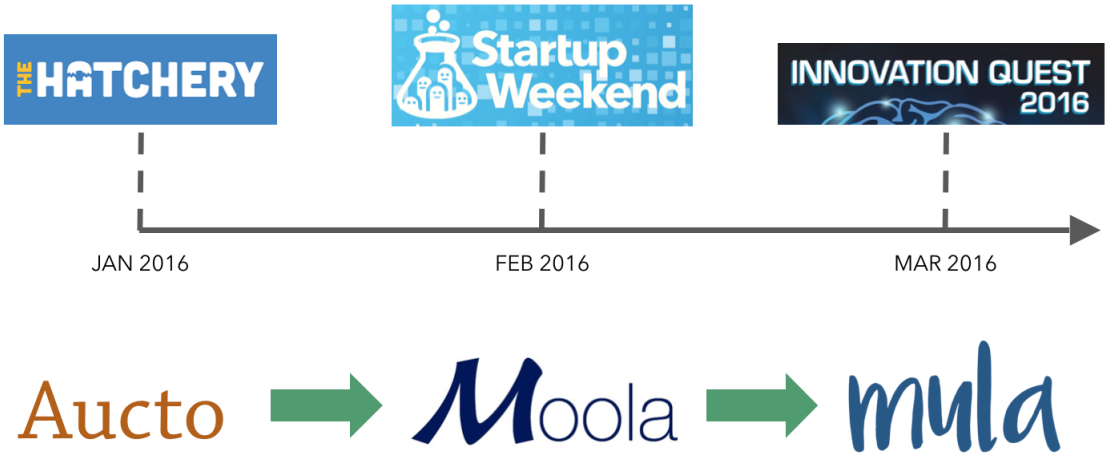

Figure 1: Winter quarter timeline.

1.3.3 Spring 2016

My co-founder and I came back to school in the spring with low morale; we were in dire need of an inspirational pivot. With that in mind, we participated in the Cal Poly Design and Dev Hackathon in April 2016, hoping to gain a different perspective on revamping the reimbursement process. We were able to assemble a team of seven students – bright developers and designers of different ages and skill levels. During the hackathon, we developed a very preliminary version of Repay. Repay takes a step back from dealing with in-app payment and focuses on a more serious issue – the process of taking pictures and submitting/mailing receipts, tracking expenses, and following company protocols in order to receive the interviewee's money back. The end-to-end application overview will be discussed in detail in the following section.

1 Introduction

Overall the hackathon was a positive experience; we worked with other wonderful developers and designers, received mentorship from talented industry representatives, and ended up winning first place!

2 End-to-End Application Overview

Repay is built as a cross-platform application, a web and mobile component, in order to streamline the reimbursement process between companies and their interviewees. It is designed such that the company HR representative interacts with our product through a web interface and the interviewee through a mobile interface. The version of Repay developed for the senior project specification enables the HR representative to add applicants, interviews, customize budgets for a specific interview, and approve/flag reimbursements created by the interviewee. Similarly, the interviewee can view budgets, request reimbursements, accept/flag reimbursement responses from the company representative, and view past reimbursement requests. This section discusses the high-level overview of the end-to-end interaction and includes detailed screenshots of both the web and mobile interfaces.

2.1 Scenario

Suppose Vivian is a human resources (HR) representative from Google and Esha is a interviewee for one of Google's coveted associate product manager positions. Vivian will be able to log in to the Repay web application, view active and past interviews, create an applicant position and interview opportunity for Esha, create and set budgets for her interview, and monitor Esha's transactions. During the time of her interview, Esha will be able to log in to the Repay mobile application, switch between active and past interviews, view remaining budgets for her current interview, request reimbursements, and view past reimbursement requests.

2.2 Web Application – Logging In

When Vivian visits the Repay application, she is greeted with an initial login page as seen in Figure 2, containing the Repay logo and slogan, "Revamping the reimbursement process". New business clients can sign up with Repay for the first time by clicking the "Sign up" button on the upper right corner of the web application.

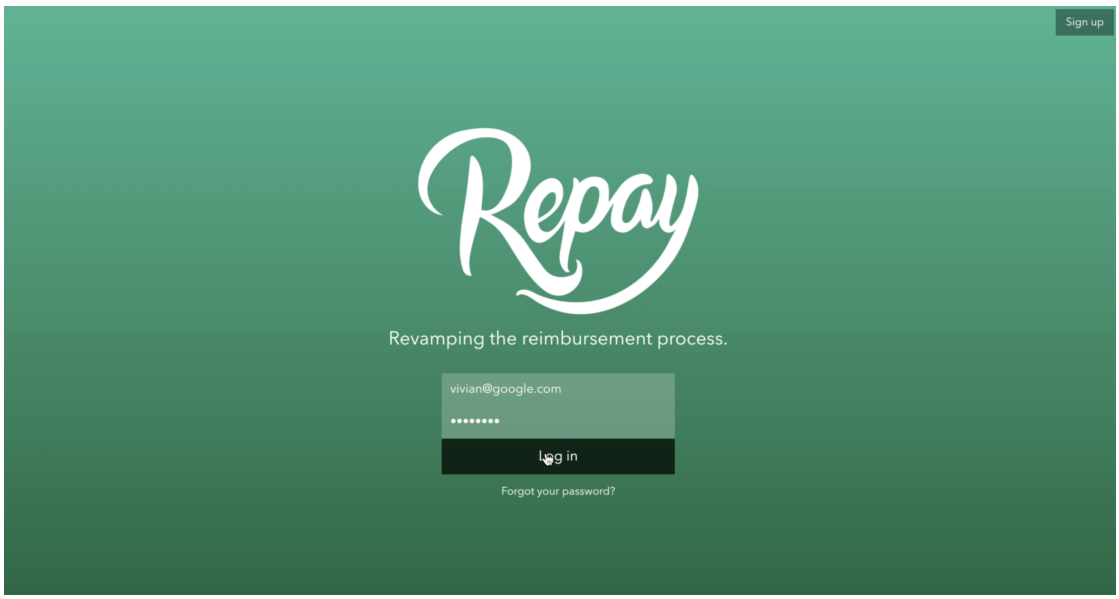

Figure 2: The login page for the web application.

2.3 Web Application – Set Up

2.3.1 Viewing Interviews & Setting Budgets

Once Vivian logs into the application, she sees a variety of pages she can visit via the navigation bar on the left-side of the web app: "Budgets", "Positions", "Applicants", "Interviews", and "Approvals". On the "Interviews" page, Vivian can view all active and past interviews. Figure 3 shows an active interview (at the time) for applicant Katelyn Hicks, who interviewed for the "Software Engineer I" position at Google from June 1, 2016 - June 4, 2016.

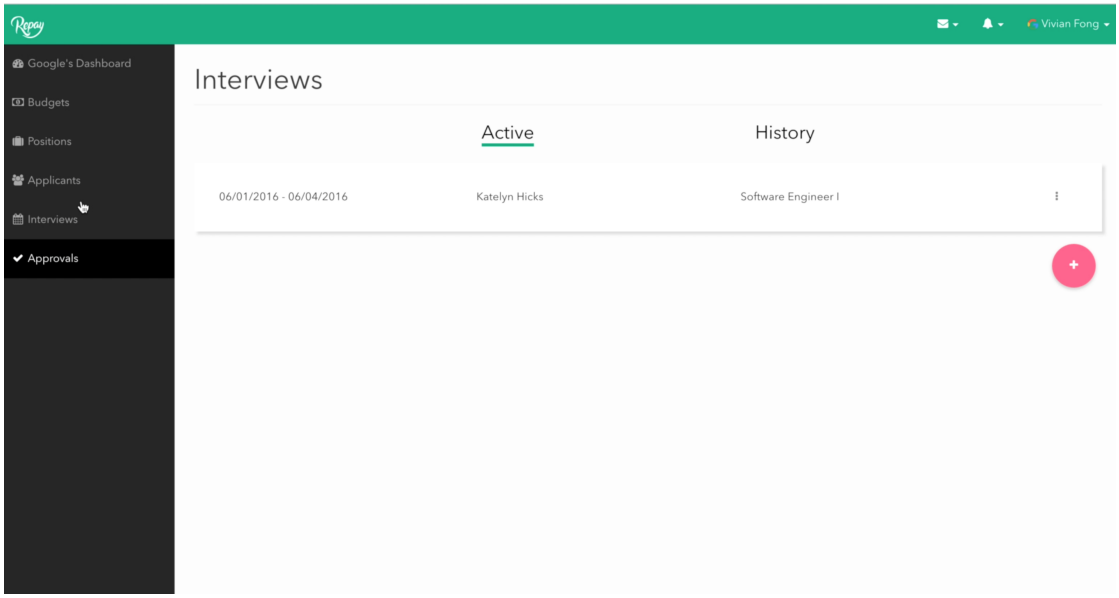

Figure 3: The "Interviews" page.

If Vivian navigates to the "Budgets" page, she can view and set budgets for each respective category. The default categories for budgets set by Repay are "Food", "Lodging", and "Transportation". As shown in Figure 4, the budgets for each of the listed categories are $50.00, $350.00, and $100.00, making a total of $500.00 available for Esha to spend during her interview.

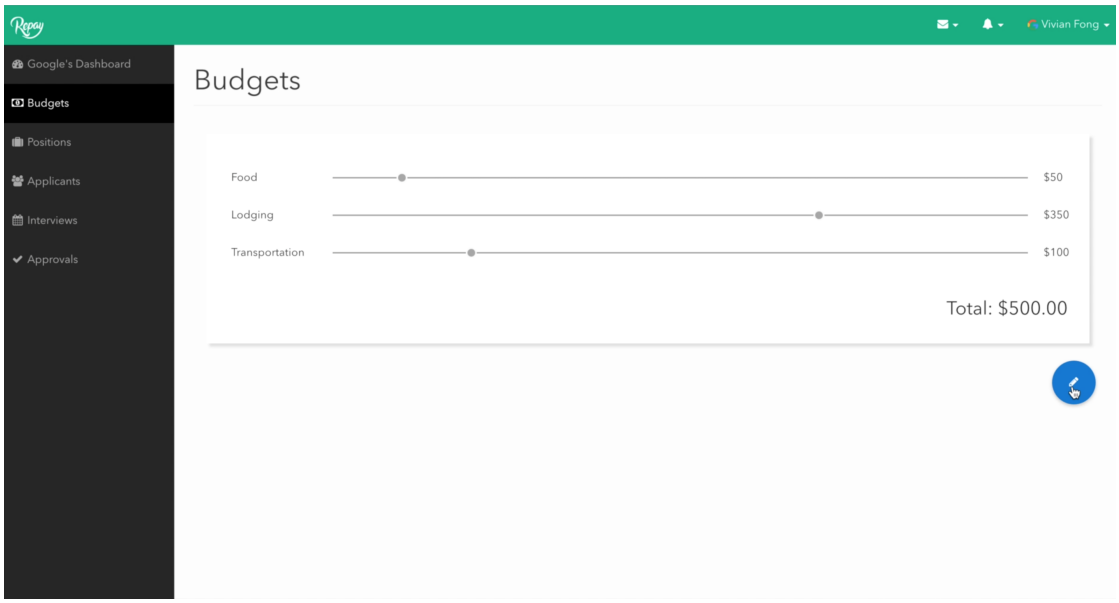

Figure 4: The "Budgets" page.

We wanted to give company HR representatives the opportunity to change, add, and set new budgets on an ad-hoc basis. If Vivian selects the blue pencil icon, she will be able to change the budgets (Figure 5). Vivian can set new budgets for any or all categories by manually entering new values or dragging the slider left and right to decrease and increase budgets. Vivian can set her new budgets by clicking the green "check-mark" icon, erase her changes to the budgets by clicking the gray x icon, or add a new budget category by clicking the pink plus icon.

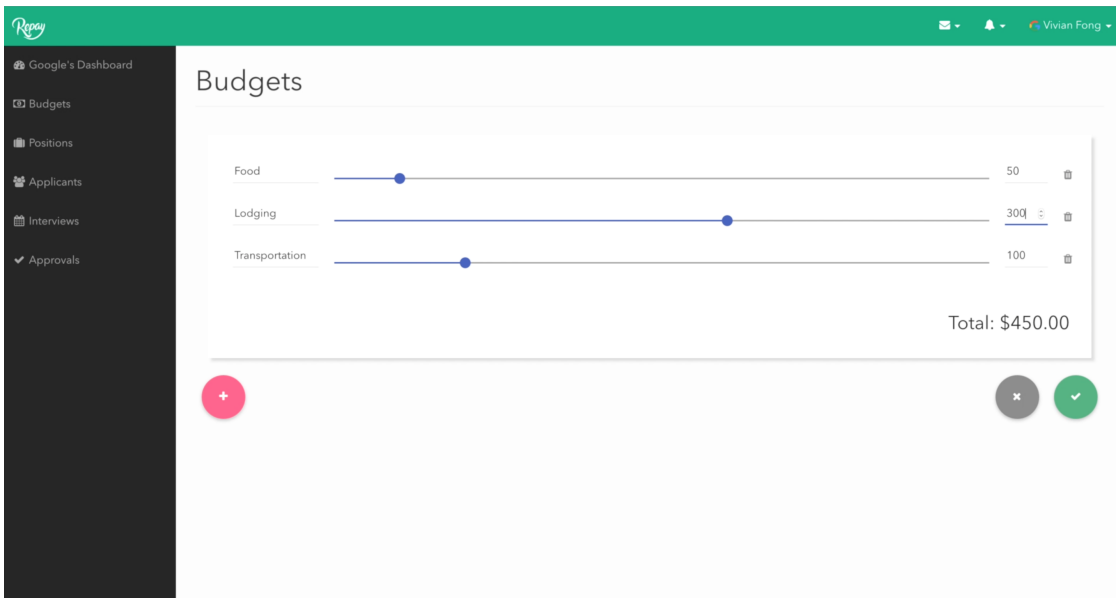

Figure 5: Editing the budgets.

2.3.2 Viewing & Adding Applicants

If Vivian navigates to the "Applicants" page, she can view and add new applicants. Figure 6 shows names, emails, and phone numbers of three existing applicants: Colin Adams, Nupur Garg, and Katelyn Hicks.

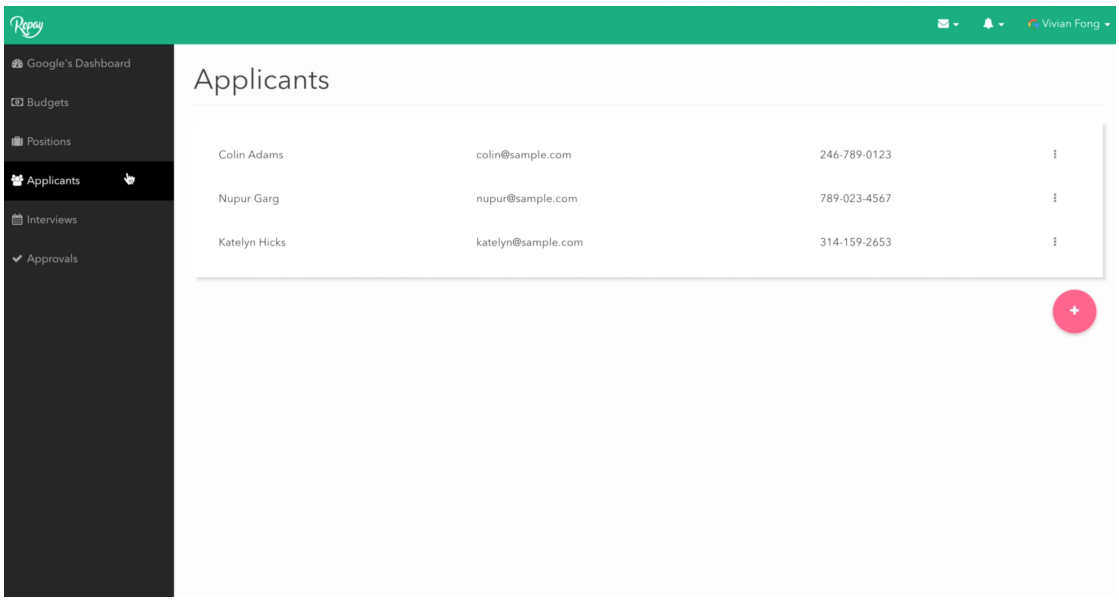

Figure 6: The "Applicants" page.

We wanted to give company HR representatives the opportunity to add new applicants very easily. If Vivian selects the pink "plus" icon, she will be able to seamlessly add a new applicant (Figure 7). She can add a first name, last name, email, and phone number. The "Send Invitation Email" checkbox is important because it auto-generates an email to send to Esha with appropriate login information for the mobile Repay application. Once Vivian has entered the necessary information, she can select the "CONFIRM" button to save all changes or the CANCEL button to cancel all changes.

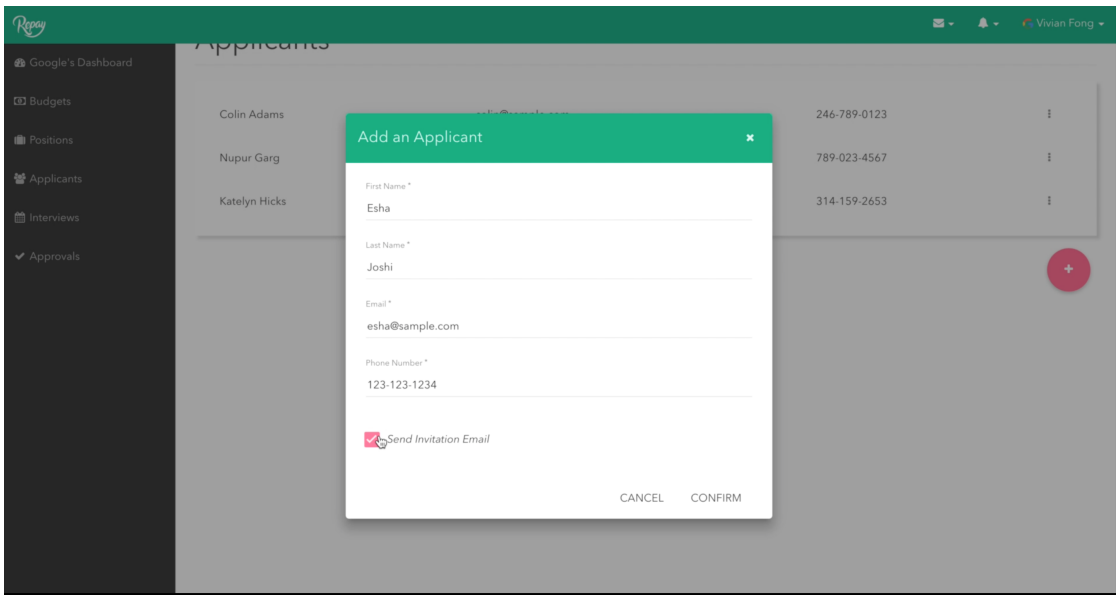

Figure 7: Adding a new applicant.

2.3.3 Adding New Interviews

When navigating back to the "Interviews" page, company HR representatives can add a new interview by selecting the pink "plus" icon (Figure 8).

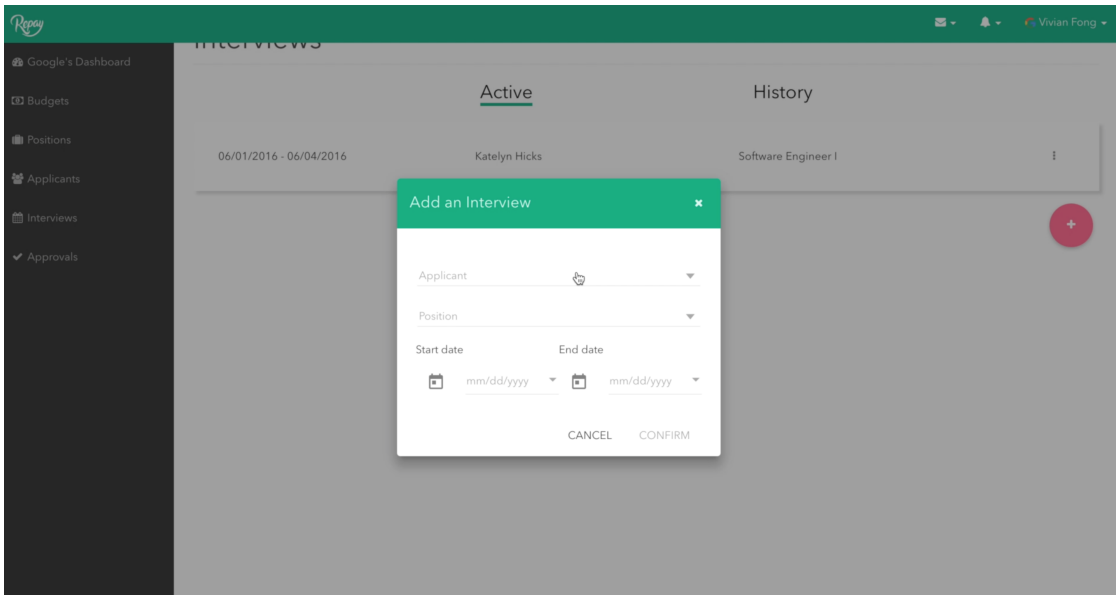

Figure 8: Adding a new interview.

In this scenario, Vivian can add an interview for Esha by inputting and selecting the appropriate content in the "Add an Interview" dialog. As shown in Figures 9, 10, and 11, Vivian can specify the applicant, position, and start and end dates by choosing the dynamically populated content from the available drop-down menus.

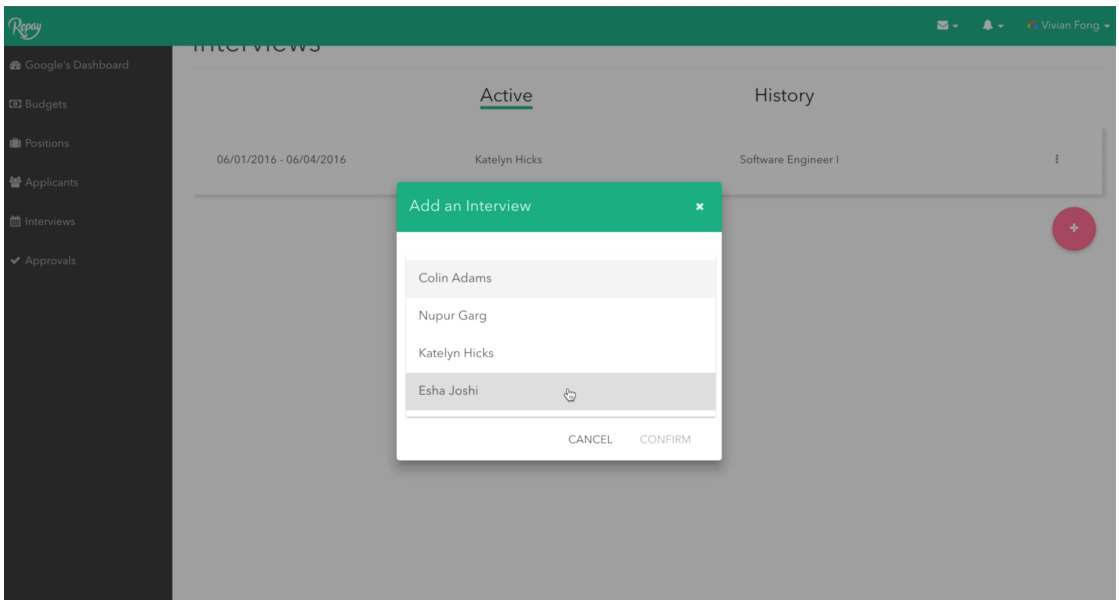

Figure 9: Specifying the interviewee.

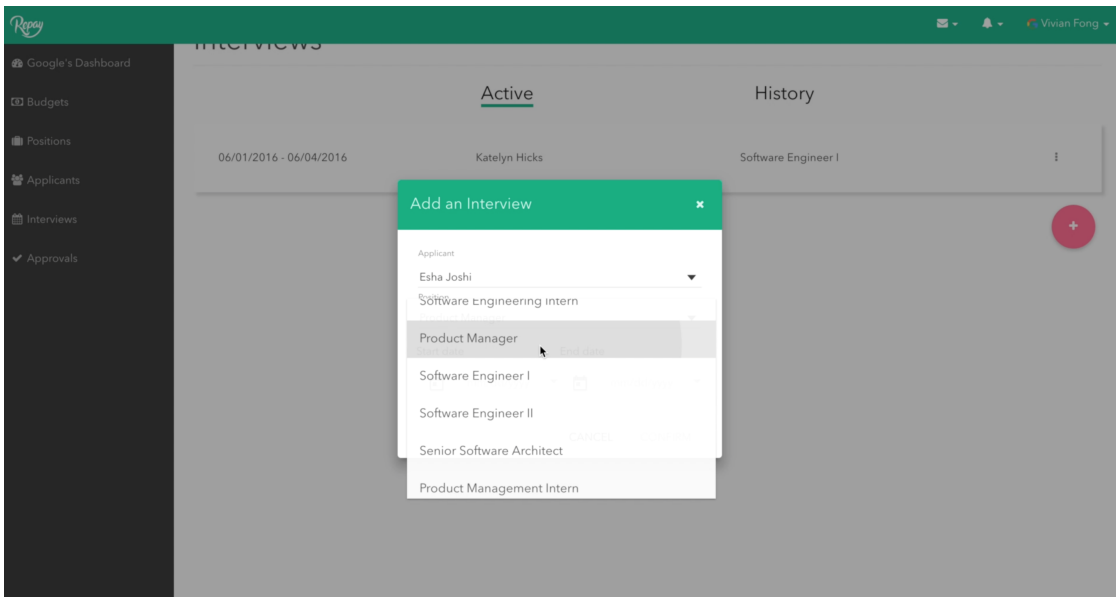

Figure 10: Specifying the position.

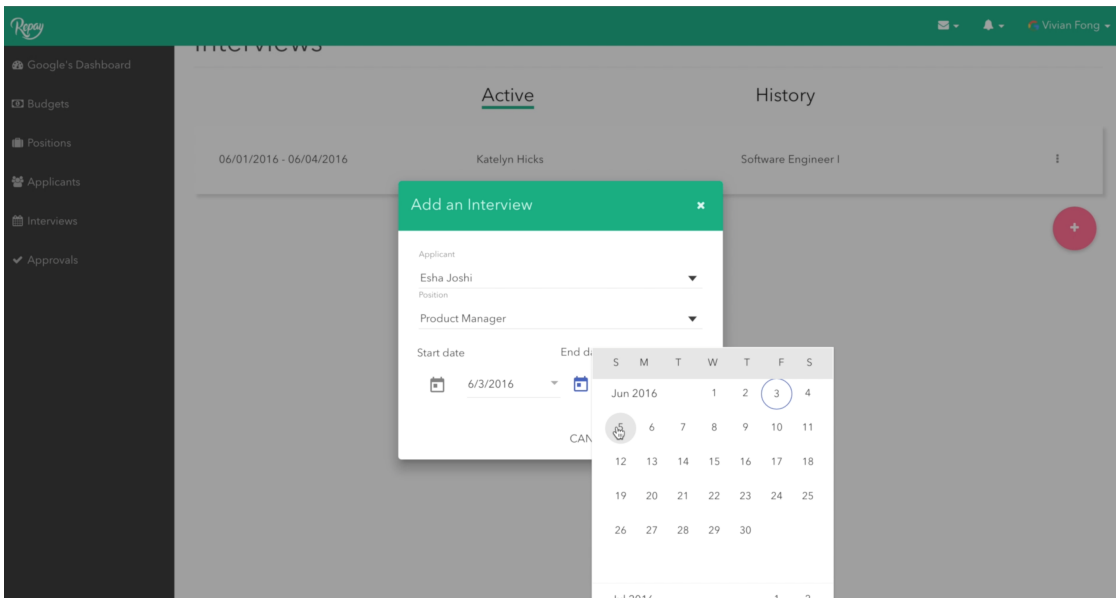

Figure 11: Specifying start and end dates.

As seen above in Figure 11, Vivian created a product manager interview for applicant Esha Joshi from June 3, 2016 - June 5, 2016.

2.4 Mobile Application – Logging In

When Esha downloads and launches the Repay mobile application for the first time, she is greeted with an initial signup screen as seen in Figure 12. The signup screen contains textfields for “email and “temporary password. Esha can select the “Sign Up button when she is ready to sign in or the “Log In button if she already has an account with Repay. Esha can view Repays privacy policy and other information if she wants to.

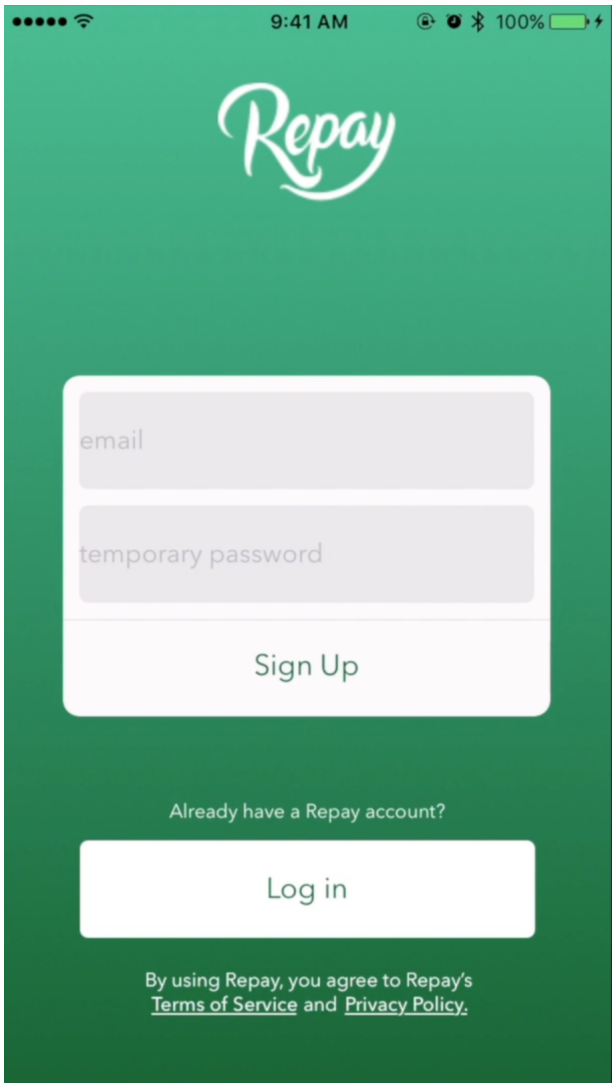

Figure 12: The login screen for the mobile app.

As seen in Figure 13, Esha entered the email and temporary password information listed in the Repay auto-generated email she received when she was added as a new applicant to the system. My team and I decided to design it this way because we wanted to simplify the signup/login process as much as possible. We want interviewees only to use the mobile application whenever they have an interview or want to refer back to previous interviews' reimbursements.

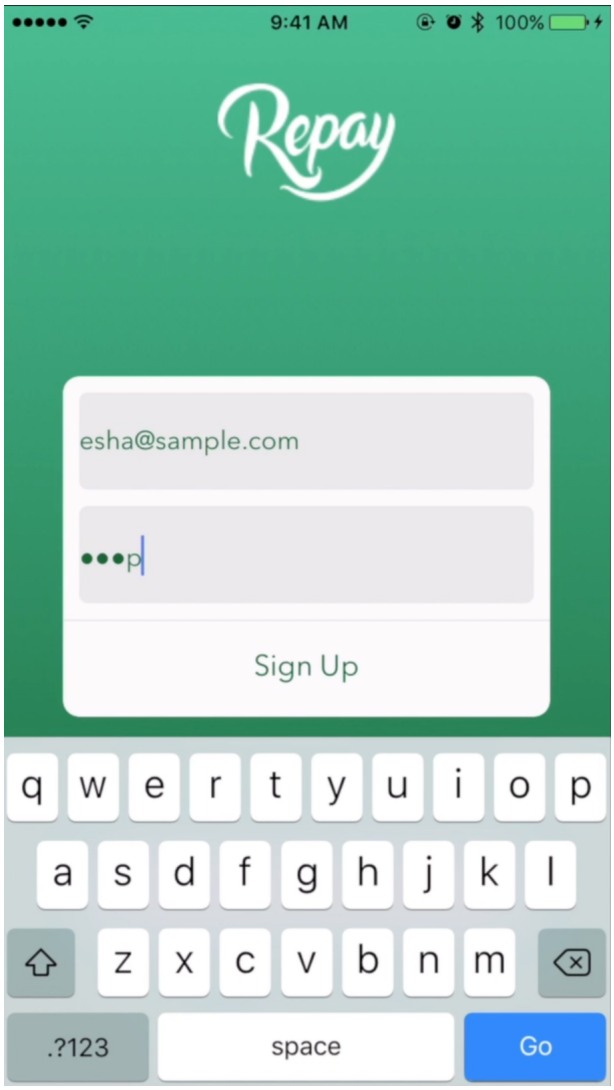

Figure 13: Entering login information.

Upon clicking "Sign Up", Esha will be prompted to enter a new password (Figure 14).

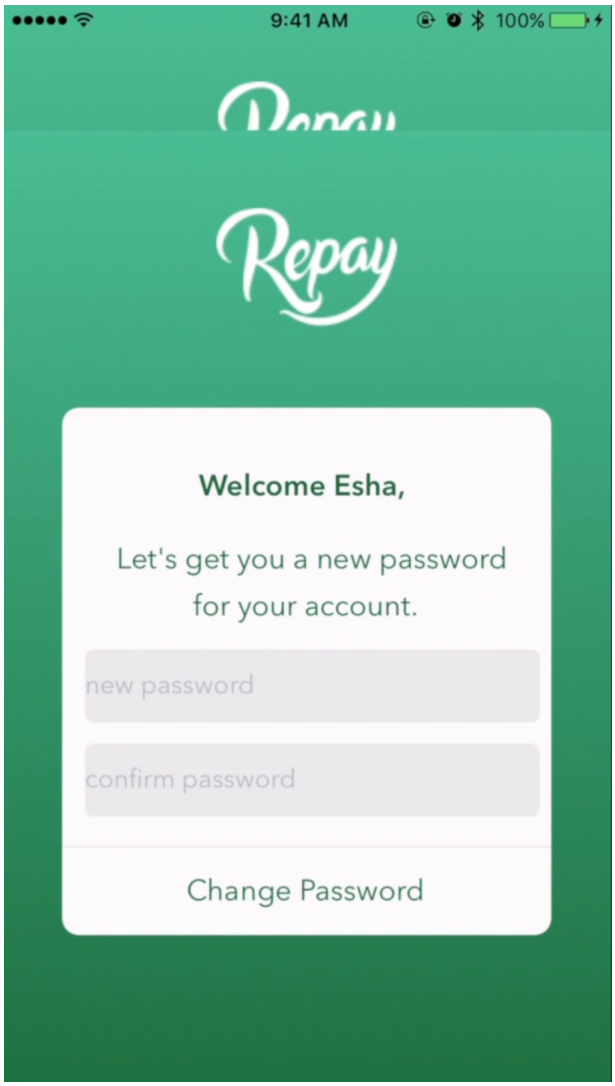

Figure 14: The screen for changing the user's password.

2.5 Mobile Application – Uploading Receipts

One of major components of the mobile application is the process of instantly uploading receipts and sending this information to the company. Undoubtedly this is the crutch of our entire mobile platform because this is what streamlines the reimbursement

process. With this feature, company representatives may start approving, denying, and reimbursing the interviewee for any expenditures while the interview is still going on.

2.5.1 Home Page

The Home Page of the mobile application, which can be seen in Figure 15, is a very handy view that aggregates the information pertaining to a single interview. The Home Page displays the name of the company and the remaining budgets interviewees have left to spend for the interview. When designing the Home Page, we wanted this to be a simple user interface, just denoting to interviewees the dynamically loaded amounts they have left to spend in each category. This easily accessible information could also help curb interviewees' impulsive spending habits by letting them know how much they have left to spend at any given time.

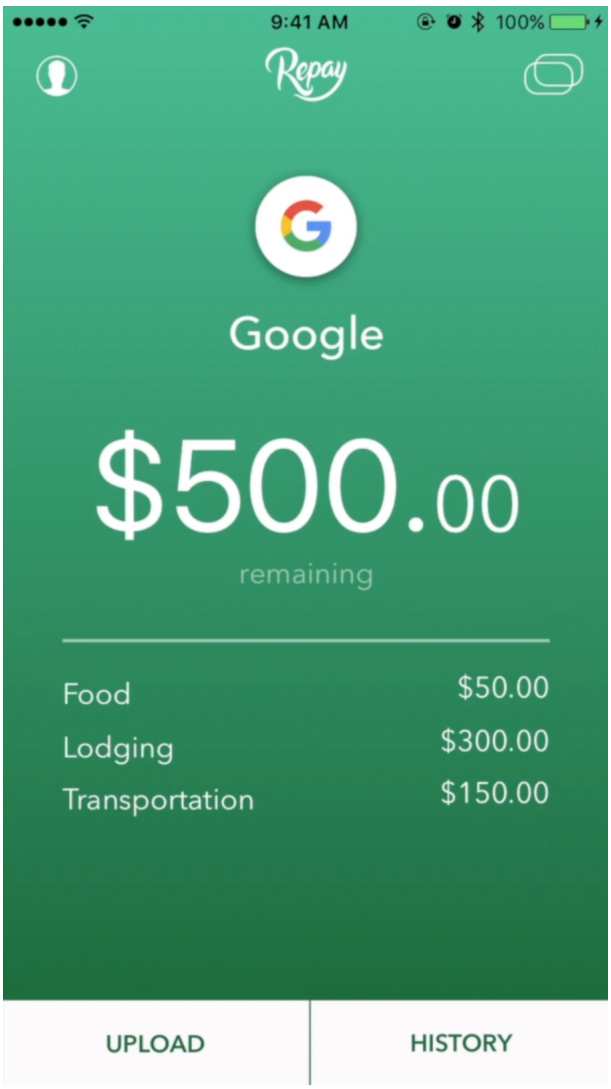

Figure 15: The home screen for the mobile app.

As seen in the previous figure, after signing into the application for the first time, Esha can view the remaining budgets for her Google interview: $50.00 for food, $300.00 for lodging, and $150.00 for transportation. She has an upper limit of $500.00 to spend during her interview from June 3, 2016 - June 5, 2016.

2.5.2 Changing Companies

We also give the interviewee the option to switch between companies and view other interviews' budgets by clicking the overlapping-ovals symbol in the top-right corner of the Home Page.

In our scenario, Esha is an applicant for one company, Google. As shown in the Figure 16, she cannot switch between any other interviews.

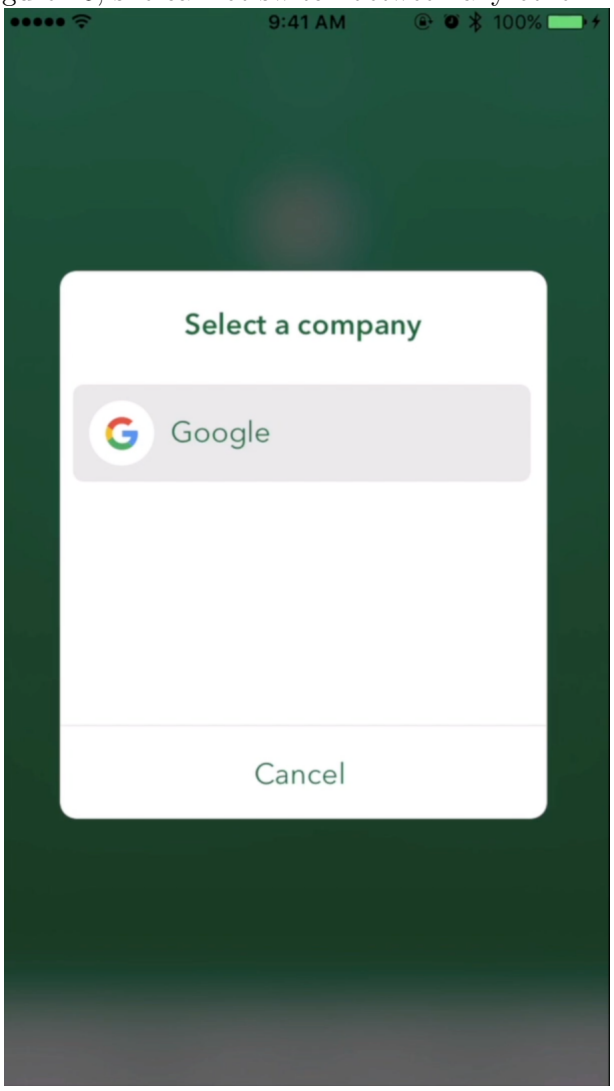

Figure 16: Modal dialog for changing companies.

2.5.3 Selecting a Category

We give interviewees the opportunity to specifically upload a receipt to one of the listed categories (specified by the company). As shown in Figure 17, interviewees can view the categories they may upload receipts to as well as the remaining amounts for each category.

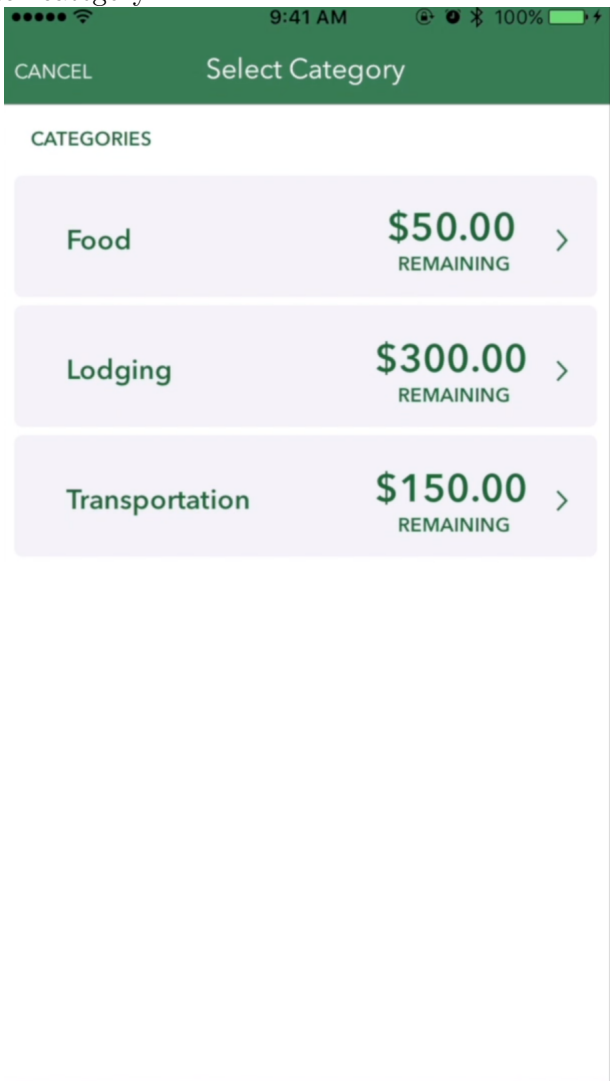

Figure 17: Selecting a category for a reimbursement.

2.5.4 Inputting Reimbursement Amounts

Upon selecting a category, interviewees may select which banking account they would like to deposit the money to and how much they wish to be reimbursed for. The process also enforces a photo of the receipt proving that transaction happened (workflow explained in following sections) (Figure 18).

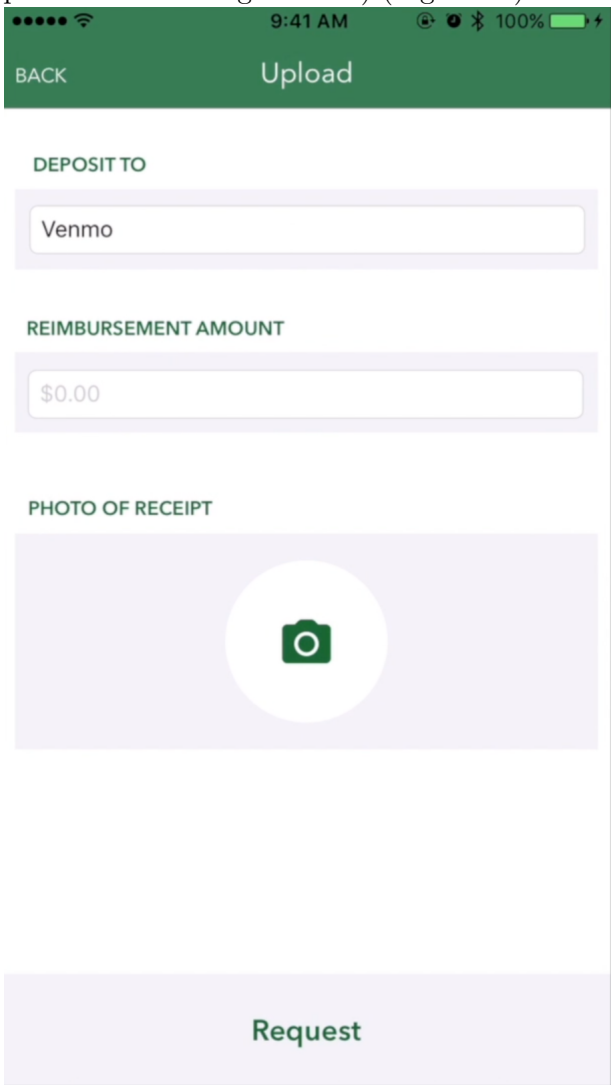

Figure 18: Creating a reimbursement request.

Suppose Esha wanted to upload a receipt in the "Food Category" for a $3.95 coffee she bought at Blackhorse Cafe on Foothill Boulevard in San Luis Obispo, CA. She would first select her preferred banking option from a drop-down menu ("Venmo" shown in Figure 19) and the amount of $3.95 to be reimbursed. When she is entering an amount in the "Reimbursement Amount" textfield, the keyboard pops up from the bottom of the mobile device's screen.

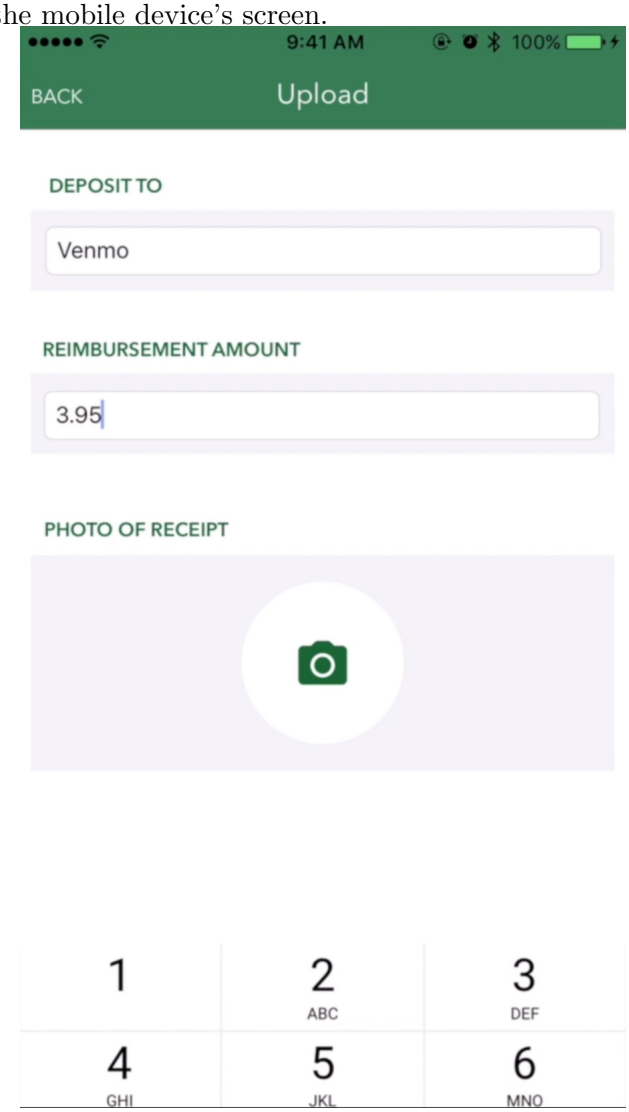

Figure 19: Requesting a reimbursement $3.95.

2.5.5 Uploading Picture of Receipt

In order to be properly reimbursed for her coffee, Esha will need to take a picture of her receipt. By clicking the green camera icon mid-way down the screen, she can activate the iOS Camera application and take a picture of her receipt to prove that the transaction actually took place (Figure 20). Esha may now select "Request" at the bottom of the screen to finalize the reimbursement request.

Figure 20: Taking a picture of the receipt for $3.95.

2.5.6 Confirming Reimbursement

Once the reimbursement request has gone through, interviewees will be presented with a screen confirming their requested category and amount. As seen in Figure 21, since Esha submitted a receipt to be reimbursed for $3.95, she is presented with a screen saying, "Your FOOD request for $3.95 is under review". She can select the "Yay!" button at the bottom of the screen to navigate back to the Home Page.

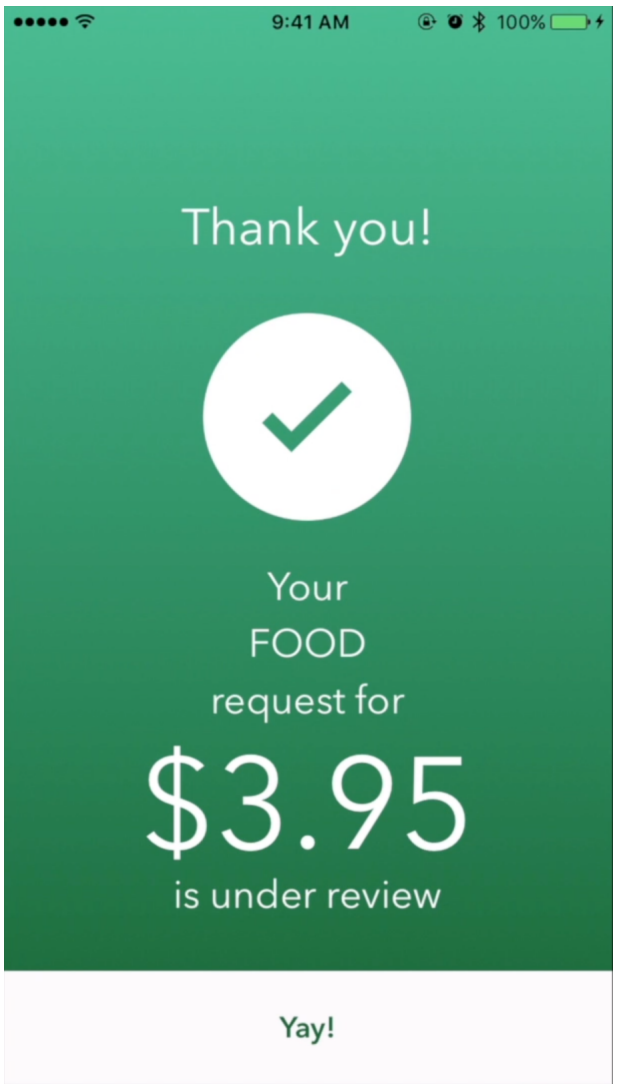

Figure 21: Confirm view for reimbursement of $3.95.

2.5.7 Viewing Updated Budgets

As discussed earlier, one major advantage of the mobile application is the ability to see the interviewee's remaining budgets very easily. As shown in Figure 22, Esha's remaining budgets for her Google interview have been updated appropriately.

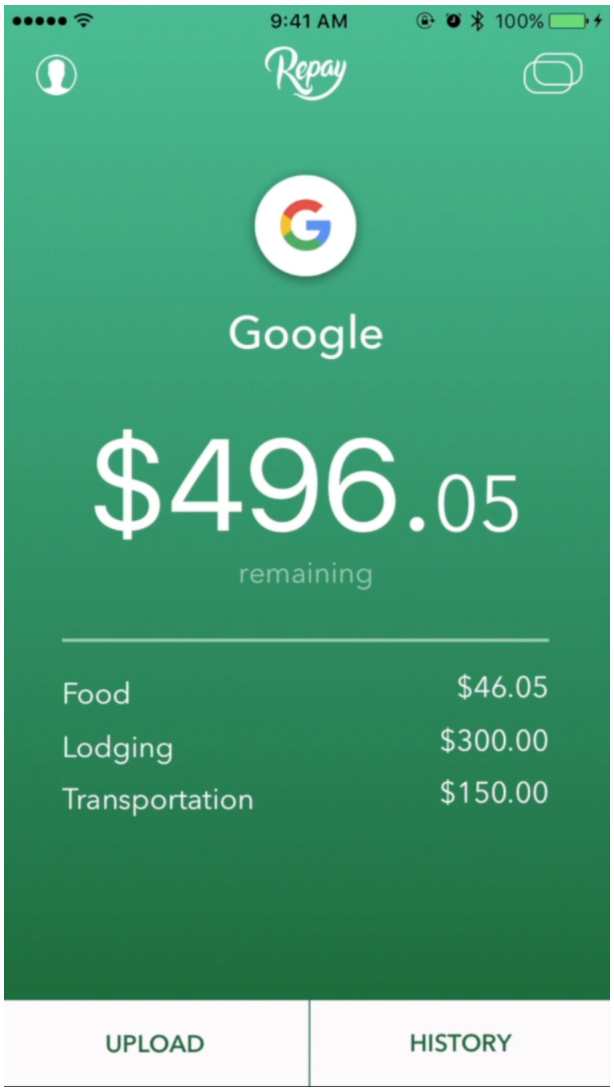

Figure 22: Home page after request reimbursement of $3.95.

2.5.8 Additional Reimbursement Workflows

The following screenshots visually show two workflows for requesting reimbursements for food and transportation transactions:

The first reimbursement workflow depicts Esha's food request for $18.36 from Bevmo. (Figure 23). The flow starts at the upper-left-most view, moving in a clock-wise fashion.

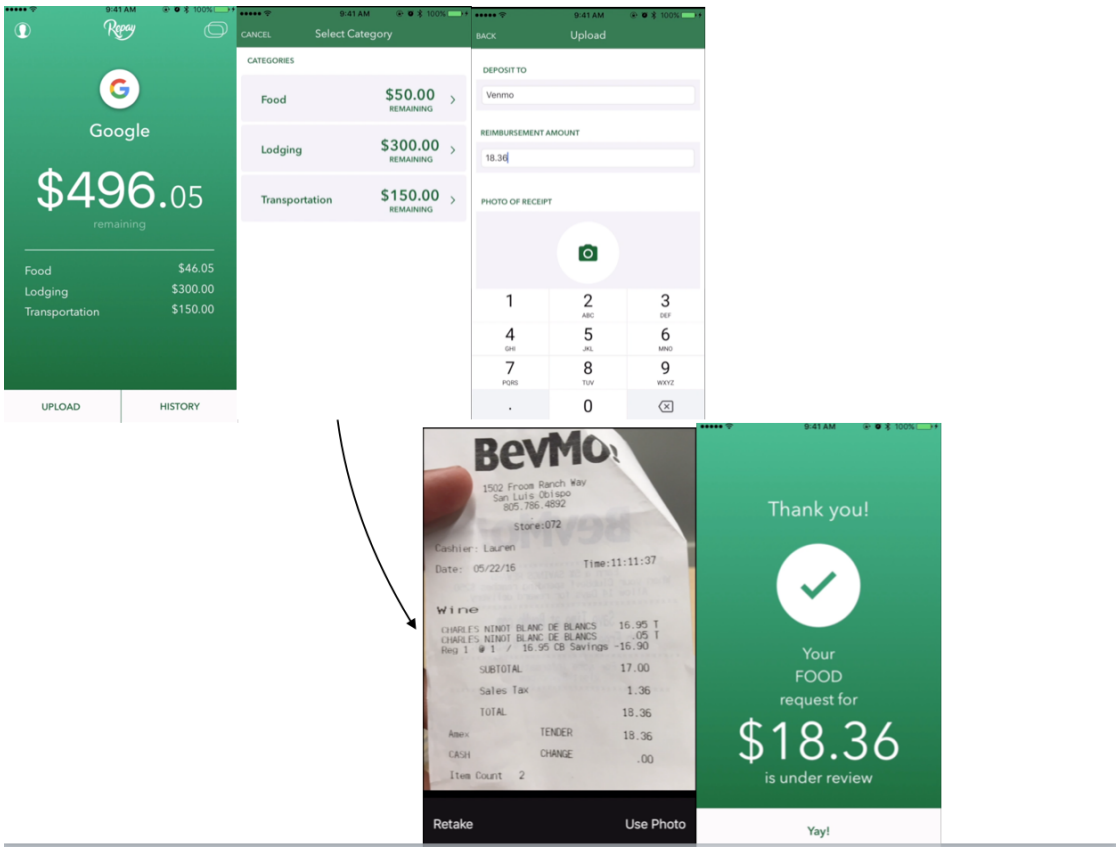

Figure 23: Workflow for Bevmo request.

The second reimbursement workflow depicts Esha's transportation request for $44.11 from Chevron Gas (Figure 24). The flow starts at the upper-left-most view, moving in a clock-wise fashion.

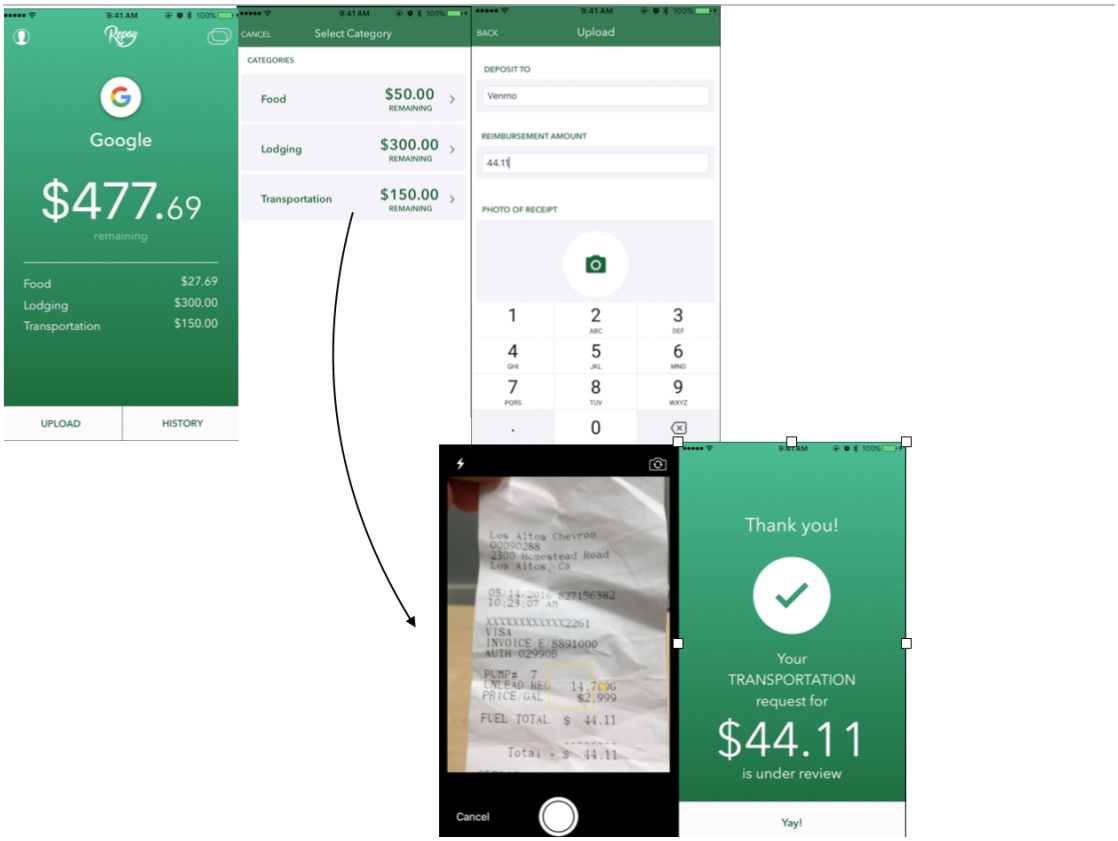

Figure 24: Workflow for Gas request.

2.6 Web Application – Approving/Flagging Receipts

When navigating to the "Approvals" page in the Repay web application, company HR representatives can view all pending and flagged receipts submitted by each interviewee (Figure 25). Depending on the receipts they receive and the amount interviewees seek to be reimbursed, they can either accept reimbursements by selecting the green "Accept" button or flag them by selecting the red "Flag" button.

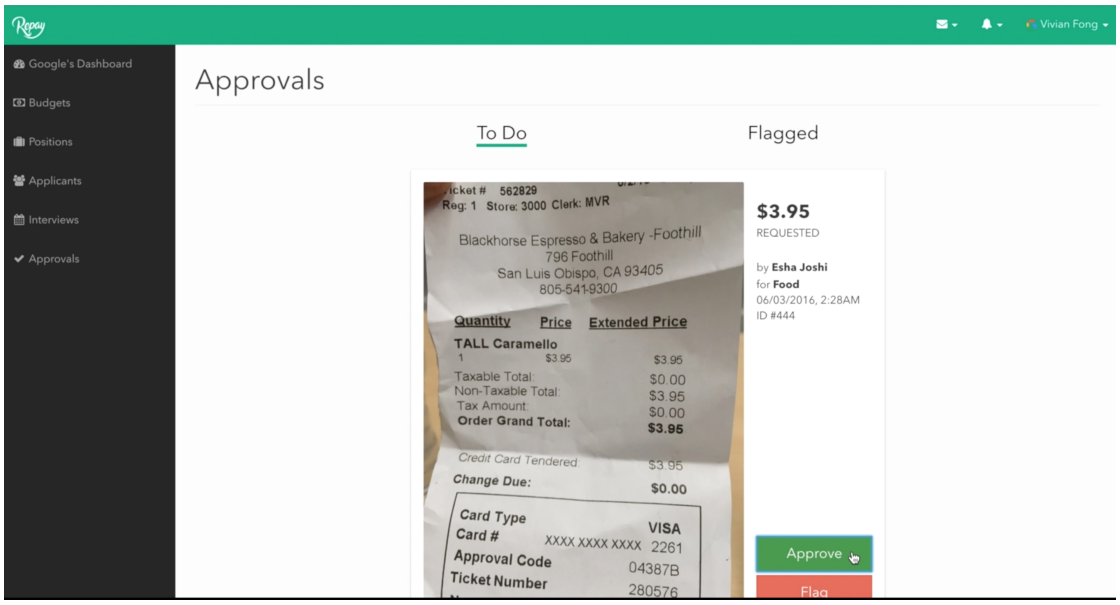

Figure 25: The "Approvals" page – approving a receipt.

2.6.1 Flagging Receipts

Google does not support reimbursing alcoholic expenditures. Vivian can choose to flag this receipt (Figure 26).

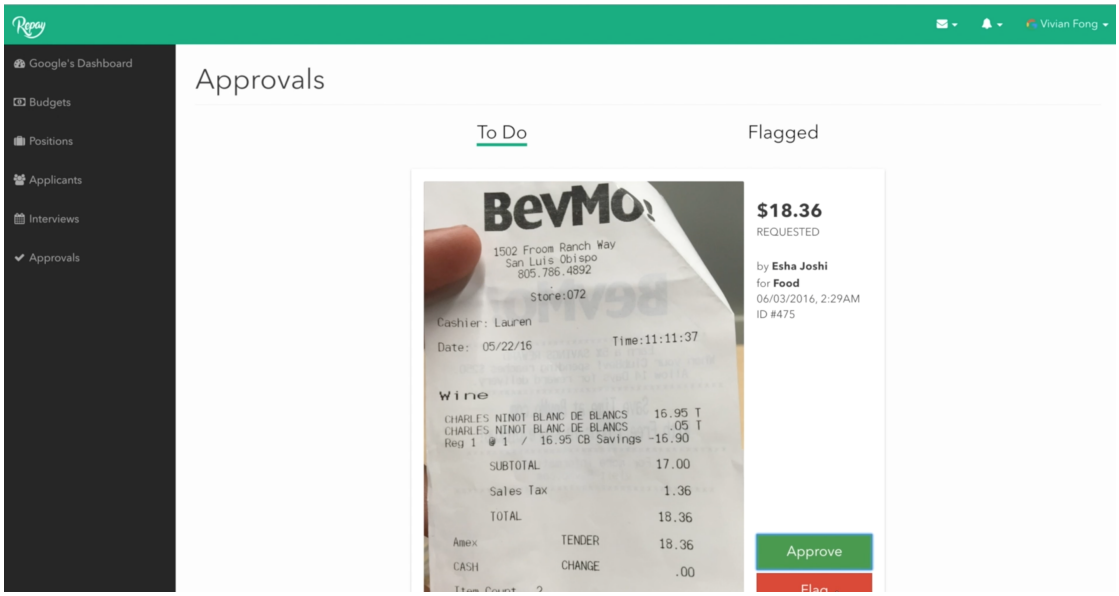

Figure 26: The "Approvals" page – flagging a receipt.

Upon selecting the red "Flag" button, Vivian will be prompted with a "Flag Receipt" modal dialog, where she can input the amount Google will actually reimburse and a reason why (Figure 27). She can then select the "Confirm" button to save her changes.

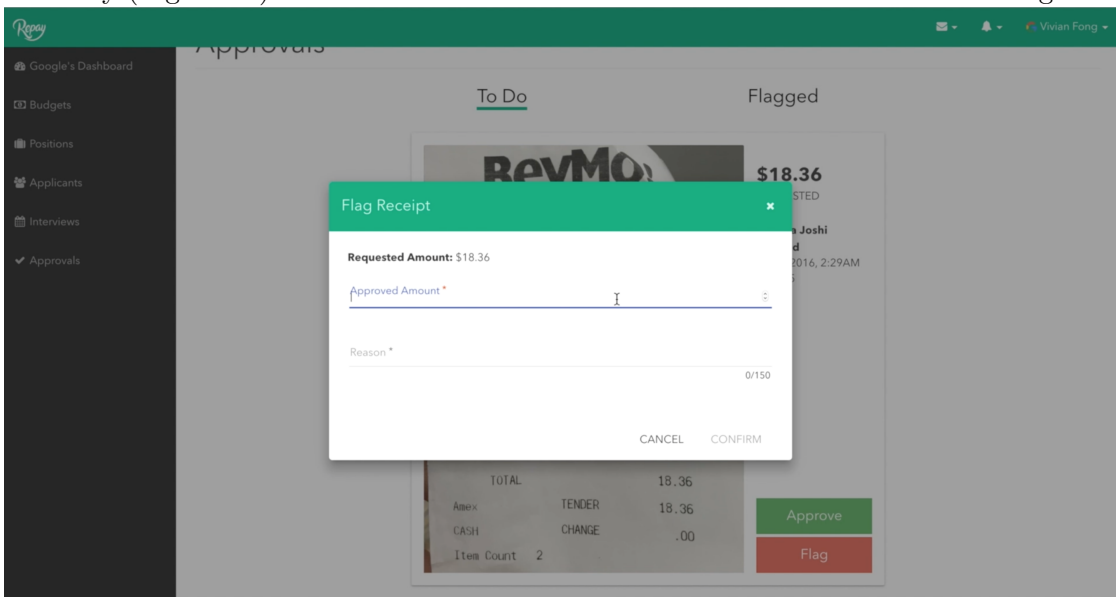

Figure 27: The modal for flagging a receipt.

As shown in Figure 28, Vivian chose to reimburse $0.00 for this alcoholic expenditure with the reason, "No alcohol is allowed".

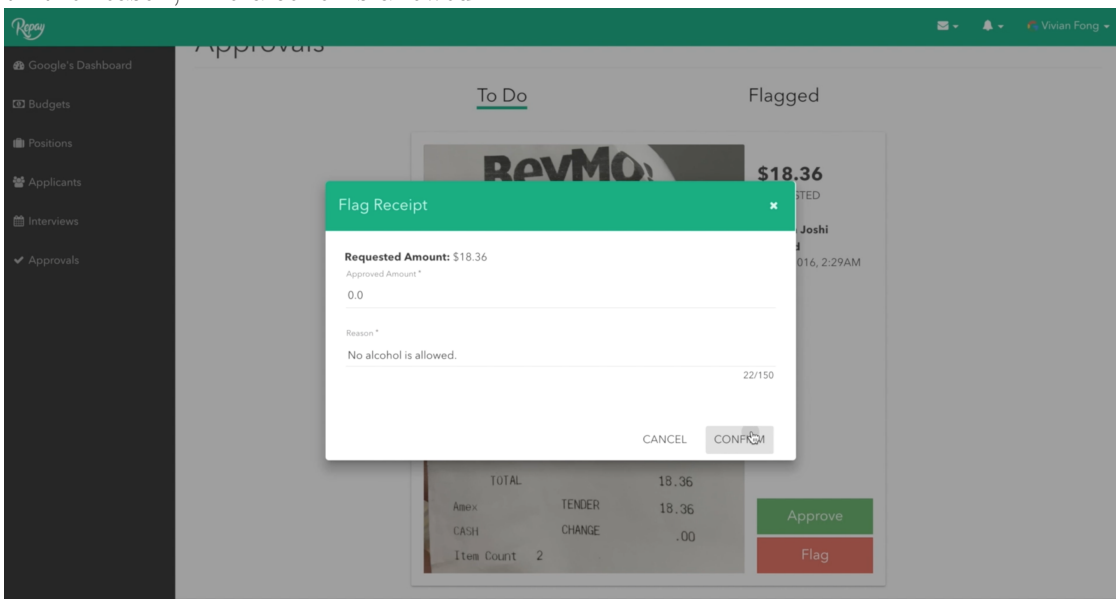

Figure 28: Flagging a receipt with the amount and reason.

2.6.2 Viewing Flagged Approvals

Vivian can view her flagged receipts by selecting the "Flagged" tab on the "Approvals" page. As shown in Figure 29, she can select the $18.36 expenditure submitted by Esha to view the amount she chose to reimburse Esha and why. Figure 30 shows the message from Vivian for this reimbursement. The message from Esha indicates that she disputed this reimbursement (explaining in following sections).

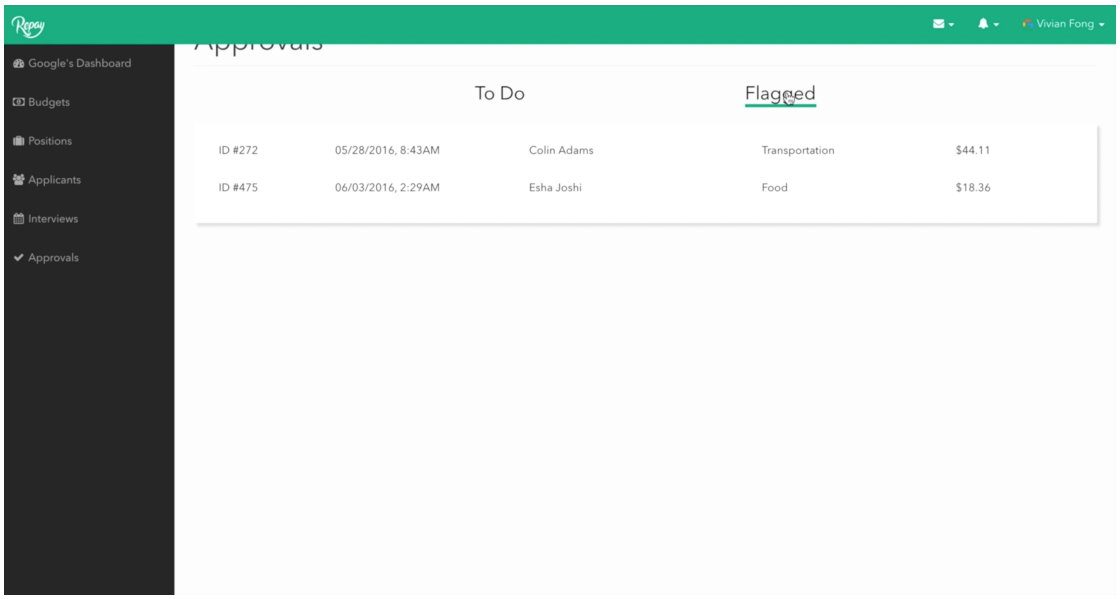

Figure 29: The "Approvals" page – flagged receipts.

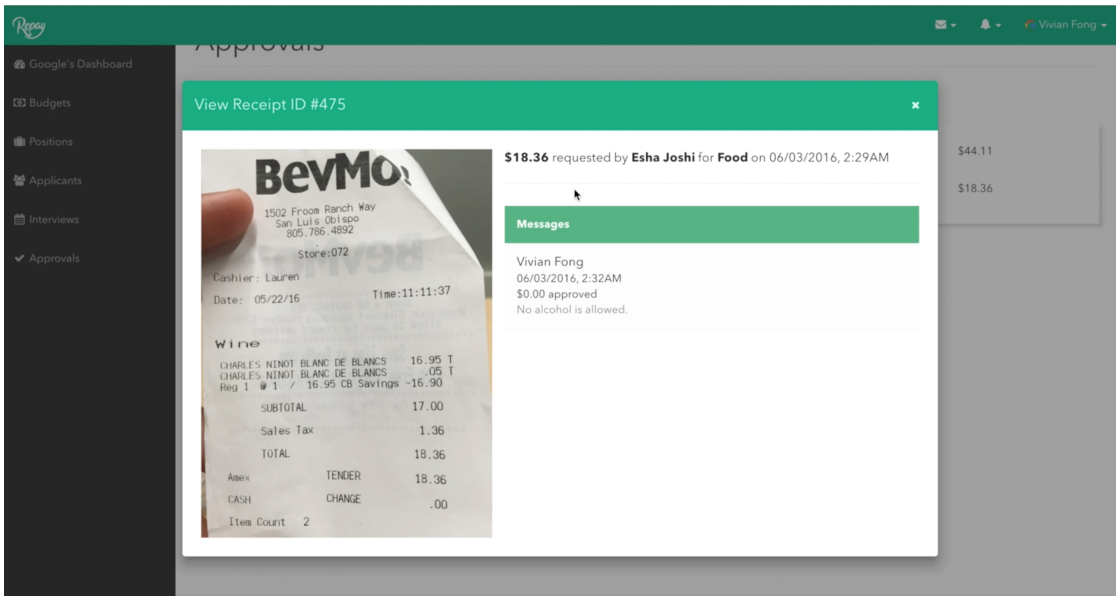

Figure 30: Viewing messages for a flagged reimbursement request.

2.7 Mobile Application – Viewing History

The second major component of the mobile application is the interviewee's ability to view all reimbursement requests for any given interview. Interviewees can load the history workflow by selecting the "History" tab at the bottom of the home page (Figure 31).

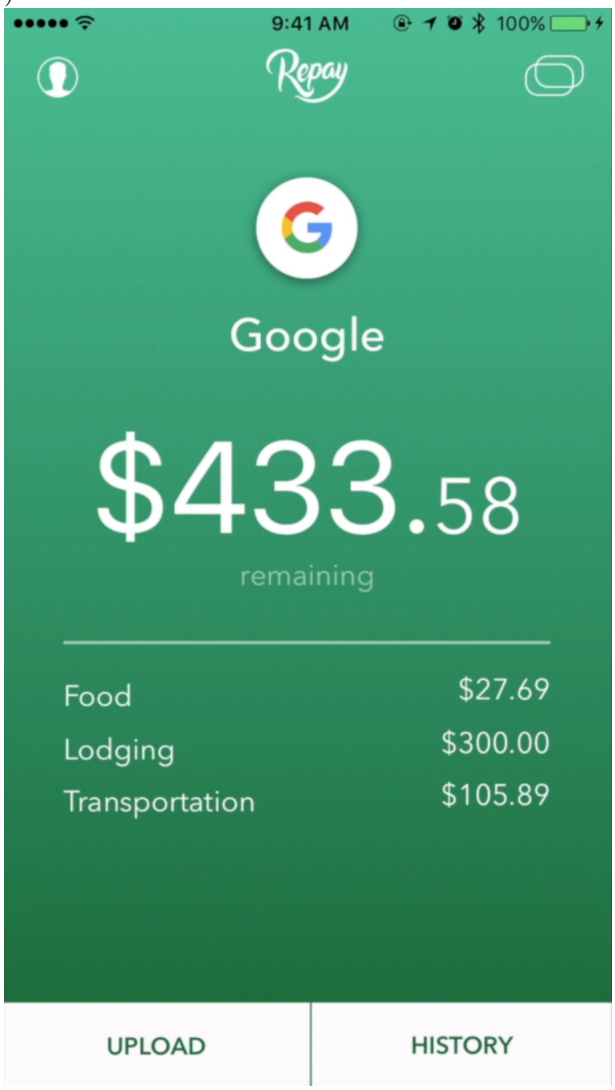

Figure 31: Home page of the mobile application (with updated budgets).

2.7.1 Viewing History

As shown in Figure 32, The reimbursement requests are visually shown as a segmented control of three segments: “Todo”, “Flagged”, and “Approved” reimbursement requests. The default control is the “Flagged” segment because it requires immediate action by the interviewee. If the interviewee does not have flagged reimbursements to address, the default control is the “Todo” segment.

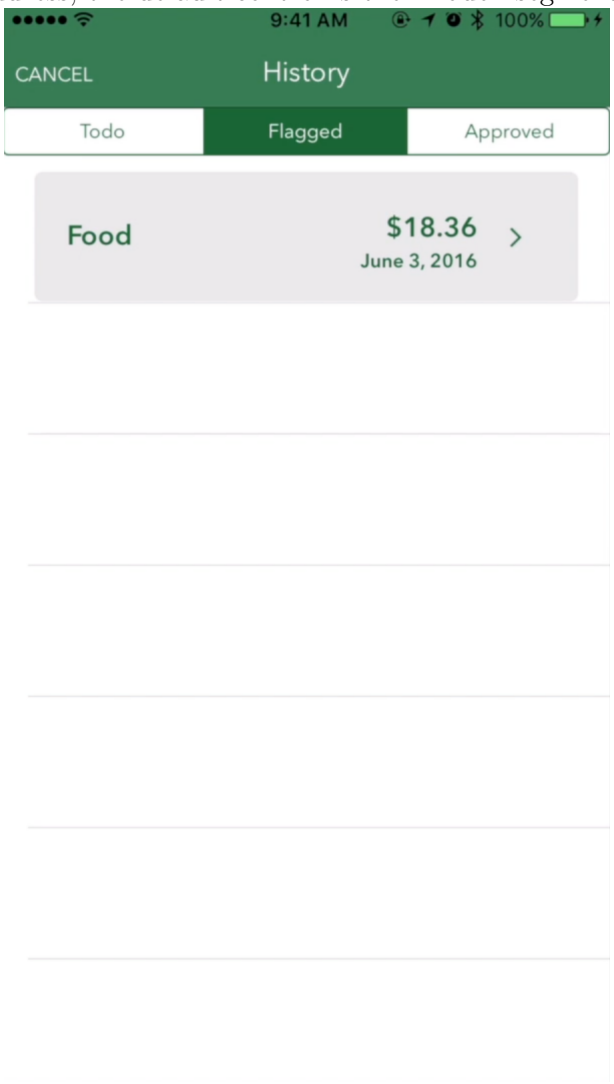

Figure 32: Viewing flagged reimbursements.

2.7.2 Viewing Approved Reimbursements

Interviewees can view any reimbursements in detail by selecting any table cell. For a table cell under the "Approved" segment (shown in Figure 33) the information would not be editable. The amount requested text field would be populated with what the interviewee requested and reason textfield would be populated with "N/A.

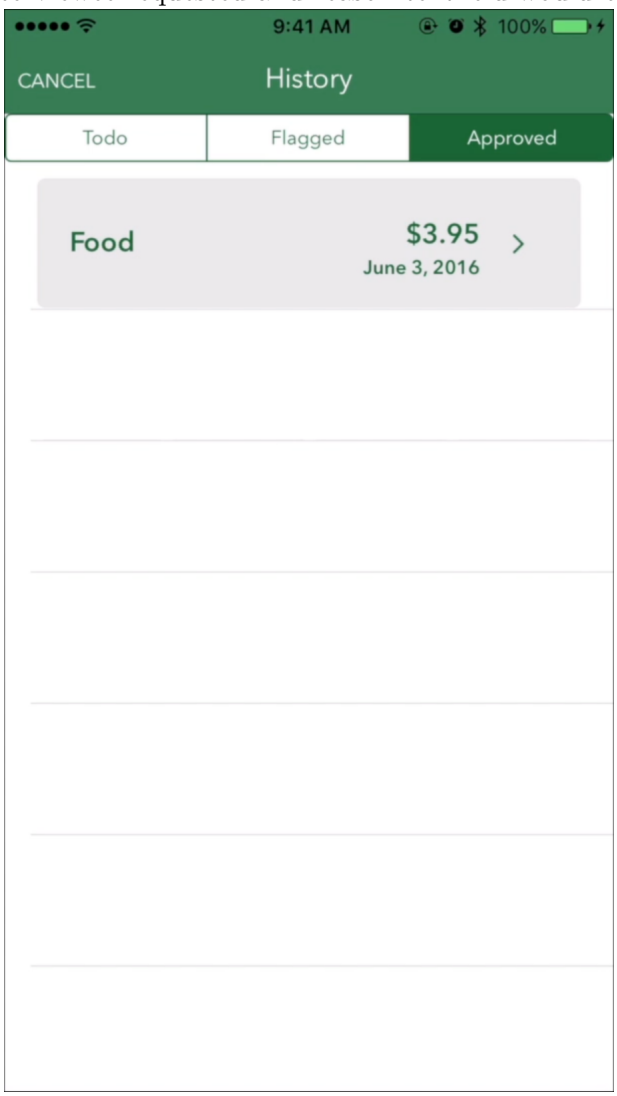

Figure 33: Viewing approved reimbursements.

2.7.3 Accepting/Disputing Reimbursements

Interviewees can view a specific flagged reimbursement in detail by selecting appropriate table cell under the "Flagged" segment control. As shown in Figure 34, interviewees can view how much money the company is willing to reimburse for that specific transaction and the reason why. Interviewees may dispute the reimbursement request by selecting "Dispute" at the bottom of the screen and inputting an amount they would like to be reimbursed and why (Figure 35).

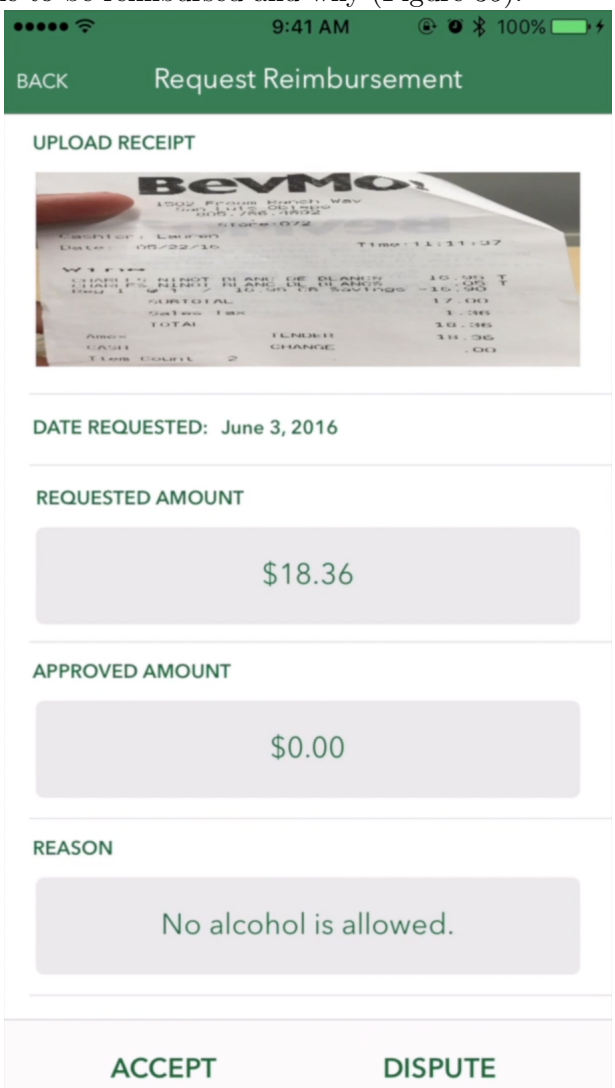

Figure 34: Viewing details of a flagged reimbursement.

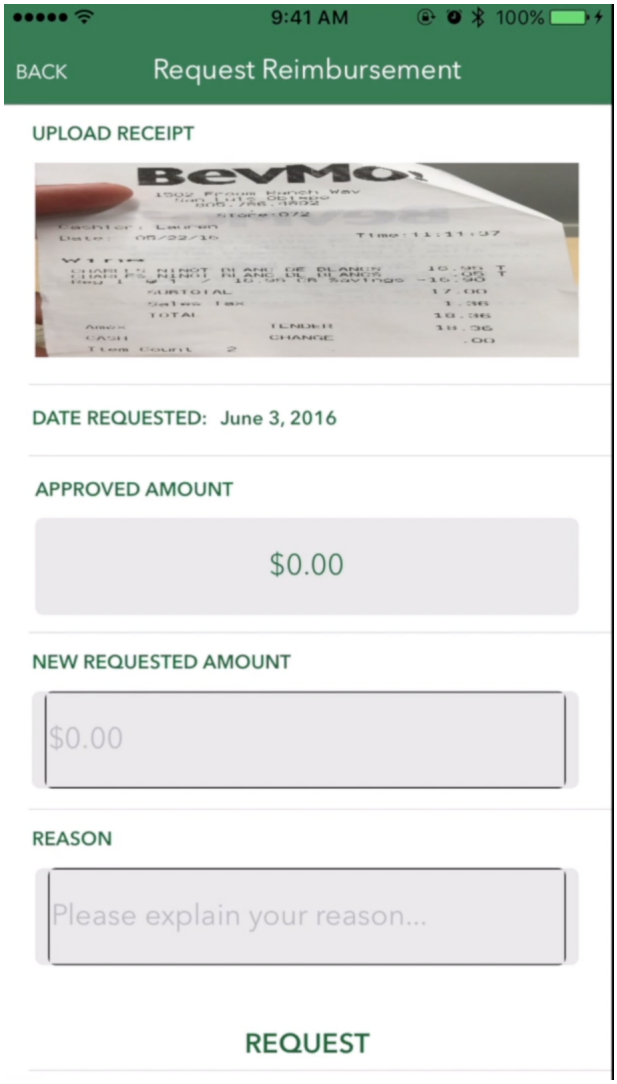

Figure 35: Screen for disputing a flagged reimbursement.

In the scenario, Vivian had chosen to reimburse $0.00 with the reason: "No alcohol is allowed." As shown in Figure 36, Esha decided to dispute this reimbursement and request back $18.36 with the reason: "Champagne isn't real alcohol!"

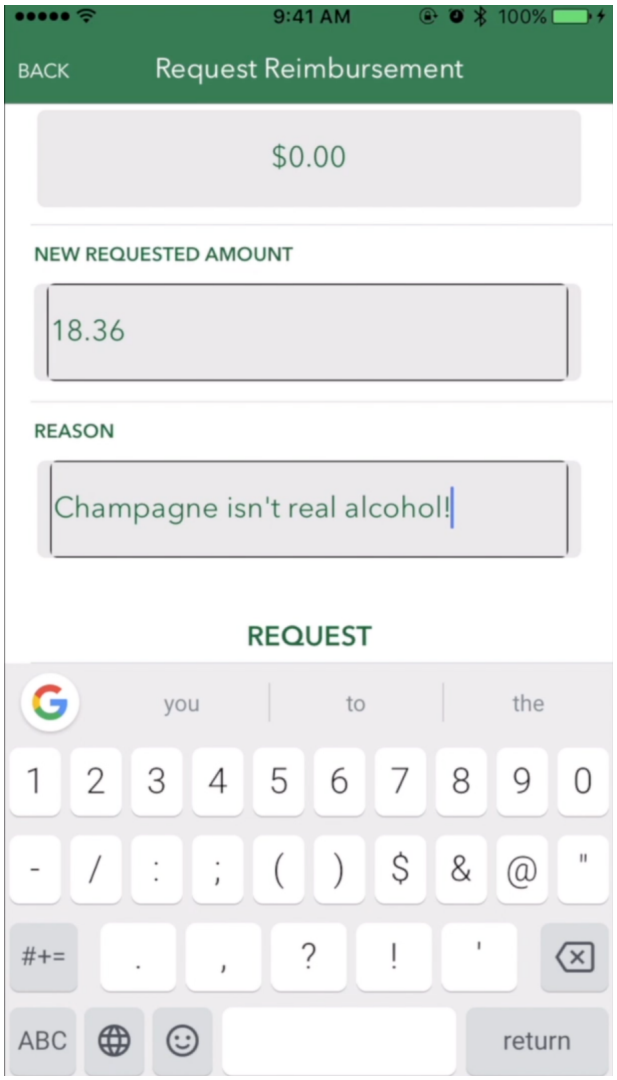

Figure 36: Disputing a flagged reimbursement.

2.8 Web Application – Disputed Reimbursements

When viewing approvals on the "Approvals" page, HR representatives can view disputed reimbursements if any exist. Using our scenario, Vivian can view Esha's disputed reimbursement (Figure 37). If she clicks the blue "Messages" button, she can view

Esha's reason for the dispute (Figure 38).

Figure 37: Viewing disputed reimbursements on the dashboard.

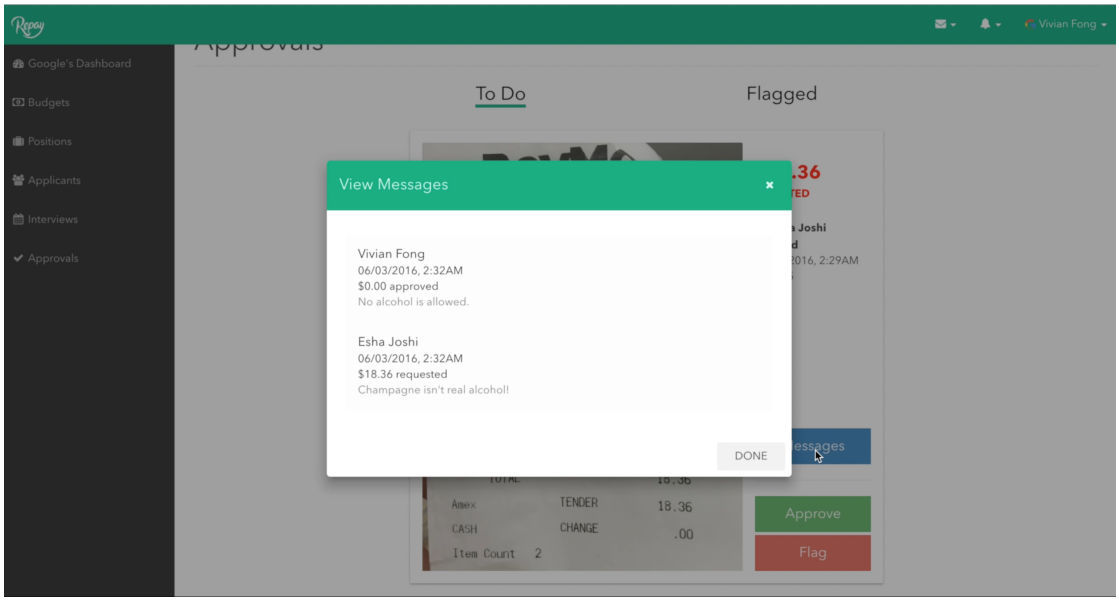

Figure 38: Viewing disputed reimbursements with messages.

Vivian can then re-flag the reimbursement, as shown in Figure 39, and again would be required to provide an amount to be reimbursed and a reason for the amount provided.

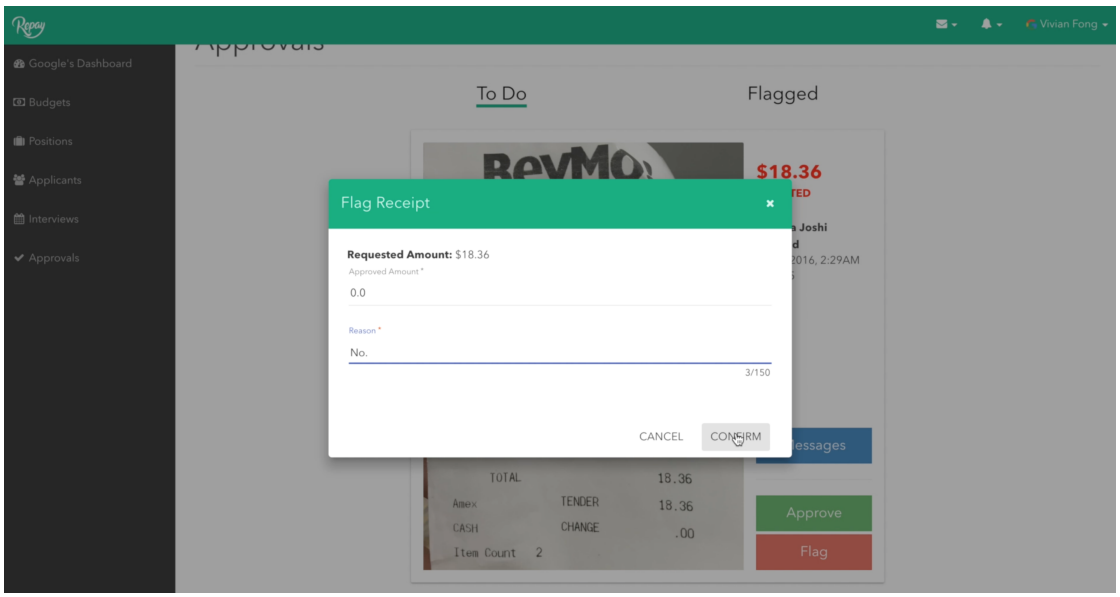

Figure 39: Re-flagging a reimbursement with an appropriate amount and reason.

2.9 Mobile Application – Disputed Reimbursements

Once again referring back to our scenario, Esha would see the re-flagged reimbursement with the associated amount and reason (Figure 40).

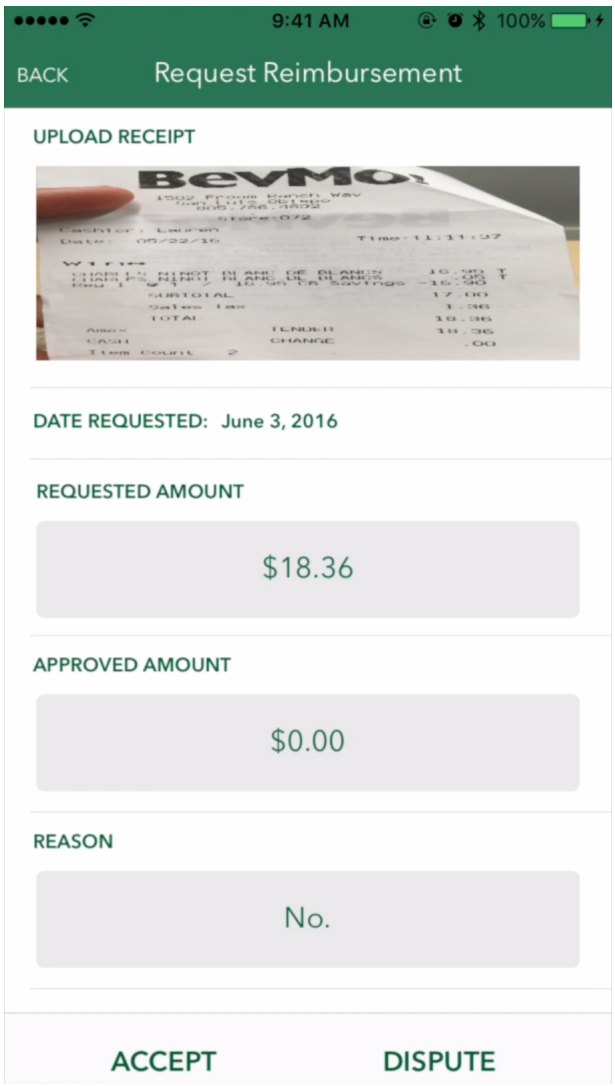

Figure 40: Viewing the flagged reimbursement with the new amount and reason.

She may then accept/deny this reimbursement. In this case, as shown in Figure 41, Esha accepted the reimbursement $0.00 was deposited in her Venmo account.

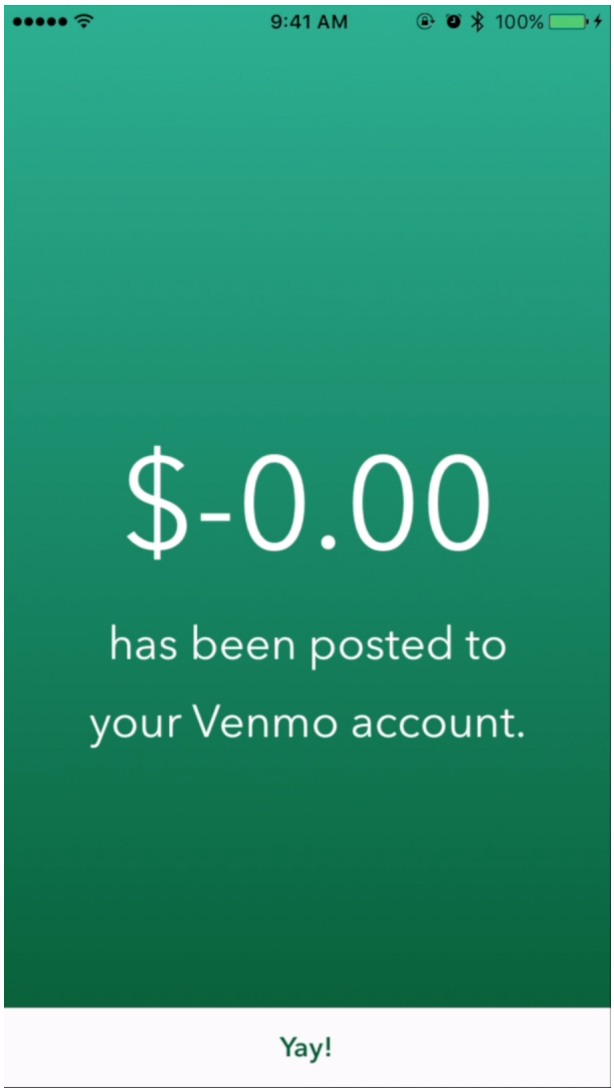

Figure 41: Screen for showing much money was deposited in interviewee's account.

3 Application Architecture

At its core, Repay is built as both an iOS application in Swift and web application in AngularJS, using Firebase as the real-time database. Both the mobile and web applications asynchronously read from and write to Firebase, a real-time database and backend as a service that conveniently displays data in JSON (JavaScript Object Notation) format.

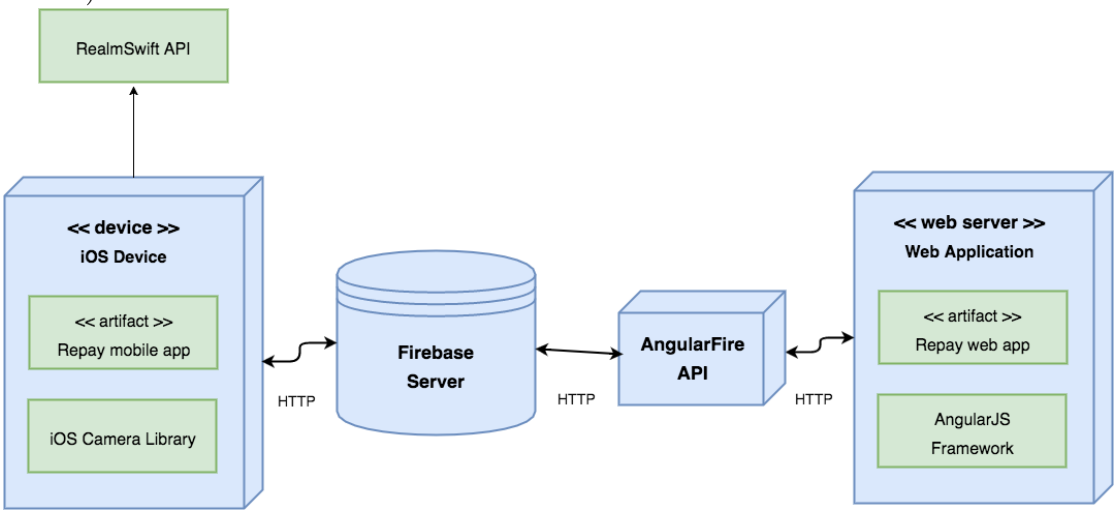

Figure 42: Deployment diagram for our system.

3.1 Mobile – Swift

The Repay mobile application is developed in Apple Swift version 2.2 using the Xcode interface. It utilizes the Model-View-Controller (MVC) design paradigm as shown in Figure 43.

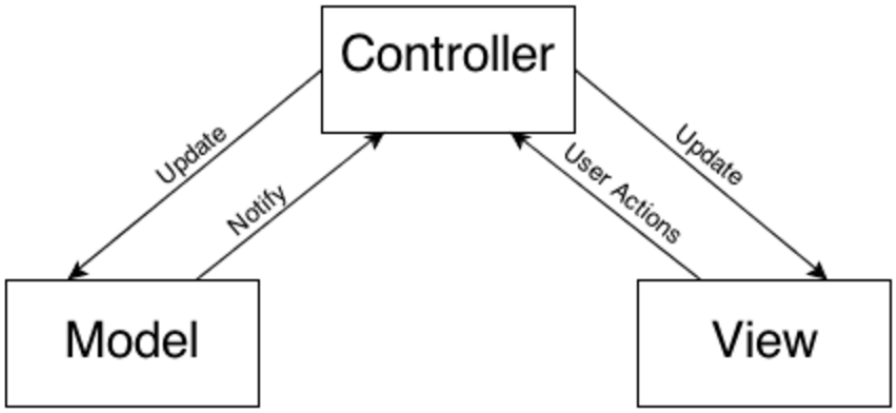

Figure 43: Visual for MVC design paradigm.

Each iOS View has a single Controller class associated with it, and the Controller updates the data in the Views. The View will then notify the Controller of actions the user performed and the Controller will either update the Model if necessary or retrieve any requested data from the backend database.

Maintaining separation and re-usability between the Models, Views, and Controllers made it extremely easy to add new features or entire Views. All functionality and static UI for each View is contained in the *viewDidLoad()* and *viewDidAppear()* methods of its Controller class. When a user navigates to a screen for the first time, functionality from the *viewDidLoad()* method will be executed; for subsequent occurrences, functionality from the *viewDidAppear()* method will be executed.

3.1.1 RealmSwift

Realm is a mobile database that enabled us to efficiently write our app's model layer in a safe and persisted way. Our Realm data Model objects are defined using regular Swift classes with a specified primary key for each data model. Realm was highly beneficial for the mobile application because it essentially served as a global hashmap that stored and persisted data locally to the mobile client; this prevented us from constantly reading from Firebase in order to render the Views. RealmSwift also supports the use of *Lists* and *Collections*, data types not natively available in Swift.

3.1.1.1 Creating Realm Objects

Adding objects to Realm is fairly easy. Creating an object requires creating a new instance of the appropriate object using the Model's initializer method.

3.1.1.2 Writing to Realm Object

All changes to a Realm object (addition, modification, and deletion) are done within a write transaction. In order to share objects between threads or re-use them between app launches, they are added to Realm via a write transaction. The following figure (Figure 44) shows an example of a write transaction, where an *updatedInterview* object is updated in Realm.

```
func updateCurInterviewInRealm(interview: Interview, budget: Budget) {
    let updatedInterview = Interview(uid: interview.uid,
                                     interviewee_id: interview.interviewee_id,
                                     position: interview.position,
                                     company: interview.company,
                                     start_date: interview.start_date,
                                     end_date: interview.end_date,
                                     total_consumed: interview.total_consumed,
                                     food_consumed: interview.food_consumed,
                                     lodging_consumed: interview.lodging_consumed,
                                     transportation_consumed: interview.transportation_consumed)

    updatedInterview.company_budget = budget

    do {
        try realm.write() {
            realm.add(updatedInterview, update: true)
        }
    } catch {
        print("Error adding/updating curInterview to Realm object!")
    }
}
```

Figure 44: Code snippet for writing data to Realm.

3.1.1.3 Reading from Realm Object

As noted by the Realm documentation, all queries return a *Results* instance, which contain a collection of *Objects*. *Results* have an interface similar to *Array* and objects contained in a *Results* object can be accessed using indexed subscripting. All queries are lazy in Realm and therefore data is only read when the properties are accessed. Figure 44 is an example of the mobile app reading an *Interview* object from Realm to update the *curInterview* object.

```
// Read curInterview object from RealmSwift
func updateCurInterview() {
    let realmInterviews = realm.objects(Interview)

    if (realmInterviews.count > 0) {
        print(realmInterviews)

        self.curInterview = realmInterviews[0]

        print("Updated Realm curInterview:")
        print("\t total food: \(self.curInterview!.food_consumed)")
        print("\t total lodging: \(self.curInterview!.lodging_consumed)")
        print("\t total trans: \(self.curInterview!.transportation_consumed)")
        print("\t total: \(self.curInterview!.total_consumed)")
    }
}
```

Figure 44: Code snippet for reading data from Realm.

3.2 Web – AngularJS

The Repay web application is developed using the AngularJS framework. It utilizes the Model-View-ViewModel (MVVM) design paradigm as shown in Figure 45.

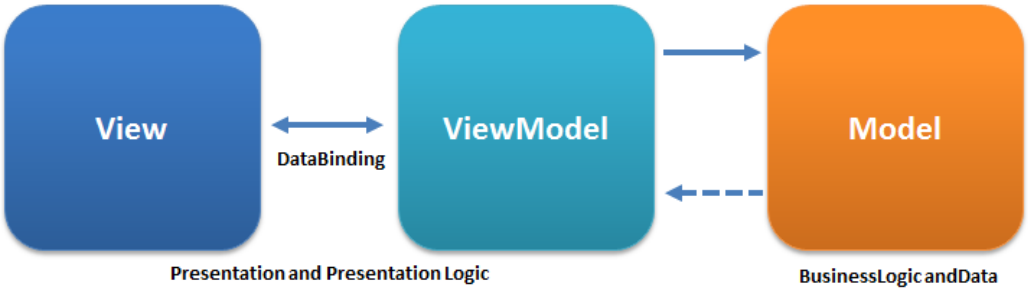

Figure 45: Visual for MVVM design paradigm.

3.2.1 AngularFire

According to the Firebase documentation, AngularFire is the officially supported AngularJS binding for Firebase. The combination of Angular and Firebase provides a three-way data binding between the web application's HTML, JavaScript code, and the Firebase database.

3.3 Firebase Backend Server

As of May 2016, Firebase provides tools to develop high-quality applications integrated with several platforms, including iOS, Android, and web. My team and I benefited from using Firebase because we were able to streamline our app's data integration using its real-time data retrieval/storage capabilities. Figure 46 is a visual depiction of how the *Receipt* object is stored in Firebase.

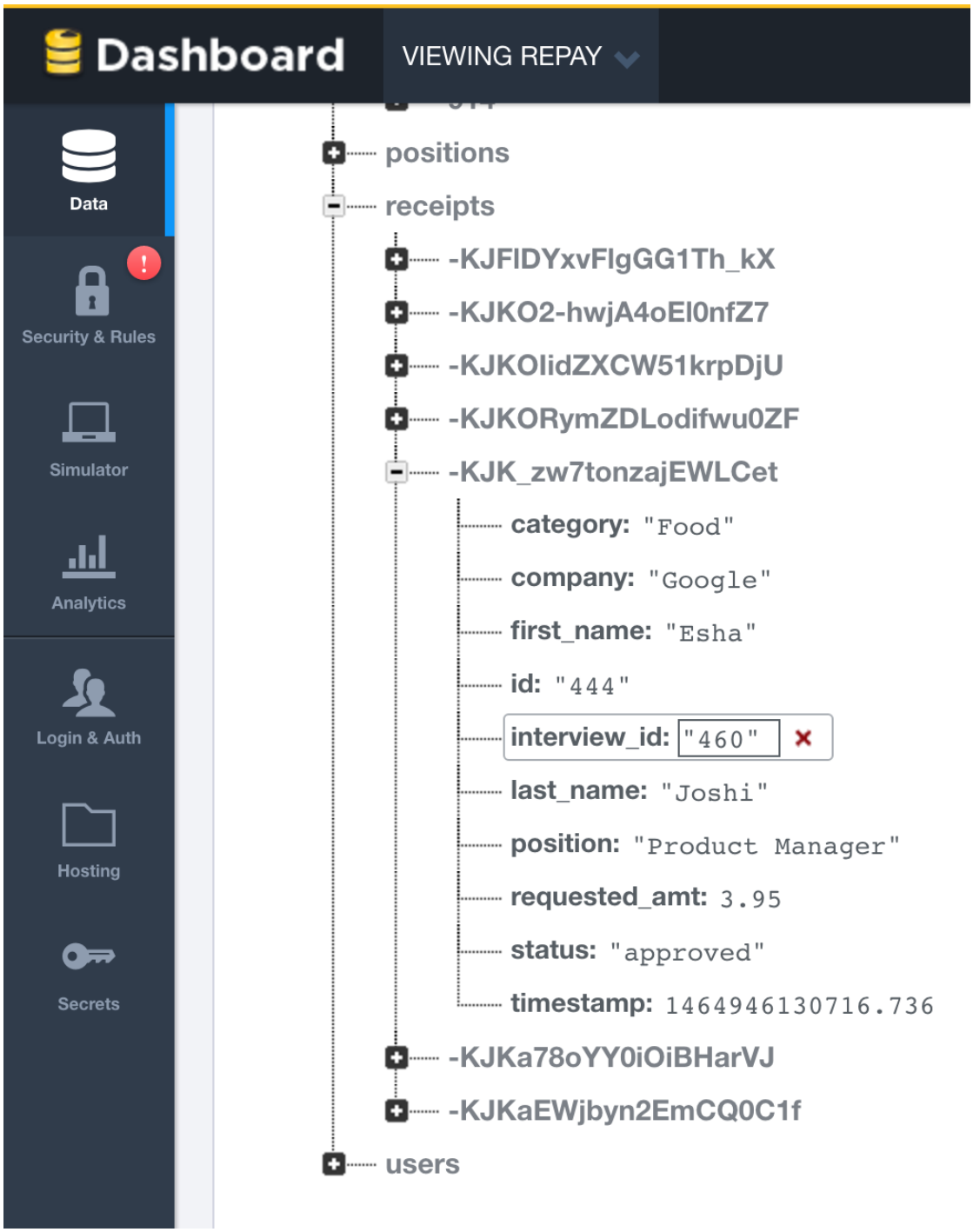
Dashboard
VIEWING REPAY
Data
Security & Rules
Simulator
Analytics
Login & Auth
Hosting
Secrets
positions
receipts
-KJFlDYxvFlgGG1Th_kX
-KJKO2-hwjA4oEl0nfZ7
-KJKOlidZXCW51krpDjU
-KJKORymZDLodifwu0ZF
-KJK_zw7tonzajEWLCet
category: "Food"
company: "Google"
first_name: "Esha"
id: "444"
interview_id: "460"
last_name: "Joshi"
position: "Product Manager"
requested_amt: 3.95
status: "approved"
timestamp: 1464946130716.736
-KJKa78oYY0iOiBHarVJ
-KJKaEWjbyn2EmCQ0C1f
users

Figure 46: Visual of how the *Receipt* object is stored in Firebase.

3.3.0.1 Writing to Firebase

As shown in Figure 47, one way of writing to Firebase is using the *setValue()* method. *setValue()* saves new data to the specified database reference (not shown in the figure) with the provided key (*receiptKeyId* in the figure). This figure shows a *Receipt* object tuple being saved to Firebase on the iOS end.

```
func writeReceiptTupleToFirebase(receiptObject: Receipt) {
    print("Writing receipt tuple to firebase...")

    let receiptTuple = ["id": receiptObject.id,
                        "interview_id": receiptObject.interview_id,
                        "company": receiptObject.company,
                        "category": receiptObject.category,
                        "first_name": receiptObject.first_name,
                        "last_name": receiptObject.last_name,
                        "position": receiptObject.position,
                        "requested_amt": receiptObject.requested_amt,
                        "status": receiptObject.status,
                        "timestamp": receiptObject.timestamp]

    let receiptKeyId = ref.childByAppendingPath("receipts").childByAutoId()
    receiptKeyId.setValue(receiptTuple);
    print("New receipt tuple of \(receiptObject.requested_amt) written to Firebase.")
}
```

Figure 47: Code snippet for writing a *Receipt* object tuple to Firebase (iOS).

3.3.0.2 Reading from Firebase

As shown in Figure 48, one way of retrieving data from Firebase is using the *Child Added* Read Event Type. The event callback is passed a snapshot containing each child's data; each attribute value can easily be obtained from the snapshot. This figure shows a *Receipt* object tuple being retrieved from Firebase on the iOS end.

```
// Read receipts from Firebase
func readReceiptsFromFirebase(interviewId: String) {
    print("Reading/sorting receipts from Firebase by interview_id: \(interviewId)...")

    ref.childByAppendingPath("receipts").observeEventType(.ChildAdded, withBlock: { snapshot in

        if interviewId == snapshot.value["interview_id"] as! String {
            let status = snapshot.value["status"] as! String
            let nReceipt = self.convertSnapshotToReceipt(snapshot)

            if status == "approved" {           // No longer needs attention
                if (!self.approvedReceipts.contains({$0.id == nReceipt.id})) {
                    print("\tApproved receipt of id: \((snapshot.value["id"])!) and amt: \
                        ((snapshot.value["requested_amt"])!)")
                    self.approvedReceipts.append(nReceipt)
                }
            } else if status == "flagged" {     // Needs to be accepted/disputed by interviewee
                if (!self.flaggedReceipts.contains({$0.id == nReceipt.id})) {
                    print("\tFlagged receipt of id: \((snapshot.value["id"])!) and amt: \
                        ((snapshot.value["requested_amt"])!)")
                    self.flaggedReceipts.append(nReceipt)
                }
            } else {                            // Needs to be looked over by business HR
                if (!self.todoReceipts.contains({$0.id == nReceipt.id})) {
                    print("\tTodo receipt of id: \((snapshot.value["id"])!) and amt: \
                        ((snapshot.value["requested_amt"])!)")
                    self.todoReceipts.append(nReceipt)
                }
            }
        }
    })
}
```

Figure 48: Code snippet for reading a *Receipt* object tuple from Firebase (iOS).

3.4 The User Model

The "User" model contains all of the information for a given user on Repay. It has fields that include the user's first name, last name, unique identifier, email, and password fields. Figure 49 shows a visual snapshot of the "User" object.

```
class User: Object {
    dynamic var uid = ""
    dynamic var email = ""
    dynamic var first_name = ""
    dynamic var last_name = ""
    dynamic var temp_password = ""
    dynamic var new_password = ""
    dynamic var confirm_password = ""
}
```

Figure 49: The "User" data model.

3.5 The Interview Model

The "Interview" model contains all of the information for a given interview accounted for with Repay. It has fields that include the interviewee's unique identifier ("user" id), company, position, start date, end date, and amounts consumed in the food, lodging, and transportation categories. It also contains a list of all the *Receipt* objects associated with this *Interview* instance. Figure 50 shows a visual snapshot of the "Interview" object.

```
class Interview: Object {
    dynamic var uid = ""
        DB)
    dynamic var interviewee_id = ""
    dynamic var position = ""
    dynamic var company = ""
    dynamic var start_date = ""
    dynamic var end_date = ""
    dynamic var company_budget: Budget?
    dynamic var total_consumed = 0.0
    dynamic var food_consumed = 0.0
    dynamic var lodging_consumed = 0.0
    dynamic var transportation_consumed = 0.0
    let receipts = List<Receipt>()
}
```

Figure 50: The "Interview" data model.

3.6 The Receipt Model

The "Receipt" model contains all of the information for a given receipt accounted for with Repay. It has fields that include the interview's unique identifier, company, category, position, user's first and last names, and amount requested by the interviewee. Figure 51 shows a visual snapshot of the "Receipt" object.

```
class Receipt: Object {
    dynamic var id = ""
    dynamic var interview_id = ""
    dynamic var company = ""
    dynamic var category = ""
    dynamic var first_name = ""
    dynamic var last_name = ""
    dynamic var position = ""
    dynamic var requested_amt = 0.0
    dynamic var status = ""
    dynamic var timestamp = 0.0
    let messages = List<Message>()
}
```

Figure 51: The "Receipt" data model.

4 Retrospective

The journey of building Repay was not always an easy one. In the last five months, my co-founder and I honed our skills, met some wonderfully-talented people, and received amazing support a series of different mentors.

4.1 Key Takeaways

My key takeaways from this large-scale project were in development and entrepreneurship.

4.1.1 Development

One of the biggest hurdles I had to overcome was learning Swift and how to develop for iOS. I needed to scour through a great deal of Apple Developer documentation in order to figure how to build every little component of the UI. Since Swift is a new language, I often struggled finding the proper tutorial to help me accomplish my implementation goals. However from building native components to integrating the app with Firebase and Realm, I truly enjoyed developing the mobile Repay app and will continue to enhance it in the future.

4.1.2 Entrepreneurship

One aspect of this senior project that was unexpected was entrepreneurship. From joining the Hatchery program, to participating in Startup Weekend and winning the Cal Poly Design and Dev Hackathon, my co-founder and I learned so much about customer development, market search, public speaking, the importance of clean and meticulously-planned UI/UX design, and effective communication while working in a team. As software engineering majors, it was refreshing yet intimidating to delve into the world of entrepreneurship among other passionate "hustlers".

4.2 Future Prospects

Although my partner and I accomplished our goals for a successful senior project, there are a plethora of features we would like to implement in the near future! A few of the features we would like to focus on next are in-app messaging, banking integration, and automated receipt analysis.

4.2.1 In-App Messaging System

One point of major feedback we received while speaking with company representatives was the notion of instantaneous guidance if interviewees face any difficulties with the app or reimbursement process. We feel that an in-app messaging system would be an ideal solution; including an additional channel of communication between the interviewee and company representative would provide a sense of comfort to the interviewee if something were to go awry. Although this has not been fully "white-boarded", our goal is for the interviewee to instantly get in touch with the appropriate company representative through the mobile application when necessary.

4.2.2 Banking Integration

The major shortcoming we faced in meeting the overall vision and scope of our project was failing to integrate with any reputable banking APIs. We knew from the start that we did not want to handle security issues and other common threats that come into immediate consideration when dealing with companies' and individuals' money. That being said, we still have hopes to connect the Repay application with Venmo, PayPal, or Square Cash. Ultimately, the possibility of creating an end-to-end holistic product – submission of receipts and secure handling of in-app reimbursements – would be incredible.

4.2.3 Automated Receipt Analysis

Currently our product allows the interviewee to upload a picture of the receipt supplemented with an amount to be reimbursed; the company representative approves/denies this reimbursement request based on manually checking whether the amount, location, date, and category of transaction match up with the company's protocols. But what if we could automate this approval process? The ideal development goal would be to have an element of computer vision in our applications such that the picture of the receipt could be easily parsed to extrapolate relevant information. We understand that computer vision algorithms for receipt analysis, optical character reader (OCR) recognition,

etc., is a stretch goal because it is technically very challenging. That being said, our third future goal would be to at least perform research and development in this area.

5 Conclusion

The experience of building the Repay mobile application was one of the greatest learning experiences in college. I gained immense knowledge in the area of mobile development and UI/UX design. I also got a chance to experience product management and learned what it means to effectively manage a product and assemble a team of smart people who share the same vision for the project as the founders do. I also spent a great deal of time interacting with enterprise potential customers (company representatives and students) and drafting requirements for both the mobile and web applications. At this moment, my partner and I do not currently have plans to deploy Repay or partner with companies to test this product. Nevertheless I am thoroughly happy with the outcome of this senior project and am looking forward to future iOS development opportunities and product management experiences.

6 References

1. MVC Picture
    a) Available at https://www.raywenderlich.com/86477/introducing-ios-design-patterns-in-swift-part-1.
2. MVVM Picture
    a) Available at https://upload.wikimedia.org/wikipedia/commons/8/87/MVVMPattern.png
3. Realm Swift
    a) Available at https://realm.io/docs/swift/latest/
4. Firebase
    a) Available at https://www.firebase.com/.
5. AngularJS
    a) Available at https://angularjs.org/.
6. AngularFire
    a) Available at https://www.firebase.com/docs/web/libraries/angular/.
7. "How mobile payments will grow in 2016."
    a) Available at http://fortune.com/2015/10/29/mobile-payments-grow-2016/.
8. "Apple Pay Is Here and There's Just One Big Problem"
    a) Available at http://time.com/money/3311917/apple-pay-iphone-iwatch-passbook/.