11/21

- Should have meeting agenda ready for the next time, push before meeting
- Model design, should we store data both ways?
    - If one direction is used more often, just store that direction. If just 1 level, 1 direction is fine. But if there are multiple levels, it's better to store both ways.
- Is it okay to use external modules? Yes.
- Anonymous name
    - If it's just used as a field in another model, probably no need to make a separate model for that
    - Have a list of anonymous names used for a babble stored as a field in each babble to eliminate duplicates?
- Babble
    - addBabble() function does not callback newBabble, new babble is not undefined? Print out the error, object might not be following the schema properly. Check database.
- When we return things? Do we want to populate or not?
    - Don't worry about performance, worry about functionality first. You should populate everything if needed.
- How should we decide which module to put a function?
    - Just need to decide amongst ourselves
    - Separation of concerns
    - Have consistency over style
- Models, importing other models?
    - Should have fewer dependencies
- Bluebird, mongoose promise
    - How to use promise: minimizing callbacks.
    - Promise definitions: won't run the code until promise is done
    - Can you get into a deadlock? Yes.
    - It's better to chain promises, treating code as synchronous
    - Are there any other advantages: code is simpler, a matter of preference
- Should we make sure inputs are valid in controller, and have spec in model to assume inputs are valid?
    - Have consistency across all the files
- Fritter-react code
    - Attach the methods to the schema? More standard.
    - Helper controller objects, schema, separate them?
    - Code styling preference
- Is it okay to look into the database while testing? Yes.
- TO-DO
    - MVP is due Wednesday, optional to demo in person
    - Meeting again next Monday

11/28

- DEMO
    - Need more UI changes
        - Comments: build a component that can be re-used, make a react component. Have an anonymous selector, an HTML component. Never mind, it looks good.
- Have the same IDs, not “==”, but toString() is “==”?
    - If toString() works, then just use toString()
- “Called done() too many times”?
    - Match the spec, the tests should be consistent with the spec
- Promise vs Normal code?
    - Mixed right now. If it’s consistent within the whole block of code, e.g, in models, in routes. Promise in AnonymousName models and routes. Callbacks in routes.
    - Not the highest priority
    - But style-wise, promise looks better
    - Should be fine, as long as the errors are handled properly and functionally it’s fine. Readability is about 2% of the grade, if readable, then good.
    - Revised if needed
- Revised design due Wed?
    - Design document update
    - Feedbacks on 11/28
    - Update or add
    - Made updates on design since MVP, should reflect that in the design doc. Address changes
    - If there are issues, Vahid will be responsive
- Project revision vs Final code hand-in?
    - No project revision, but demo on 12/7
    - Current calendar on stellar is good
- Error?
    - Something goes wrong, we should tell the client. Errors should be propagated back to the client.
    - Client should not know secret things about the database. “Rep exposure”
    - Something has gone wrong, but the message content should be discrete.
    - 1: send 500 error, then display a message
    - 2: send the error string
    - Advantage of 1 over the other: if you can tell the user what to do, it’ll be better. Make specific error message, easiest to send from the server.
    - Error propagation: if there is an error in schema or model, (err, result), use the same design through the code. err is a string. In route handler, if (err) send that error back, but maybe not the actual error because it’s too specific sometimes.
    - Console.log?
        - Have some sort of flag to indicate whether you are in development or not.
        - Route everything through some function.

        - Example code: error handling. There is a flag there.
        - Development vs Production
        - Console.log is also a good way
- Live update
    - Every 5s, check updates, if notifications not clicked then no more updates.
        - Don't worry about sending too many queries. Every 5s should be fine because the request is fast. Every 2s or 1s should be fine.
    - Counting number of collections in the database. Is this too much?
        - In the session cookie, store: when is the last time you got all the babbles?
        - Look at the most recent babble? Still need to sort all babbles to find it.
        - Count is fine?
        - Is there a way to get latest babble without looking through all the babbles? Do not think so.
        - Count is fine.
    - Comment section.
        - At this point, maybe it's better to switch to sockets.
        - Learning curve is steep.
            - Need to use a third party library
            - Need to change code on the server side??
        - Hard decision to make…
        - If you can make this work, then this is easier. But sockets will definitely work.
        - To do the same we did with babble for comment, we have to do so for all the babbles, which might seem redundant most of the times.
    - Worry about functionality over performance first. Don't worry about too many calls at first.
    - Think about switching to sockets later.
    - Watch out for update concurrency issues with sockets. Send updates through sockets, other read and write can use AJAX.
    - Count of comments on each babble should be fine.

12/05

- Email verification code location? User model? Routes?
    - It's okay for now.
    - Two general strategies: a separate email file; put in a environment variable
    - Can also set a flag
    - Put instructions in the README for configuration purposes, need to npm install the extra email verification module\
- Email verification test: treat it like a front-end test.
    - For things you can't check with code, test manually
    - Probably not able to write unit tests for email verification
    - Worry about if an email sending action is successful, rather than if the email is actually sent. Or more like, pretend the email is sent and check the rest of the actions.
    - Leave the test cases as it is, but clarify in the README
- Email verification vs. register
    - Include register() as a utility function in tests. Separation of testing code and production code
- Sending email as our personal emails, masking it with babble@mit.edu. When we deploy it, what do we do?
    - Maybe register another email rather than using personal emails
    - MIT email that we can use to send? Ask IS&T. Set up mail client to send emails from.
    - What we have is okay too. We can just create another email
- Error messages, propagating to routes? (NVM, what does this actually mean?)
- Trying to update the leaderboard and updateLimits weekly, HOW?
    - Have another instance of leaderboard students in the schema
    - When you pull the top 10, just pull all the documents.
    - Check the last time you updated the list, if already a week, then re-pull the top 10 students from users and update the schema documents
    - Another strategy: create model/schema for update events. If past a week, create a new event. Each user has a rank. When you create the update event, you create new ranks.
    - REAL WORLD: cron. Runs in the background for periodically events.
- Testing with while loops
    - Vahid's guess: while loop is running, starting the functions every iteration, by the time it gets to the bottom, none of the above have finished.
    - Use console.log to see if the previous statements have finished.
    - commentUser is not printing, probably not finishing
    - Create functional for loop that does the same thing. Each iteration calls back the previous iteration. Either calls the next iteration or done().
    - Asynchronous loops: running the next one before the previous ones finish
    - Vahid will send the link
- babble.addComment, comment.addComment

    - They are doubly linked. When updated, need to update both sides. Is this okay? How to make this better?
    - It's not that bad to import this model into another model. It's probably better to import another model than doing doubly linked methods.
    - Import comment into babble. Create comment in babble, set babble ID as the current babble.
- Weds: get feedbacks. No more design decisions