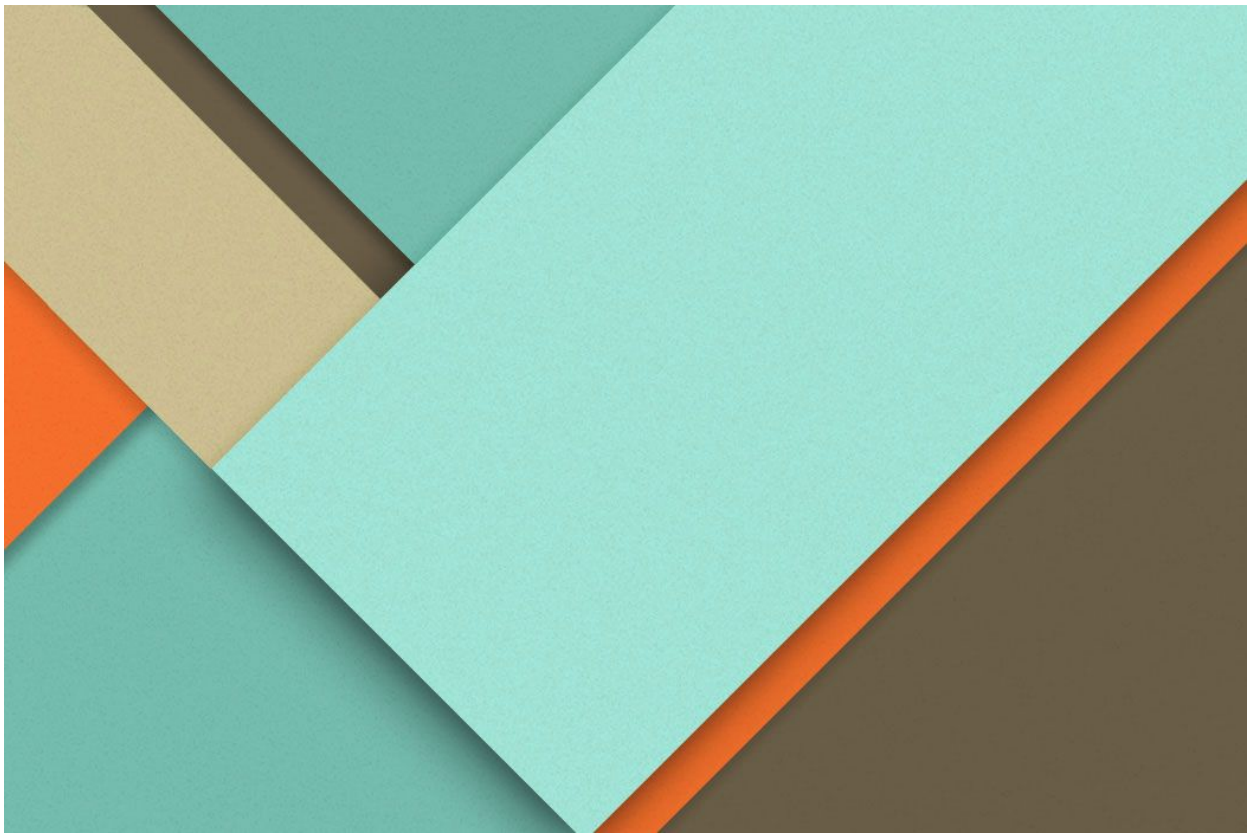
sawtooth



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.

Description

Donor.Me is a proof-of-concept for organ donation and matching that has been implemented on Hyperledger Sawtooth. For a system that handles patient records, security of the information being stores is a necessity, and the system needs to be transparent and auditable (at a higher-level). It is also imperative that the records are reliable, and that there is no chance for replicating data or identity fraud. There cannot be any middle-men or third-parties who can bypass the system and procure or sell organs to patients.

Blockchain networks address these issues by providing distributed ledgers that are immutable in nature, and control the user access to the network. The distributed ledgers ensure that the data cannot be manipulated by any of the parties involved, as each participant of the network has a copy of the database, and there is a protocol (consensus) that ensures that there is an agreement regarding the state of the ledger.

Hyperledger Sawtooth, being an open-source business blockchain, is suitable for this scenario. The network being Byzantine Fault-tolerant, ensures that the network is kept up and running even if a node is not online or is disconnected. The permissioning (policies and roles) in Sawtooth ensures that there is control over who is allowed to do what kind of transactions.

Components

I. Client

The client application is developed using Express & Node.js, with Handlebars templates for the web pages.

II. Transaction Processors

The Donor.Me system consists of two Transactions Processors (one for each Family, DOHKERALA and DOHTVPM). The transactions processors are developed in Javascript.

Users

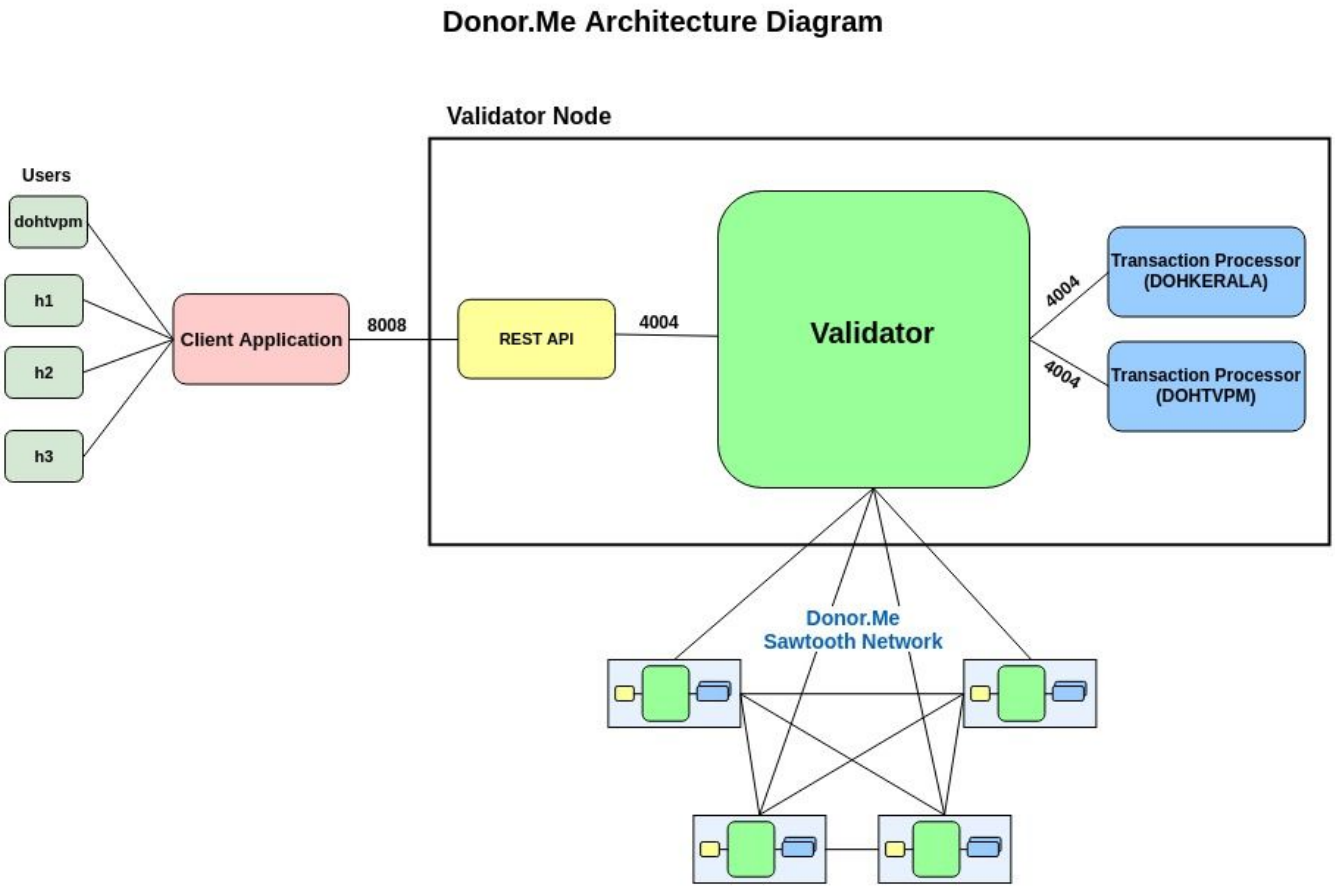
I. Department of Health

The Department of Health (using login 'dohtvpm'), can view all the Patients awaiting organ donations, check for matching donor organs, and view the log of all organ matches completed in the Donor.Me system.

II. Hospitals

The registered hospitals in the Trivandrum network are 'h1', 'h2', 'h3'. The hospitals can register Patients and Donors, update Donors, and view Patient Lists, Donor Lists, Block Lists, search for Transaction Receipts, and view Batch Lists.

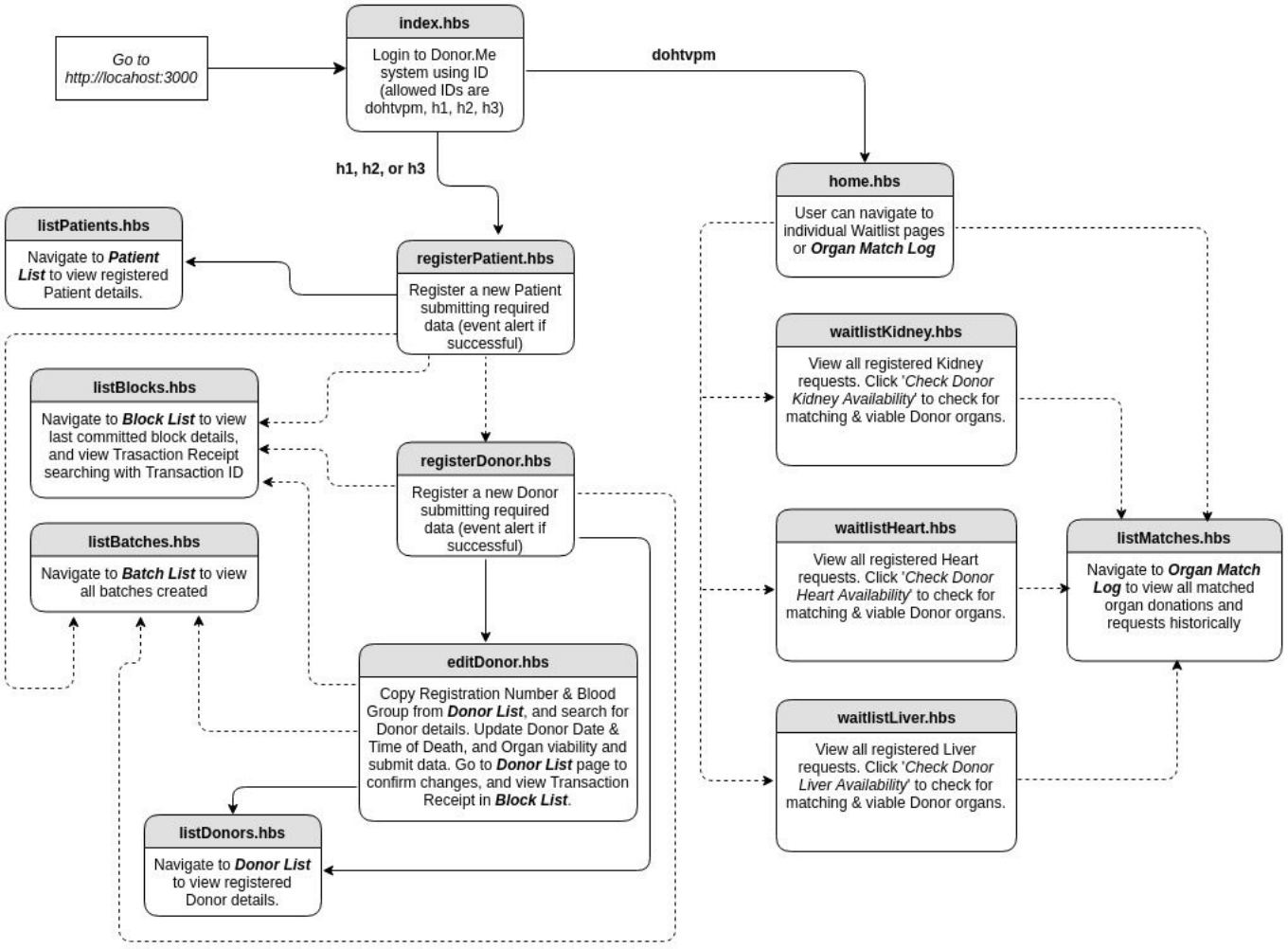
Architecture Diagram



Code Flow

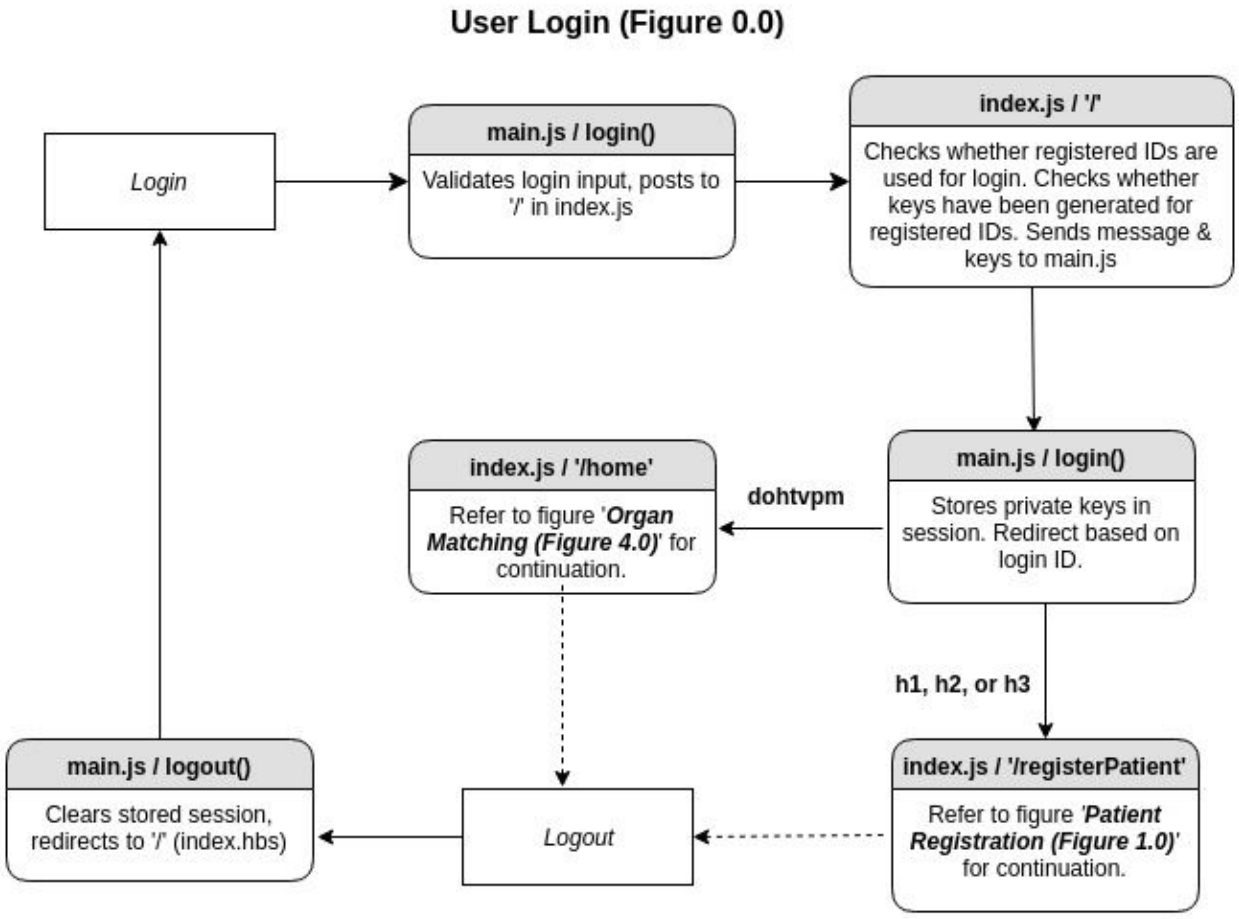
I. User Interface

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



II. User Login

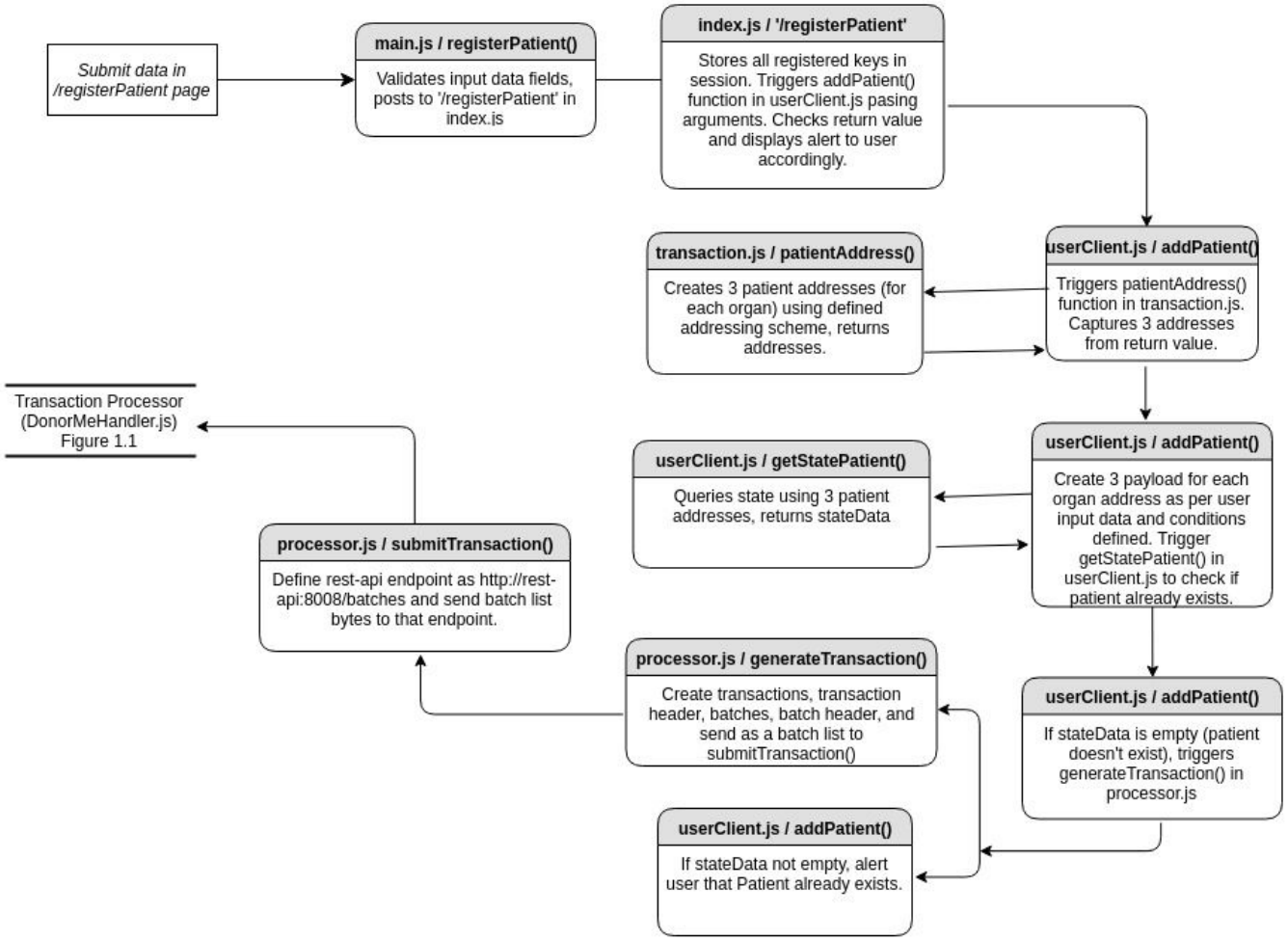
The below figure (Figure 0.0) describes the code flow and actions involved when a user logs into the Donor.Me system.



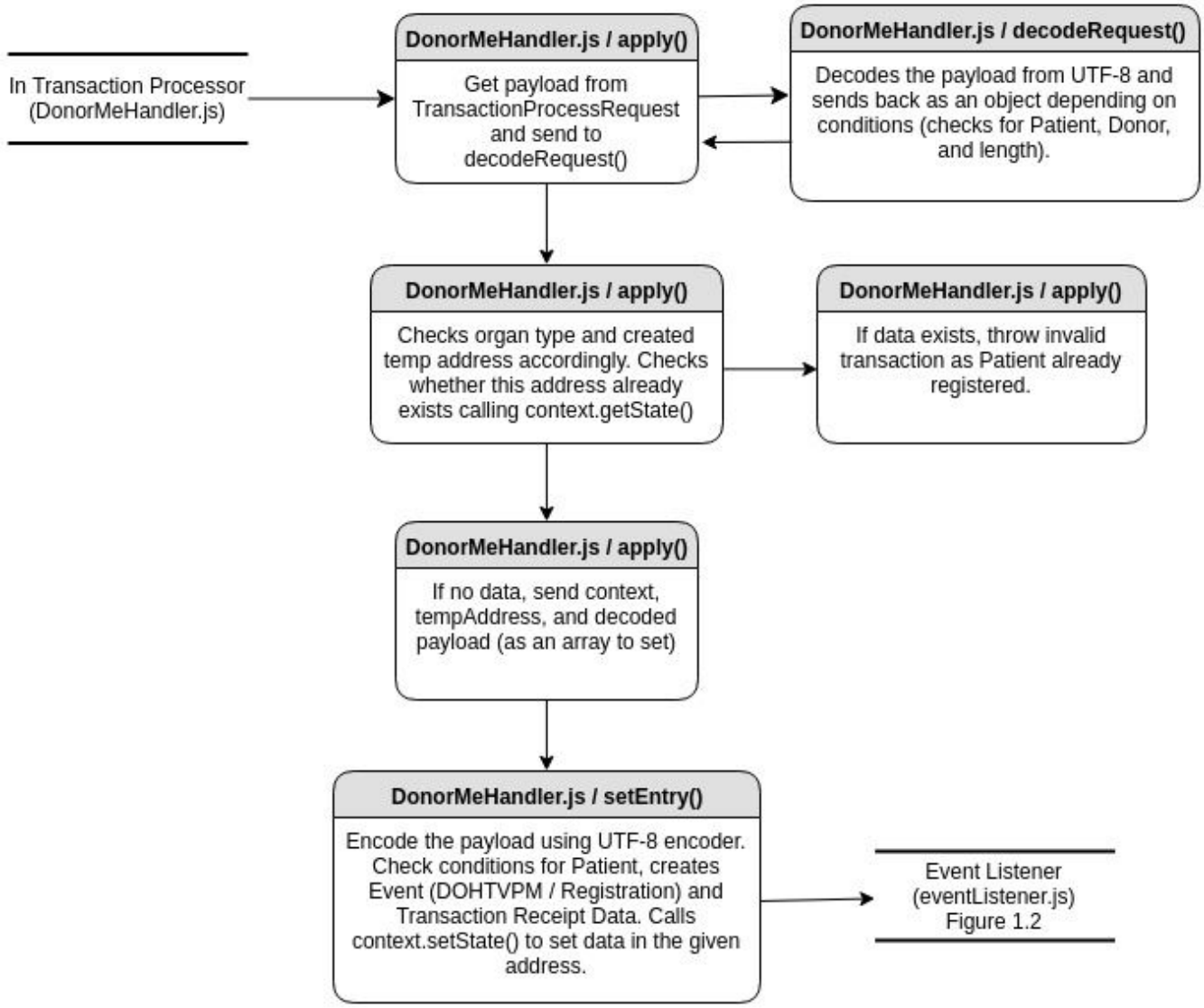
III. Patient Registration

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.

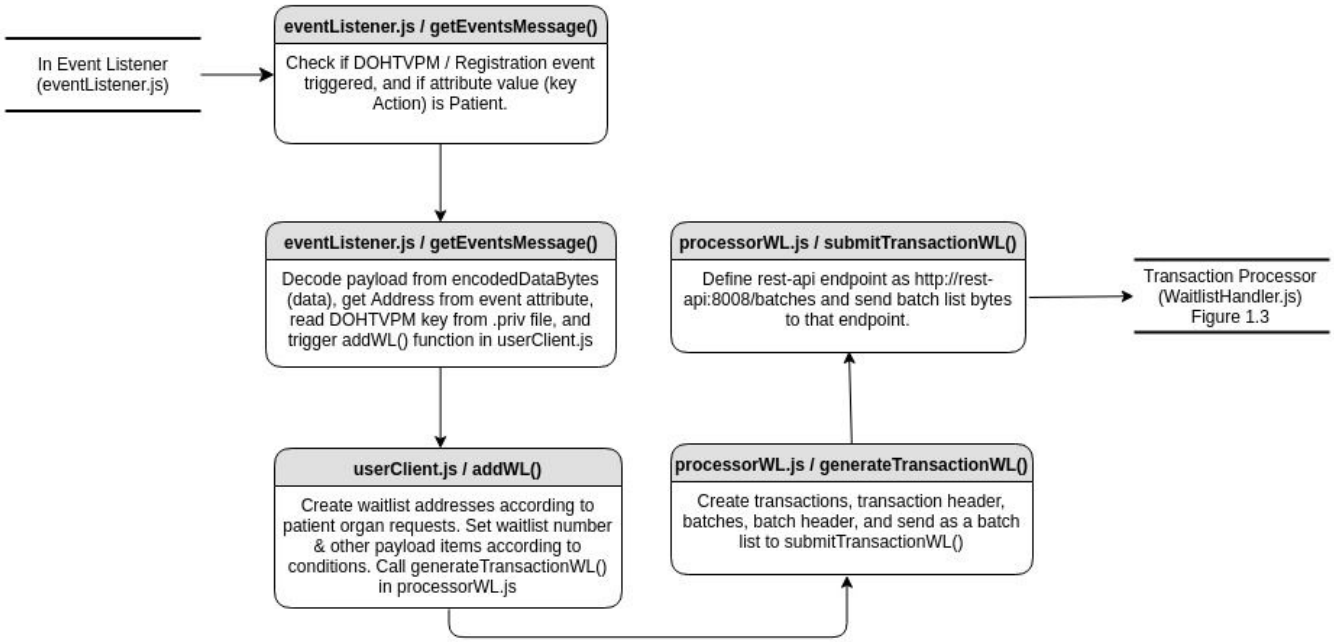
Patient Registration (Figure 1.0)



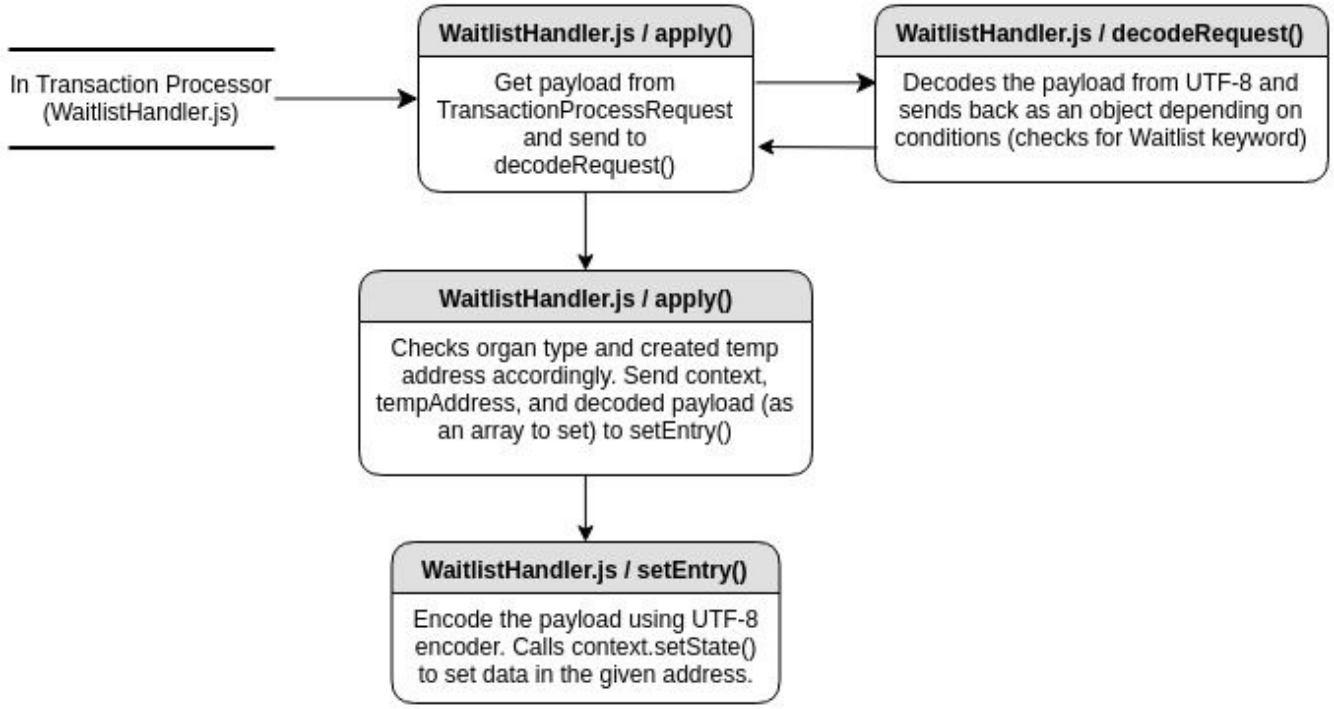
Patient Registration (Figure 1.1)



Patient Registration (Figure 1.2)

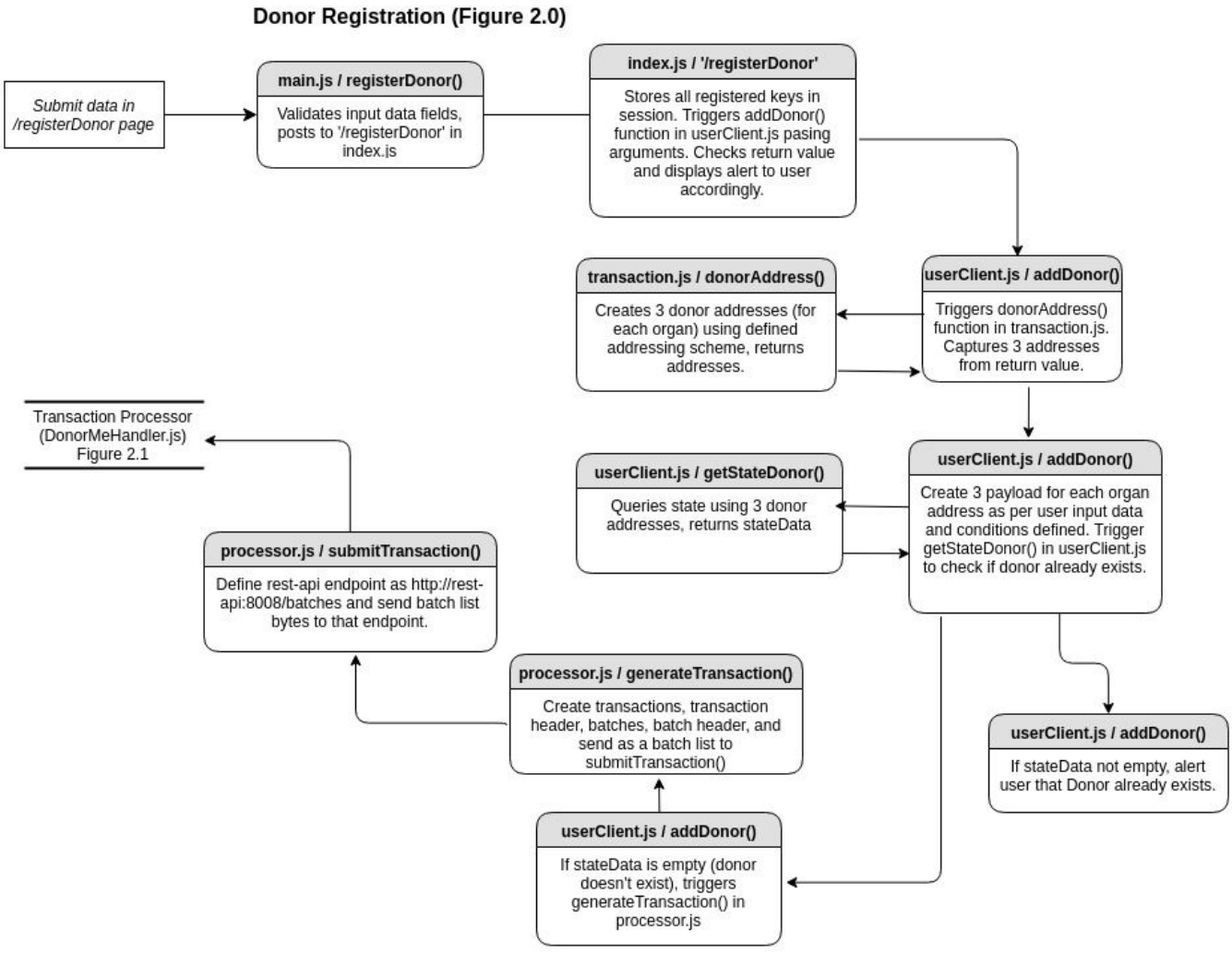


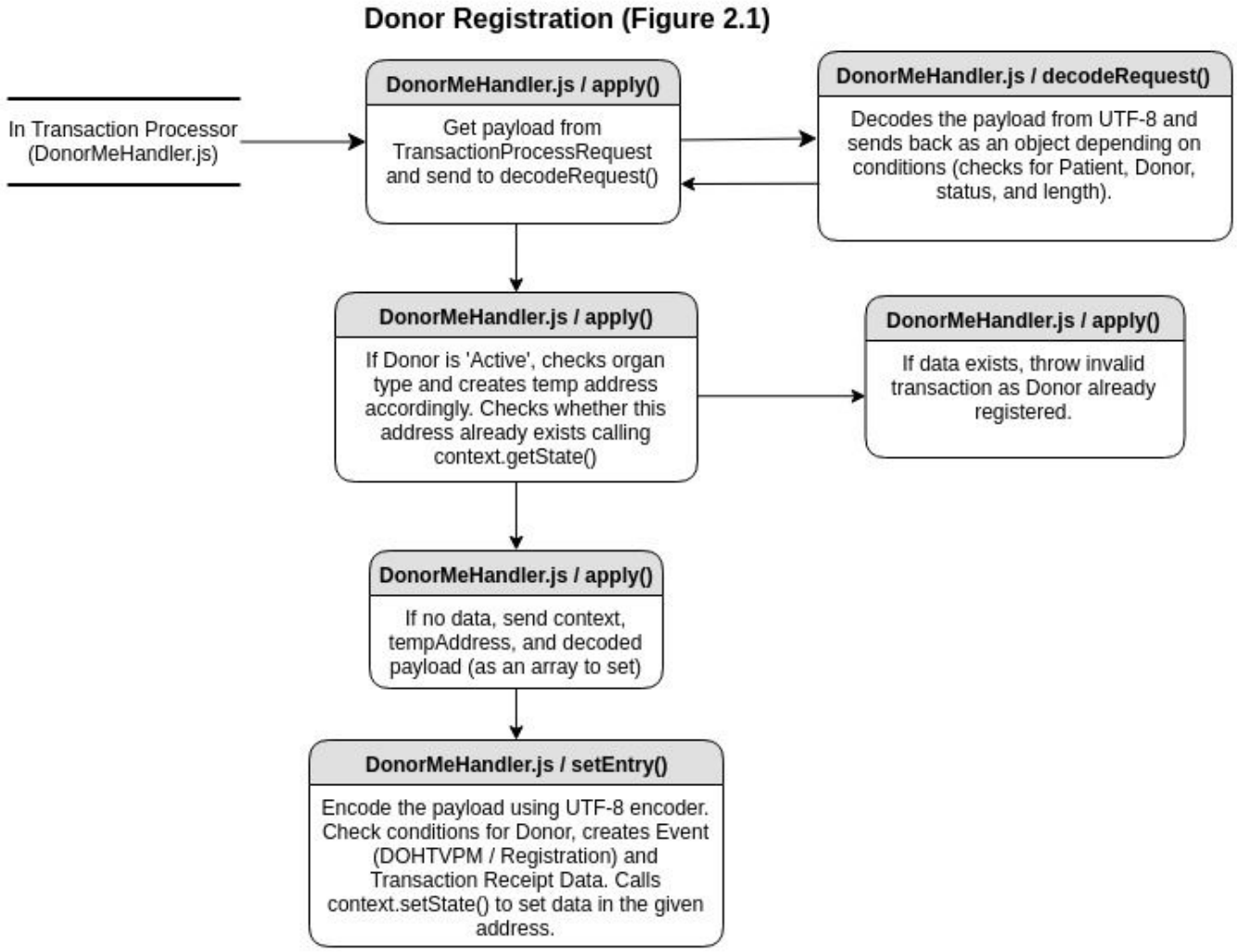
Patient Registration (Figure 1.3)



IV. Donor Registration

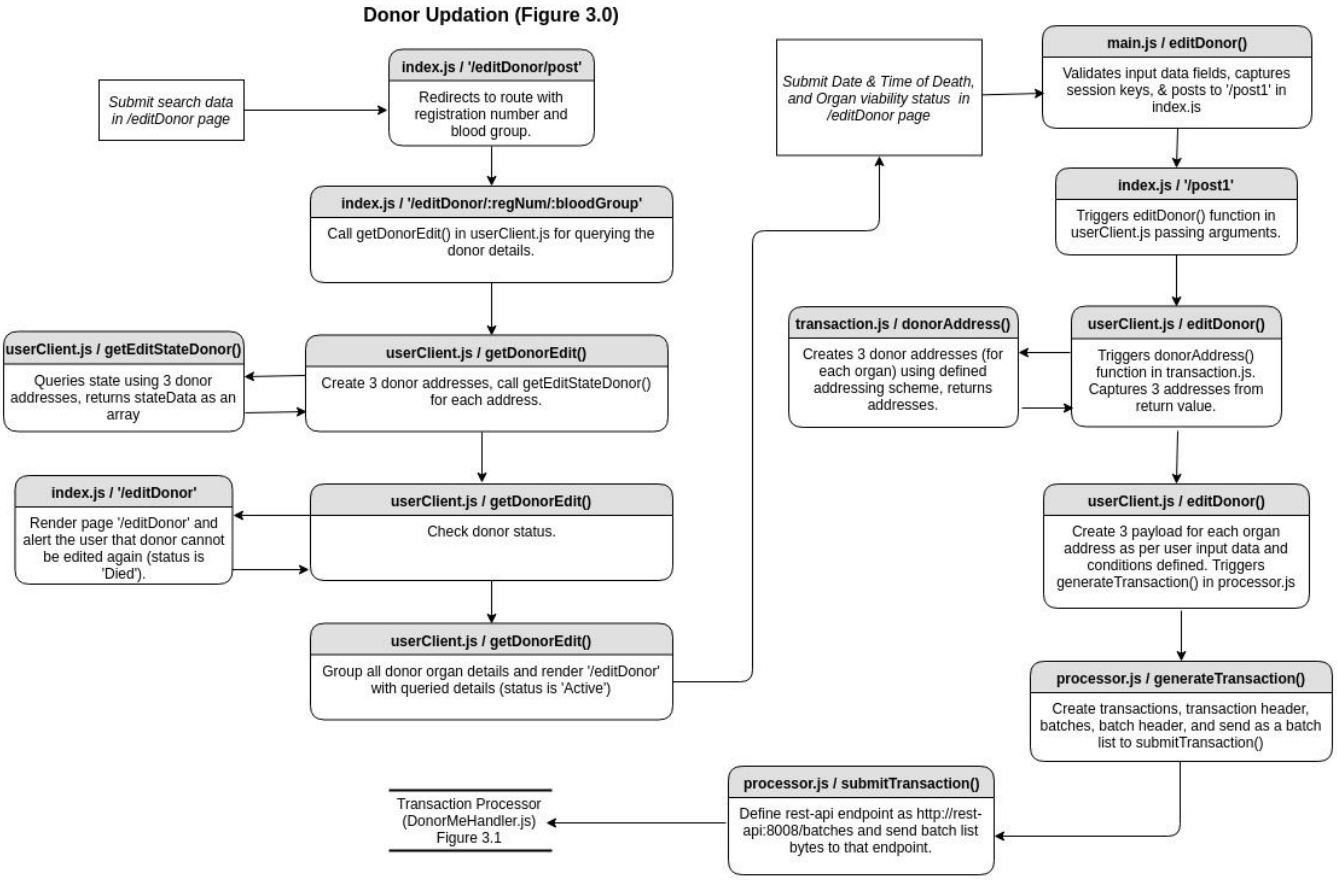
The below figures (Figure 2.0 & 2.1) describes the code flow and actions involved when a Donor is being registered by a hospital into the Donor.Me system.



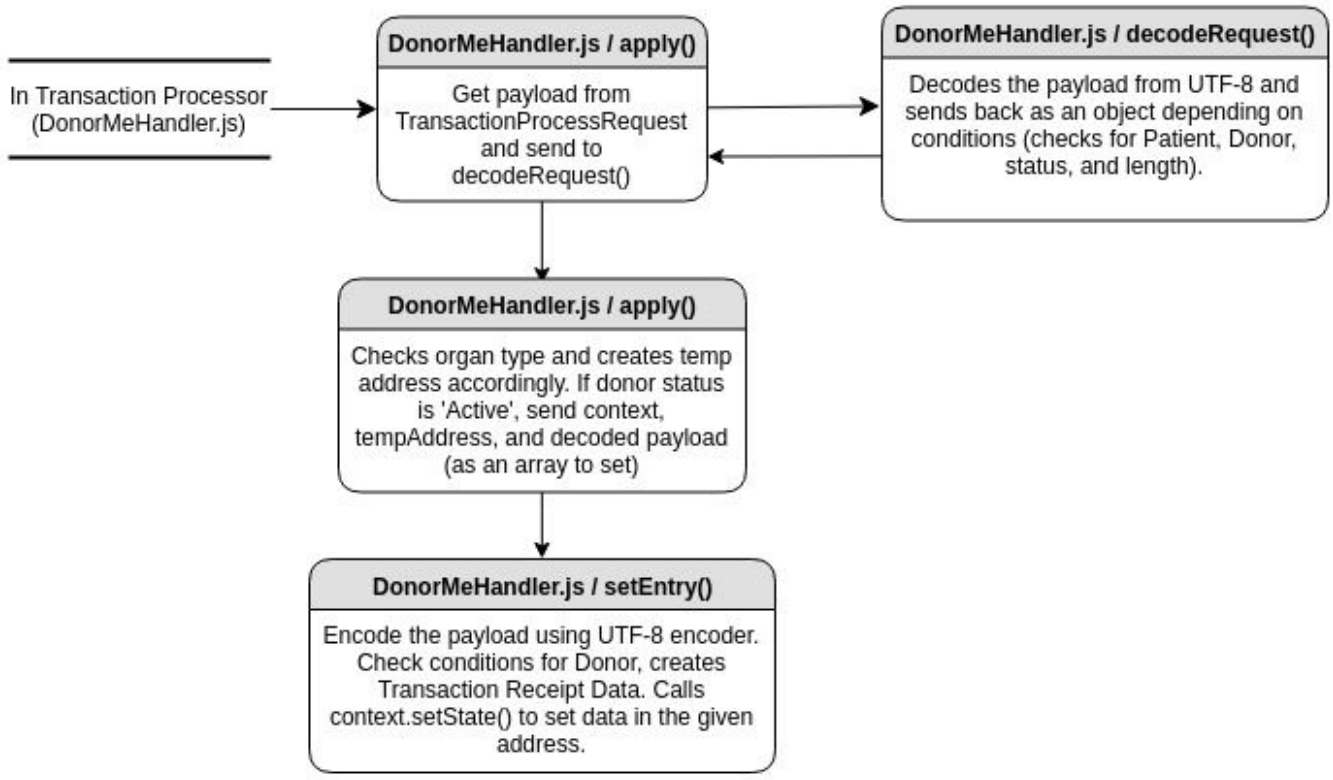

Donor Registration (Figure 2.1)
In Transaction Processor (DonorMeHandler.js)
DonorMeHandler.js / apply()
Get payload from TransactionProcessRequest and send to decodeRequest()
DonorMeHandler.js / decodeRequest()
Decodes the payload from UTF-8 and sends back as an object depending on conditions (checks for Patient, Donor, status, and length).
DonorMeHandler.js / apply()
If Donor is 'Active', checks organ type and creates temp address accordingly. Checks whether this address already exists calling context.getState()
DonorMeHandler.js / apply()
If data exists, throw invalid transaction as Donor already registered.
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.



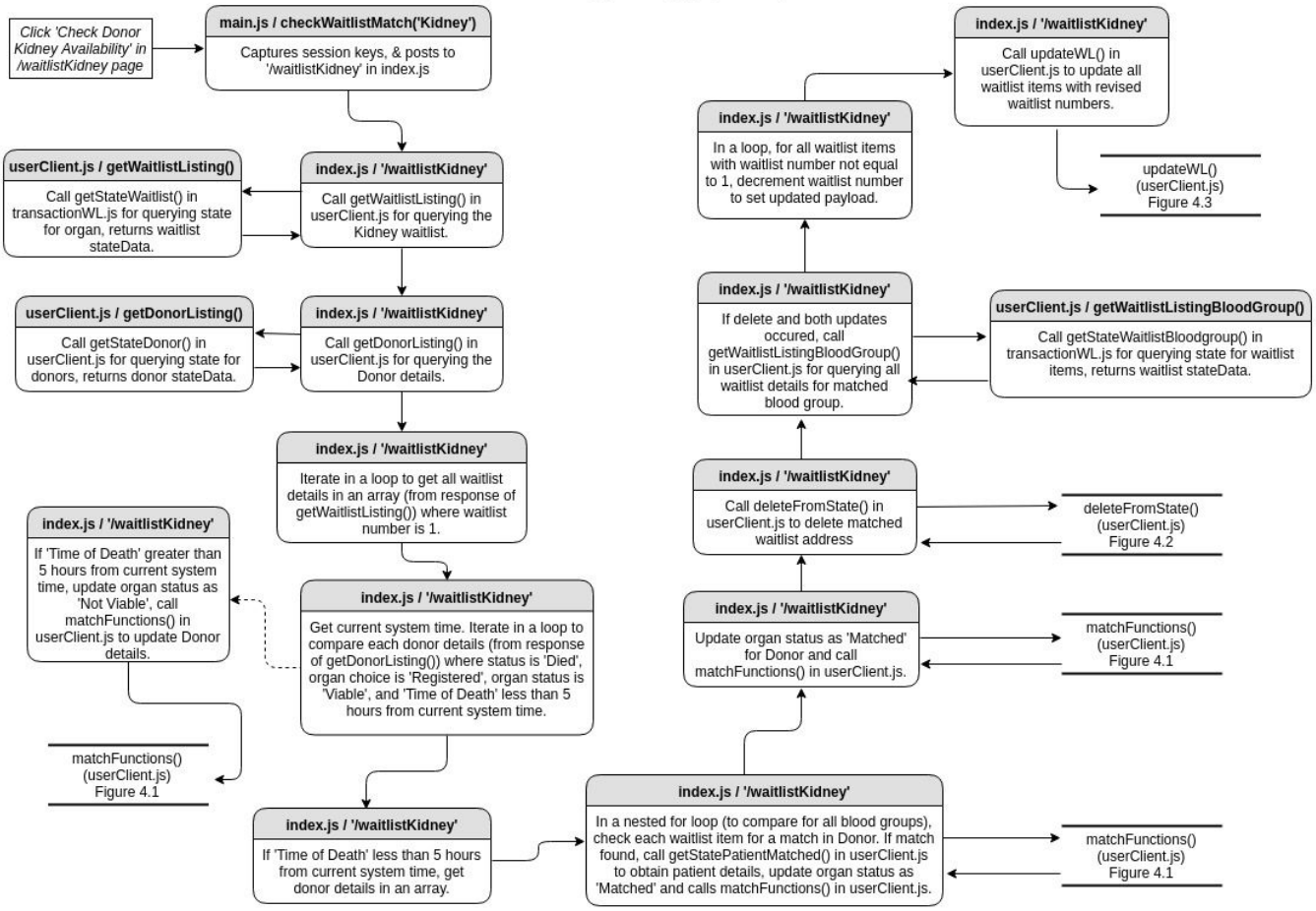
Donor Updation (Figure 3.1)

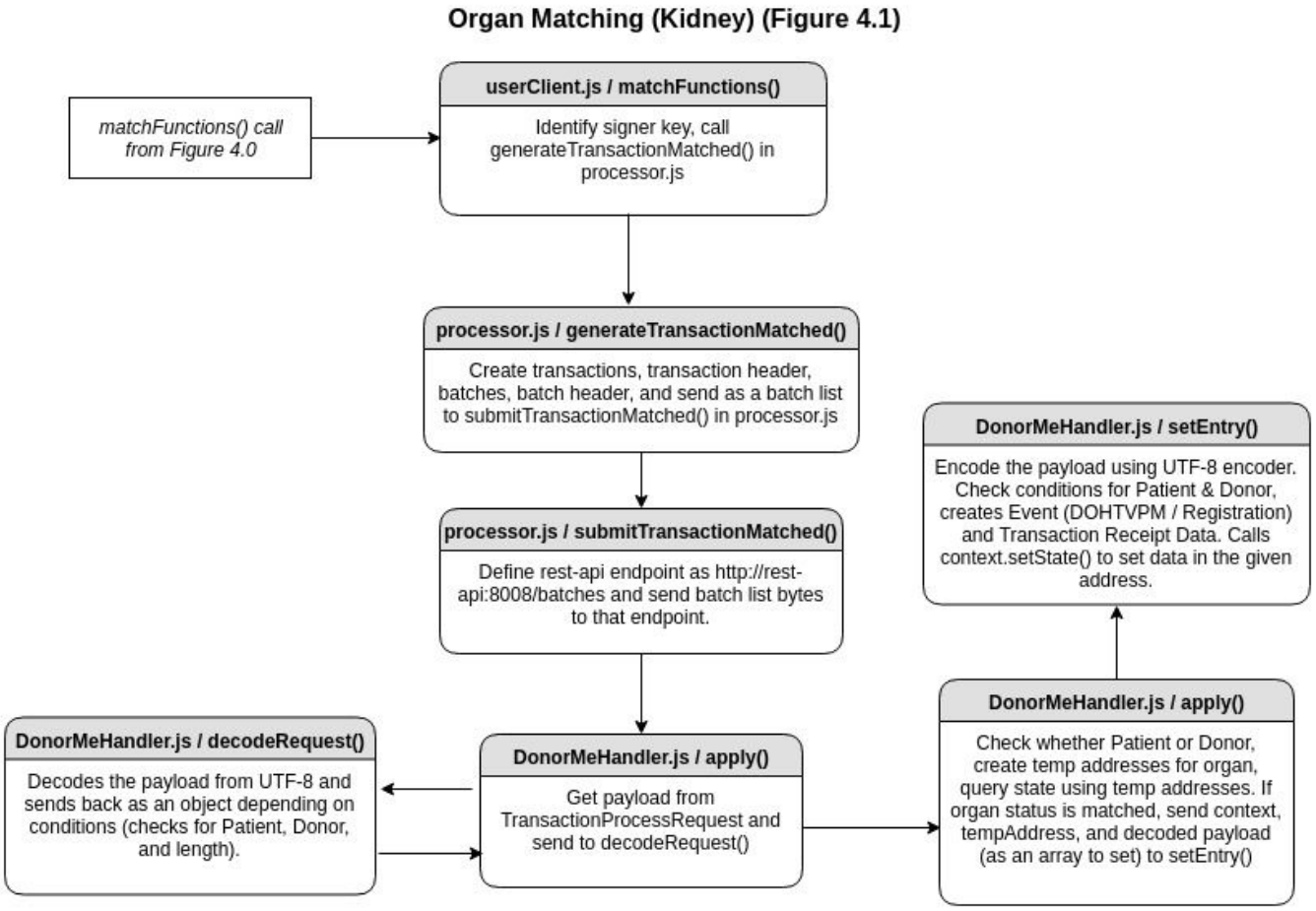


VI. Organ Matching

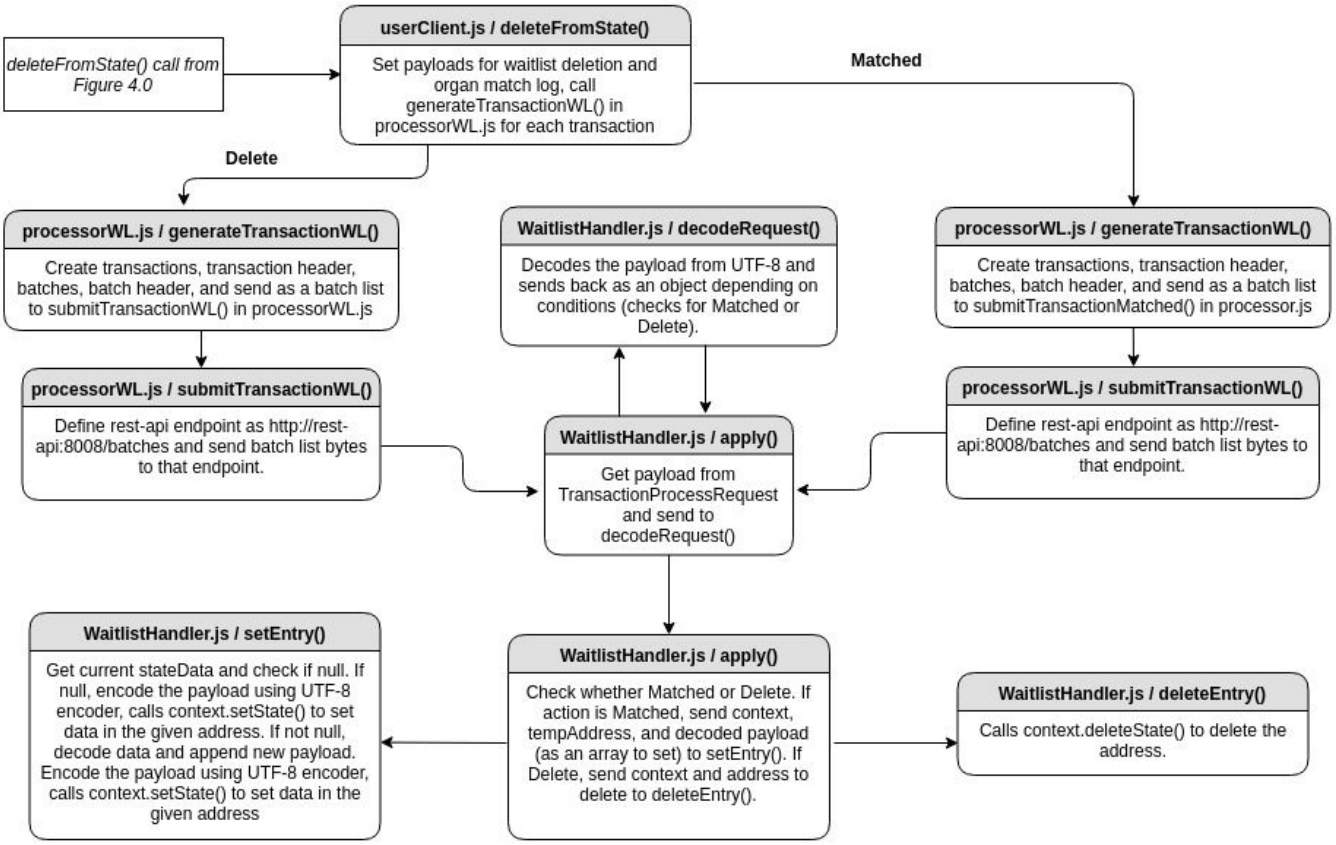
The below figures (Figure 4.0 to 4.3) describes the code flow and actions involved when a registered Donor organ & Patient organ request is being matched by the Department of Health in the Donor.Me system. The figures given are for Kidney matching, the process for Heart and Liver matching are the same,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

Organ Matching (Kidney) (FIgure 4.0)




Organ Matching (Kidney) (Figure 4.1)
matchFunctions() call from Figure 4.0
userClient.js / matchFunctions()
Identify signer key, call 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
processor.js / submitTransactionMatched()
Define rest-api endpoint as http://rest-api:8008/batches and send batch list bytes to that endpoint.
DonorMeHandler.js / decodeRequest()
Decodes the payload from UTF-8 and sends back as an object depending on conditions (checks for Patient, Donor, and length).
DonorMeHandler.js / apply()
Get payload from TransactionProcessRequest and send to decodeRequest()
DonorMeHandler.js / apply()
Check whether Patient or Donor, create temp addresses for organ, query state using temp addresses. If organ status is matched, send context, tempAddress, and decoded payload (as an array to set) to setEntry()
DonorMeHandler.js / setEntry()
Encode the payload using UTF-8 encoder. Check conditions for Patient & Donor, creates Event (DOHTVPM / Registration) and Transaction Receipt Data. Calls context.setState() to set data in the given address.

Organ Matching (Kidney) (Figure 4.2)



Organ Matching (Kidney) (Figure 4.3)