# Deliverable: Project Evaluation Document

Our original planning worked out well overall, although there were a few times when better planning may have allowed us to get tasks done by certain deadlines without having to make last-minute scrambles. For example, during the breaks of the semester such as fall break and Thanksgiving break, there were times when we didn't have big milestones due until two weeks later and we ended up taking one week off and working the next week to prepare for the next milestone. However, we would usually end up having a lot to accomplish within one week, especially since our project members would have other extracurriculars and classes to work on during that week, which sometimes led to last-day scrambles. With more planning and working steadily on it constantly, we might not have had to rush as much right before our advisor meetings. In addition, one large and slightly unpleasant surprise occurred during our beta demonstration milestone, in which we were making changes to our application on Render even within an hour before the demonstration. Although we had thought the Render application was ready by the time we had our advisor try to sign onto the application on Render as a new user, the application encountered an error as soon as he tried. We were able to have him log in as an existing user instead in order to show him our app's functionality, but it definitely taught us to avoid making any large last minute changes up to an hour before any demonstration or meeting in case something went wrong and we didn't have time to fix it.

We made many good choices throughout the building of our application that worked out well, such as live updating of pages and in-app and SMS notifications. We were also able to implement a clean calendar UI that works as intended and build a backwards-compatible app that allows us to get users to make necessary changes to their data without having to reinitialize the entire database from empty. On the other hand, some changes were less intuitive and took longer to implement. For example, we spent a while trying to figure out which UI to use for our calendar/scheduling, when the user wanted to look at a schedule for their matches or pick times to match with someone else at. At first, we looked at using the Toast Calendar UI, which is a vanilla JavaScript implementation of a calendar. However, we eventually realized that the calendar was too complicated and wouldn't do exactly what we wanted it to do. For example, the calendar was meant to show once-occurring events and to move from week to week, which meant that adding an event meant that it wouldn't keep showing up, although we wanted users to be able to set a weekly gym session with their partner to encourage them to keep going to the gym. We instead just wanted a calendar without dates that would remain the same week-to-week. The Toast Calendar UI was almost too extensive and yet not extensive enough–there was no way to remove the dates and set recurring events in its documentation, and the source JS file had way too many functions and code to sort through and figure out what to change to get what we wanted. As a result, we ended up having to research for another UI to implement, and we settled for one that had defined functionality aside from a few basic functions, which allowed us to modify the UI to do what we wanted based on its more understandable code. Overall, the process would have been much faster if we had searched for an easier UI to use, but we spent lots of time on trying to use the Toast UI since we thought it looked prettier (which it did). This also affected our database–before we made the calendar UI change, we stored each user's schedules in a schedules table in our database. However, the new UI that we ended up using represented schedules using a JSON string format, which doesn't require an entire database table to be stored in. Instead, the calendars would have been fine as a column in the other database tables, like the user and request tables.

That being said, there were a lot of other pleasant surprises, such as when we finally got a good calendar UI implementation working, or when we figured out how to implement the matchmaking algorithm, or when we figured out how to implement modals that didn't accidentally pop up behind other modals. Our biggest challenges included the calendar, profile set up and validation, live updating of pages with Ajax and database refreshes, and notifications, but we were able to work through them all and didn't have to leave a feature undone that we aimed to complete at the beginning.

If we were to do a similar project in the future, we would likely try to get user feedback throughout our application building process, not just mostly at the end. We did two main periods of user evaluations–one was getting user feedback about our Figma designs and getting a general survey about application ideas, such as what type of gym interests would users like to be matched by during the matchmaking process, and the other was more towards the end of the semester after our beta demonstration, when we had most of the functionality completed by then. It would have been more helpful throughout the semester to get recurring feedback from the same users, even after smaller milestones like the basic functionality. Instead of testing the completed functionality all at once, it might have been better to get feedback on a few functions at a time (ie. the Find a Buddy page, notifications, modifying incoming requests, etc.)

From this entire process, we learned that user designs and interfaces can be tricky–something that looks good to you might not look as good to someone else. Getting feedback at the end can also be tricky as well, since it's possible that at the end some functionality or design that you've worked on the entire time ends up being unnecessary or useless. Working with different languages wasn't too difficult, since we mostly used the tech stack that was taught in lectures (Jinja, Flask, HTML, CSS, and Bootstrap), although working with JavaScript was sometimes a bit difficult. Systems also didn't pose much of a problem to us, although we ran into some (small) problems with PostgreSQL. Testing was also a little intensive, since there were many edge cases where there could be errors (for example, a user might accept a request that has overlapping times with other requests that would cancel them, or a user might click on the Modify button on the Incoming requests modal without having modified any times first). Working with so many schedules for each user and allowing users to modify times and send them back to each other can lead to many edge cases, which led to us spending lots of time alerting the user whenever they happened.

If we had more time, we would like to implement mobile compatibility, since the application also makes sense as a phone app that Princeton students could pull up and use anywhere, especially at Dillon if they need to confirm a workout time or buddy. We could also attempt to simplify our UI currently, since new users might find it intimidating or time-exhausting to go through the process of selecting times, matching with others, and waiting for their responses. These are all ideas we could implement in the future if we had more time or were to redo the project, but in the meantime, we definitely took away lessons about planning a project, getting user evaluations, and implementing a system that can handle errors gracefully with testing.