Lauren Hibbs
The Juicerz
2/4/20

Tech Debt

I investigated using sonarqube for technical debt, but as a lot of our technical debt has to do with cloud service integration and lambda functions I felt that manually recording technical debt by task was the best strategy. Martin Fowler suggests calculating tech debt by having teams estimate how much "interest" was paid against problem areas of the system. I have estimated the extra work that would go into the current sprint if we were to reach our acceptance criteria for each task.

As the number of completed tasks has increased over the quarter I am not included completed tasks that have accrued no technical debt.

Issue	*Title/Reason*	*Tech Debt(hrs)*
115	View setup screen only on login There is currently a bug with the setup screen where the user is advanced to the logged in page after setup but the user is not signed into the app	3
114	View Charging Data in Graph The requirements for the graph were unclear and have changed. The backend charge log table is not stored in a manner that is suitable to produce a graph of the data. This tdatabase able should be remade so that it includes appropriate start and end time of the charge and whether it belongs to primary or secondary device. Additionally, the graph is not displayed in a logical location within the app.	8
112	Single table for User,Notifications, ChargeSchedule These fields should have been added to the same table originally as specified. This was avoided to avoid making changes to existing code. Therefore I have counted work on this task as technical debt.	2
----	ChargeSchedule Table The ChargeSchedule table doesn't appear to contain enough information to be used by the app for reporting data in a graph or determining which device has been used. Some features of the charge logs are not clear. Database schema should be updated and clarification with the customer is needed	6
----	Lambda / API Gateway We have many API gateway endpoints, one for each lambda function. A thoughtfully implemented API design would have multiple actions at	? >20

> each endpoint, and would allow for more readable and maintainable frontend code. Lambda functions themselves contain useless or commented code that makes them difficult to read. Calls to the database use plaintext for sql that makes them vunerable to sql injection attacks.

An evaluation of the metrics above shows a strong reduction in technical debt.

There are some problems with the front end, such as notifications and other settings needing to be moved to different pages. However, this is due to changing and unclear requirements of the customer and I don't believe constitutes technical debt.

On the backend, most of the work that needs to be done is updates to database tables. This stems from an unclear definition of log data and a lack of live data to work with. More work should have been done to understand these things initially and create a clear database schema and is now difficult to correct because of the many locations our tables are accessed from.

Last metrics evaluation, Lambda was the source of a lot of our technical debt, and it still is. Lambda has poor documentation and we are still now always successful in creating new lambda functions that work. We have attempted to learn the correct ways to do this and correct existing functions, but changes to lambda functions often cause them to break and aren't reversible. Because of these factors it doesn't seem feasible to attempt to tackle these code smells at this point.

Another point of improvement last metrics evaluation was security. Our lambda code performs sql queries by parsing plain text, however this takes minutes to fix. We integrated with AWS Cognito, so calling API functions now requires a token that is given only to authenticated users.