Smart Medication Dispenser

Matt Boyd, M.S. Electrical & Computer Engineering
Evelyn Liu, M.Eng. Biomedical Engineering
Yutong Wang, M.Eng Mechanical Engineering

Github: https://github.com/mcboyd-bu/EC544-Smart-Pill-Dispenser

Software & Network Architecture (See Diagram in Appendix A)

Blockchain & Crypto Modules

We selected Hyperledger Fabric as our permissioned blockchain to store patient medication records, including what medications the patient is taking and when they have taken them. We significantly modified the provided sample network "Fabcar" to build our network since it included all of the necessary network components, including private Certificate Authorities for both the Doctor and Patient "organizations". After the network is created, we generate certificates for both the doctor and patient and store them on a virtual machine where our Hyperledger Fabric "client application" can use them to query the network on the desired users' behalf. These certificates provide both authentication and authorization since they include logical organization and role attributes to provide fine grained access control.

API Modules

We utilized **Twilio's APIs** to send text and voice call alerts to patients and their families if a medication hasn't been taken at the designated time. The alerts escalate in this order: text to patient, call to patient, text to family member, call to family member. The Twilio API uses an account identification string ("SID") and secret authorization string ("Auth Token") to provide access to the API. We utilized the Twilio NodeJS SDK to access the API programmatically rather than using REST endpoints because it was simpler to secure and setup. We utilized an AWS Lambda function to access Twilio so that the SID and Token were stored in a single location in the Lambda function, rather than with our NodeJS application which actually triggers the alerts and, in our design, would have been on devices in patients' homes.

We chose **NodeJS** as our "server" software to both run the Hyperledger Fabric "client application" and to be a web application server. NodeJS is a traditional choice for "serverless" applications, and is the execution environment also used in the AWS Lambda functions used to access the Twilio APIs. We chose it because it is lightweight and feature-rich and would have worked just as well on a Raspberry Pi, as we originally intended.

In its role as our web application server, NodeJS served the web pages for our mock-UI and was coded as an endpoint to receive API requests to access Hyperledger Fabric data, for example from Telegram and Mycroft.

We set up a **Telegram Bot** named “Medication Dispenser Bot” to help authorized physicians retrieve medication records from the Hyperledger Fabric blockchain. The /patients command lists their patients’ names and ages. The /progress command is used to track side effects or symptoms. The /prescriptions command lists a patient’s medications and escalation level (from the alerting sequence). The /meddetail command provides information on a certain medication. The chatbot also reminds a user of these commands when a faulty command is entered. A webhook is set using url commands between the bot and the AWS API Gateway. Physicians can send messages to the bot via a chat window; their chat ID and message is forwarded as part of a JSON string. The AWS API Gateway forwards the message to the AWS Lambda service for processing. The access token for our Telegram bot stays hidden from public view throughout the process. (See Appendix C for Telegram dialogues and Appendix D for Hyperledger Fabric logs)

Decentralized Computing Modules

We created 3 **AWS IAM** users with varying roles and policies. The roles were mainly used for the following services: AWS Lambda, AWS API Gateway, AWS Cloud9, and Amazon EC2.

For security, the Telegram bot access token is stored in our **AWS Lambda** functions so that it does not have to be uploaded to Github. The “send-telegram-message-2” Lambda function receives Telegram messages as JSON events from the AWS API Gateway. The Lambda handler reads in each event’s chat ID, text, and entities. The entities are checked so that commands and their associated parameters can be discovered and forwarded to Hyperledger Fabric. Any “parameter 400” code errors are sent to the terminal log and also back to the AWS API Gateway POST method for display. A separate Lambda function called “send-telegram-message” is called by NodeJS to send Hyperledger Fabric data relevant to each command back to the Telegram chat window. This function formats the data properly for physicians to read. Multiple JSON test events are set up in the Lambda service for both functions based on test calls to PipeDream and Postman from Telegram and Hyperledger Fabric.

Four mock data tables were first set up in Raspbian Debian in **MariaDB** on a Raspberry Pi 2 to be integrated into the Raspberry Pi 4 as soon as Mycroft was set up. Hyperledger Fabric also temporarily uses that data before Mycroft “webcalls” to the Hyperledger Fabric endpoints could be set up. The medication table includes all prescriptions assigned to all patients, including the dosage, doses per day, frequency, remaining doses, restrictions, and progress variables. The table refers to patient IDs that are foreign keys to the patients table including patients’ names; the medication table also includes arrays of foreign keys to the restrictions and progress variables tables. The variables represent symptoms and side effects that physicians can track. Mycroft pulls from the progress variables table to check up on patients and record the dates of occurrence. (See Appendix B for our mock tables.)

We set up **Mycroft** on a Raspberry Pi 4 to represent the voice of the dispenser, and connected it to a Mycroft account. We developed 3 skills: First, it answers questions about the medication table, including the indication, dosage, frequency, and process. Secondly, it reminds the patients to take pills at the appropriate times in the day. We can also ask for a patients’ medication status after the alerting process ends. Thirdly, when patients indicate that they feel dizzy or nauseous, the dates of the occurrences get recorded on Hyperledger Fabric for physicians to access. The Raspberry Pi 4 sends JSON data to the Hyperledger Fabric via a REST endpoint call, using a POST url. Physicians can then decide whether the patients’ prescriptions need to be adjusted.

Implemented and New from Original Proposal

The Telegram bot commands were originally implemented on Repl.It. However, to keep it serverless, we decided to set up an **AWS API Gateway** so that the "send-telegram-message-2" function can be automatically triggered every time a physician sends a message. API Gateway uses the RESTFUL API and treats any calls from Telegram as an HTTP POST request.We set up the testing page so that any 400 error codes for missing variables from AWS Lambda is displayed in JSON format in the POST method without leaking sensitive information.

PipeDream was used to identify JSON events sent out by Telegram, Twilio, and the Raspberry Pi 4 with Mycroft attached. It facilitated testing and troubleshooting in AWS API Gateway, AWS Lambda and Hyperledger Fabric, as we progress on different parts of the project separately. **Postman** was used to test the error codes in API Gateway to bypass the need for including an access token parameter in plain sight while connecting the Telegram resource url.

Failures & Items Not Implemented from Original Proposal

New Hyperledger Fabric Network

We originally planned to build a new Hyperledger Fabric network from scratch, including the logical organizations, servers, nodes, and Certificate Authorities. The documentation for starting this process from scratch was lacking, though, and would have taken many dozens of hours on its own. The next idea was to take the provided "Fabcar" sample network, designed for tracking car ownership and transfers, and edit all parts to reflect our scenario, including renaming the network and all associated parts and pieces. We spent a few hours renaming components and attempting to run this scenario, but it just created errors; clearly we missed some of the renamings and it just wouldn't function. After about 5 hours spent on this endeavor we gave up and chose to use the Fabcar sample network as-is and then add our own custom code on top of it. For example, we wrote custom functions to store our medication info on the network and wrote our client application from scratch, but there are still places in the code that refer to "cars" and not medications.

SMS and Call Responses via Twilio

We originally intended the alerting functions via Twilio to be a two-way communication. So the patient could receive a text message alert that it was time to take their medication, and could reply with one of a small number of preset responses (e.g., "Press 1 to be reminded again in 1 hour" if they were away from home). There was no technical challenge to getting this setup - Twilio supports the functionality and we could have routed the responses through Lambda back to NodeJS. As usual in technical projects, some other aspects just took longer than anticipated (see Hyperledger Fabric network) and we simply ran out of time to get it implemented.

Scheduling Process Overly Simplified

Our scheduling process for the medication reminders is significantly simplified from what would be needed in a production design. For example, we have no way to allow the skipping of a dose or a timeout after which a dose is considered "missed". We also hard-coded the medication times with no method for the patient to change them or the ability to delay a reminder if they're away from home. This simplification is a feature of the exploratory nature of this project, and not a "failure" perse - it is indicated here for completeness.

Mycroft Connecting to MariaDB
MariaDB is run on a local server from a distant Raspberry Pi 2, which made it inconvenient to access by the Raspberry Pi 4 running Mycroft because of isolation of the developers. To temporarily bypass this, we hardcoded intents into the Raspberry Pi 4 directly using the Mycroft API. In the future, it will be easy to transfer these information when the developers are in close proximity.

Mycroft Sending Data to Hyperledger Fabric
Mycroft sends data to the Hyperledger Fabric network. However, adding the POST requests to access Hyperledger Fabric confuses Mycroft; it no longer identifies intents from our speech and responds with confusion. We contacted developers of Mycroft, but they did not have a solution either. As a result, we rolled back the feature.

Technical Skills Learned

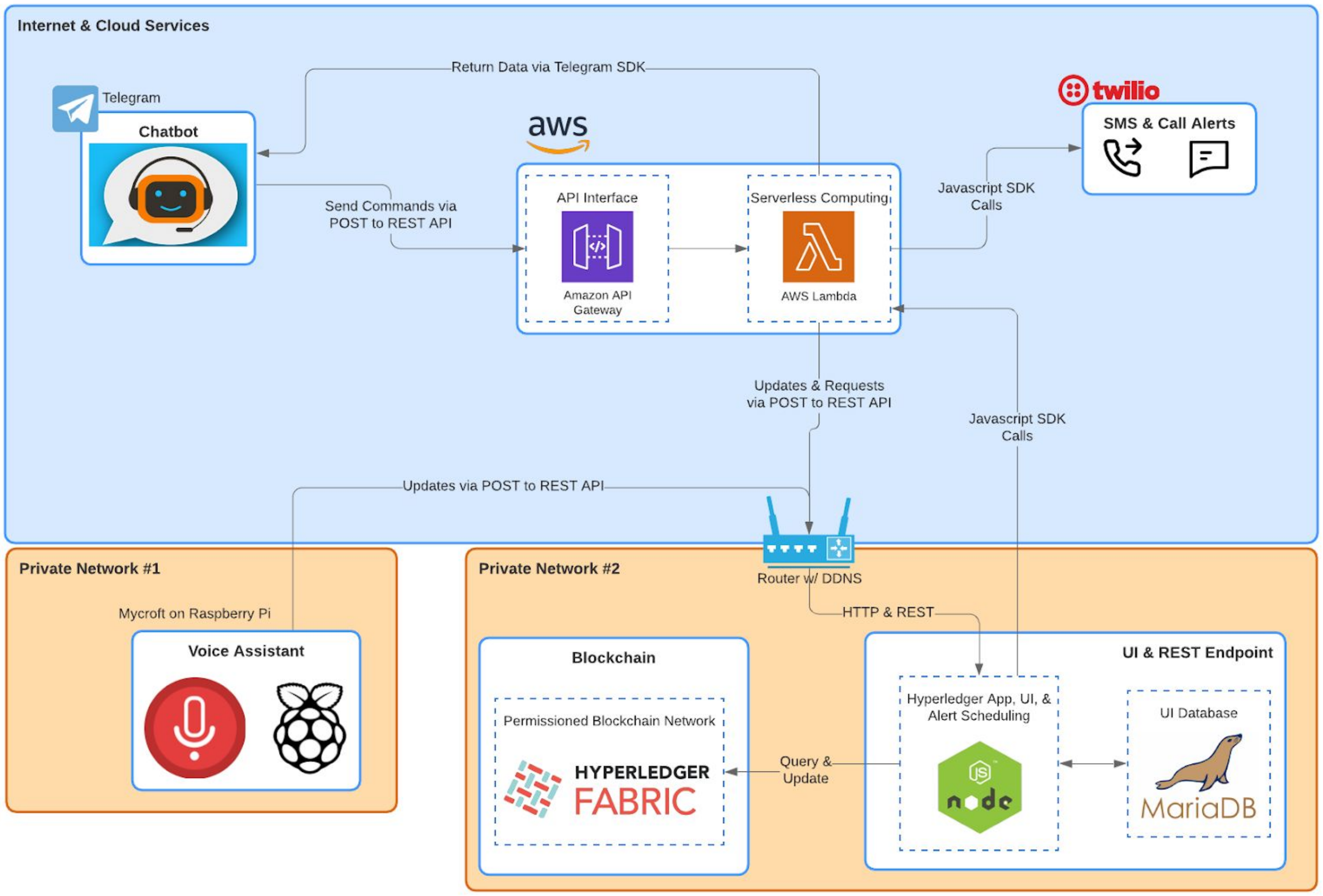
- Hyperledger Fabric Network Architecture
- Mycroft development
- Google AIY VoiceKit, speech recognition
- Telegram API, Bot API
- Serverless computing: API Gateway, AWS Lambda
- RESTFUL APIs: Webhooks, HTTP POST
- Github, incl. Security and Secrecy
- Endpoint testing
- MySQL structure
- IAM roles
- Linux

References

- Hyperledger Fabric: Process and Data Design
- Hyperledger Fabric: Writing Your First Application
- Task Scheduling in NodeJS
- Twilio: Using the Twilio Node.js Helper Library with AWS Lambda Layers
- AWS: Lambda SDK for Javascript: Setting Credentials in Node
- Mycroft: Mycroft-skills-kit
- Mycroft: How to build adapt intent
- Raspbian Installation
- How to install MariaDB on Raspbian and the Raspberry Pi
- How to connect Python programs to MariaDB
- Node.js Tutorial
- Bots: An introduction for developers
- Telegram Bot API
- AWS Lambda Getting Started
- Using Python 3 with Repl.It
- Creating Functions Using the AWS Lambda Console Editor
- Github: cfn-news-to-telegram
- What Is A Webhook?
- Cross-Origin Resource Sharing
- How to Write Your First AWS Lambda Function
- Building a Telegram Bot with AWS API Gateway and AWS Lambda

- Integrating Your Serverless Telegram Bot with AWS API Gateway
- Serverless Telegram bot on AWS Lambda
- IAM Roles

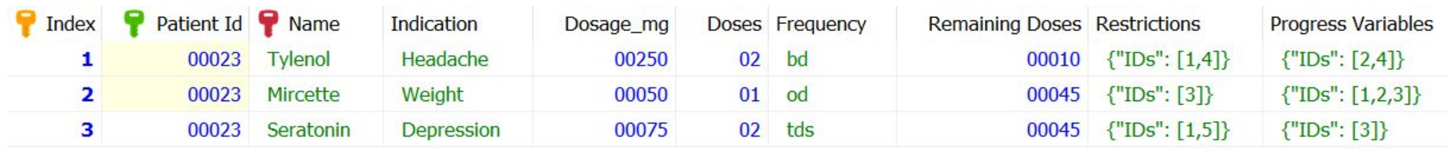
Appendix A: System Diagram



Appendix B: Mock Data Tables stored on the Raspberry Pi 4

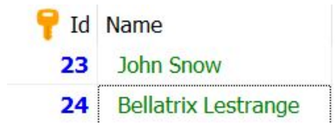
Note: some values within these mock tables have been changed for testing purposes.

Medication Table



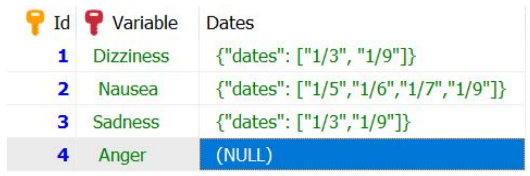
Index	Patient Id	Name	Indication	Dosage_mg	Doses	Frequency	Remaining Doses	Restrictions	Progress Variables
1	00023	Tylenol	Headache	00250	02	bd	00010	{"IDs": [1,4]}	{"IDs": [2,4]}
2	00023	Mircette	Weight	00050	01	od	00045	{"IDs": [3]}	{"IDs": [1,2,3]}
3	00023	Seratonin	Depression	00075	02	tds	00045	{"IDs": [1,5]}	{"IDs": [3]}

Patients Table



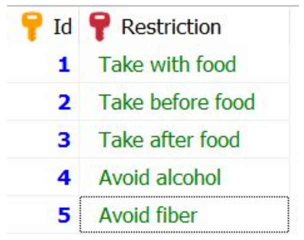
Id	Name
23	John Snow
24	Bellatrix Lestrange

Progress Variables Table



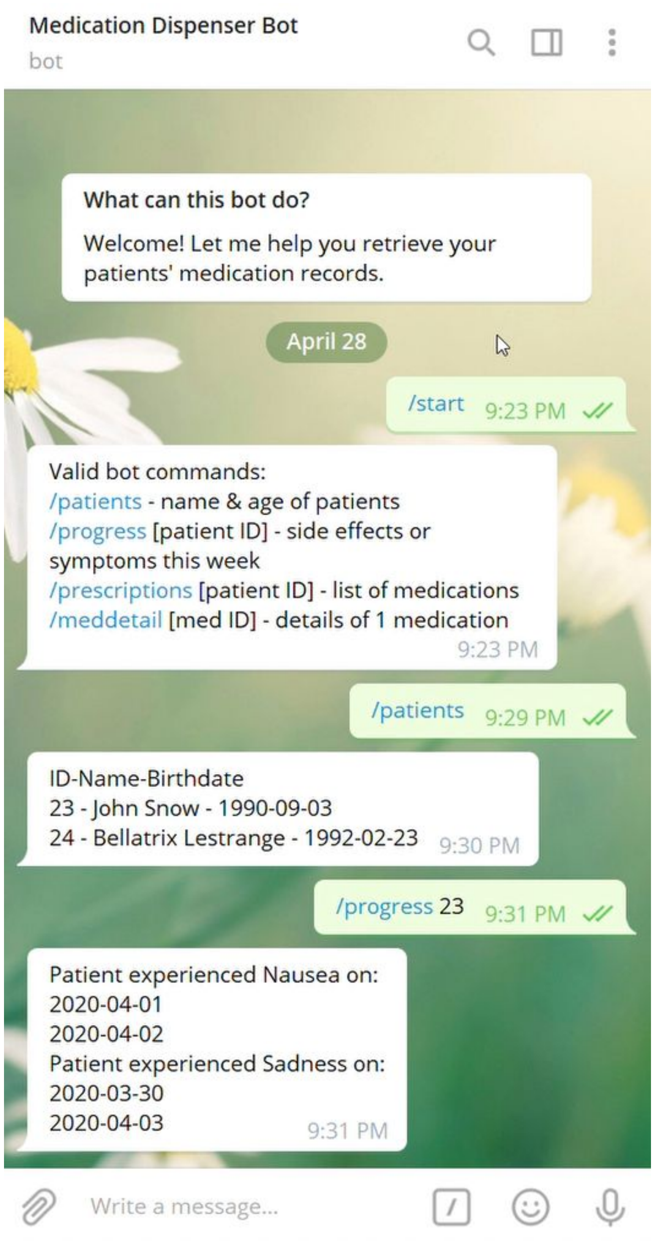
Id	Variable	Dates
1	Dizziness	{"dates": ["1/3", "1/9"]}
2	Nausea	{"dates": ["1/5","1/6","1/7","1/9"]}
3	Sadness	{"dates": ["1/3","1/9"]}
4	Anger	(NULL)

Restrictions Table



Id	Restriction
1	Take with food
2	Take before food
3	Take after food
4	Avoid alcohol
5	Avoid fiber

Appendix C: Telegram Dialogues



Medication Dispenser Bot
bot

What can this bot do?

Welcome! Let me help you retrieve your patients' medication records.

April 28

/start 9:23 PM

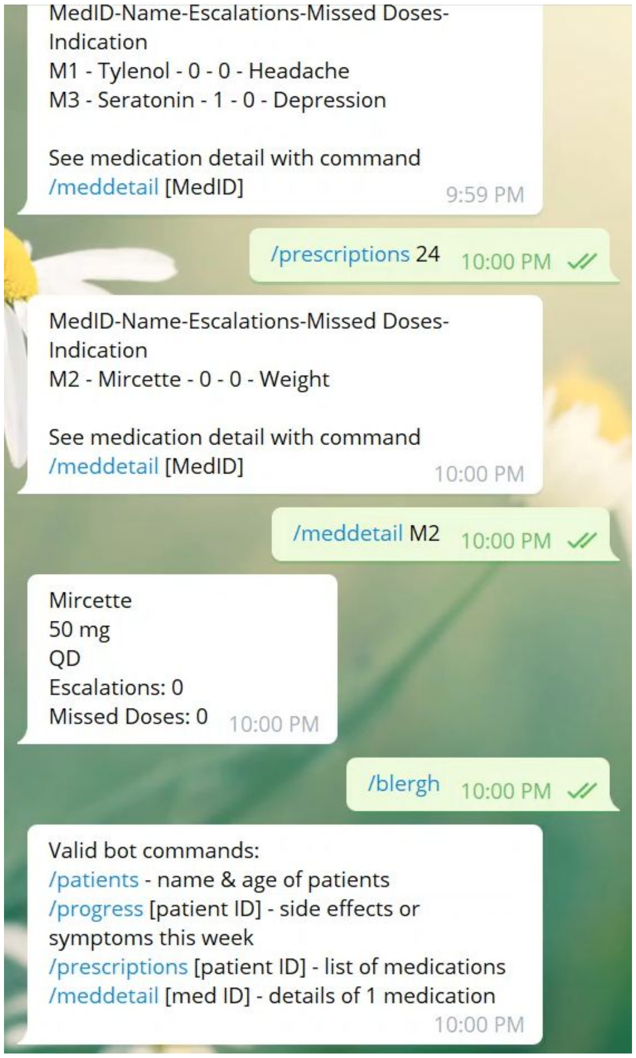
```
Valid bot commands:
/patients - name & age of patients
/progress [patient ID] - side effects or symptoms this week
/prescriptions [patient ID] - list of medications
/meddetail [med ID] - details of 1 medication
```
9:23 PM

/patients 9:29 PM

```
ID-Name-Birthdate
23 - John Snow - 1990-09-03
24 - Bellatrix Lestrange - 1992-02-23
```
9:30 PM

/progress 23 9:31 PM

```
Patient experienced Nausea on:
2020-04-01
2020-04-02
Patient experienced Sadness on:
2020-03-30
2020-04-03
```
9:31 PM



```
MedID-Name-Escalations-Missed Doses-Indication
M1 - Tylenol - 0 - 0 - Headache
M3 - Seratonin - 1 - 0 - Depression

See medication detail with command
/meddetail [MedID]
```
9:59 PM

/prescriptions 24 10:00 PM

```
MedID-Name-Escalations-Missed Doses-Indication
M2 - Mircette - 0 - 0 - Weight

See medication detail with command
/meddetail [MedID]
```
10:00 PM

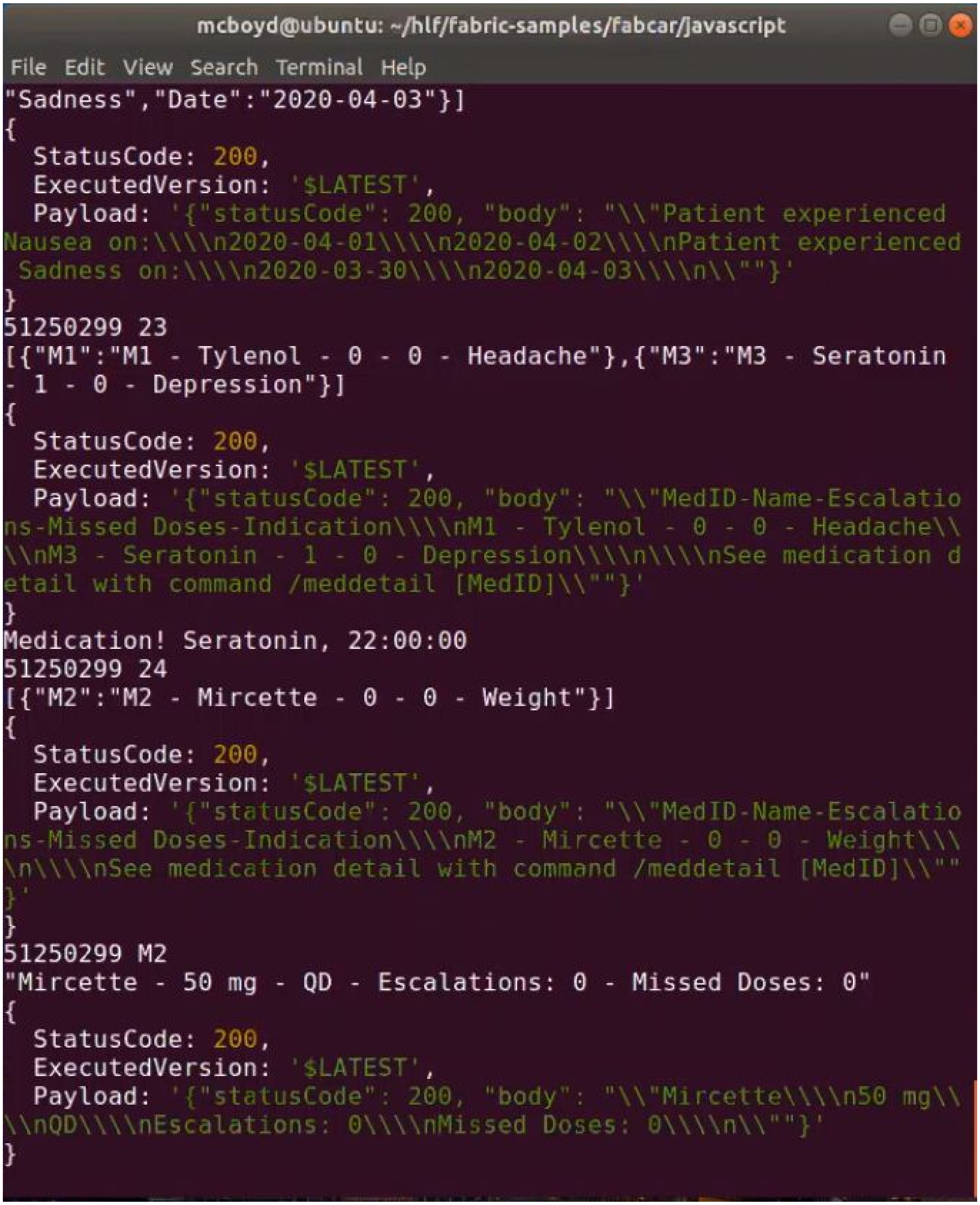
/meddetail M2 10:00 PM

```
Mircette
50 mg
QD
Escalations: 0
Missed Doses: 0
```
10:00 PM

/blergh 10:00 PM

```
Valid bot commands:
/patients - name & age of patients
/progress [patient ID] - side effects or symptoms this week
/prescriptions [patient ID] - list of medications
/meddetail [med ID] - details of 1 medication
```
10:00 PM

Appendix D: Hyperledger Fabric Logs



mcboyd@ubuntu: ~/hlf/fabric-samples/fabcar/javascript

File Edit View Search Terminal Help

```
"Sadness","Date":"2020-04-03"}]
{
  StatusCode: 200,
  ExecutedVersion: '$LATEST',
  Payload: '{"statusCode": 200, "body": "\\"Patient experienced
Nausea on:\\\\n2020-04-01\\\\n2020-04-02\\\\nPatient experienced
 Sadness on:\\\\n2020-03-30\\\\n2020-04-03\\\\n\\""}'
}
51250299 23
[{"M1":"M1 - Tylenol - 0 - 0 - Headache"},{"M3":"M3 - Seratonin
- 1 - 0 - Depression"}]
{
  StatusCode: 200,
  ExecutedVersion: '$LATEST',
  Payload: '{"statusCode": 200, "body": "\\"MedID-Name-Escalatio
ns-Missed Doses-Indication\\\\nM1 - Tylenol - 0 - 0 - Headache\\
\\nM3 - Seratonin - 1 - 0 - Depression\\\\n\\\\nSee medication d
etail with command /meddetail [MedID]\\""}'
}
Medication! Seratonin, 22:00:00
51250299 24
[{"M2":"M2 - Mircette - 0 - 0 - Weight"}]
{
  StatusCode: 200,
  ExecutedVersion: '$LATEST',
  Payload: '{"statusCode": 200, "body": "\\"MedID-Name-Escalatio
ns-Missed Doses-Indication\\\\nM2 - Mircette - 0 - 0 - Weight\\\
\n\\\\nSee medication detail with command /meddetail [MedID]\\""
}'
}
51250299 M2
"Mircette - 50 mg - QD - Escalations: 0 - Missed Doses: 0"
{
  StatusCode: 200,
  ExecutedVersion: '$LATEST',
  Payload: '{"statusCode": 200, "body": "\\"Mircette\\\\n50 mg\\
\\nQD\\\\nEscalations: 0\\\\nMissed Doses: 0\\\\n\\""}'
}
```

Note: this is just a snapshot of a demo with Telegram