[ReactiveX, or Rx\*](http://reactivex.io/rxjs/), is an API for asynchronous programming with observable streams.

observables are streams.

 an observable is a "stream" that represents a collection over time.

[**https://auth0.com/blog/rxjs-advanced-tutorial-with-angular-web-speech-part-1/**](https://auth0.com/blog/rxjs-advanced-tutorial-with-angular-web-speech-part-1/) **god**

"Reactive programming is based on propagating and responding to incoming events over time, declaratively."

RxJS is a library for reactive programming using Observables, to make it easier to compose asynchronous or callback-based code. This project is a rewrite of [Reactive-Extensions/RxJS](https://github.com/Reactive-Extensions/RxJS) with better performance, better modularity, better debuggable call stacks, while staying mostly backwards compatible, with some breaking changes that reduce the API surface.

 Node.js is an open source server framework.

Node.js allows you to run JavaScript on the server.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

## What Can Node.js Do?

* Node.js can generate dynamic page content
* Node.js can create, open, read, write, delete, and close files on the server
* Node.js can collect form data
* Node.js can add, delete, modify data in your database

## What is a Node.js File?

* Node.js files contain tasks that will be executed on certain events
* A typical event is someone trying to access a port on the server
* Node.js files must be initiated on the server before having any effect
* Node.js files have extension ".js"

Node.js can be used in database applications.

One of the most popular NoSQL database is MongoDB.

<https://www.w3schools.com/nodejs/nodejs_mongodb_find.asp>

## Find One

To select data from a collection in MongoDB, we can use the findOne() method.

The findOne() method returns the first occurrence in the selection.

**Anthem oss Technical Architecture overview**

**Objective**

Understand the underlying framework architecture and technology stack

**Content**

**Domain Model** - For any application we first have to define the business entity/entities. This would be the first step in the process of building the product.

**State** - The value of every entity and its corresponding attributes is referred to as state by the framework. There could various events for example generated based on the state and certain other things associated with the state. To get the history of changes that happened on an entity attribute, we would need audit to be enabled.

**Config** - Once we have the domain model, we can define the configuration for the view, workflow and the rules. The view definition configs, the mapping to the domain model, the workflow(if any) and the corresponding view and core domain rules can be written.

#### Configuration

* Business Entity configuration
* View configuration
* Business Rule configuration
* Workflow Configuration

**Command** - The command is the instruction that the framework understands to execute and come back with an output. It is similar to writing the traditional method calls for button click to do some business logic but just that we have standardized the process of writing such to the domain specific language that the framework understands.

#### Command

* Query DSL: The framework processes information using url. The url is based on a query dsl structure. It consists of two parts;
* Target application identifier: Anything prior to /p identifies the application associated with the request
* Domain identifier: Anything post /p identifies the domain for which the request is to be processed.
  + Action
* \_new: Creates a new instance for the model
* \_get: Fetches the instance of the model referenced by the Id
* \_save: Saves the model into the database
* \_replace: Replaces the model state
* \_update: Updates the model state
* \_remove: Removes the model from the database
* \_search: Searches the model based on a search criteria
* \_process: Executes asigned workflow process or custom hanlders

 All standard BPM functions are available for creating business processes. Business process can be defined to manage business entity lifecycle and for creation of stateless processes that executes complex business functions.

Steps for configuring a BPMN process as a function

* Create a BPMN process with a unique process id
* Define d config url with action as **\_process**,**fn** as **\_bpm** and **processId** as the name of the process id. Ex: **@Config(url="/p/patient:<!/.m/id!>/\_process?fn=\_bpm&processId=createcaseforpatient")**

**Function Handlers** are an abstraction within the framework to execute a common set of instructions for a given **Action**.

Given an action, a particular function handler can be executed by specifying a value for the query parameter **fn**.

Example configuration for a \_set function handler

|  |
| --- |
| @Config(url="/p/patient:<!/.m/id!>/\_process?fn=\_set")  **process** is the **Action** and **set** is the Function Handler  **Predefined Function Handlers**  There are several default function handlers defined within the core framework to handle common framework  instructions, such as setting the state of a parameter (***set***) or adding a parameter value into a collection (***add***). |

Predefined Function Handlers

| **Handler** | **Action** | **Description** |
| --- | --- | --- |
| \_param | \_get |  |
| \_default | \_nav |  |
| \_initEntity | \_new |  |
| \_add | \_process | Adds parameter value to a collection. |
| \_addCollection | \_process |  |
| \_bpm | \_process | Invokes a statless bpm process. |
| \_eval | \_process |  |
| \_set | \_process | Sets parameter value. |
| \_setByRule | \_process | Set parameter value through a rule. |
| \_example | \_search |  |
| \_lookup | \_search |  |
| \_query | \_search |  |

# Config

A class with @Config annotation is used to perform an action on button click. In most cases, the action is to retrieve values via HTTP Rest calls from database (MongoDB), and display on the web page.

The possible Actions are: -

* get for HTTP GET
* new for HTTP post
* update for HTTP update
* delete for HTTP delete
* search for searching
* nav for navigation
* process for custom process/ work-flow definitions

## Configs

@Configs is a collection of configuration calls. Multiple calls to the database can be made simultaneously using @Configs, where each call will be represented by @Config.

As shown in the example, when the submit button is clicked, three http calls that are being made viz. update, process, and navigation.

@Configs(

{ @Config(url="~/client/org\_name/\_update"),

 @Config(url="~/client/org\_name/\_process?fn=\_set&amp;url=/p/cmcase/\_search?fn=query&amp;

where=cmcase.patientReferred.firstName.eq('&amp;lt;!/.m/patientReferred/firstName!&amp;gt;').and(cmcase.patientReferred.lastName.eq('&amp;lt;!/.m/patientReferred/lastName!&amp;gt;'))"),

 @Config(url="~/client/org\_name/\_nav?pageId=vpAdvancedCaseSearch") })

 @Button(type=Button.Type.PRIMARY, formReset=false)

 private String submit;

# Domain

Core Config configuration @Domain annotation persists data.

Core config @Domain will always be followed by @Repo that will specify the way data is persisted.

**includeListeners={ListenerType.persistence, ListenerType.update}** of @Domain specifies that the data will be persisted.  
**value=Database.rep\_mongodb** of @Repo specifies that a class with @Domain annotation will use MongoDb for persistence.