Lauren Hibbs
The Juicerz
2/4/20

Tech Debt

Martin Fowler suggests calculating tech debt by having teams estimate how much “interest” was paid against problem areas of the system. For example, a task took 5 days but would have taken 4 days if the prerequisite problems with the system had been solved. I have estimated the extra work that would go into the current sprint if we were to reach our acceptance criteria for each task.

JUIC-106 Store a Log when Session Starts 0 extra work
JUIC-98 Interact with Slider to Control Charging (Front end) 0 extra work
JUIC-107 Detect disconnected charge 0 extra work
JUIC-114 Charging Graph 0 extra work
JUIC-109 Login and Credentials are Remembered 0 extra work
JUIC-63 Settings Page 2 hrs extra

Currently, the changes for the front end are not merged together. I am including all the work of merging the changes from the last sprint into this single front end task.

JUIC-108 Submit setup configurations 3 hrs extra

The team still has insufficient knowledge of the API gateway. If endpoints were setup correctly, it would be easy to add a put to the user settings table. However, we are still unsure how to create get/put/post operations on a single endpoint which should be the minimum acceptance criteria for this task.

JUIC-110 Add deviceIds to a database (admin) 4 hrs extra

If past tasks had been fully completed, it would be easy to make a new endpoint to add deviceIds. See JUIC-108.

JUIC-112 Submit JUIC-108 securely 8 hrs extra

The entirety of this task is to make up for past security technical debt, so I have included the entire estimate of the task as technical debt.

JUIC-79 Receive an IOS Push Notification 6 hrs extra

Researching and designing architecture to receive a push notification should have been included in the vertical prototype. Implementation of this feature is an appropriate task for this sprint, so I have not included the entire estimate as extra work, only the time to research and design.

An evaluation of the metrics above shows that we have not accumulated much technical debt on the front end. This is likely because our front end is not sufficiently complex to have code smells or require refactoring. However, our changes from the last sprint are still unmerged which is required before we go further in the project.

On the backend, there is a lot of technical debt created by implementing previous tasks quickly in ways that do not reuse code. Before continuing, we must consolidate the different

endpoints that we have in our API gateway. For example, different endpoints for getting and putting different information about a user rather than a single endpoint containing multiple POST operations. Lacking this framework means that each new operation requires setting up a new endpoint through AWS amplify which is also very time consuming because the team has still not fully mastered the technology. Additionally, there are security flaws that need to be addressed that also concern the API gateway. These things will continue to cause technical debt again in future sprints unless they are addressed.