Massachusetts Institute of Technology

6.170 FINAL PROJECT
“ibetcha”

Lecturers

Adam Chlipala, Mark Day, Daniel Jackson

Submitted by

Zulsar Batmunkh

Saadiyah Husnoo

Dong Hyug Lim

Dana Mukusheva

Submission Date

November 25th, 2014

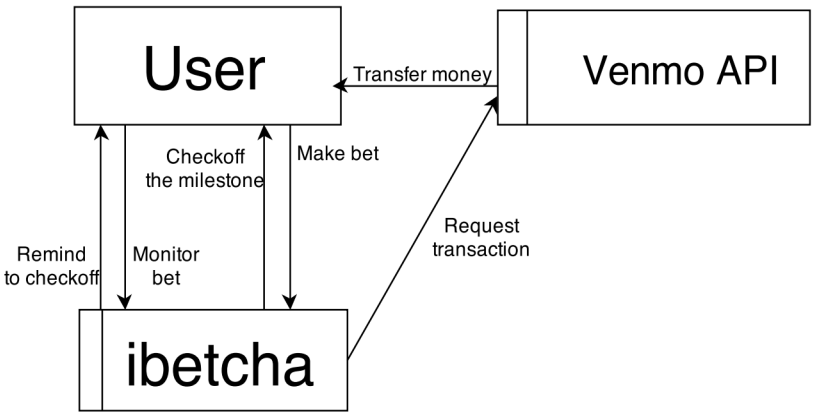
I. Motivation

ibetcha is an online app to help users follow through with their resolutions by betting money on it with friends. It enables users to build a support network, select a group of trusted friends and seek their support to see to it that he/she completes the task. We think that doing so will add peer pressure and monetary incentive to the user's resolutions.

II. Purposes

1. **Help a user follow through his/her resolutions.** While it's easy to make resolutions, they're often hard to keep after the first few days. Therefore, ibetcha will help users stick to their resolutions and not to give up even when it gets harder to follow.
2. **Make a reliable support network for the user:** ibetcha allows the user to choose a group of trusted friends to monitor him/her. It does so by providing incentive for the support network to regularly check on the user's progress.
3. **Help a user keep track of his/her progress:** Each user can look back on the history of resolutions and get further motivation for future resolutions.

III. Context Diagram



The app is accessible to any user who has a valid Venmo account. We use the Venmo API to proceed with money transfers. The user can invite new users by sending email invitations.

IV. Concepts

ibetcha incorporates the following concepts to support our purposes:

Betcher[(1)]**:** a user who makes the bet, puts money on a stake and chooses his/her monitors in order to fulfill his/her resolution.

Bet[(1)]**:** the representation of the deal between a betcher and his/her monitors. The Bet contains such fields as a Bounty (see below), Monitors (see below), a creator, frequency (e.g., daily, weekly, weekends only, etc.), start date, end date. The Bet is a central concept of our app and fulfills the primary purpose of helping users to follow their resolutions.

Bounty[(1)]**:** amount of money put on the bet. The bounty is chosen by the betcher and can't be changed after the bet was created. In case of successful task completion, the bounty is returned to the betcher, otherwise, it is evenly distributed between monitors. The bounty is used in the user/system interactions and system/Venmo interactions.

Monitor[(2)]**:** a friend chosen by the user to keep track of the user's progress. The user has a group of three or five monitors for progress. The monitor is empowered to give Checkoffs (see below) to the betcher.

Milestone[(3)]**:** a point in time that requires an action (checkoff) from the monitors. Milestones are generated by the app based on the start date, end date and frequency (for example: how often the person will run) of the bet. Milestone states include:

- "inactive": before the date of the milestone
- "open": on the date of the milestone, until the next day or until a checkoff is received.
- "pending": on the day after the milestone date until a checkoff is received or the bet expires ("Drop date" of bet past with some checkoffs still pending)
- "success": if monitor gives a positive checkoff
- "failed": if monitor gives a negative checkoff
- "closed": if bet expires and state of the milestone was "pending action". Or if any milestone gets a "failed" checkoff, the other "pending", "open" and "inactive" milestones become "closed".

Note:

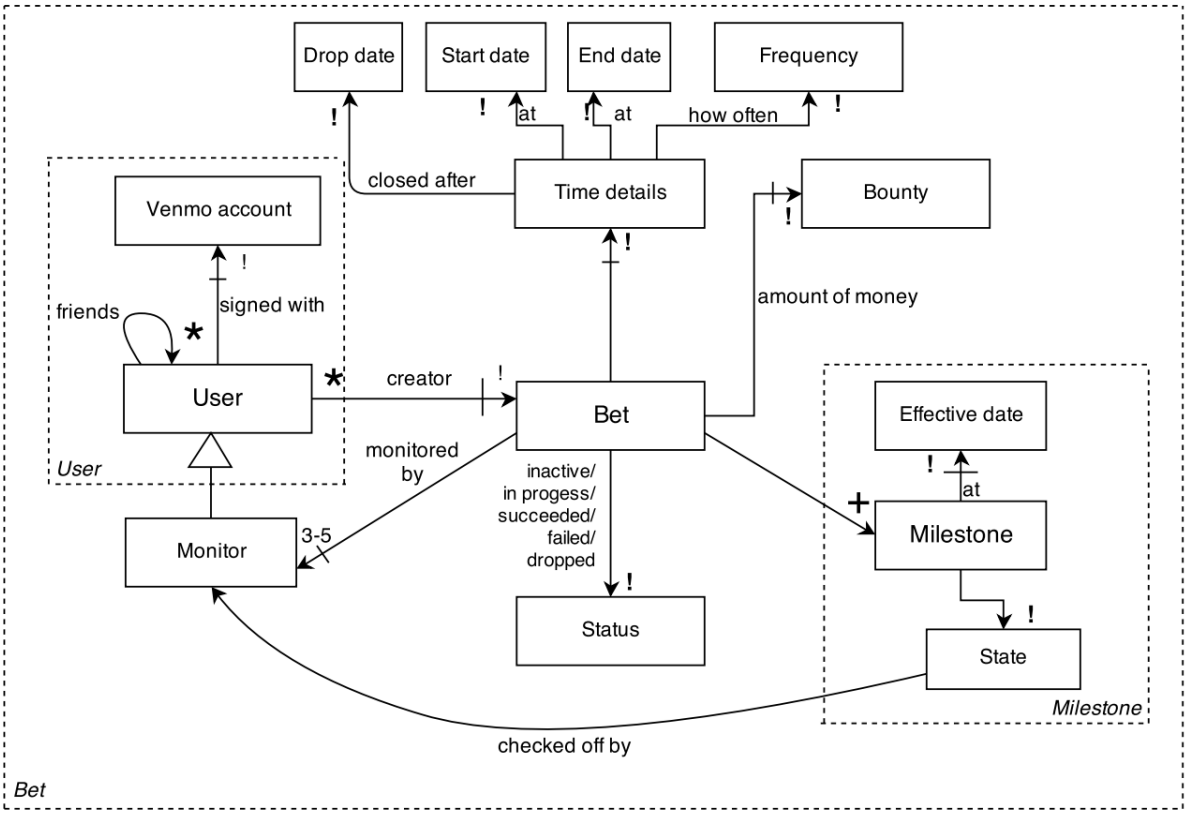
[(1)] Corresponds to purpose number 1

[(2)] Corresponds to purpose number 2

[(3)] Corresponds to purpose number 3

V. Data model and data design

1. Data model



2. Collections

- Bet
- User
- Milestone
- FriendRequest - stores data related to pending friend requests: “from”, “to”; data is removed once the request was accepted or rejected
- MonitorRequest - stores data related to pending monitor requests: “from”, “to” and “bet”; data is removed once the request was accepted or rejected
- MoneyRecord - stores data related to pending friend requests: “from”, “to”, and “amount”; data is removed once the request was accepted or rejected

VI. Behavior

1.User Interface workflow

The workflow of the app includes such actions as:

- a User signs up for an iBetcha using Venmo account
- a User creates a Bet object, chooses the Bounty and Monitors (at least 3 no more than 5) among his friends, chooses the timing details of the Bet. User cannot edit or delete the Bet.
- friends selection process: a User can invite his/her friends (handled via email) to be his/her ibetcha friends (or create an ibetcha profile and then become a friend).
- a User can reject friend request
- monitor selection process: a User can select Monitors for his/her new bet from the list of his/her ibetcha friends.
- a User can be invited to be the monitor of the Bet and checkoff the Milestones of the Bet creator.
- a User can reject an invitation to be a Monitor.
- a User who is a Monitor to some Bet will receive a reminder if none of the fellow Monitors of the same bet did not check the Milestone after its effective date has passed.
- if a Monitor decides to fail the Milestone, he/she will be asked to confirm his/her actions and possibly, leave some feedback.
- a Bet creator will be able to see the final result of his/her bet after the end date has passed.
- Monitors will receive money in case the Bet they were monitoring has failed.
- if all Monitors do not respond to the emails and do not checkoff the user, when the End date passes the Bet creator will NOT lose the money.

2. Security Concerns

- Since money is involved with ibetcha, there's a lot of risk involved regarding user protection. Extensive measures should be taken to protect users from unauthorized charges, credit card frauds or theft of personal information or other crimes.
- Not having robust security system, thereby exposing our website to cross-site scripting attacks. As a result, attackers could inject scripts into our app and users could be redirected to web content controlled by the attacker under the guise of our app.
- Not correctly implementing authentication and session management might allow outside attackers to compromise users' passwords or personal information and assume users' identities.
- Hackers might get into our system disguised as authenticated users and might disrupt the app, interacting with other users and falsely checkoffing them to hoard their money.

- Malicious users might try to drown our server with a vast number of requests or actions that will degrade our app performance (could create multiple number of bets with maximum frequencies, and creating many pending checkoffs, that would require a lot of memory repeatedly and sending repeated emails to users).
- When users invite other users via email, we need to make sure that no other information is exposed, and the url included in the email is not used for purposes other than signup or invitation
- We need to make sure that a user cannot spam people via our email functionality.
- XSS attacks. Hackers could inject client side scripts into our web pages and use our app to send malicious code to users and could exploit the trust that user has for our website.
- CSRF attacks. Hackers could exploit the trust that our app has in the users. an attacker could trick a user into accessing a website or clicking a URL link that contains malicious or unauthorized requests. Thus, if user's authentication session is still valid, an attacker can use CSRF to launch any desired requests against our app, without the app being able to distinguish whether the requests are legitimate or not.

3. Mitigations:

- To secure users personal information further, we use Passport.js to authenticate the users and hash passwords while stored in the database. Moreover, we'll minimize the sensitive data stored in ibetcha by using Venmo to secure money transfers. Therefore, all transactions will be processed through Venmo and users' credit card and other personal information would not be stored in ibetcha.
- We ensure that monitors can be chosen only from the friends list; thus no strangers can modify user's profile and his/her ongoing bets. We also encourage interaction between users as an informal identification to identify outside attackers as early as possible. Moreover, we could also introduce extra security questions whose answers are only known to the trusted parties to secure user identities.
- Furthermore, most of the objects created in the back-end are immutable and safely stored in the MongoDB. Only recognized and authorized users can access and mutate fields. We will also sanitize our inputs.
- For all API requests, we can check for the login information and return false if not logged in.
- In all emails, we do not expose any personal information. In addition, we can check if a user has an email before sending an email, or we can make the email a required field for signup process.

- To prevent spams, we can limit emailing non-users to just invites. The rest of the emails, we check if the recipient of the emails are the members of ibetcha. Also, the user does not get to write the content of the emails.

VII. Design Challenges

1. Challenges (Not Including Implementation)

1.1 How to check-off the user and make sure that the bet is completed? How to track the user progress if the bet is expected to be over a long period?

Potential Solutions:

- *User will select as many monitors as he desired, who will track his/her progress.* With this option, the checkoff system would be more flexible and would not depend on the monitors too much. Nevertheless, having many monitors delegates the responsibilities of the monitors, and more monitors would be likely to depend on each other and not do their checkoffs. If there are too many monitors, it would complicate the checkoff system and furthermore, we would need to have a universal checkoff strategy that is applicable for all different possibilities.

- *User will selected either 1 or 2 trusted monitors.* Having either one or two monitors will have the advantage of having trusted monitors who would be likely to do their checkoffs on time. Nevertheless, the checkoff system will depend too heavily on the responsible monitors (if they're busy) and also if the number of the monitors is too few, the bet would lack peer pressure.

- *User should be able to select only a limited number of monitors (3 to 5).* Having not too much, but not too few monitors, provides nice balance between dependence on monitors and also the checkoff strategy. If one monitor is busy, others could fill in his position by checking off the user. Furthermore, in this case, since the user's selecting his/her trusted friends, maybe only one checkoff would be sufficient to confirm user's progress.

- *User will have the option to select a different frequency for the bet and his/her progress will be tracked over milestones.* If the bet is over a long time, this provides nice flexibility for the track of the user progress and keeps the user motivated over the long-term. Furthermore, user would be able to see his/her progress over different periods.

Our choice:

- Therefore, we decided to proceed with the option of choosing only a limited number of monitors (currently, we're deciding the number to be in range from 3 to 5) with also the option to have a different frequencies for the bet. As noted above, this would make the checkoff system simple and consistent and also decentralize the responsibilities between monitors.

1.2 How to prevent users from betting trivial amounts of money?

Potential solutions:

- *The User will have a minimum amount of bounty(threshold) required per bet.* The bounty threshold should not be either too much or too small.

- *We assume that user will bet however much his resolution is worth.* Peer pressure will check back on this; Bob's friends would judge him if he bets 10 cents on his resolution to get a beach body.

Our choice:

- Currently, we're deciding the range of the bounty to be -> possibly minimum -> $5 and maximum -> $50.

1.3 How many checkoffs are needed to confirm that the user succeeded? What if there are inconsistencies in the checkoffs?

Potential Solutions:

- *Taking the majority of the checkoffs* -- If there are inconsistencies among the checkoffs, this would be a nice way to resolve the conflict. However, this would be difficult to coordinate among multiple monitors since one or more monitors may be out of reach (no Wifi, psets, vacationing in Hawaii… etc)

- *Only one checkoff is enough (either failed or succeeded)* -- Since user is selecting his/her trusted friends, it will not be very likely that the friends would be interested in falsely reporting user's progress and claim that the user failed when in fact the user succeeded his/her goal. Furthermore, there would be a limited number of monitors (either 3 or 5), thus only one checkoff would be sufficient. However, this approach faces the problem of "delegation of responsibility." Each person in the group may think "Well, another monitor can do the check-off."

Our choice:

- Choose the second option and require only one checkoff. If during either one of the frequencies, if one of the monitors say that the user did not follow his resolution, then the bet is over and user will be considered as failed at his/her bet. Similarly, if one monitors say that user actually did what he promised to do, the user will be checked off as having completed his/her duty.

1.4 How to prevent monitors from falsely checking off the user and hoarding his/her money?

Potential Solutions:

- *After the decision is made, ask other monitors for confirmation.* It would prevent any monitor from falsely checking off since others also have to agree to the decision. It would be highly unlikely that all the monitors would collude. But this option would be hard to manage since some monitors might be busy or what if all of them are busy? Then, we might need have to have another strategy for concluding other users' confirmation period and acceptance cutoff.

- *User will select his/her trusted friends as monitors.* Therefore, this will ensure that the monitors will be trustees who have genuine interest in helping their friend achieve his/her goal.

Our choice:

- We'll rely on the user's (the user who made the bet) judgement on choosing his/her own monitors and therefore, chose to follow the second option. If the user has no ibetcha friends, he/she would also have the option to invite his/her friends into ibetcha by sending them emails with appropriate ibetcha signup link.

2. Code Design Challenges

2.1 How to implement the friending feature? To whom should the users be able to send friend requests? How should they be able to send friend requests to non-ibetcha users?

Potential Solutions:

- *Users could send friend requests to only ibetcha users.* This option is necessary since if a friend is already ibetcha user, they should somehow be

able to become friends. Even though this option provides a lot of flexibility but having this option alone is very limiting. We should have extra features to attract more users.

- *We could allow users to invite their facebook friends by using Facebook Graph API and getting users' friends from his/her facebook account and sending them a* . But in order to use Facebook API, they required us to upload a complete app in order to have access. But inviting friend is one of the biggest features or our app and it was not viable to have the complete app and wait until we get approved.

- *Users could also send friend requests using email. Users could provide their friends' emails and we could send appropriate links to the provided addresses.* The link will redirect to a page where the friend can sign up and automatically become friends with the user who invited him/her. But then it could be slightly complicated since we would have to remember initial user's (inviter's) username or user id in the link and when the invitee signs up also need to use it to make the users friends. Also, in order to be consistent, we might also need to design the monitor request similarly.

- *Same as the solution above but the link will just redirect to our Login Page and the friend then will signup and become ibetcha user and send a separate friend request*. But invitee will not become friends with the user who invited him. The user would have to manually send a separate friend request to the user who invited him/her.

Our choice:

- We chose to go the third option since we had complications using Facebook Graph API. We would not have to remember the inviter's information in the link provided. The users will have the both options send a friend request either to an ibetcha user using his/her username (the other ibetcha user could either accept or reject the request) or to their friends using their email address (the invitee will signup using the link provided in the email and become ibetcha user and then, will have to send a friend request to become friends with the inviter.)

2.2 How should the users add monitors to his/her bet? How and when should users be able to send monitor requests?

Potential Solutions:

- *Send the monitor requests when creating the bet and not allow adding monitors or sending monitor requests later.* This makes betting process simple and

makes implementation easier. If the bet has less than three monitors, we can drop the bet; otherwise, the bet can be started with all the milestones generated.

- *Send the monitor requests after the bet is created*. We could let the user add monitors after the bet is created. Then, once the user initializes a bet, this approach allows the betcher to add in more monitor requests in case invited monitors have rejected the requests. The user would not have to create the bet again and invite different monitors. But it also makes the betting process a little more complicated and we would have to generate a logic for monitor addition and waiting for the response from the invited monitors.

Our choice:

- Send the monitor requests when creating the bet: The second approach makes possible a scenario in which a user makes a bet and adds monitor requests, but the friend accepts the request after start date of the bet. We decided that even though we can just ignore the fact that the friend accepted the monitor requests, it would create misunderstanding about the bet. Sending the monitor requests just during the bet creation is both simpler and cleaner for implementation. Furthermore, by the start date whether the bet is dropped or not, pending monitor requests that need approval will be dropped automatically.

2.3 How long should the minimum limit for the milestone checkoff be? Should it be 12 hours/a day/ couple days ...etc?

Potential Solutions:

- *The Milestone object should be set status "Open" and accept monitor inputs only during its effective date*, if none of the monitors checked off the Betcher, then the Milestone objects is set to be "Closed". However, this solution does not provide with much flexibility for monitors when none of them have an access to the Internet and ibetcha in particularly.

- *Another solution is to set the status of the Milestone objects to "Open" and make it accept monitor input any time after*. However, the disadvantage of this solution is that there would be no way to differentiate between currently open and late Milestones. Thus, the reminder system would be impossible to implement.

- *One solution would be to open the Milestone object for monitor input as the effective date comes, and if there is no input, then mark it as "Pending Action"*

and keep reminding the users who are monitoring bets with “pending” Milestones.

Our choice:

- In order to enable email reminder and make a checkoff system more robust to the monitors’ access to ibetcha, we decided to choose the last solution, when we make Milestone objects open for monitor input but flag them as late via setting the status “Pending Action”

2.4 When to update the milestones of a certain bet? Do the monitors need to be reminded to checkoff the betcher? The problem is that we need to run a script within the server at certain times of the day. The script will do a few things. First, it will go through all milestones of the betcher and update the status depending on what date it is. Second, if any of the monitors have failed checkoff active or pending milestones, the script will send all the monitors a reminder email to checkoff the betcher on that milestone. Third, the script will run through the bets and start the bets that have at least three monitors, drop the bets that don’t have enough monitors, and grant success status to the bets that have all milestones that are successfully checked off.

Potential Solutions:

- Cron job (node.js npm package). The advantage of using native for node script is that it has very simple interface. However, there is a risk that in the deployed version, the cron job will be unstable.
- Heroku internal scheduler. The advantage of this solution is that usage of the native heroku script will ensure the regular function call. However, even with the built-in scheduler, there is a possibility of missed tasks. Nevertheless, since our scheduler does not call on massive and computationally heavy functions (our database is relatively small at this stage), the risk is very small.

Our choice:

- We chose node-cron over heroku because of simpler interface and API.

2.5 How to enforce cron job functions to run sequentially?
The cron job is updating the status of Bet objects and states of Milestone objects based on their date information. However, since the functions implied interactions with database and mutating collections, all the changes could have run asynchronously. For example, if the Bet object was marked as “Dropped” because its drop date has passed and there were still unchecked milestones, we wanted to mark all corresponding milestones as “Closed” since none of them should be accepting any monitor input. This required changing the status of all corresponding milestones.

However, if changes would happen asynchronously, some of the milestones would have not changes and its monitors would have gotten reminders even though the bet object was dropped.

Potential Solutions:

- Manipulate collections one after another using callbacks. This solution would ensure the sequential function calls, but require very nested structure of the code. Such big chunks of code is hard to read and debug

- Use external libraries that enforce sequential function calls, so the code looks neater and more concise.

Our choice:

- We discovered a handy and well written npm package “async” that supplies all the functions that let the user control the order in which functions are called.

VIII. API Design:

USER:

METHOD	URL	Description	JSON returned
POST	/users/login	login to ibetcha	{success: true, content:user JSON} {success:false, err: error}
GET	/users	get a list of users	{success: true, content:JSON of users} {success:false, err: error}
GET	/users/current	get the currently logged in user information	{success: true, content:{user: User object, requests: [MonitorRequst objects]}} {success:false, err: error}
GET	/users/payments	get all the payments that the current user owes other people	{success: true, content:{from: [User objects], to: [User objects]}} {success:false, err: error}
GET	/users/friends/:username	get the friends of the given user	{success: true, content: [User JSONs]} {success: false, err: error}
GET	/users/logout	log out	{success: true, content:"Successfully Logged out"} {success: false, err: error}
POST	/users/new	create new user	{success: true, content:User JSON} {success:false, err: error}
POST	/users/login	log in	{success: true, content:User JSON} {success: false, err: error}
POST	/users/emailinvite	send email invites	{success: true, content: "success"} {success: false, err: error}
GET	/users/:user_id	get user profile	{success: true, content: User JSON} {success:false, err: error}

BET:

METHOD	URL	Description	JSON returned
POST	/bets	create a Bet object	{success: true, content: Bet object} {success:false, err: error}
GET	/bets	get all Bet objects of the currently logged in user	{success: true, content: [Bet object]} {success:false, err: error}
GET	/bets/:bet_id	get a Bet object	{success: true, content: Bet object} {success:false, err: error}
GET	/bets/:bet_id/milestones/pending'	get all pending Milestone objects of specified Bet	{success: true, content: [Milestone object]} {success:false, err: error}

MILESTONE:

METHOD	URL	Description	JSON returned
PUT	/milestones/milestone_id	edit a Milestone object	{success: true, content: Milestone object} {success:false, err: error}

FRIEND REQUEST:

METHOD	URL	Description	JSON returned
GET	/friendRequests	get all the friend requests of the user	{success: true, content: [FriendRequest object]} {success:false, err: error}
POST	/friendRequests/byEmail	sends friend request to the given email	{success: true, content: FriendRequest object} {success:false, err: error}
POST	/friendRequests/byUsername	sends friend request to a user with given username	{success: true, content: FriendRequest object} {success:false, err: error}
POST	/friendRequests/:requestId/accept	accepts friend request	{success: true, content: FriendRequest object} {success:false, err: error}
POST	/friendRequests/:requestId/reject	rejects friend request	{success: true, content: FriendRequest object} {success:false, err: error}

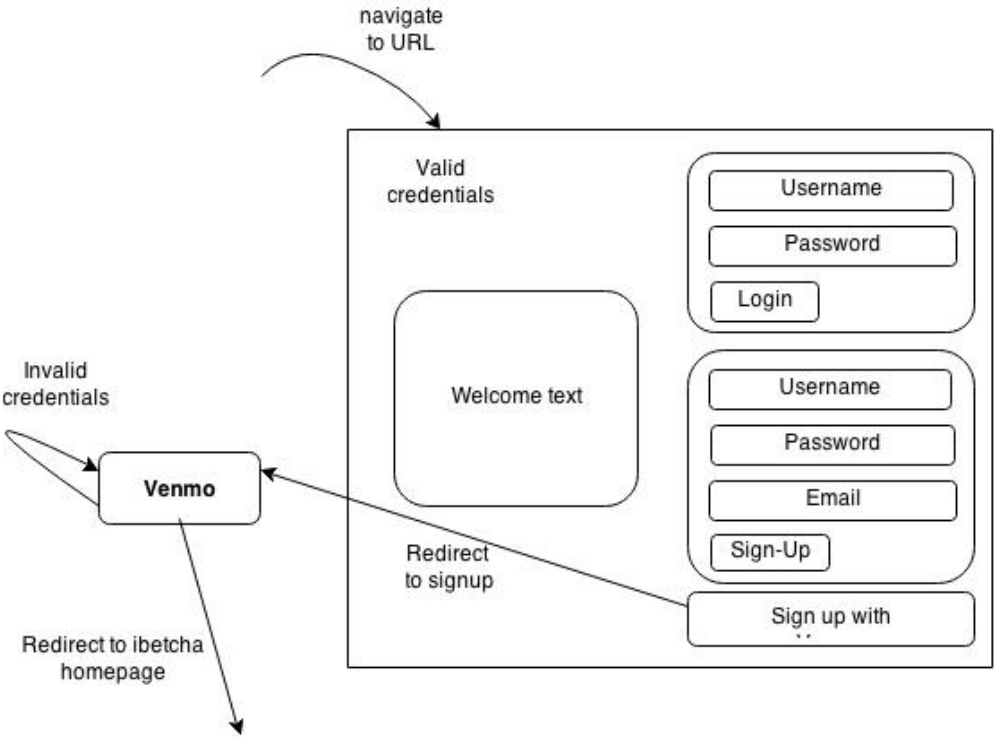
MONITOR REQUEST:

METHOD	URL	Description	JSON returned
GET	/monitorRequests/	gets all the monitor requests that currently logged in user received	{success: true, content: [MonitorRequest object]} {success:false, err: error}
POST	/monitorRequests/	creates a new monitor request	{success: true, content: MonitorRequest object} {success:false, err: error}
POST	/monitorRequests/:requestId/accept	accepts a monitor request	{success: true, content: MonitorRequest object} {success:false, err: error}
POST	/monitorRequests//:requestId/reject	rejects a monitor request	{success: true, content: MonitorRequest object} {success:false, err: error}

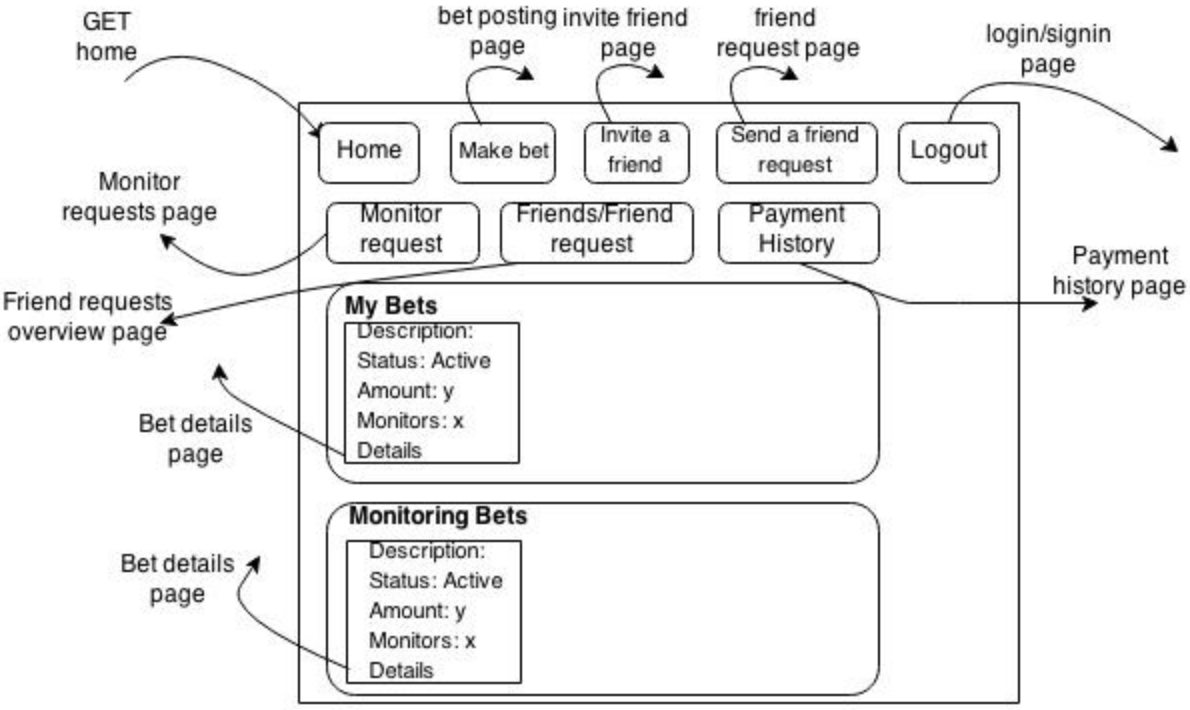
PAYMENT REQUEST:

METHOD	URL	Description	JSON returned
GET	/paymentRequests/paid/:paymentId/claim	Update the payment to display claim: payer has paid payee	{success: true, content: Payment object} {success:false, err: error}
GET	/paymentRequests/received/:paymentId	Delete the payment record to show: payee confirmed the payer's claim	{success: true, content: confirmation JSON} {success:false, err: error}

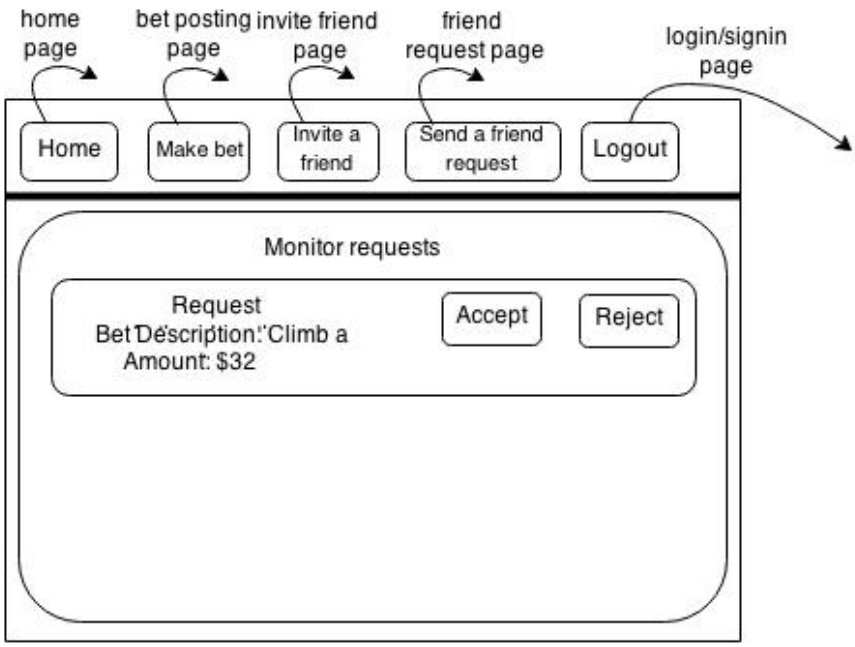
IX. Wireframes



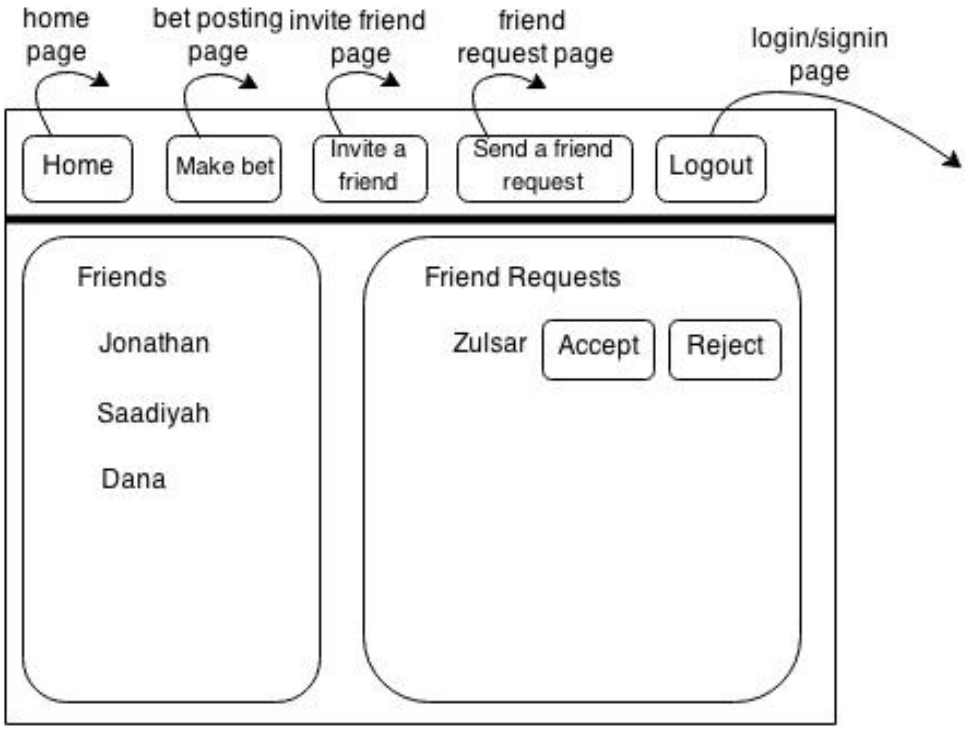
⬆ Sign-up page



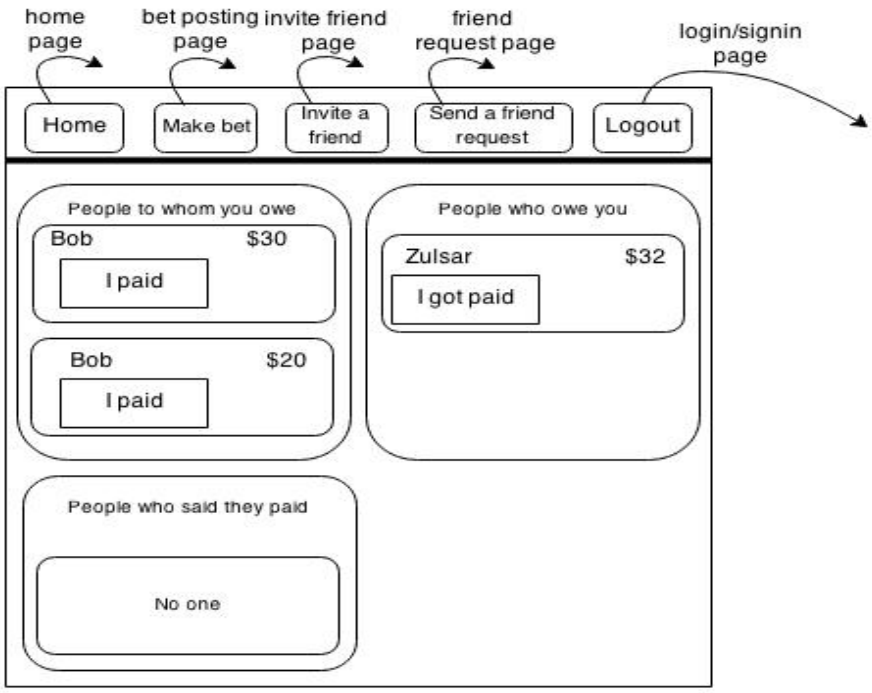
⬆ Home page



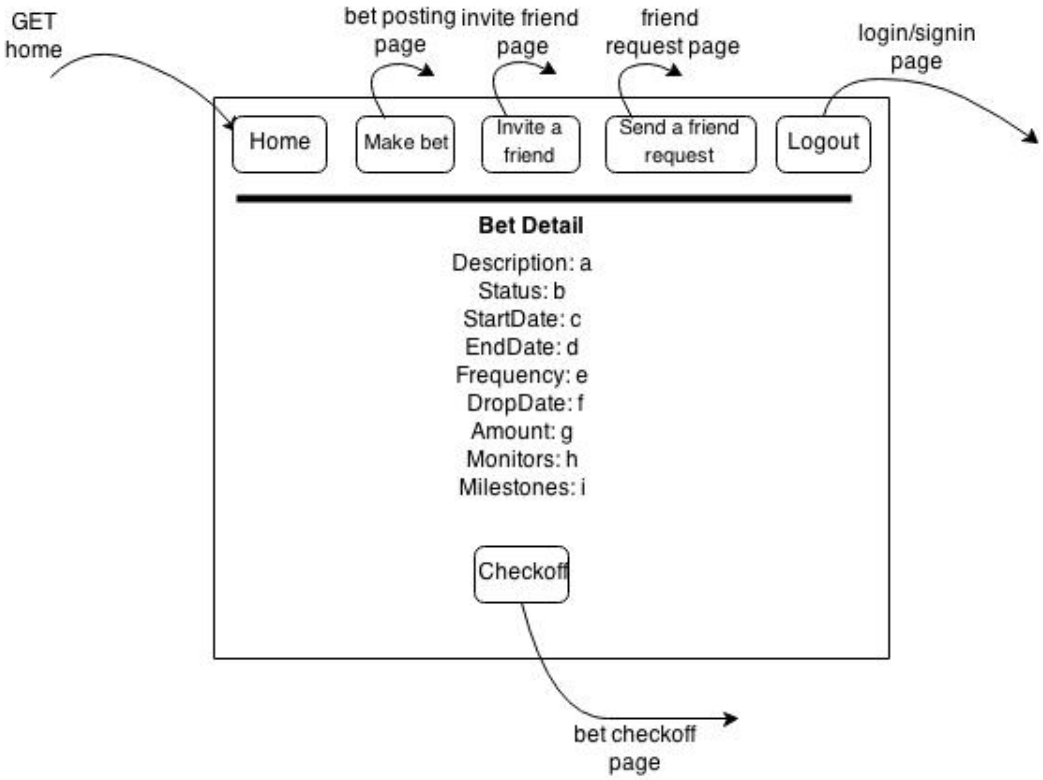
⬆ Monitor requests page



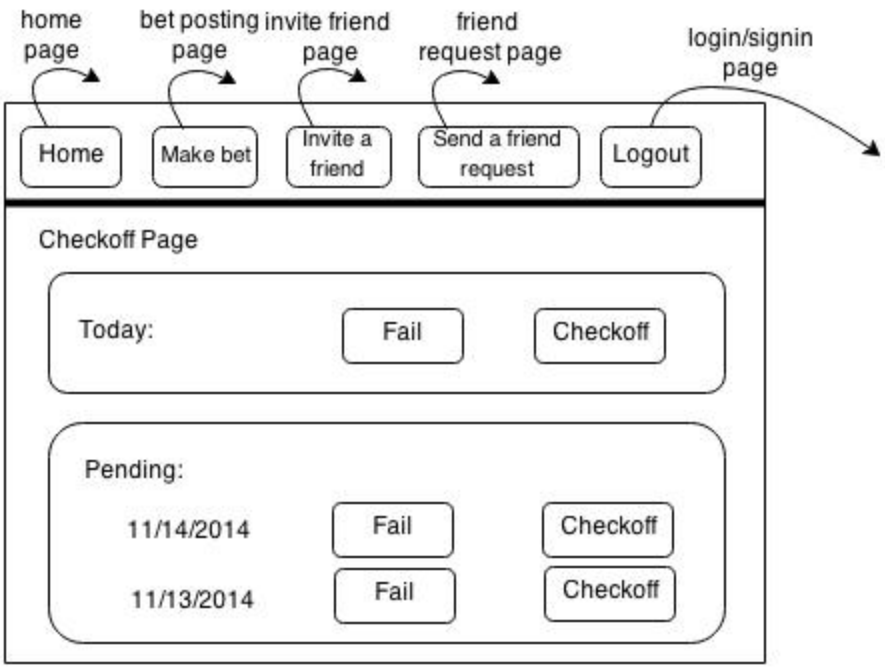
⬆ Friends requests overview page



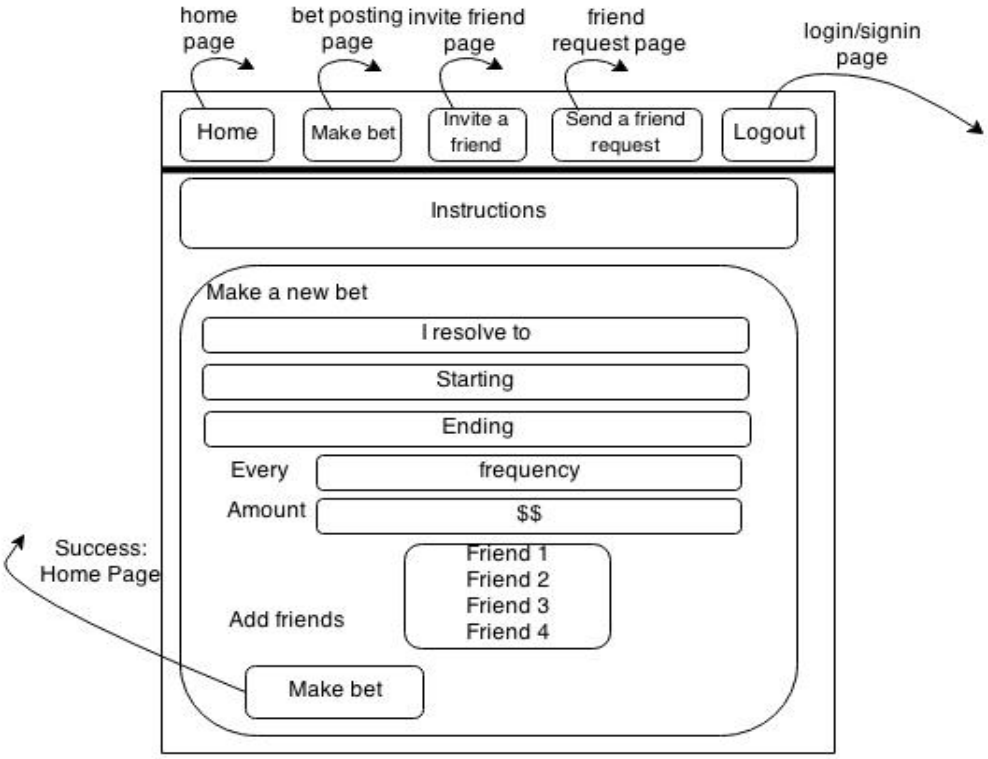
⬆ Payment history page



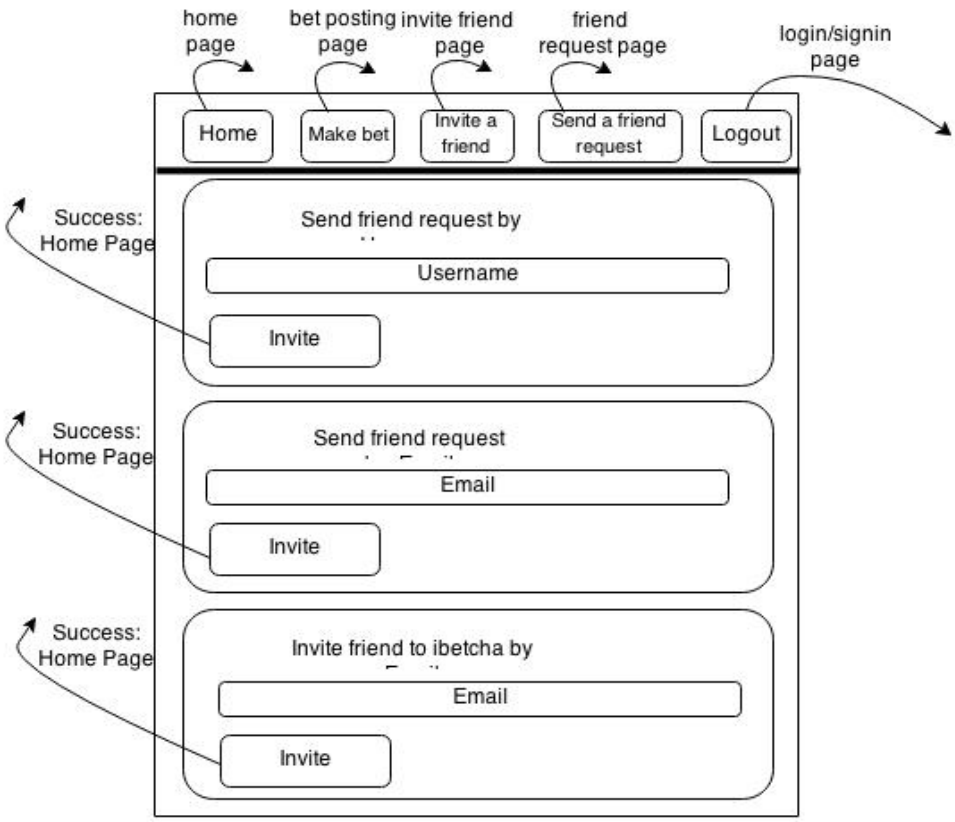
⬆ Bet-details page



⬆ Bet checkoffs page



⬆ Make bet page



⬆ Send a friend request page