VistA: a first-class citizen in the JSON-centric future of Health IT

Rob Tweed
M/Gateway Developments Ltd
@rtweed

M/GATEWAY DEVELOPMENTS

What is JSON?

- JavaScript Object Notation
- A simple and compact syntax for describing Objects of any level of complexity
- Built-in to JavaScript
  - dynamically creating objects
  - importing objects
  - exporting objects
- Increasingly being adopted in other languages

JSON v XML

- Both describe hierarchies
- JSON is rapidly replacing XML as the *lingua franca* for data exchange and description
  - less verbose
  - easier to parse
    - almost no overhead in JavaScript
  - just as readable by humans
- Can be cross-converted:
  - JSON to XML can lead to ambiguities

Data transfer

- JSON is now the preferred syntax for describing data for transfer between systems

JSON initiatives in Healthcare

- ONC
  - Mitre / Cypress Server
- HL7 FHIR
- SMART / Harvard Medical School/ Josh Mandel
  - JSON-LD
    - self-defining JSON
  - CCDA Receiver
    - CCDA XML > JSON conversion

JSON initiatives in Healthcare

- VA:
  - Virtual Patient Record
  - Health Management Portal
  - Vista Novo: HL7 FHIR
- etc...
  - eg OpenEHR interfacing via JSON

What does JSON look like?

- var object = {};
- var array = [];

Simple name/value pairs

- var person = {
  name: ‘Rob Tweed’
  };

  person.name = ‘Rob Tweed’

As many as you like

- var person = {
  firstName: ‘Rob’,
  lastName: ‘Tweed’
  };

  person.firstName = ‘Rob’
  person.lastName = ‘Tweed’

A property can be an array

- var person = {
  firstName: ‘Rob’,
  lastName: ‘Tweed’,
  children: [‘Simon’, ‘Helen’]
  };

  person.children[0] = ‘Simon’

or another object

- var person = {
  firstName: ‘Rob’,
  lastName: ‘Tweed’,
  children: [‘Simon’, ‘Helen’],
  address: {city: ‘Reigate’, country: ‘UK’}
  };

  person.address.city = ‘Reigate’

or arrays of objects

- var person = {
  firstName: ‘Rob’,
  lastName: ‘Tweed’,
  children: [‘Simon’, ‘Helen’],
  address: {city: ‘Reigate’, country: ‘UK’},
  bikes: [
  {make: ‘Trek’, model: ‘Madone 4.5’},
  {make: ‘Cannondale’, model: ‘SuperSix Ultrega Di2’}
  ]
  };

  person.bikes[1].make = ‘Cannondale’

etc, etc

- Complex objects of arrays of arrays of objects of arrays.....
- Hierarchical tree
  - in memory, in JavaScript

Parsing JSON

- Incoming JSON-formatted string:
  - var obj = JSON.parse(string);
- Converting a JavaScript object to a JSON string:
  - var string = JSON.stringify(obj);

The Rise of JavaScript

- JavaScript is the natural home of JSON
  - JSON supported in many other languages too
- Grew up in the browser
- Now becoming dominant in the server:
  - Node.js

Node.js

- Server-side JavaScript
- originally a project by Ryan Dahl
- open source
- sponsored by Joyent
- massively popular
- Modules available for anything you can think of
- Approved technology at the VA

JSON persistence?

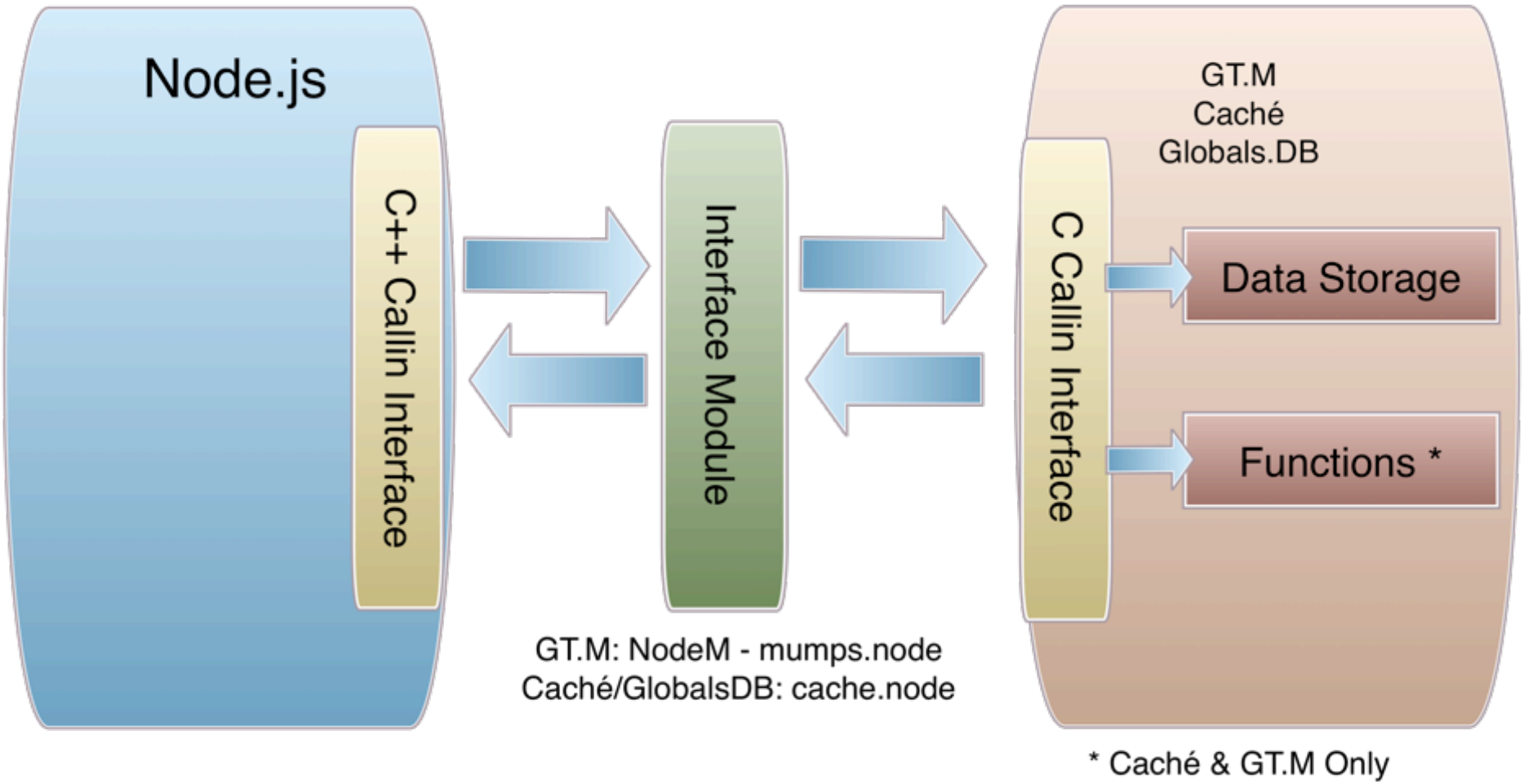
- Document databases:
  - MongoDB
  - CouchDB

JSON persistence?

- Document databases:
  - MongoDB
  - CouchDB
  - and Mumps too
    - GT.M - via NodeM interface
    - Caché - via built-in Node.js interface
    - GlobalsDB - via built-in Node.js interface

    - hierarchical database, so a natural and very efficient fit

Node.js Interface



JavaScript Document Storage

```
var gridData = [
   {col1: 1, col2: 1, name: ‘rec1’},
   {col1: 4, col2: 4, name: ‘rec4’}
 ];
var session = new ewd.mumps.GlobalNode(‘%zewdSession’, [4020]);
session.$('newGridData')._setDocument(gridData);
```

```
^%zewdSession("session",4020,"newGridData",0,"col1")=1
^%zewdSession("session",4020,"newGridData",0,"col2")=1
^%zewdSession("session",4020,"newGridData",0,"name")="rec1"
^%zewdSession("session",4020,"newGridData",1,"col1")=4
^%zewdSession("session",4020,"newGridData",1,"col2")=4
^%zewdSession("session",4020,"newGridData",1,"name")="rec4"
```

JavaScript Document Storage

```
^%zewdSession("session",4020,"newGridData",0,"col1")=1
^%zewdSession("session",4020,"newGridData",0,"col2")=1
^%zewdSession("session",4020,"newGridData",0,"name")="rec1"
^%zewdSession("session",4020,"newGridData",1,"col1")=4
^%zewdSession("session",4020,"newGridData",1,"col2")=4
^%zewdSession("session",4020,"newGridData",1,"name")="rec4"
```

```
var gridData = session.newGridData._getDocument();

[
   {col1: 1, col2: 1, name: 'rec1'},
   {col1: 4, col2: 4, name: 'rec4'}
 ];
```

Invoking Mumps code

- Can invoke functions from within the back-end JavaScript module:

```
var result = ewd.mumps.function('getPatientVitals^MyEHR',
                                params.patientId,
                                params.date);
```

Invoking Mumps code

- Can invoke functions from within the back-end JavaScript module:

```
var result = ewd.mumps.function('getPatientVitals^MyEHR',
                                params.patientId,
                                params.date);
```

This is the equivalent of the Mumps code:

```
set result=$$getPatientVitals^MyEHR(patientId,date)
```

Invoking Mumps code

- Can invoke functions from within the back-end JavaScript module:

```
var result = ewd.mumps.function('getPatientVitals^MyEHR',
                                params.patientId,
                                params.date);
```

This is the equivalent of the Mumps code:

```
set result=$$getPatientVitals^MyEHR(patientId,date)
```

Then use *_getDocument()* to retrieve Vitals from Global to corresponding JSON

Browser-based Applications

- Browsers used to be limited to “web applications”
  - HTTP Protocol
  - Ajax to add dynamic access to back-end
    - still limited by HTTP protocol

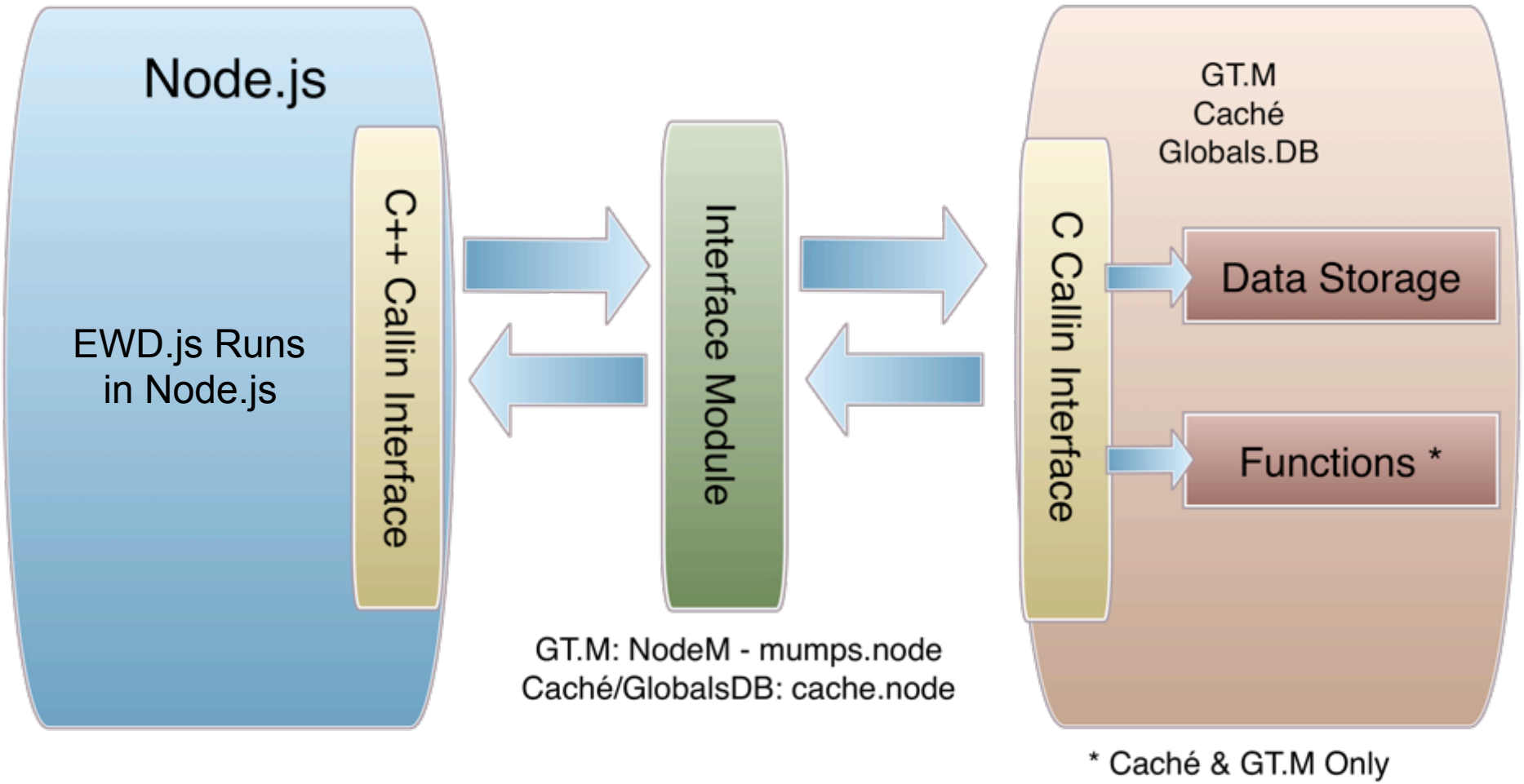
Browser-based Applications

- Now it's a client-server environment
  - lightweight
  - event-driven
- WebSockets
  - HTML5
  - bi-directional socket connection
  - event-driven at each end
- No more polling!

EWD.js

- Framework for JavaScript applications
- Mumps databases abstracted to appear to be JSON stores
- Also supports MongoDB
- Fully event-driven
- Client/server in the browser

Node.js Interface



EWD.js = 100% JSON

- Browser to/from back-end:
  - JSON messages via WebSockets
- Data storage:
  - JSON storage in MongoDB or Mumps
    - _setDocument(JSON)
- Data retrieval:
  - JSON from MongoDB or Mumps:
    - _getDocument()

So VistA is fully JSON-enabled

- Today!
- Automatically, without any other technologies than Node.js + EWD.js

JSON initiatives in Healthcare

- ONC
  - Mitre / Cypress Server
- HL7 FHIR
- SMART / Harvard Medical School/ Josh Mandel
  - JSON-LD
    - self-defining JSON
  - CCDA Receiver
    - CCDA XML > JSON conversion

JSON initiatives in Healthcare

- VA:
  - Virtual Patient Record
  - Health Management Portal
  - Vista Novo: HL7 FHIR
- etc...

Reinventing the JSON Wheel?

- Opportunity to consolidate and co-ordinate the various JSON initiatives
  - one common JSON representation of the patient?
  - VistA data to/from JSON
  - EWD.js:
    - browser-based access
    - Web Service access

HL7 FHIR Using EWD.js

Or

EWD.js on 

HL7 FHIR = REST

- RESTful interfaces are a key part of HL7 FHIR
  - http://localhost:8081/fhir/patient/@1/observation
  - Returns JSON
    - specific HL7 FHIR syntax

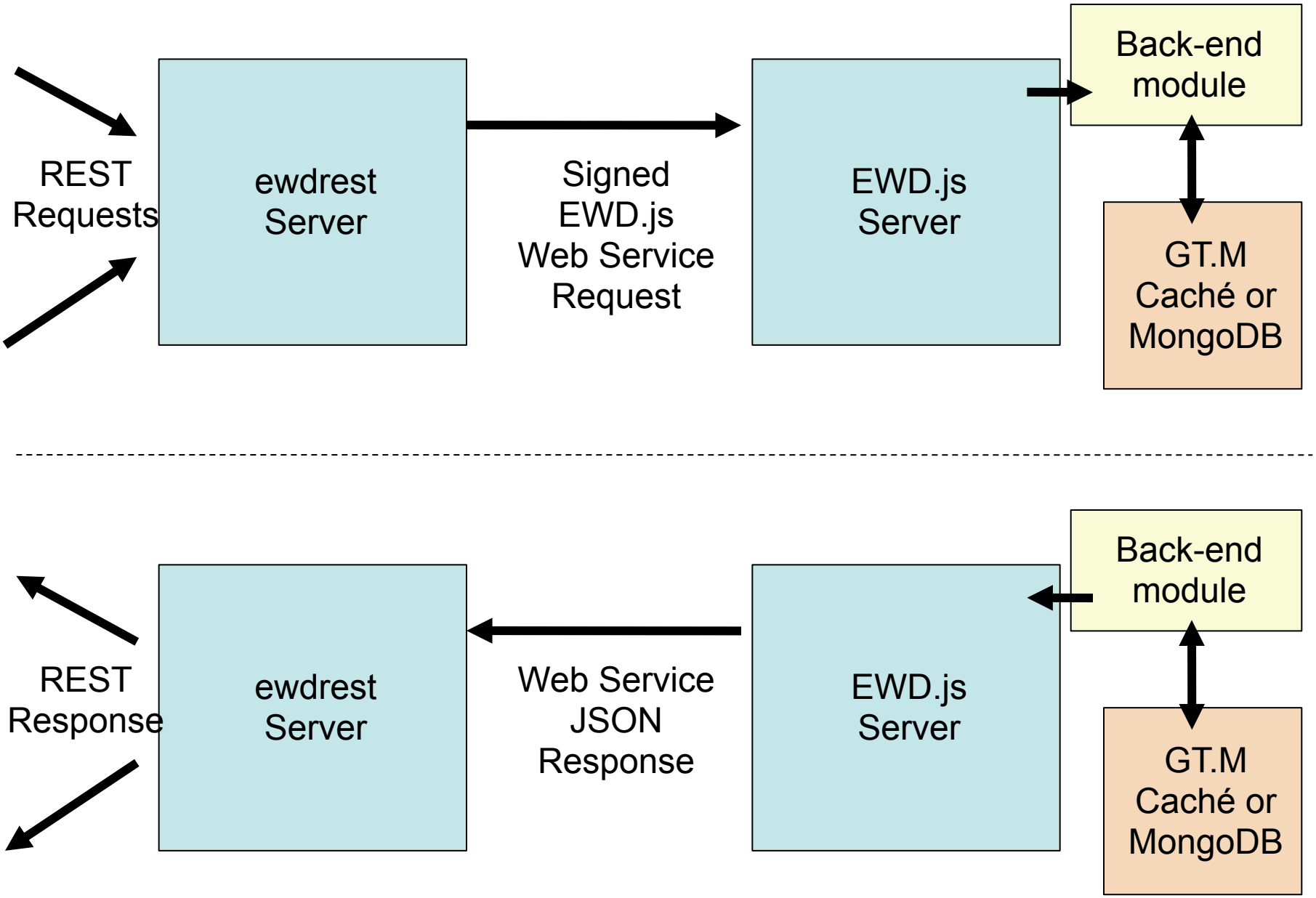
EWD.js and REST

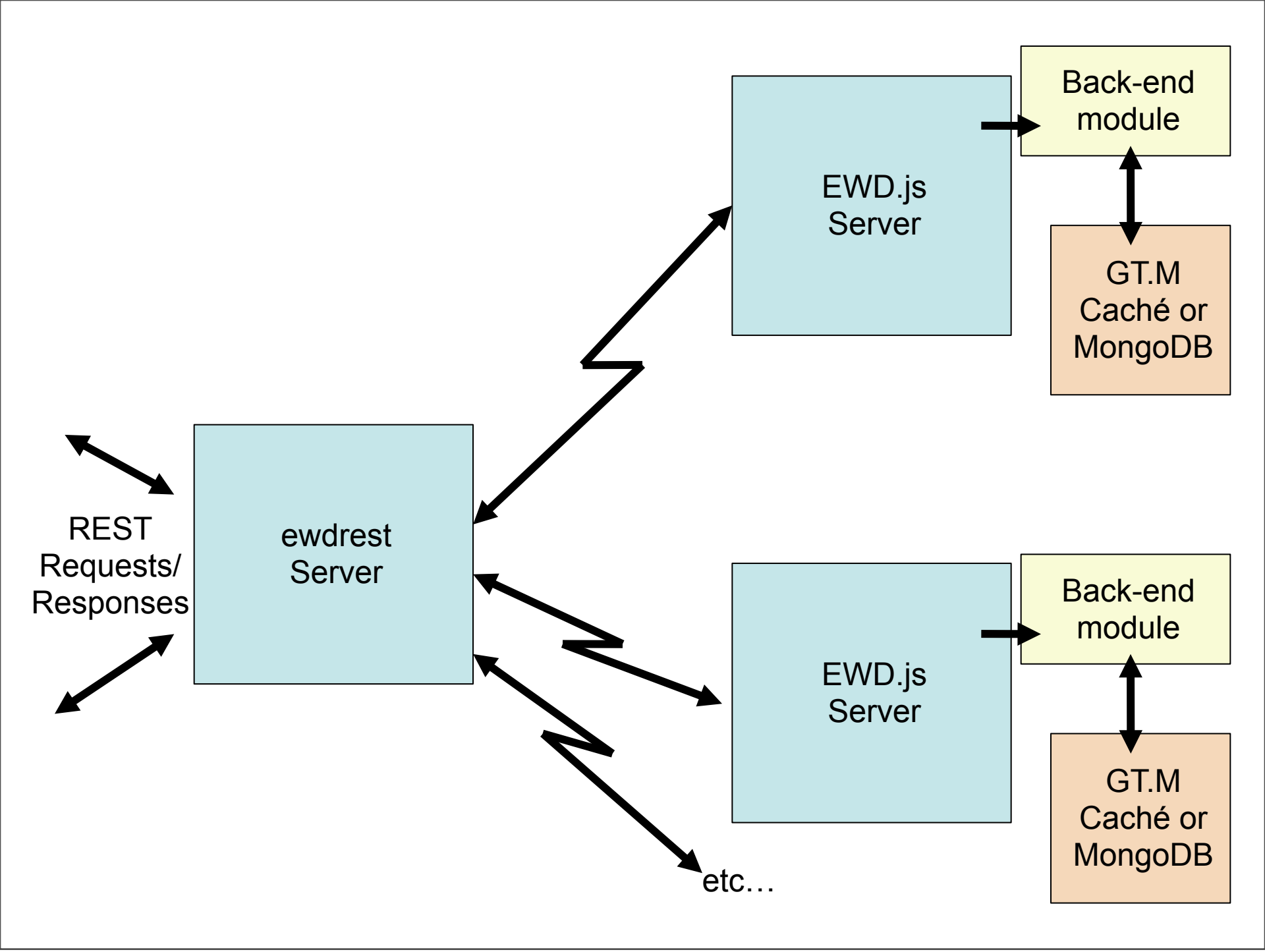
- EWD.js doesn't have a REST interface built in
  - but it does have secured HTTP-based WebService interface built-in
  - Node.js: Restify - off-the-shelf REST server module

- Restify-based module: *ewdrest*
  - rewrites REST URLs as digitally-signed EWD.js HTTP Web Service Requests

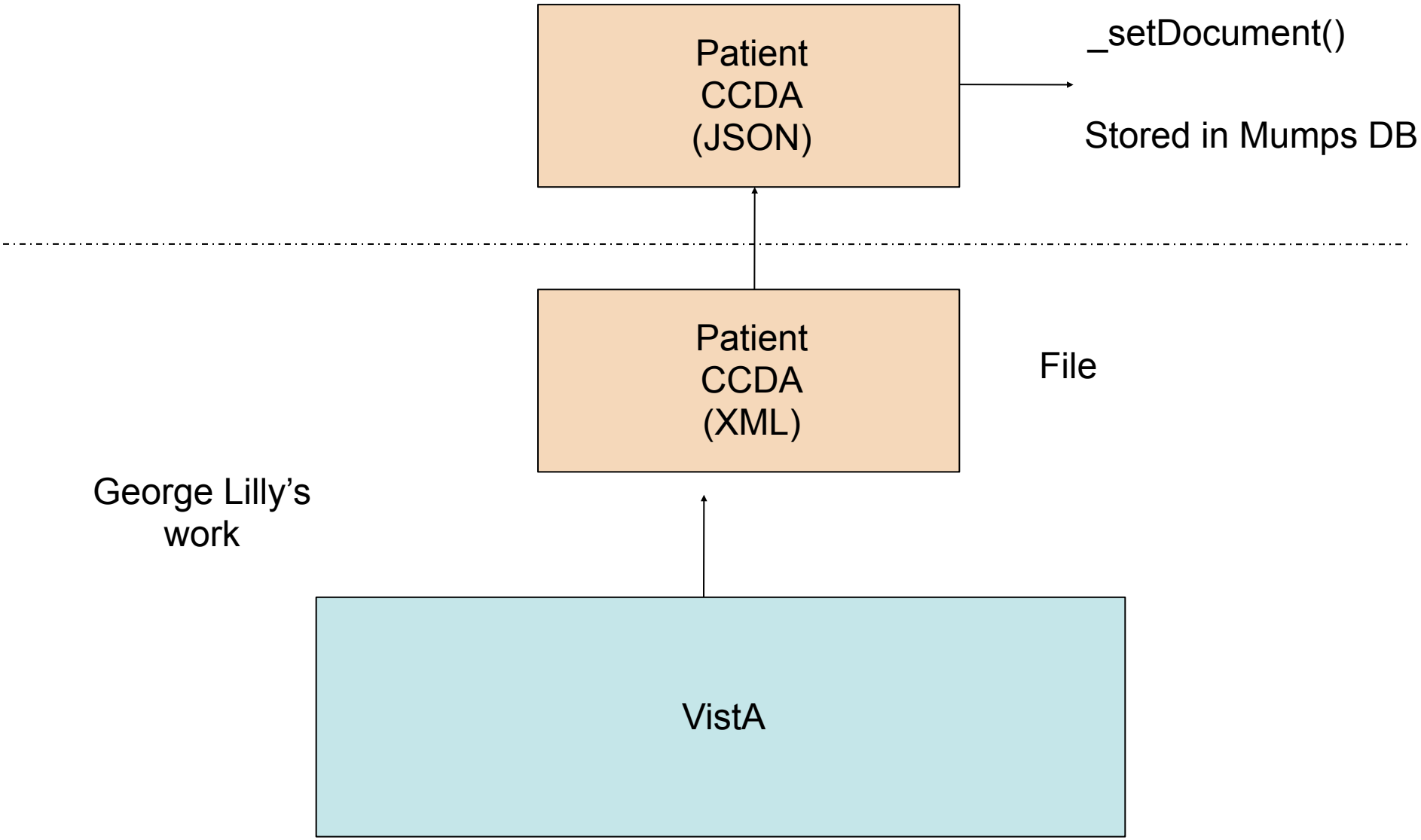
HL7 FHIR handling in EWD.js

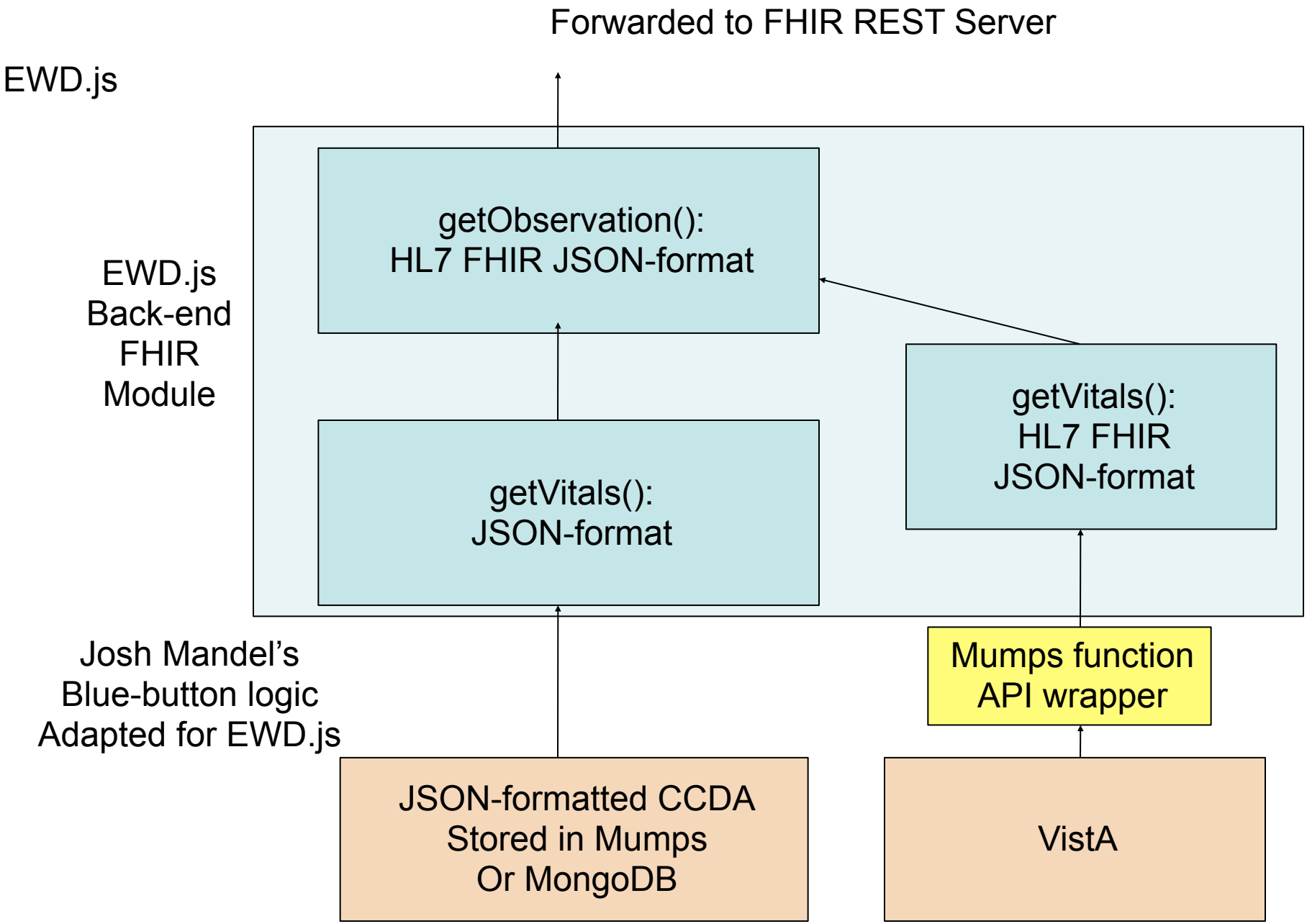
- FHIRServer module for EWD.js
  - parsing incoming FHIR requests
    - interfacing to VistA / FileMan APIs
      - read and write
  - Current version limited to what's required for the Vista Novo demo
      - /observation : gets all observations for a patient
      - /observation/create: saves a new observation
    - ready for extension by community
    - will be posted on GitHub
    - will be Apache 2 licensed

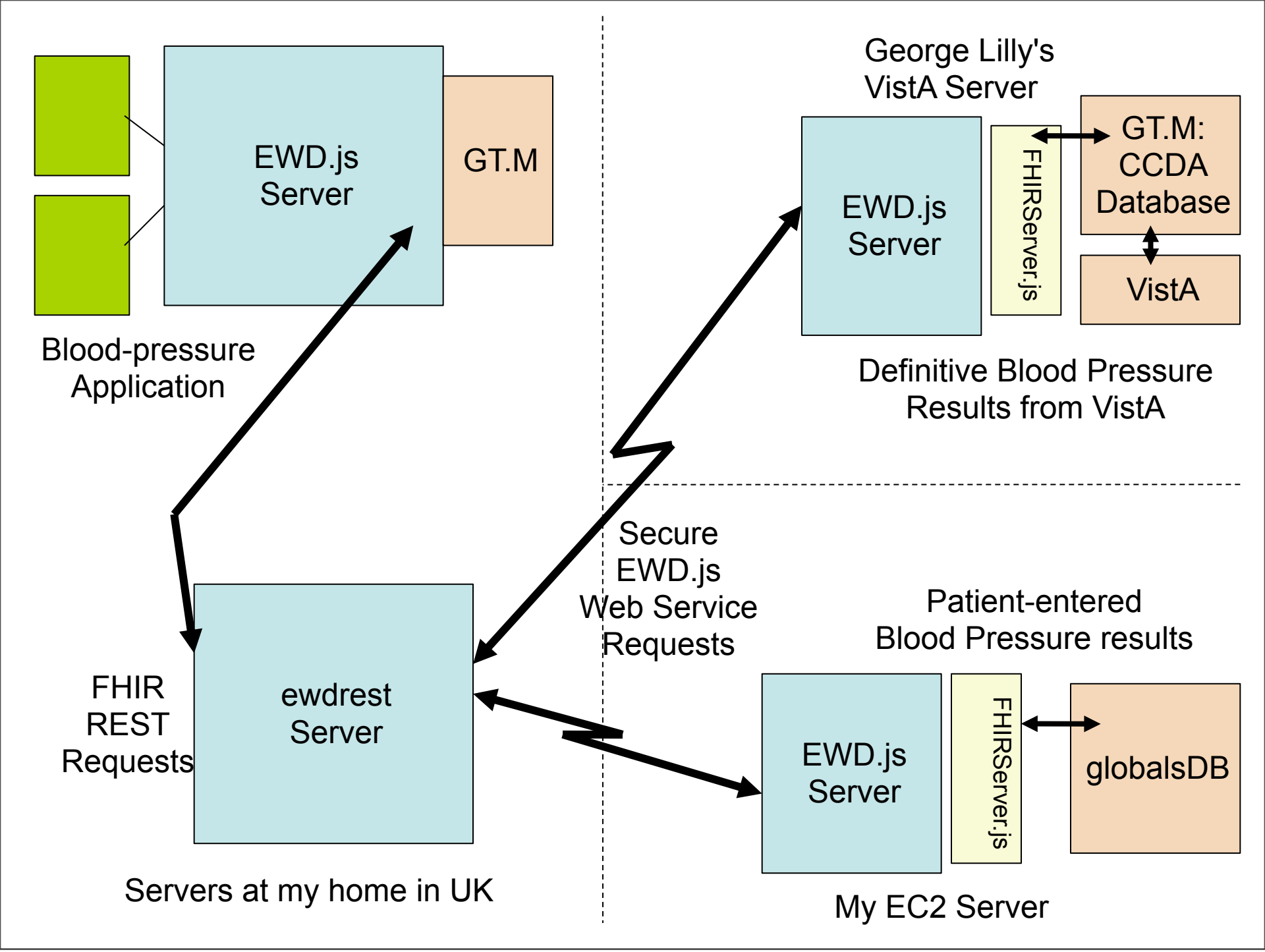

REST
Requests
ewdrest
Server
Signed
EWD.js
Web Service
Request
EWD.js
Server
Back-end
module
GT.M
Caché or
MongoDB
REST
Response
ewdrest
Server
Web Service
JSON
Response
EWD.js
Server
Back-end
module
GT.M
Caché or
MongoDB


Back-end module
EWD.js Server
GT.M Caché or MongoDB
REST Requests/ Responses
ewdrest Server
Back-end module
EWD.js Server
GT.M Caché or MongoDB
etc…

EWD.js / Node.js




Forwarded to FHIR REST Server
EWD.js
EWD.js
Back-end
FHIR
Module
getObservation():
HL7 FHIR JSON-format
getVitals():
JSON-format
getVitals():
HL7 FHIR
JSON-format
Josh Mandel's
Blue-button logic
Adapted for EWD.js
Mumps function
API wrapper
JSON-formatted CCDA
Stored in Mumps
Or MongoDB
VistA


EWD.js
Server
GT.M
Blood-pressure
Application
George Lilly's
VistA Server
EWD.js
Server
FHIRServer.js
GT.M:
CCDA
Database
VistA
Definitive Blood Pressure
Results from VistA
Secure
EWD.js
Web Service
Requests
FHIR
REST
Requests
ewdrest
Server
Patient-entered
Blood Pressure results
EWD.js
Server
FHIRServer.js
globalsDB
Servers at my home in UK
My EC2 Server