Vikas: Let’s start with Data Model. What’s the Update model?

EJ: Students giving updates about their progress such as pictures, text, videos.

Roman: Collaborators?

Vikas: You could create another relation between the student and the project, You can have a box between students and projects. Or you can have different number of arrows which each conveys different types of relationships.

EJ: Advisors?

Vikas: If you allow users who are not in the system to be collaborator or Advisor, then your code is going to be a bit messier. Also, you have to think about all the cases. You also have cover edge cases so that the collaborators/advisors will be linked to everything. It’s not something you want to do on an MVP.

Richard: What about just providing their emails and make it as sleeping accounts

Vikas: That’s another valid approach. You’ll have to look up if the person has a sleeping account whenever they try to register.

Roman: How to implement different levels of authorization?

Vikas: Standard: Use before_actions: Various before_actions. There’s also a gem called kenken.

Roman: What about the model part?

Christian: creator_id, writer_id

Vikas: Write functions that implicitly exclude other types of users.

Roman: What if the student graduates?

Vikas: That’s valid.  Throw in a check function if the user has passed the graduation. I did kickstarter for scholarships. Require a graduation date and check the year every time. When demoing you are going to have to change the graduation date in the database and refresh and re-log in.

Roman: What did you do for payments?

Vikas: We didn’t do payments. Actually, we did. But it takes some learning time. Prioritize it later if you can.

Vikas: Wait a second. Oh we did implement payments. My bad.

Richard: Just use email for advisors?

Vikas: Then do it just as an attribute of the Project model.

Roman: (Gives a description about the MVP)

Vikas: Sounds reasonable. The UI doesn't have to be beautiful. It has to be functional. If Ajax is not core to your MVP, you should prioritize other back-end stuff.

Vikas: How will you be handling users?

Christian: Devise

Christian: I have a question though. What do you suggest I do to represent the two types of users:students and alums. Should I create two separate tables? Or use STI (single table inheritance)

Vikas: So that was actually what I was curious about. Normally on a high scale you would avoid STI if the types are too different because you would be left with alot of null columns, but since you are focusing only on this project you can include this in design decisions that you felt that it was unecessary to worry about table space.

Richard: What if we want to scale?

Vikas: Then you wouldn't want to use rails. Expanding the API on different machines, etc.

Richard: How to make bookmark for dashboard?

Vikas: You could use partial views to reuse the selection tool for projects. You can force the users to bookmark. Put the bookmarks into the database. Just create a new relation between users to projects.

Vikas: Don't spend too much time on the production of your app.