**Methereum DevPost Paper**

**By Patrick Kijek, Alexi Falquier, Karrin Melton and Jay Pan**

**Inspiration**

Methereum will revolutionize the way every college, university and lender can provide financial aid and permanent records like transcripts and visas for students. For far too long, the financial aid institution has had a bureaucratic approach to accepting and processing loans, which takes weeks to months during processing and leaves the student at a severe disadvantage when making payments or applying to jobs. To make matters worse, the onus resides on students to pay off their debt for each course before classes begin. Slow transactions of these sorts can cause students to become ineligible for class registration when holds are added to their accounts. Some students may also need a transcript or Visa to find a job: slow transactions also hamper students’ ability to complete applications in timely manners. Methereum will use blockchain technology to make financial aid and permanent record transactions instant with encrypted records across all systems throughout the network due to its distributed ledger populated with smart contracts. Methereum will also keep a record of each student’s transcript and Visa, if applicable, that can be distributed instantly.

**Challenges**

We ran into issues from the very beginning, since none of us had prior experience with the implementation of Blockchain technology beyond theoretical research. Our team had to focus on learning so many new ways to program in an entirely new language, Solidity. This included how to weave smart contracts throughout our repository and mapping authorized users to various value types based on the level of authorization appropriate to each type of user.

On the backend, we had to learn how Solidity and Blockchain technology would access various forms of financial aid balances. We needed to break down the transaction process to its critical elements, when the payment is either sent or received, and learn how our technology stored this information as well. Much of this time was spent noticing the similarities between other coding languages, however, the open-source community had projects with a similar framework, which we were able to adapt and dissect to help us along the journey.

On the front end, ours web developer did not have experience with Web3, Javascript or Materialize. We had to learn how to make the code developed with these three tools interact with our backend on Solidity. We created a prototype that displayed a homepage where a student search for his private key: the challenge their lied in how to cross-reference the Solidity files we had created.

We went on to incorporate our currency of Qtum with Solar and Docker. Nobody on the internet has published adequate information on implementation of Solar or Docker, which is why we struggled through an entire day of troubleshooting. It could have been easy if the instructions had been more clear, but we also ran into frequent trouble where our files were not all cloned into the correct repositories after we repeatedly followed the instructions, which were only written for Linux.

**Accomplishments**

We provided a solution to the most maddening process a student has to go through every time when borrowing money or requesting documents like transcripts and Visas. We feel a lofty sense of accomplishment, because of the sheer amount of applications we learned to implement through the repository and CLI, which would make a modern man crumble under the weight of his own stress.

**What We Learned**

We learned how to use a plethora of new tools such as Solidity, Web3, Javascript, Materialize, and Remix for compiling front and back ends to our web application. We learned when not to force an Internet of Things device into a blockchain process, and also how to moonwalk in a space suit.

**What’s next**

It’s time to authorize schools to use our network of distributed ledgers to add jargon to our buzzwords.

**Built With**

We used the most stable release of tears.drop on top of the dev.developer API.