Flair: Social Product Discovery

Tanmay Chordia, Yash Shirsath, Alec Wang, Aditya Radhakrishnan
CIS Senior Design 2019

Abstract

Your friends know your preferences better than you do, and certainly better than Facebook does. What if you could use this information to connect people to products they actually care about? Flair is a recommendation engine that rewards users for finding friends who care about particular products. Our repeated recommendation algorithm quickly discovers the people in a social network graph who care most about a product, giving advertisers maximum bang for their buck. Users improve product recommendations, while interacting with advertising for the first time in a fun and engaging way.

Motivation

People in the modern day are peppered with ads around the clock. The average internet user easily sees dozens or even hundreds of ads across platforms such as Facebook, Google, Amazon, and Instagram. However, most people very rarely (if ever) interact with these ads, despite the billions of dollars Facebook and Google have spent on machine learning algorithms to better target ads. Most people no longer register seeing the ads at all, and have learned to simply filter them out. On the other hand, people really do have the need to buy things that solve their problems, and they rarely find these things in ads. At best, ads just familiarize people with products so they have some idea of what to look for in the future.

We have identified several deep rooted problems in the ad targeting space in Table 1. All of these issues guided our development of Flair: a platform specifically designed to cultivate social product discovery.

Solution

Our solution is to create a fun social environment in which friends recommend products for each other and discover products they might like. We do this by incentivizing users to buy products on our app (with discounts) and by sharing ad revenue with users who make good recommendations (with an in-app gold mechanism).

The following is a basic user-perspective flow of our app, for test user Alice.

1. Alice logs in and verifies email.
2. Alice sees a popup of the current product and chooses to buy on Amazon, recommend to friends, or skip.
3. If Alice want to recommend she see a list of Tinder-style cards of their friends. When she swipes right her friends are sent the product.
4. After three recommendations, Alice moves on to the next product.
5. Eventually, those three friends will see the recommended product.
6. If Alice buys, everyone who recommended the product to her gets gold.

Figure 2 helps visualize the user flow.

Technical Approach

UX

One of the most important parts of any app is the UX (user experience). UX is also one of the most tricky parts of an app to properly develop. While we all agreed on using the familiar swiping mechanism for recommending products to users, almost every other design aspect was contentious. We also found our disagreements to be difficult to reason around since much of the appeal of a UX is more of a gut feeling than anything quantifiable. For this reason, we spent several months developing a design thesis and a design process before we reached a final design.

UX Motivation We established early on that we wanted to app to be easy to use, fun, competitive, and intuitive. We wanted the app to also be visually appealing in terms of color scheme and layout. For these reasons, we decided to stick to relatively simple layouts involving one or two pages accessible through a bottom bar as well as a simple and vibrant color scheme.

Design Process We quickly realized that developing good design as a team was difficult from a process perspective. Our approach was to quickly generate a wide diversity of designs and then trim them down until we reached a mutually agreeable design. We used Sketch to quickly iterate through mockups of our app. To maximize our diversity of designs,

Problems	Description	Solution in Flair
Inconvenience	People are forced to watch ads before videos	Distinct time and place for product discovery
Irrelevance	People rarely want the product in an ad	Recommendations from friends who know what you want
No Active Engagement	Passively view ads	Actively interact with ads and share them

Table 1: Summary of problems and solutions

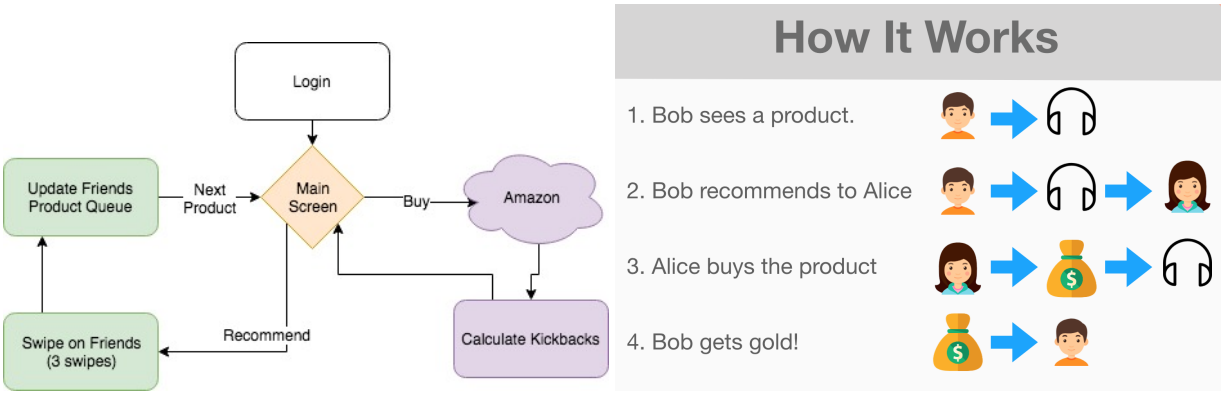

Figure 1: Left: Flowchart of user activity on Flair. Right: Simple visualization of the recommendation incentive system.

we split into different pairs each meeting and listened to different genres of music, browsed different internet sources, and spoke to different friends while creating our mockups. After about 2 months of following this process, we eventually converged to our final design (you can see our design mock-up iterations in Appendix A).

Final Design and Layout Ultimately, we decided on a design with the name Flair which represented the personality flairs being uncovered by our the recommendations on our app. The Flair design had a color scheme inspired by flare, a play on words of the title. We felt the warm colors were friendly and inviting to our users.

The Flair design combined aspects of a variety of dating app card-based swiping mechanisms to form an intuitive user experience. This is displayed prominently on the main screen of the app. Secondly, the Flair design utilized a leaderboard for the in-app currency. We decided to put the product above the leaderboard to remind users of the current product being shown. Finally, we added a feed of recent purchases and recommendations (inspired by Venmo) to make the app more fun and competitive. You can see the final UX design in figure 13 in Appendix A.

Card Based Interactions As stated above, we wanted the frontend to be as aesthetically pleasing and user-friendly as possible. We decided the right mechanic is a swipe based user interaction that Tinder popularized. The animations for swiping a card as well as all network requests and calculations are run asynchronously in JavaScript leading to many potential race conditions that had to be dealt with correctly. First of all, we decided to have a local cache of card objects because cards are expensive to load through a network request due and we found that users want to swipe through cards quickly without having to wait for a network request to finish. Therefore, when the cache reached 70% depletion, we launched a background request to fetch new cards and replenish the cache.

Optimistic UI Furthermore, in the name of usability, we implemented optimistic UI. This is a design practice of updating state locally even if network requests to update the state in our persistence layer have not yet finished. This makes the UI more seamless for users, i.e. the votes remaining changes directly after a swipe (as expected) rather than after a network request to update Neo4J completes. However, this may lead to inconsistencies between client state and persistence layer state (if a network request failed for some reason). We resolved this by synchronising persistence layer state and client state on cache depletion.

Tech Stack

Frontend Tech Stack The frontend of this application was built using React Native, a NodeJS-based platform that transpiles JavaScript into native Android and iOS code. We chose to use React Native for its ease of use, rapid iteration time, and rich library of third-party components that allowed us to essentially access native functionality through our JavaScript code.

Another reason for choosing react native over native iOS is the ability to use Expo. Expo is similar to containerization software like Docker. It allows the JavaScript source code to be streamed directly to a client container on the users device. The JS is then rendered into to native code within the client container App. This has many benefits. First of all, there is a lot of helpful built-in tooling and remote debugging that comes with Expo. But most importantly, Expo allows us to hot push client-side updates to our Beta Test users instantaneously over the air. The client container only has to

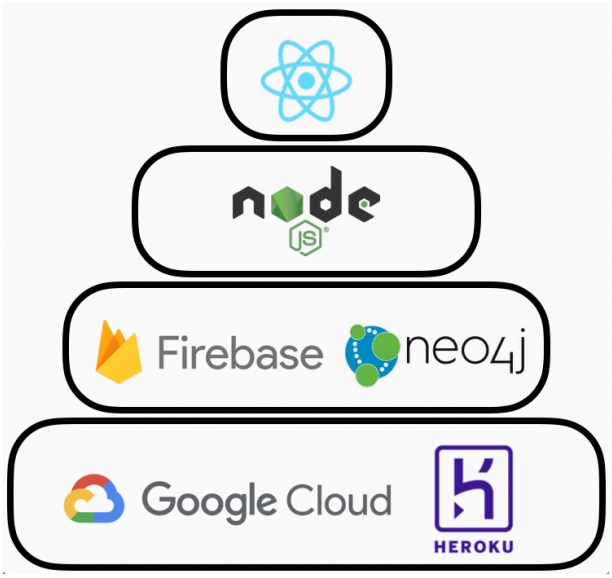

Figure 2: The Flair Tech Stack

go through App Review once, and basically never has to be updated. Compare this with the multi-day process of submitting an iOS app for review, and then the potentially infinitely long process of waiting for users to update their apps. Because the goal of this project was an exercise in shipping live code, user testing, and rapid iteration on feedback, this feature was invaluable to us.

Backend Tech Stack The tech stack of our backend consists of an Express server, Facebook authentication, and both Neo4j and Firestore databases. Using Facebook authentication allows us to directly access the users Facebook profile picture and friends list (only friends who were already using the app and accepted the permissions) which allows us to have a shorter onboarding process. Our user actions and algorithms which we will describe below are all graph-based so the graph database Neo4j was a clear choice for our primary database of user actions and data. We have an additional Firestore database for time series data such as purchase and recommendation history since graph databases are poorly equipped for handling such data. We host separate test and production versions of the server hosted on Heroku, and test and production versions of the databases on GCP (Google Cloud Platform).

Algorithms

Three key algorithms underpin the incentive structure that makes our app work.

1. *Cards Algorithm* The first and simplest algorithm is the cards algorithm, which determines the order in which you see other people on the main recommending screen. Since we want to capture latent social information, we want to show users their closest friends first since users know the most about them and their likes and dislikes. To this end, we show users their friends ordered by their number of mutual friends (a proxy for closeness) plus some Gaussian noise (so you do not see the same people in the same order every time).

2. *Conversion Kickback Algorithm* The kickback algorithm is our mechanism for rewarding good recommendations. When a user purchases a product, we receive a conversion fee which is split as follows: 20% is our margin, 20% is given as a discount for the purchasing user, and the remaining 60% is distributed as a kickback among those who recommended the product to the purchasing user. We want to reward recommenders roughly proportionally to their contribution to the purchase. We proxy this by constructing a reverse BFS tree of the recommendations rooted at the purchasing user and assigning shares to each recommender equal to 0.5^d where d is the recommenders depth in the tree (see Appendix B for an example). The normalized shares of each recommender then represents the percentage of the recommenders kickback each recommender receives. We distribute these kickbacks through our in-app currency gold which can then be directly redeemed for cash.

3. *Product Algorithm* This gold (net of redemptions) also conveniently doubles as a proxy for how good a user is at recommending products. We want to show people the most relevant products to them based on other users recommendations. Consequently, when a user moves to a new product, we show that user their current most recommended product weighted by the quality of those recommendations defined as the natural log of the recommenders gold (see Appendix B for an example). In combination, these three algorithms allow our app to achieve its goal of converting latent social information into great product recommendations.

Evaluation

Our evaluation was two-fold. We first verified that our algorithms actually showed the right products to the right people, and that over time everyone who wanted to buy the product did. Secondly, we recruited 20 people to use the alpha version of the app and got user experience data from them.

Simulation

We created a simulation in which we ran all of our algorithms with a Python script on a simulated graph of 100 users and 1000 products. We used the NetworkX graph library and the Numpy linear algebra library to implement the product recommendation and kickback allocation algorithms. Our algorithm runs as follows: we initialize a graph of users and products, then iterate through the users and simulate actions on their part. After their swipes / purchases, we update the users product queues and kickbacks according to the algorithms described earlier.

We formally define the problem setting as follows with the following three definitions and two assumptions.

1. Let $G_f = (V, E)$ be the undirected friends graph, in which users u and v are neighbors if u and v are friends. Let $n = |V| = 100$. G_f is randomly generated as a small-world graph.
2. Let U_{vp} be the utility matrix that gives the utility that user v receives from product p.
3. Let E^u_{vp} be user u's estimate of U_{vp}.
4. Estimation. Your friends have an unbiased estimate of your utility for a product. Formally, $E^u_{vp} = U_{vp} + N(0, \sigma)$, with adjustable noise parameter σ.
5. Smoothness. Your utility vector across products is correlated with your friends utility vector across products. Formally, the vector $\rho(U_a, U_b) > \alpha$, for tunable parameter α. Note that this is nontrivial to implement. If a and b are friends, and b and c are friends, but a and c aren't friends, there is implicit correlation between a and c despite this not being explicitly modelled in the graph.

   We modelled this by drawing $U_{:p}$ from an n dimensional multivariate gaussian distribution $N(0, \Sigma)$. We define $\Sigma = (I + \alpha A)^{-1}$, where I is the identity matrix, and A is the adjacency matrix of G_f. In other words, we define the precision matrix of the distribution so that friends have partial correlation at least α. As long as $|\alpha|$ is sufficiently small, this matrix is positive semidefinite by Gershgorin circle theorem, and so the distribution is well defined.

At runtime we use the following algorithm to generate the graph.

1. Randomly initalize G_f using the Watts-strogatz small world graph generation algorithm
2. Randomly draw each column of U (a vector of length n that represents the utility vector of a particular product) from $N(0, \Sigma)$ to satisfy the smoothness criterion.
3. Loop
   (a) Select a user u.
   (b) Show them the next product p from their product queue.
   (c) Show them a list F of their top 20 friends, sorted by mutual friends, plus some noise.
   (d) For $f \in F$, they swipe right on a friend if $E^u_{fp} > \beta$ (that they estimate that their friend is β standard deviations above the mean in terms of liking this product).
   (e) They buy the product if $U_{up} > \beta$. After a purchase, U_{up} is set to 0 (the user won't buy twice).
   (f) We update product queues and kickbacks based on the algorithms mentioned earlier.

Simulation Results

The goal of the simulation is to demonstrate that our algorithm is precise and complete: it both shows users product they like, and shows products to all users who want to buy them. The following graphs of user purchasing behaviour over time demonstrate these results. The results of the simulation are shown in Figure 3.

These graphs show that our algorithms meet all of the criteria we designed. The recommendation plots show that the number of 'Best' recommendations (recommendations that led to a purchase), is very high for the first several days. This means that users are mostly seeing products they really like. Furthermore, the purchasing converges very rapidly until the entire market is saturated. The market saturates because we assume that users won't buy the same product twice, and that user preferences don't evolve over time. Furthermore, we assume that each user interacts with 10 products per day. Thus, after 100 users interacting with 50 products each, we converge to the purchase limit (in other words, everyone who wanted to purchase a product does purchase the product). On the other hand, with random recommendations, only 12% of possible purchases are made. Our algorithm is so effective that every product is purchased about the same number of times as the total number of purchases under a random recommendation algorithm! This result is quite astonishing, as each user only actually interacts with 5% of the product set in order to find every product that they want to purchase.

Furthermore, our algorithm distributes a very large amount of money back to the users, with a right skewed distribution. In other words, most users receive about 100 gold, but the best recommenders receive as much as 500.

Overall, our simulation shows that our algorithms are both precise and complete. Users see products that they like most of the time, and after interacting with only 5% of the products given, the simulation converges to the purchasing limit.

User Testing and Feedback

To validate whether our app was usable, we conducted user beta testing. We reached out to fellow Penn students with access to iOS devices, and asked them to download our app via TestFlight, an iOS service specifically made for running beta testing. We ended up getting more than 20 users to actually use the app. We also distributed a survey asking questions about users experiences using the app to help them offer meaningful feedback to us. This survey can be found HERE. The questions are broken up into the following categories: onboarding, usage, understanding, feed, and prod-

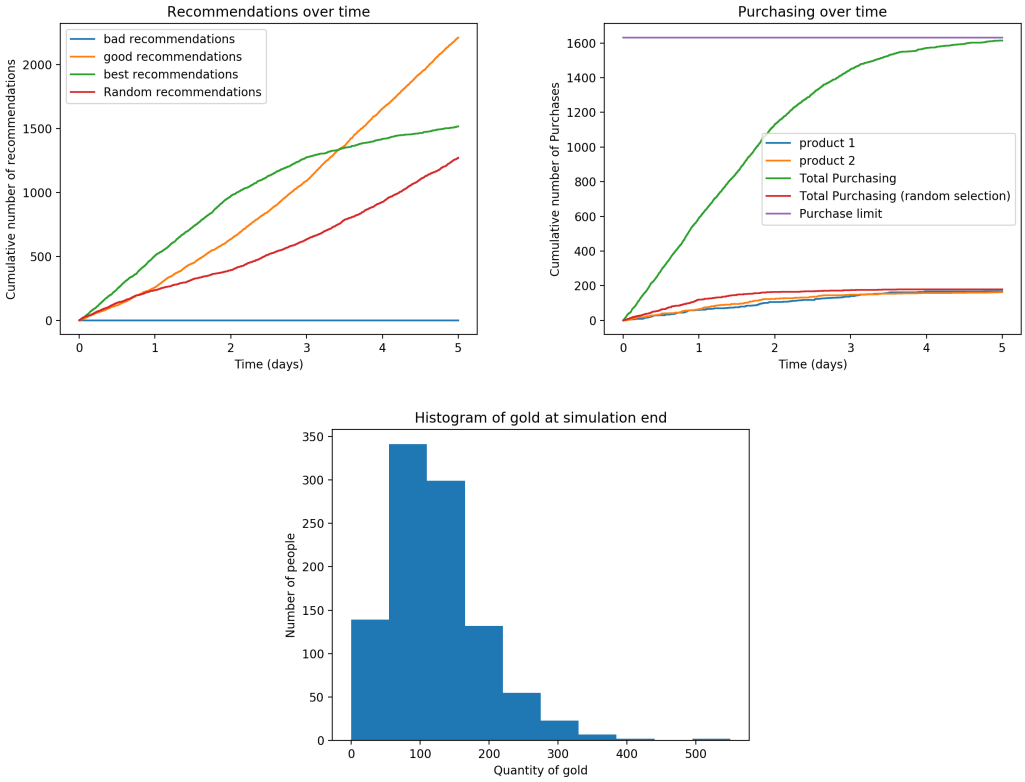

Figure 3: Simulation results over time. *Left:* Recommendation behaviour over time. Shows that the number of 'Best' recommendations starts our in the lead, but slows down as all of the products listed saturates the market, at which point 'Good' recommendations and random recommendations start to take over. This is largely due to the fact that when a user purchases a product, their future utility of that product drops to 0. Overall, these plots show that users will see content that they like, and that products will very rapidly saturate the market (in a period of only 5 days, in this model), assuming that users interact with 10 products on the platform per day. *Right:* The number of purchases over time converges to the purchase limit after only 50 product interactions per person (5 days simulation time). People only need to interact with 5% of the product set for them and everyone else to see all of the products they would like to buy. *Bottom:* The distribution of gold at the end. Most users have around 100 gold, and some users have as much as 6 times the mean (for making better recommendations)

ucts, with a final general category for the questions that did not fit in any of the others. Feed and products were two of the three screens used in the app (with the third being the simple leaderboard screen), so it made sense to ask questions related to them. Onboarding is a critical component of the app, as this is where users gain understanding as to how to use the app. If they do not get it, they will leave. Naturally, understanding is thus also important to gauge independently. Exploring usage is important to see how users interact with the app.

Conducting user testing ended up being a valuable exercise. We received a number of positive reaction from users, with many expressing admiration for the UI and the ease of use of the app. We also received critical feedback ranging from bugs that had to be fixed immediately, to new ways of interacting with the app to provide better user experiences, to simple things like making sure the tutorial fit on the screens of older, smaller phones. Here is a comprehensive list of all the fixes we made over the course of user testing, and the rationale behind them.

Resizing the tutorial We found that the iPhone 6 and 6S had issues with the tutorial fitting on the screen. Surprisingly, the button to end the tutorial was off the screen, and thus the app ended up hanging for some users. We resized the tutorial to solve this issue

Including a skip button Many users complained that some of the products on our catalog did not fit their interests, and they wanted to be able to skip products until they came across an interesting one. We debated internally whether we wanted to add this feature, since we felt the whole point of a platform like ours is not for users to engage with products they find interesting, but for them to engage with products their friends might find interesting. Another point in favor of the skip button is that it is difficult for us to put out targeted products for users when they first start using the app because we have no additional information about them. Better product selection happens when your friends start recommending products to you. We settled on adding a skip button with a limited number of skips per day, so as to encourage users to explore new products, but giving them some opportunities to browse around.

Improving the product catalog A lot of users wanted to see products that better fit their everyday lives (like household cleaning items) rather than just niche products. We added some items in this category so as to diversify our catalog.

Adding events to the product catalog Beyond just wanting good products in the catalog, some users expressed excitement that the platform could be used to find out about new events, such as concerts, wine tastings, and the like. We decided that our app fit perfectly with the manner in which people find out about events: word of mouth through their friends. If someone hears about a concert where an artist their friend really likes is performing, they are pretty likely to reach out and let them know. Our platform just makes this process simpler, and offers an avenue for promoters of little-known-events to get the word out.

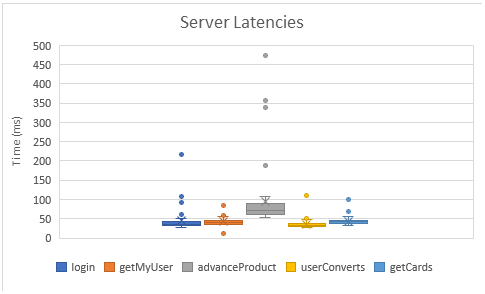

Figure 4: Boxplot of Server Latency Times

We thus added events in Philadelphia from EventBrite and other similar event/ticket websites to our product catalog.

Server Latency Testing

Before deploying our app to users, we wanted to verify that they would have a seamless experience using the app. We ran latency tests on all our major server routes, which consists of the login, getMyUser, advanceProduct, userConverts, and getCards routes. The login route is used to login to the app. The getMyUser route is used to access important metadata associated with a user, including their profile picture, gold, votes, and so on. The advanceProduct route is used to advance the user to the next product after they have either finished voting on the current one, or have skipped it. The userConverts route runs the conversion kickback algorithm. The getCards route is used to get the set of user cards that will be displayed to the user. The box plot below shows the distribution of latencies for each route, with each of the routes having been called 50 times to generate this graph.

Ethical Considerations

As with any consumer-product facing product, there are a few important ethical considerations to consider. Primarily, we are concerned with protecting consumer data. To that end, we have incorporated features into our app so that consumers transparently understand what information they are allowing us to access.

Users use Facebook to login to our app. This makes it easier on our end to protect user information, since passwords and other sensitive information is thus secured by Facebook. By agreeing to login via Facebook, users allow the app to receive access to their profile picture and friends list. We need this information to allow full functionality of the app, as the whole point of the app is to recommend products to your friends. If users do not wish to share this information, using the app would not be of any benefit to them.

One feature that users can opt out of is the feed. Our feed screen allows other users to see what the most recent actions taken across all users are, but if you elect to, you can choose to have any actions involving you to be excluded from this feed. We believe that this is a necessary feature to have, since some users would be uncomfortable with others knowing that they have purchased a certain product, or have recommended that product to a friend.

We also elected to use SSL and HTTPS within our app to make our routes secure. This protects the app against server-side attacks. Additionally, we made use of Cloud Firestore's Security Rules functionality to define access controls and prevent user information in the Firestore database from being accessed by malicious entities. We took similar steps with our Neo4j database by defining authentication and authorization rules for the same purpose.

Acknowledgements

Thanks to Brett Hemenway and Ani Nenkova for all of their help and support throughout the process of doing this project.

Appendices

Appendix A

All iterations of our design mock-ups are shown below.

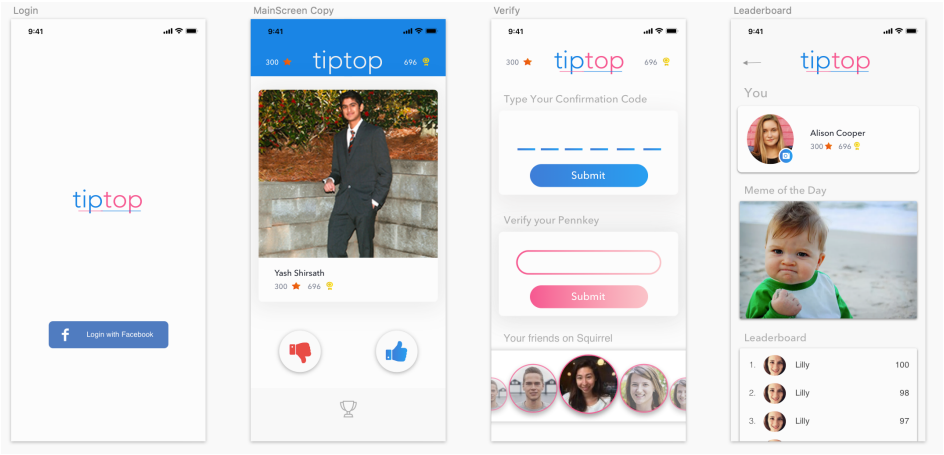

Figure 5: TipTop Mock

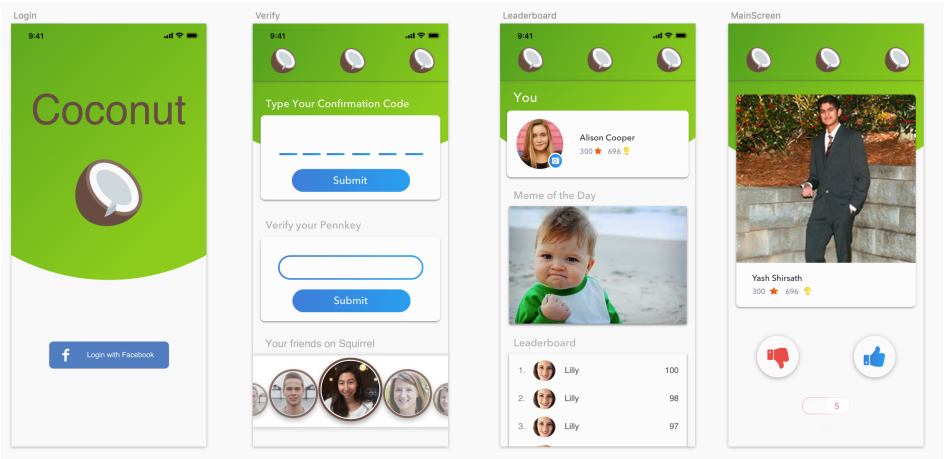

Figure 6: Coconut Mock

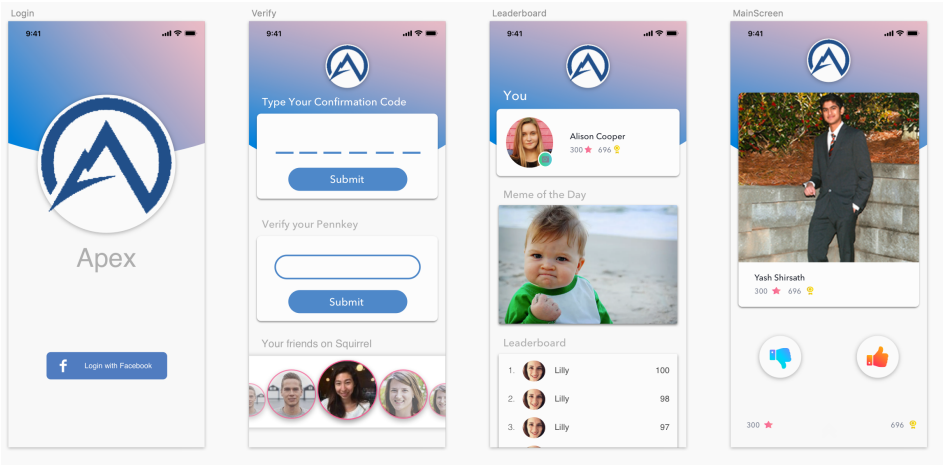

Figure 7: Apex Mock

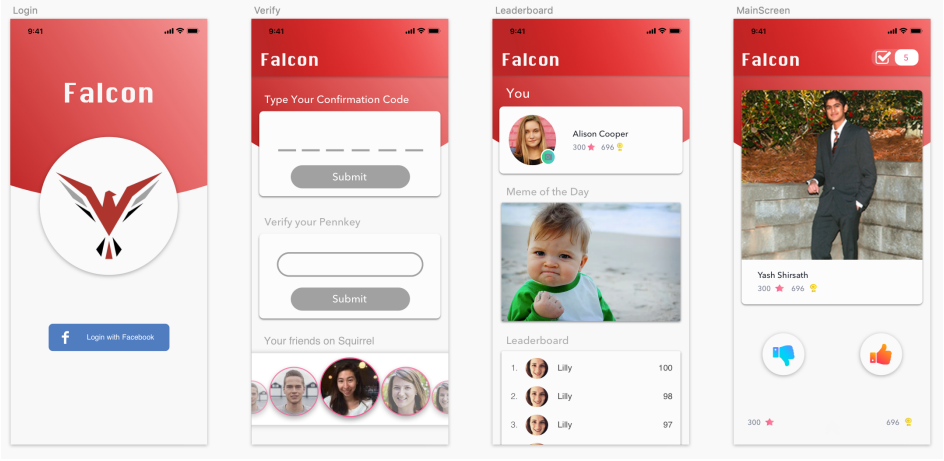

Figure 8: Falcon Mock

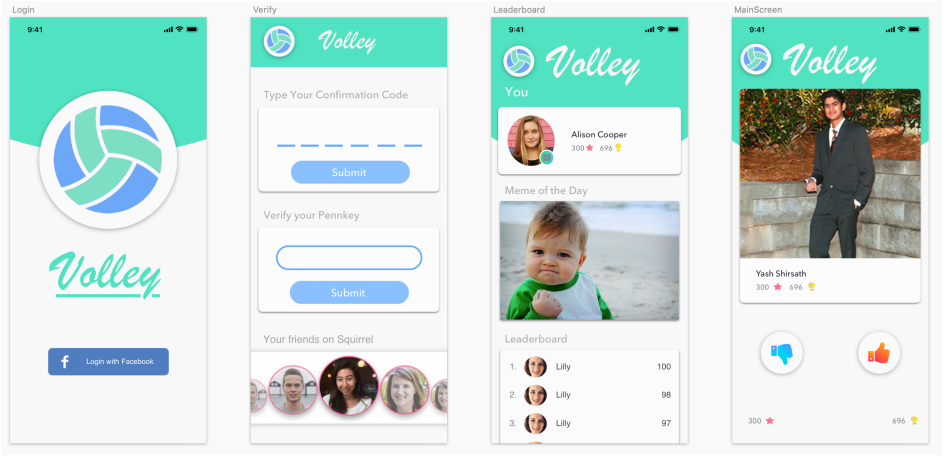

Figure 9: Volley Mock

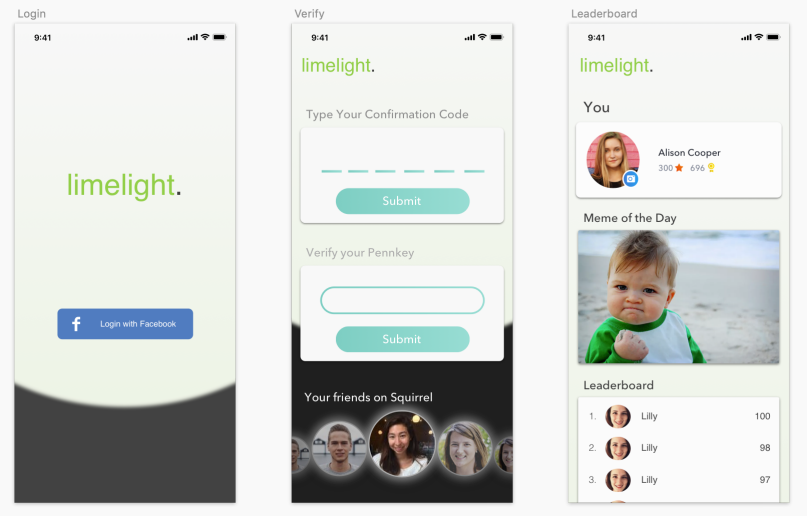

Figure 10: Limelight Mock

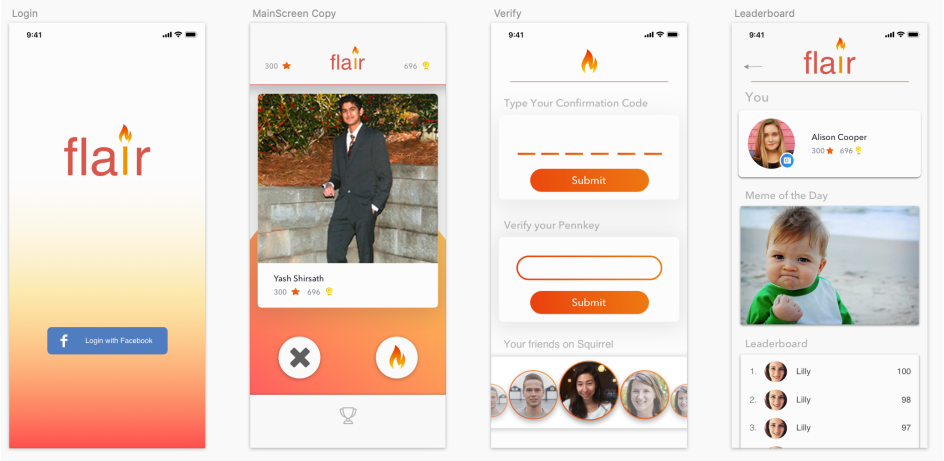

Figure 11: Flair Mock #1

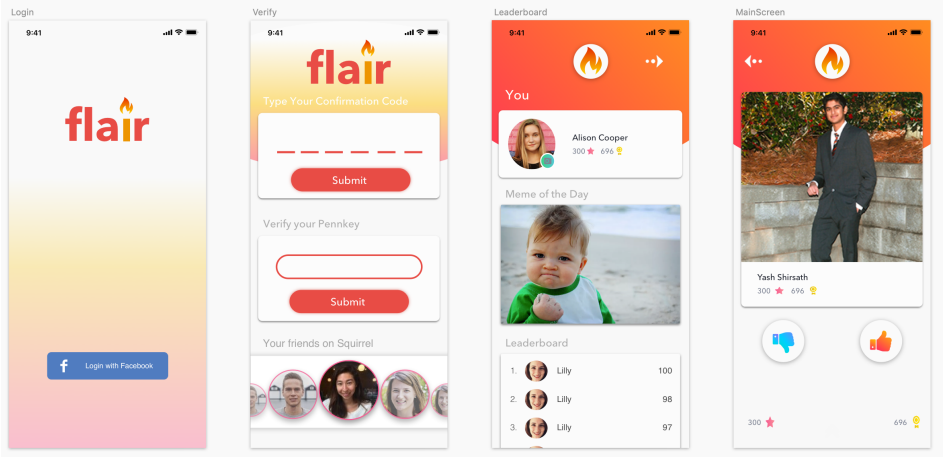

Figure 12: Flair Mock #2

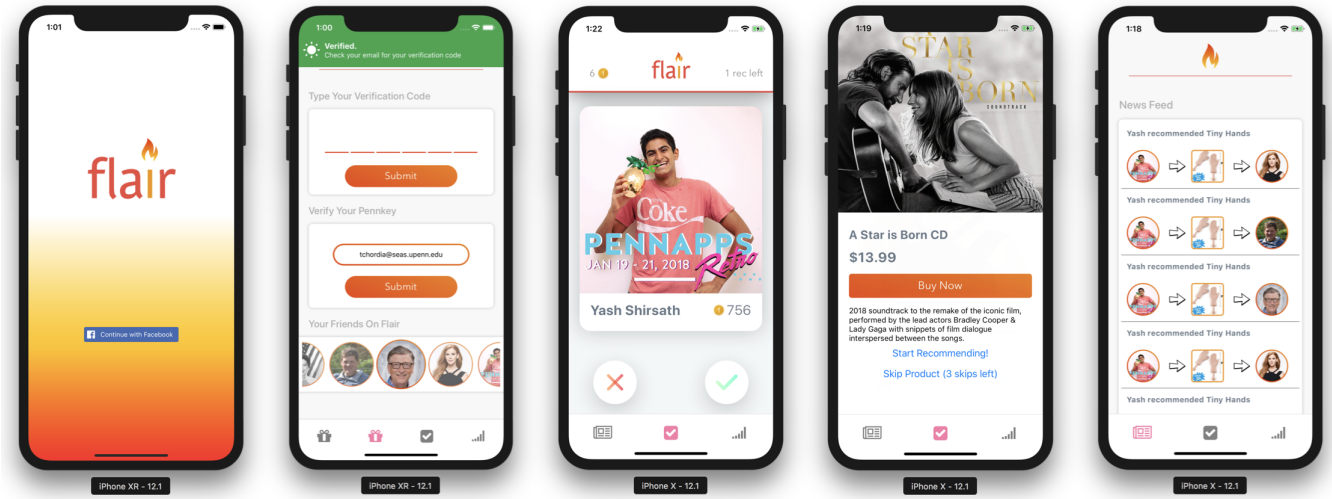

Figure 13: Flair Final Design

Appendix B

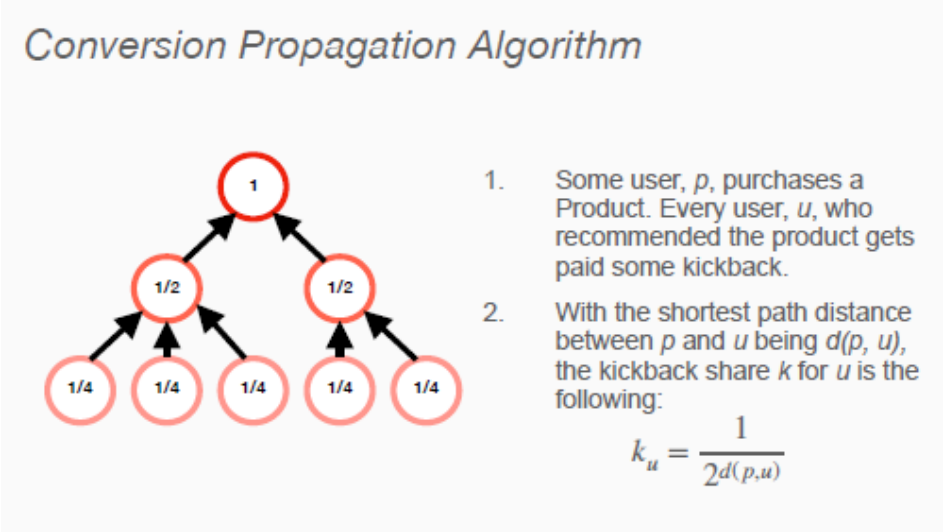

Figure 14: Conversion Kickback Algorithm Example

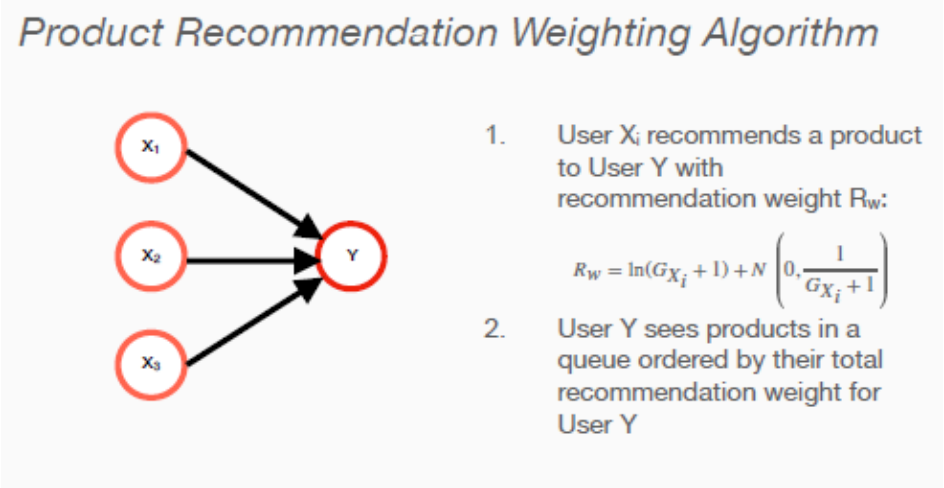

Figure 15: Product Recommendation Algorithm Example

Yash Shirsath, Tanmay Chordia, Alec Wang, Aditya Radhakrishnan

M&T Integration Lab

NEED

The current paradigm of ad delivery relies on inferring user characteristics by browsing history, likes, and other online behavior. However, current methods do nothing to address the wealth of information about user preferences stored in social networks: your friends are very likely to understand your preferences and your pet peeves. Management scientists call this social understanding *transactive memory*. Transactive memory is currently a huge and completely untapped source of information about customer needs. Our ad platform is the first to allow advertisers to access transactive memory to better target ads.

SOLUTION

Our solution is a novel approach to display ads. We've created an interactive ad experience that combines an Ad distribution platform with a social network. Users are asked to recommend products to their friends within the ad slot. A typical experience for a user might go as follows:

1. The user sees our full screen ad on a website or an app
2. At the bottom of the ad are a selection of friends that we think might be interested in the product.
3. The user either clicks the product if he/she is interested, or clicks a friend to recommend the product to.
4. Users who are recommended the product will see the product in a future ad, and can see who recommended it to them.
5. Users who consistently make good recommendations are rewarded with points / monetary reward

On the back end, we distribute an open source sdk (software development kit) for app developers to integrate our ad unit into their mobile apps. They then use the Flair Ad Network to match with advertisers and monetize their apps.
The project described in the accompanied technical report is a prototype of what a specific ad slot will look like. It is currently a standalone app, but will be integrated into an SDK that 3rd party developers can import to easily display our full screen popup ads. Ad buyers can then bid on the ad slots in a similar fashion to current ad distribution networks.

STAKEHOLDERS

Similarly to traditional ad networks, Flair operates on a 4 party system: Ad Buyers, Ad Unit Sellers, Ad network, and Ad Viewers. We're using an auction model to auction ad slots (eg. slots on mobile devices) to advertisers (eg. Nike). Our in house recommendation system will determine which users to target based on industry standard targeting methods and which of their friends to present as secondary alternatives. As individuals recommend their friends and narrow down which users are most interested in this product, we will command a higher CPC (cost per click) and CPM (cost per thousand impressions).

Other stakeholders include governments and other privacy lobbies. As can be seen by the growing scrutiny of the large incumbents in the space, these stakeholders are gaining a larger pulpit and more power over traditional advertisers.

VALUE PROPOSITION

Flair solves two problems simultaneously. First of all, virality research shows that users are much more engaged with content when it includes their friends. Secondly, ads are extremely relevant because users, are personally recommending them for their friends. Due to the 6 degrees of separation rule of social graphs, ads are almost guaranteed to efficiently be routed to those who are most interested in viewing them.

On a competitive note, one might argue that Facebook is harnessing information within a social network. This is a fair point: they do a good job of gathering data about you specifically and targeting based on that. However, they don't do such a great job of unlocking the information that user a knows a's friends.

A critical piece of our value proposition is related to the aforementioned governments and privacy lobbies. Incumbent advertisers are in hot water right now because their value proposition to end users is orthogonal to their value proposition to advertisers. I.e. they provide some service to consumers with an ad network bolted on to the side to allow the service to be free for users. This causes misalignment of incentives and confusion around who owns the data. Better targeting often involves gathering more and more personal data. However, Flair is different. We've gamified the ad itself. A large part of the reason users use our platform is because they get better product recommendations and they actually enjoy recommending products to their friends.

CUSTOMER SEGMENTATION

Because we are an ad distribution platform operating on a B2B2C model, we have targeting decisions for three different types of stakeholders: ad slot sellers, ad buyers, and ad viewers. Our targeting decision for ad viewers and ad buyers is dependent on who we choose as our ad slot sellers. This is because ad slot sellers each cater to specific customer demographics and ad ad buyers want to target those demographics.

We believe the best ad slot sellers to target are mobile games for many reasons. Firstly, there is ample supply of mobile games that turns over very frequently. And these game developers are always looking for the best way to monetize their mostly free apps. Secondly, mobile games already utilize interactive/playable full screen popup ads. Thirdly, mobile games cater to a younger demographic who will theoretically be more familiar with our swiping mechanics because of dating apps like Tinder.

MARKET RESEARCH

Every new media format for ads has had its pioneer. Google AdX with search ads. Facebook with social ads. Youtube for pre-roll video ads. AdMob for mobile ads, and DoubleClick for display ads. Which each new frontier of advertising, the trailblazer develops the real estate (i.e. grows visibility and customer acceptance of the new format) and spawns a plethora of competitors and M&A targets for incumbents. We assume that the same tried and true process will occur as well with this format.

The net digital ad revenues for the top three players are projected to be $102.4 Billion, $67.2 Billion, and $30.5 Billion for Google, Facebook, and Alibaba respectively. The total market size of digital ad spending is $129.34 Billion, representing year over year growth of 19%. Furthermore, digital ad spending will surpass traditional ad spending for the first time in 2019. On top of this, mobile ad spending will make up a full 66% of this [Source].

From this, we can conclude that the future of advertising is digital and mobile. While the current ad distribution networks already have huge reach, none of them are currently not harnessing the transactive knowledge stored in social networks and providing the same value proposition as us. Because we have a unique approach to monetizing this knowledge, we project that we will command an increasing share of this massive and growing market.

REVENUE MODEL

On our platform, the money flows from ad buyers to ad sellers. Because we own the platform, we take a portion of this as a fee. We retain 20% of this revenue as our gross margin, and disburse the remaining 80% as kickbacks to our users as incentives for good recommendations. Furthermore, Under these assumptions we can model out revenue as follows:

Revenue Projections ($1,000)

	2017	2018	2019	2020	2021
Global Digital Ad Spending*	$273,000.00	$286,650.00	$300,982.50	$316,031.63	$331,833.21
Mobile Spending (CAGR 19.27%)	$125.34	$149.49	$178.30	$212.66	$253.64
Market Penetration	0.00%	0.00%	0.50%	1.00%	5.00%
Projected Revenue	$0.00	$0.00	$0.89	$2.13	$12.68
Margin	$0.00	10%	20%	20%	20%
Projected Disbursements	0	$0.00	$0.71	$1.70	$10.15

*Source