Sprint 1 Retrospective Document

Eric Aguilera, Justin Bonner, Carson Harmon, Dan Zheng

What went well?

During sprint 1, we finished all of the user stories listed in our sprint 1 planning document. We even finished some stretch goals and are ready to hit the ground running for sprint 2.

#	Task description	Estimated Time	Owner
1	Set up back end server and deploy to Heroku	1 hr	Dan
2	Set up database	1 hr	Dan
3	Create database model for users	2 hrs	Eric
4	Add back-end controller for user account creation with input validation	2 hrs	Justin
5	Add front-end view for user account creation with input validation	2 hrs	Carson

Completed. We successfully deployed our Node.js server to Heroku: pushing updates is simple and convenient. We used the Heroku add-on JawsDB MySQL to host our database. A `Sequelize` database model for users was created along with the associated front-end and back-end controllers.

#	Task description	Estimated Time	Owner
1	Implement user authentication using username and password	3 hrs	Justin
2	Add back-end authentication middleware	3 hrs	Dan
3	Add front-end view for login	2 hrs	Eric

Completed. User authentication works successfully. In addition, we used `express-flash` and `express-validator` to validate the login form and display feedback notifications. Password are hashed and salted using `bcrypt`. We also added support for external login providers such as Google and Facebook.

#	Task description	Estimated Time	Owner
1	Create database model for recovery tokens	1 hr	Eric
2	Add back-end controller for recovery process	3 hrs	Dan
3	Add front-end interfaces for recovery process	3 hrs	Justin

Completed. The `Sequelize` model for recovery tokens was successfully created. Using the crypto package, we generate recovery tokens for users that expire after one hour. Using nodemailer, can email the user a link, with the recovery token in the url, to reset their

password. Upon visiting the link. a user may reset their password until the token is no longer valid.

#	Task description	Estimated Time	Owner
1	Add back-end controller for adding/updating profile information	2 hrs	Carson
2	Add front-end view for adding/updating profile information	2 hrs	Carson

Completed. After logging in, users can visit their profile page to edit their user information. From the profile page, a user can modify the fields they want and submit the changes to the database.

#	Task description	Estimated Time	Owner
1	Create database model for courses	1 hr	Eric
2	Add back-end logic that uses Purdue course API (purdue.io) to populate database with courses	3 hrs	Dan
3	Add back-end controller for searching courses	3 hrs	Justin
4	Add front-end view for searching courses	3 hrs	Carson

Completed. The courses page contains a search bar which was implemented using Algolia to provide a convenient way to autocomplete course names. From the courses page, a user can input their Purdue career account credentials to automatically pull their courses using the Purdue.io API. Users can also manually input their courses using a search bar.

#	Task description	Estimated Time	Owner
1	Add back-end controller for adding a course to account	1 hr	Justin
2	Add front-end interface for adding a course to account	1 hr	Justin

Completed. From the courses page, a user can search for their class in the search bar and manually add their courses.

#	Task description	Estimated Time	Owner
1	Add back-end controller for returning user courses	1 hr	Eric
2	Add front-end view for viewing user courses	1 hr	Justin

Completed. From the courses page, a user can view a list of all their courses. For each course, there is an info button and a deletion button.

#	Task description	Estimated Time	Owner
1	Add back-end controller for viewing course students	1 hr	Carson
2	Add front-end interface for viewing course students	1 hr	Carson

Completed. From the courses pages, a user can click on a course to navigate to a course page, which lists specific information about the course as well as a list of users in the course.

#	Task description	Estimated Time	Owner
1	Create database model for groups	1 hr	Eric
2	Add back-end controller for creating a group	3 hrs	Eric
3	Add front-end interface for creating a group	3 hrs	Eric

Completed. The `Sequelize` model for groups was successfully created. To create a group, a user can navigate to the chats page and enter a corresponding chat name and description. After creating the chat, their list of chats will be updated. A user can click on any of their chats to view their members and manage the chat.

#	Task description	Estimated Time	Owner
1	Add back-end controller for deleting a group	1 hr	Carson
2	Add front-end interface for deleting a group	1 hr	Carson

Completed. From a chat page, a user can delete the chat and all of its associated data. In the next sprint, we plan on restricting who will be allowed to delete a chat to the oldest member of the chat.

#	Task description	Estimated Time	Owner
1	Add back-end controller for inviting people to a group	2 hrs	Dan
2	Add front-end interface for inviting people to a group	2 hrs	Dan

Completed. From a chat page, a user can enter an email of another user that they want to invite. Once invited, the invited user will show up in the list of chat members. If the email doesnt exist or the user is already in the chat, an error notification will be shown.

#	Task description	Estimated Time	Owner
1	Add back-end controller for leaving a group	1 hr	Dan
2	Add front-end interface for leaving a group	1 hr	Dan

Completed. From a chat page, a user can leave the chat. If they are the last person in the chat, the chat will be deleted.

What didn't go well?

In our sprint 1 planning document, we assigned parts of features between different team members. This created awkward problems when one team member couldn't start working because another part of a feature had to be done first. This happened frequently and limited the efficiency of our team.

How should we improve?

In the future, we will improve our sprint planning by assigning full features to team members rather than partial features. During sprint 1, we found that it was difficult to collaborate at times because multiple members were assigned to different aspects of the same feature: for example, one person may have worked on front-end while the other worked on back-end. However, this was not ideal because we had to wait on each other to complete tasks. In future sprints, it will be more natural for one person to complete all aspects of a feature.