6.170 Final Project - “Ibetcha”
Sunday, December 7th, 2014
Team members: Zulsar Batmunkh, Saadiyah Husnoo
Jonathan Lim, Dana Mukusheva

Team reflections

Evaluation

Good:

1. The idea of the project is very unique and not targeted to MIT community only.
2. We had a good initial design for main concepts. We did not add any substantial changes to our concepts in the entire development process.
3. We learned a lot of quirks of Javascript (issues with setDate and getDate functions, format of the Date object in MongoDB vs Javascript vs heroku, etc.).
4. We learned how to use web frameworks such as Angular.js and Passport.js.
5. We invested a good amount of time to design and implement the abstract data type, trying to think ahead of possible changes and extensions (generating milestones, changing the status of the bets).
6. We successfully catched most of the edge cases and provided efficient error handling both in the server and client sides. The server side notifies that there was an error and gives a more detailed description of the error. The client side displays descriptive messages in the UI, so user is guided in her/his interactions with our system.
7. We learned how to refactor code by reducing callback hells and separating concerns between controllers and models. Our current code is very modular and easy to read.
8. The application had an ambitious number of features, and they all worked in the way we expect.
9. Our UI is good-looking and not crowded with unnecessary or misleading details.
10. We had efficient team meetings with positive atmosphere by keeping to work ethics norms and successfully brainstorming and discussing different ideas and potential solutions.
11. We communicated with staff members whenever all of our teammates were unsure about some aspects or had questions regarding our app features (e.g., cron job, datat design).

Bad:

1. We initially wrote very long functions and router methods with a lot of callbacks. It was hard to understand each others code as a result and it took some time to refactor code and then debug all the created mistakes.
2. We haven’t separated models from controllers right away, so our router methods contained a lot of code related to model functions and were a little bit messy.

3. We did not incorporate any external APIs into our project. In the beginning, we planned to use Facebook and Venmo APIs in our feature to make our app very cool, but sadly, they required our app to be already complete and fully working in order to get approval. Therefore, it was impossible to incorporate them from the beginning and we had to use different alternatives.
4. According to Venmo, our app is considered as illegal since it involves betting and did not let us use their API. We lost a lot of time communicating with them and explaining the idea of our app and then trying to convince them to let us use their API. However, we failed and got rejected.
5. Consequently, it took a while to develop our own payment mocking system and integrate it with the existing app.
6. Since our original plan did not work out, we spent a lot of time in redesigning our features and deciding on what to do based within the deadline using the resources that we had.
7. Wiser time management would have improved our coding schedule and prevented us from pulling an all-nighter.

Lessons learned

1. Invest more time in R&D stage before committing to the coding stage.
   Our initial project idea relied on the Venmo and Facebook APIs. Simple API functions and the availability of corresponding node packages misled us in the design stage. More thorough research on the terms of use for each API would have revealed the necessity to have a complete app or legal issue related to the betting.
2. Have a detailed project schedule. Detailed planning from the beginning for each milestone of the project made all of our key objectives clear and prevented us from spending too much time on a particular feature or a bug.
3. Modularize code from the first iteration of development. The code that initially contains very long and nested functions and later on modularized is more likely to contain bugs. We ended up spending a lot of time debugging which would have been reduced by modularization.
4. Communicating with each other and keeping everyone updated so that everyone's always on the same page and we can solve issues in time.
   When some members are busy and all working separately, effective communication was really helpful in keeping all members informed on what each other has been working and whether if they had encountered a serious issue that involved redesigning of our project.
5. Ask for feedback as often as possible. It is very helpful to know the opinion of the person from the outside of the team since she/he can detect some missed bugs or give a valuable advice on UI.
6. Get testing from an outsider. A user who is not familiar with the app should be able to navigate through the functionalities without confusion. Otherwise, UI/UX is not good enough.