trickle

let the money flow.
let the money drip.
make it rain.
streamline your expenses.

GROUP MEMBERS

KEVIN MOODY
kmoody@stanford.edu

BEN MITTELBERGER
bmittelb@stanford.edu

ADAM SCHEXNAYDER
schex93@stanford.edu

DESCRIPTION

Our proposed project is a mobile and web application to facilitate easier expense management within a hierarchical organization. We envision a mobile/web app that could give a financial manager of an organization the ability to create spending sub-groups, allocate funds, set up spending rules, and process individual reimbursements. The underlying monetary technology that we are leveraging is Venmo, which facilitates easy account-to-account money transfers.

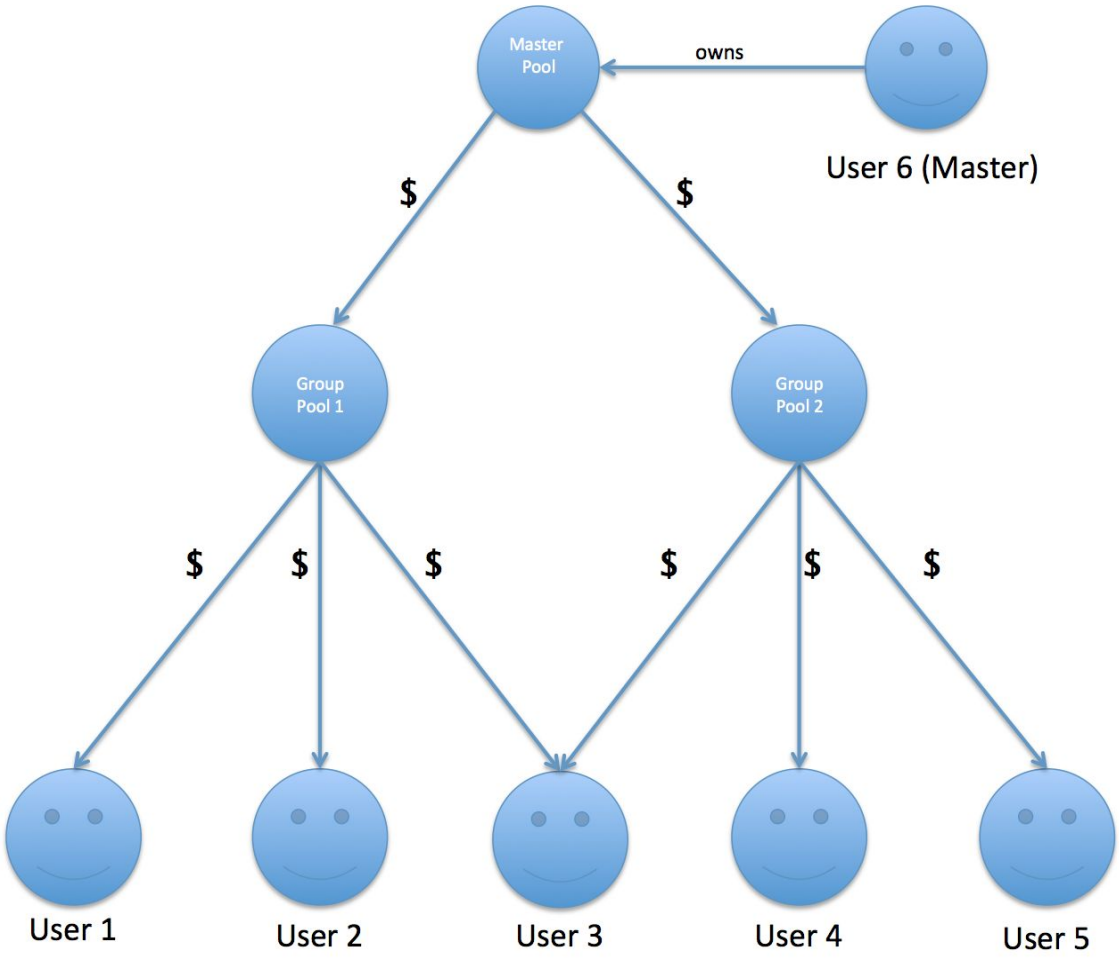
The hierarchy will be modelled using a tree-like structure in which all of a group's funds are dumped into a root node, known as the "master pool." Linked to the master node is an arbitrary number of sub-nodes, each known as a "group pool." From each group pool is linked an arbitrary number of other group pools, and any number of user accounts. Each user account is owned by a single person, and is connected with their venmo account. The user accounts can be connected to any number of group pools. The idea is that money flows down from the master pool into the group pools, which then "trickles" down to user

accounts in the form of reimbursements through venmo payments.

The master pool and group pools are managed by user accounts with special privileges. These “admins” are set by the organization “master” (also a user account), which is the user who created the original master pool. The admins are able to create sub-pools beneath the group pool that they run, and add/remove users to the group pools that they manage. The most novel feature here is the ability for pool admins to create a set of spending rules for their money. These rules would determine the allowed spending patterns for the users connected to this pool. The rules set for a pool would propagate to any connected sub-pools that exist lower in the hierarchy. Some example rules include: “Total reimbursements by pool cannot exceed $1000 per week” or “Any individual reimbursement over $100 requires special approval by the admin.” An additional perk of having these rules is that an administrator could set reimbursements to be auto-approved if they pass a strict criteria that they set. This set-and-forget rule-making could significantly reduce the amount of busy-work that a pool manager has to go through.

The general workflow for the app will work like this: A user would create a new master pool, thereby creating a new “organization.” They would then create new group pools that represent subgroups of the organization, defining their functions and rules. Finally, they would invite other trickle users to join the groups, either as admins, or simply members of the group. While using the app, a user would make a purchase for the organization, and open that organization’s tab in the trickle app. They would then add a new expense by giving it a tag, a description, an amount, and upload a picture of the receipt. Depending on the rules set by the admins, they may send it in for approval, or receive an immediate reimbursement. The money would flow from the master venmo account to the personal account. All of these actions will be transparent to the admins, and logs of any request will be available to admins up the hierarchy for later bookkeeping and analysis.

DIAGRAM OF TRICKLE HIERARCHIES



NEED FOR TRICKLE

We have found in our own personal experience that it is difficult to organize the splitting and reimbursement for group funds, especially in a setting where there are multiple sub-groups handling different aspects of an organization. The difficulty lies in effectively setting up pools of money for specific functions of the organization. Each sub-group that handles a function has a distinct subset of group members, and has its own set of financial requirements. An example of this layout will be described in the audience section. What financial managers end up creating are large and complicated spreadsheets that are interconnected, but require manual changes to be made at every level. Once these spreadsheets are set up, reimbursements often need to be made one at a time, with

cross-checking to ensure that each individual amount is correct, and that no one falls through the cracks. We believe that a simple interface for determining spending rules and processing reimbursements for complex sets of financial needs would be a useful tool.

AUDIENCE

We think that this product would be used by small to medium sized organizations that have a clear hierarchy or groups that have specific financial needs. Examples of these include casual sports leagues, clubs, and greek organizations. These groups have expenses that range from equipment, to food, to miscellaneous invoices. Their size means that managing reimbursements requires significant effort. Their casual nature also denote that they are most likely unwilling to pay for a heavy-weight solution like concur.

As an example, let us look at a hypothetical club soccer team. Each member would pay their fees into the team's central authority, who would deposit those funds into the team's venmo account, run by the financial manager. The master pool might have 3 group pools: equipment, social, and fees. Let's take a look at the equipment committee, which has multiple users. One is responsible for uniforms, while another may be responsible for balls and cones for practice, yet another may be responsible for mini-goals and corner flags. The users would buy the equipment with their own money and file for reimbursements up the tree. The same process would happen with the other two committees, likely with different users. Without trickle, the financial manager would have to collect and process each reimbursement request individually, tallying them up and figuring out where it fits in the budget. With trickle, the rules applied to the requests would guarantee compliance with budgetary restraints, and the process for actually paying out those reimbursements would be streamlined.

TECHNOLOGIES USED

INTRODUCTION

Ultimately, trickle would be a mobile application that is backed by a web admin interface. Due to the ten week time restriction, however, the initial product will be iOS-first. Its architecture will consist of five primary components: (1) iOS client, (2) API server, (3) cache server, (4) primary database server, and (5) long-term data store server. The following sections will describe each component in details and how they interact.

COMPONENT 1: IOS CLIENT

The iOS client will form the user-facing product. It is responsible for allowing the user to interact with historical data, create new payment rules, and file reimbursements. The codebase will consist primarily of presentation logic and user interaction flows. It will be

written primarily in Swift, with occasional Objective-C as necessary, on xcode. The client application will be tested on iOS simulators, iPhone 6s, and iPhone 5s. It will communicate with the API server with JSON over HTTPS.

COMPONENT 2: SERVER

The API server will serve as the core business logic server, enforcing financial rules and serving as the liaison between all present and future clients (iOS, Android, web, etc.). The API server will be hosted within an Ubuntu image running on an Amazon Micro T2 EC2 instance. The web server will be written in Node.js using the Express.js framework, daemonized by the forever javascript library, and served behind the nginx reverse proxy.

COMPONENT 3: CACHE SERVER

The cache server will be used to provide reduced latency responses for common and repeated queries and operations. It will be implemented as a Redis in-memory data store hosted on Amazon ElatsiCache.

COMPONENT 4: PRIMARY DATABASE SERVER

The primary database server will serve as the central relational store for all data pertaining to users, groups, organizations, permissions, financial rules, transactions, and reimbursements. It will be implemented as a PostgreSQL database hosted on Amazon RDS.

COMPONENT 5: LONG-TERM DATA STORE SERVER

The long-term data store server will handle storage of large files, such as the images of receipts attached to transactions and reimbursement requests. Amazon S3 will serve as the object storage server.

ADDITIONAL LIBRARIES

The Venmo API will be used to process authorized payments. Node-postgres will be used as an adapter to connect to the primary database, and Sequelize will be used as the object relational mapper (ORM) that maps entities to database tables. Node_redis (backed by hiredis) will be used as an adapter to connect to the cache server. Tesseract open-source OCR engine, created by Google, may be used to provide receipt-reading capabilities to the reimbursement submission process.

COMPONENT INTERACTION

During a typical request flow, the iOS client will send a JSON API request over HTTPS to the API server. The API server will process the request, and if necessary, request the appropriate data from the cache server. If the cache server does not contain such data, the API server will then request the data from the primary database server. Finally, the API server will send the formatted JSON API response via HTTPS back to the iOS client. In some cases, such as during asset or image retrieval, the API server may direct the iOS client to request files directly from the long-term data store server.

RESOURCE REQUIREMENTS

In order to build and run trickle on the iOS platform, Apple developer accounts would be required. Additionally, Amazon AWS credits may be necessary to power the EC2, RDS, ElastiCache, and S3 instances if the product requirements exceeds the respective free-tier capabilities.

COMPETITION

Venmo is a peer to peer payment application that utilizes users' Facebook friend networks in order to facilitate virtual payments. Users of Venmo have a virtual balance of money that they can use to "pay" other Venmo users by transferring part of said balance to other users' accounts. Users can link their bank accounts to their Venmo profiles in order to make withdrawals and deposits to and from their Venmo balance. While Venmo only focuses on direct peer to peer transferring of funds, trickle focuses on helping organizations transfer funds from a master organization account to the organization's individuals' accounts. Thus Venmo's audience is individual consumers while trickle's audience is mid-sized organizations. That being said, both apps utilize the ability to transfer funds between users, which is why trickle will use Venmo's API's for for this exact functionality. One way to think about our app is that trickle is an organization wrapper around Venmo's core functionality; however we want to design trickle in a way such that it could wrap around any payment software. The reason we want to wrap around other technologies is that we want to focus our development energy on trickle's unique organizational functionality instead of payment functionality that many other applications have already developed.

Concur is an enterprise software application that enables large corporations to better manage the creation and repayment of expense reports made by their employees. In essence, Concur has both the payment and organizational functionality that trickle would have. However, the fact that trickle is made for medium sized organizations but Concur is built for large corporation causes several key differences. First, of Concur is enterprise software that requires licensing fees to use. Medium sized organization would not want to waste any of their limited funds on such a large scale and highly customizable application. Similarly, trickle will be designed for ease of use. Concur requires a group leader's approval before any reimbursements can be made; whereas, trickle will utilize sets of rules that enables the ability to have automated reimbursements. trickle will focus on incorporating an easy setup process with an integrated gui system to allow users to view their organization in a node-graph manner. Concur has a very complicated setup process and does not have such a gui system because it is dealing with organizations that are too complex and complicated.

Paypal is similar to Venmo in that it's main functionality is peer to peer payments online. Paypal also has the expanded functionality of easily handling payments between merchants and consumers. Just like Venmo, Paypal does not offer the same organization and automation for mid-sized organizations like trickle does. Paypal would be another payment plugin that we hope our app would be able to wrap around if they user would rather use them instead of Venmo for transferring funds.

Splitwise is an app that allows members of a group to very easily track who owes what to each other. This is targeted specifically for roommates and trip members. Not only does it track expenses for the group, but you can also use Venmo or Paypal to "Settle Up" and pay whoever you owe money to. Splitwise is very similar to trickle in that it tracks expenses and utilizes payment apps to facilitate the transfer of funds accordingly. However, it does not offer any hierarchical configuration for mid-sized organizations. This is strickly for groups where every member is on the same level and there is no "master pool" of money for the organization.

POTENTIAL APPROACHES

One potential approach for tackling the problem that trickle solves is eliminating the hierarchical graph structure and just setting rules up per individual in the organization. This would make the app much more simple, but add much more complexity for the member of the organization. Additionally, the rules could potentially make less sense since the graph allows add much needed context to the reimbursement requests.

Another potential approach is to eliminate the "master pool" of money and the group members would simply pay each other for every reimbursement similar to Splitwise. This is impractical since this is not how organizations actually work in the first place. Additionally, this would mean that every single member would have to take part in every single reimbursement transaction.

RISKS

One risk is that the Venmo API's could be difficult to use. None of the group members have had to use Venmo API's before which means we are making many assumptions on their functionality. There could easily be troubles trying to make transactions on behalf of the user which could lead to complications when trying to automate payments. Additionally, there are several security risks when storing user information for an app that has access to their bank accounts. We will need to be very careful with where we store the users' Venmo information.

There are some risks with relying on Amazon EC2 and Amazon S3 because they are not machines that we have 100% control of. That being said, Amazon does have a great reputation in this field and far more experience than we have. However, we will have to trust that Amazon's machines never go down or our app will too. Additionally, we will be storing very sensitive information on these machines, so we will also have to trust Amazon's security efforts. Again, this really shouldn't be much of a problem since historically Amazon has never had these issues, so this should be a very minimal risk.

NEXT STEPS

To begin work on the product, provisioning profiles for each of the developers will first need to be created. This way, the product can be downloaded and run on iOS devices. Simultaneously, all of the Amazon AWS instances will need to be initialized and configured. At this point, work can begin from multiple angles. Design sketches will be created to storyboard the client iOS application, database tables will be constructed in PostgreSQL, and business logic API calls will begin to be fleshed out.