Requestr - MVP Reflection

What We Learned From It

- Working in a team is very different from working alone because there are a lot more variables that you can't control, such as work contribution, meetings, outside factors, etc.
- We learned to not underestimate time constraints.
- Web-programming is hard… We were very confident with how the application should be structured at the beginning. However we had to deal with annoyances, e.g. how javascript imports scripts, which did not mesh with how we wanted to deploy the code. This forced us to consider how to restructure the application while maintaining good programming principles.

What Went Well

- We were able to schedule/hold consistent meetings where everybody showed up.
- Everyone in the team held their weight. No one was flaking when it came to work.
- We were able to tackle huge problems/issues relatively quickly
- Each person was able to do his assigned role relatively well; the problems we encountered did not arise because of a person's programming skills but rather small details regarding the frameworks used.

What Could have Been Improved

- Our MVP was deemed "too simplistic" and we needed more features to make it a viable product.
- We didn't really have a concrete plan on division of labor in the beginning
- We underestimated the time needed to complete the MVP.
    - This was related to not having longer meetings/more meetings.
    - We spent too much time debugging.
        - Some of the bugs were basic problems with communication between people working on different modules.
    - We had to rush a lot near the end to add features to the MVP (tags and discussion).
    - We should've spent more time considering what would be critical to the application, in a sense making the MVP a little polished rather than crud.
- The code was not the best designed due to lack of time.
    - i.e. "style=" everywhere, violated MVC model when displaying "my requests" in public/requests.js
- The MVP still feels incomplete (no delete button, no view my accepted requests, etc.).
- We could have better assigned the roles instead of frontend/backend. amount of time spent for each person was lopsided (i.e. some people did a lot more work than others).
- We didn't utilize pair-programming until later (it was useful).

[Update]

Everything that wasn't specific (targeted towards MVP) still applies to our time spent during the final stages. In addition, there is the following:

What We Learned From It

- We learned a lot about the importance of communication and leadership. In particular, a team without either of those will perform poorly. As such, there should've been a person that took initiative in ensuring deadlines and conditions were met.
- We learned not to underestimate code collision. At times, there would be people that were working on different features on the same file and it led to some nasty conflicts (it was a huge file). We ended up only demoing most of the features because neither of the users had all of the developed features (due to features getting overwritten).
- In relation to above, we learned to separate huge files quickly (we had only 1 single controller frontend javascript file before we separated it later) before it can lead to problems.

What Went Well

- We made the right choice in not choosing to use any web frameworks as the basis of our final project (i.e. angularJS and the like). This kept our code relatively simple and made it so that we could understand all of our code better
- In addition, the features that we chose to implement for complexity, messaging and venmo integration, were very useful and appropriate for our requests app. In particular, Venmo integration allowed for users to pay each other with rewards in a safe and secure manner, which is a great thing for an app like Requestr.
- Everyone came out of this with even more experience in web development and software development in general, dealing with issues both technical and non-technical that aren't just coding. It was like an extended hackathon that taught us how to handle pressure and deadlines.

What Could have Been Improved

- There was a lack of communication within our team. When deadlines approached, no one took the initiative to ensure that whatever needed to be submitted for that deadline was delivered on time. For email, responses took, on average, half a day, despite there being urgency. As a result, teamwork suffered and meetings were only held sparsely and only until right before the deadlines. The lack of communication also led to several other occurrences, such as causing teammates to meet on campus when there was no verbal agreement of doing so, the general lack of responsibility concerning what features should've been implemented but didn't (and how to implement it) as well as a horrible collision of code (overwriting features) when two people worked on the same file independently at the same time.
- In relation to above, there seemed to be a general lack of disregard towards the importance of time management and product delivery, which also attributed to everything

above. Emails that I sent out (William Wong) about important deadlines did not receive responses until almost half a day later.

- In relation to the above 2 bullet points, we needed someone in the team to be a leader, someone that took initiative and ensured that things were going properly on time on our schedule. Instead, we didn't have that and it led to very poor performance in the final product delivery, both time-wise and content-wise.
- There was no code review done at all (mostly due to time constraints) and there was no code quality check either.
- Overall, the lack of team dynamics led to a rushed, final codebase that was still lacking in a few features and didn't have cohesion. I would've preferred it if we had met more often and everyone was generally more active in making sure the logistics and teamwork were there and not just code.

What We Could Do Differently In the Future

- We definitely need to meet more often and spend more time on our project
    - On that note, we should be very aware of the time it takes to develop parts of our applications and not underestimate it.
- We should use pair-programming more often. It helps one person catch errors that the other person didn't see.
    - Pair programming alleviates the issue of integration problems because there's fewer hands coding.
- We need to better divide our roles such that each person has relatively the same amount of work to do.
    - Don't let one person handle a majority of the work on a single part of the application.

David Wu's Peer Review

William Wong

- Will spent a lot of time working on the project. We originally split the work frontend and backend and he ended up having to do much more than us on the frontend and did a great job. The code got a little more messy without comments after he had to work on it for so long, but that is fine.

Rishad Rahman

- Rishad wrote almost all the functions for the schemas. I need to work on communicating better with him so we don't get so many bugs with slightly different spellings of functions or

incorrect parameters being passed between modules. All of the database functions he wrote were packaged together in one file at the beginning which was hard to read. He later split it up though.

Johnathan Root

- Johnathan got what he planned to work on done efficiently. He was always first to arrive for meetings despite living across the bridge in Boston. He actively participated in discussions for every decision we made. When he had to leave early from meetings, he made sure to always check-in and stay in contact if we needed him.

William Wong's Peer Review

Rishad Rahman

- He's a good devil's advocate, but sometimes tried to find unfeasible solutions to problems that had better alternatives. He's a great idea thinker and his backend/database code was solid, but he could have worked on testing/proofreading code more before integrating it with others' (i.e. writes great standalone code, but not necessarily the best to integrate due to trivial bugs, etc.).

David Wu

- When writing code to be integrated, there seems to be a lot of errors (albeit usually simple ones). He wasn't that familiar with how node.js and express worked despite being the backend person. He's easy to communicate with and bounce ideas off of. He's a bit passive sometimes during decision-making. He's otherwise a solid coder.

Johnathon Root

- He always came the earliest to meetings out of the team despite traveling the farthest. Sometimes seemed to go off and do his own thing or was silent. However, he's a very solid worker/active contributor to the team. He's great to bounce ideas off of, but could work on debugging skills more.

Rishad Rahman's Peer Review

David Wu

- I worked with David for the most part since he was using the schema methods I created from the routes. At the beginning we had some trouble because we were writing code separately, and tested at the end, hence it made things hard to debug when we needed to integrate. What we should've done was work on related functions side by side and test those functions at the same time i.e. modular testing. We did this when we were working on tags and it worked well; we were able to see the problems with each other's code relatively easily.

Jonathon Root

- I didn't really get much input from Jon. It was hard to tell whether he felt good about how the implementation was going on or not, but that may be a result of us not working on

similar aspects of the project. I should discuss with him about the functionality he is working on and how I could assist from my end.

William Wong

- Will was very experienced with certain patterns on the front end but when problems came up, it seemed like a lot of my suggestions were futile. I wish that I was able to help more but it seemed like he knew what he was doing so I hesitated to persist in alternative design ideas as it might've resulted in wasted time. It would've been better to discuss these issues at the beginning rather than trying to contribute after some implementation was done. I will focus on this in the future.

Johnathon Root's Peer Review

William Wong

- Hard to communicate as much when it came down to the wire (i.e. working alone on a feature with no other work available). Great to pair-program with. Dozed off a little bit during the meetings, but that might have been due to the lack of sleep. Could work on programming less messier.

David Wu

- Didn't really work with that much since our features were the most disjoint/orthogonal. Seemed like a good coder and definitely had insightful thoughts to share for app design. Maybe should have worked on programming more or taking on more programming roles himself. Has no technical issues.

Rishad Rahman

- Seems like a competent worker but idle at times. Could voice ideas a bit more and try to implement some other things that are not just what he is comfortable with, in this case the schema methods.