sawtoeth



Donor.Me

A blockchain-based Organ Donation & Matching System 09.05.2019

Jito K J Anoop E Varghese Lakshmi R

Overview

The Kerala Government has an initiative (KNOS - Mrithasanjeevani) established in August, 2012 to maintain records of patients waiting for organ transplants, and for citizens to register as potential organ donors. This initiative is targeted to resolve the ethical and legal issues surrounding live and deceased organ transplantation.

The proposed solution (Donor.Me) is a blockchain network, created by the Government of Kerala's Department of Health. Hospitals that are approved for organ transplant are registered in the network by each District's Department of Health, and the hospitals are permitted to register patients who are awaiting transplant, and donors who are willing to donate one or more organs. These hospitals can also update the donor's status and viability of donated organs at the donor's time of death. Patients that are registered are added into organ waitlists, and allocated waitlist numbers based on their blood group and the number of pending organ requests. The Department of Health (District) can check for matching organ donations for the patients with waitlist number 1 in their respective blood groups, and matches the patient details with donor details, removes the patient from the waitlist, and moves the other remaining patients up the waitlist. The Department can view the historical log of all the matches made in the Donor.Me system.

Goals

- 1. Lorem ipsum dolor sit amet, consectetuer adipiscing elit
- 2. Sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Specifications

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.

Lorem Ipsum

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan.

Milestones

I. Lorem ipsum

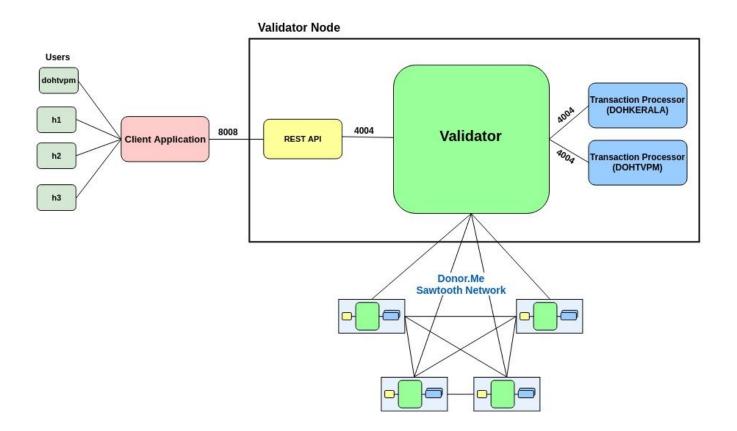
Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

II. Dolor sit amet

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Architecture Diagram

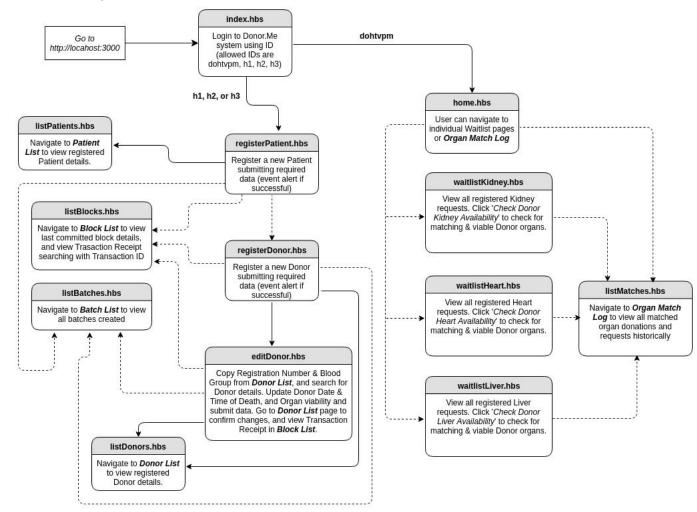
Donor.Me Architecture Diagram



Code Flow

User Interface

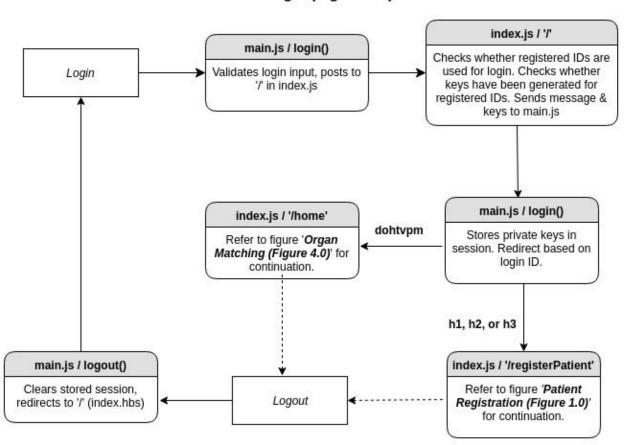
The below figure describes the code flow in the User Interface and the interactions between different components.



II. User Login

The below figure (Figure 0.0) describes the code flow and actions involved when a user logs into the Donor.Me system.

User Login (Figure 0.0)



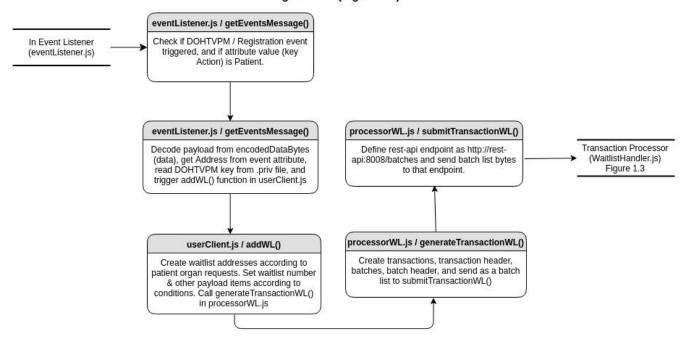
III. Patient Registration

The below figures (Figure 1.0 to 1.3) describes the code flow and actions involved when a Patient is being registered by a hospital into the Donor.Me system.

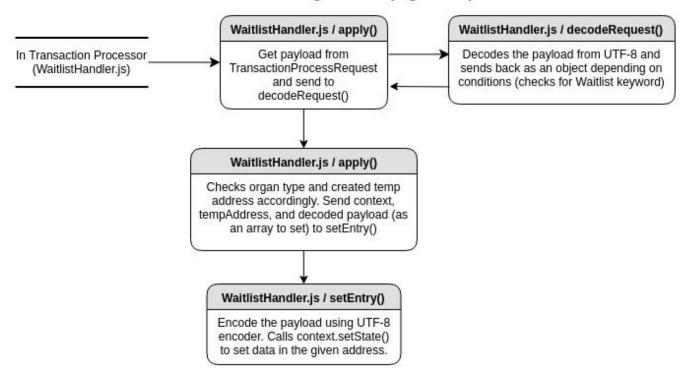
Patient Registration (Figure 1.0) index.js / '/registerPatient' main.js / registerPatient() Stores all registered keys in Submit data in Validates input data fields, session. Triggers addPatient() /registerPatient page posts to '/registerPatient' in index.js function in userClient.js pasing arguments. Checks return value and displays alert to user accordingly. userClient.js / addPatient() transaction.js / patientAddress() Triggers patientAddress() Creates 3 patient addresses (for each organ) using defined function in transaction.js. Captures 3 addresses addressing scheme, returns addresses. from return value. Transaction Processor (DonorMeHandler.js) Figure 1.1 userClient.js / addPatient() userClient.js / getStatePatient() Create 3 payload for each Queries state using 3 patient organ address as per user input data and conditions addresses, returns stateData defined. Trigger processor.js / submitTransaction() getStatePatient() in Define rest-api endpoint as http://restuserClient.js to check if patient already exists. api:8008/batches and send batch list bytes to that endpoint. processor.js / generateTransaction() Create transactions, transaction userClient.js / addPatient() header, batches, batch header, and send as a batch list to If stateData is empty (patient submitTransaction() doesn't exist), triggers generateTransaction() in processor.js userClient.js / addPatient() If stateData not empty, alert user that Patient already exists.

Patient Registration (Figure 1.1) DonorMeHandler.js / decodeRequest() DonorMeHandler.js / apply() Decodes the payload from UTF-8 and In Transaction Processor Get payload from sends back as an object depending on TransactionProcessRequest (DonorMeHandler.js) conditions (checks for Patient, Donor, and send to and length). decodeRequest() DonorMeHandler.js / apply() DonorMeHandler.js / apply() Checks organ type and created If data exists, throw invalid temp address accordingly. Checks transaction as Patient already whether this address already registered. exists calling context.getState() DonorMeHandler.js / apply() If no data, send context, tempAddress, and decoded payload (as an array to set) DonorMeHandler.js / setEntry() Encode the payload using UTF-8 encoder. Check conditions for Patient, creates Event Listener Event (DOHTVPM / Registration) and (eventListener.js) Transaction Receipt Data. Calls Figure 1.2 context.setState() to set data in the given address.

Patient Registration (Figure 1.2)

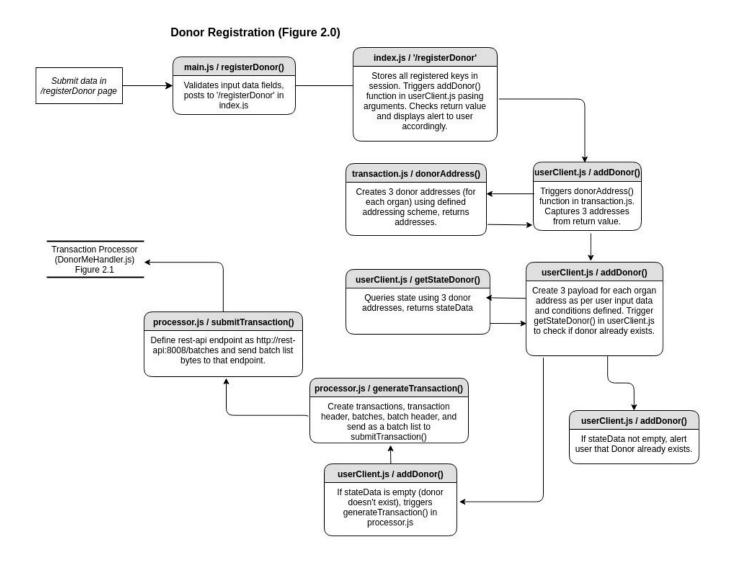


Patient Registration (Figure 1.3)



IV. Donor Registration

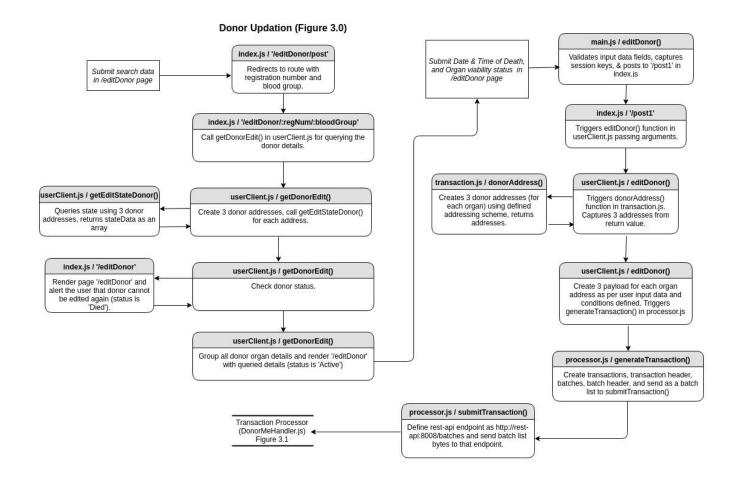
The below figures (Figure 2.0 & 2.1) describes the code flow and actions involved when a Donor is being registered by a hospital into the Donor.Me system.



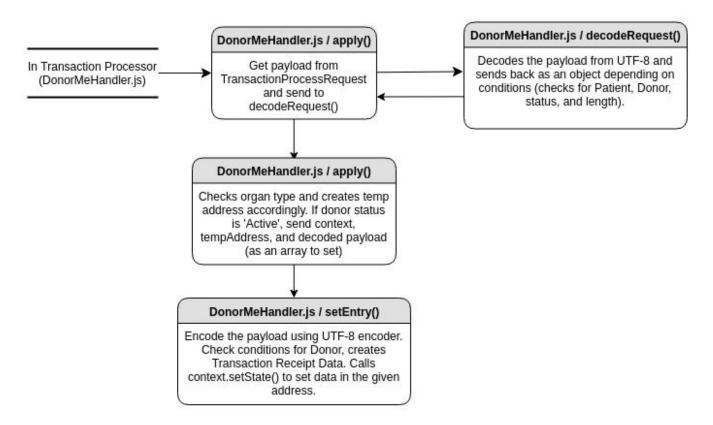
Donor Registration (Figure 2.1) DonorMeHandler.js / decodeRequest() DonorMeHandler.js / apply() Decodes the payload from UTF-8 and In Transaction Processor Get payload from sends back as an object depending on TransactionProcessRequest (DonorMeHandler.js) conditions (checks for Patient, Donor, and send to decodeRequest() status, and length). DonorMeHandler.js / apply() DonorMeHandler.js / apply() If Donor is 'Active', checks organ If data exists, throw invalid type and creates temp address transaction as Donor already accordingly. Checks whether this registered. address already exists calling context.getState() DonorMeHandler.js / apply() If no data, send context, tempAddress, and decoded payload (as an array to set) DonorMeHandler.js / setEntry() Encode the payload using UTF-8 encoder. Check conditions for Donor, creates Event (DOHTVPM / Registration) and Transaction Receipt Data. Calls context.setState() to set data in the given address.

V. Donor Updation

The below figures (Figure 3.0 & 3.1) describes the code flow and actions involved when a Donor is being updated by a hospital in the Donor.Me system.

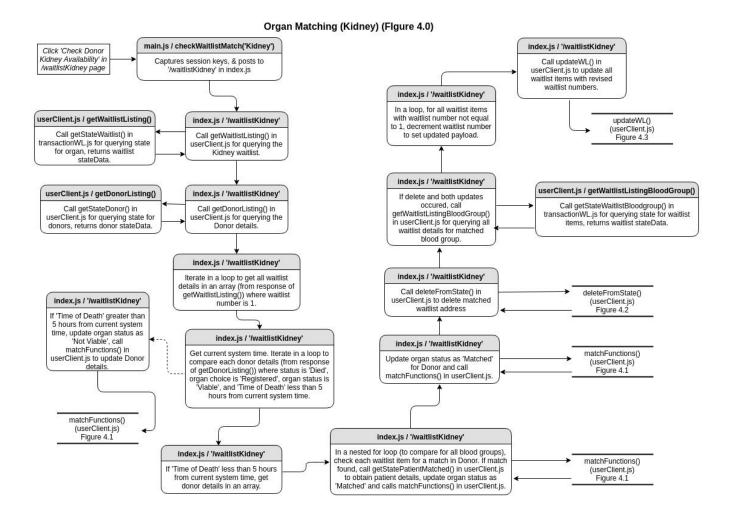


Donor Updation (Figure 3.1)



VI. Organ Matching

The below figures (Figure 4.0 to 4.3) describes the code flow and actions involved when a registered Donor organ & Patient organ request is being matched by the Department of Health in the Donor. Me system. The figures given are for Kidney matching, the process for Heart and Liver matching are the same, with the difference being the respective function names and page names.



userClient.js / matchFunctions() matchFunctions() call Identify signer key, call from Figure 4.0 generateTransactionMatched() in processor.js processor.js / generateTransactionMatched() Create transactions, transaction header, batches, batch header, and send as a batch list to submitTransactionMatched() in processor.js DonorMeHandler.js / setEntry() Encode the payload using UTF-8 encoder. Check conditions for Patient & Donor, creates Event (DOHTVPM / Registration) processor.js / submitTransactionMatched() and Transaction Receipt Data. Calls context.setState() to set data in the given Define rest-api endpoint as http://restaddress. api:8008/batches and send batch list bytes to that endpoint. DonorMeHandler.js / apply() DonorMeHandler.js / decodeRequest() Check whether Patient or Donor, DonorMeHandler.js / apply() create temp addresses for organ, Decodes the payload from UTF-8 and query state using temp addresses. If Get payload from sends back as an object depending on organ status is matched, send context, TransactionProcessRequest and conditions (checks for Patient, Donor, tempAddress, and decoded payload send to decodeRequest() and length). (as an array to set) to setEntry()

Organ Matching (Kidney) (Figure 4.1)

userClient.js / deleteFromState() Matched deleteFromState() call from Set payloads for waitlist deletion and Figure 4.0 organ match log, call generateTransactionWL() in processorWL.js for each transaction Delete WaitlistHandler.js / decodeRequest() processorWL.js / generateTransactionWL() processorWL.js / generateTransactionWL() Decodes the payload from UTF-8 and Create transactions, transaction header, Create transactions, transaction header, sends back as an object depending on batches, batch header, and send as a batch list batches, batch header, and send as a batch list to submitTransactionWL() in processorWL.js conditions (checks for Matched or to submitTransactionMatched() in processor.js Delete). processorWL.js / submitTransactionWL() processorWL.js / submitTransactionWL() Define rest-api endpoint as http://rest-Define rest-api endpoint as http://rest-WaitlistHandler.js / apply() api:8008/batches and send batch list bytes api:8008/batches and send batch list bytes to to that endpoint. that endpoint. Get payload from TransactionProcessRequest and send to decodeRequest() WaitlistHandler.js / setEntry() WaitlistHandler.js / apply() Get current stateData and check if null. If null, encode the payload using UTF-8 Check whether Matched or Delete. If WaitlistHandler.js / deleteEntry() encoder, calls context.setState() to set action is Matched, send context, Calls context.deleteState() to delete the data in the given address. If not null, tempAddress, and decoded payload decode data and append new payload. Encode the payload using UTF-8 encoder, calls context.setState() to set data in the address. (as an array to set) to setEntry(). If Delete, send context and address to delete to deleteEntry().

Organ Matching (Kidney) (Figure 4.2)

given address

Organ Matching (Kidney) (Figure 4.3)

