**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version #** | **Date** | **Author / Editor** | **Version / Revision Comments** |
| 1.0 | 11/20/2018 | Vinod Durisala |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contents**:

* Architecture of the Nimbus framework
* Detailing on two main technologies used
* How @Config annotations are used in the framework with examples

**Architecture of the Nimbus framework:**

* **Nimbus Objective:**

Users would be able to draw parallel between how a traditional application is developed and how an application can be developed using Nimbus Platform.

**Benefits**:

* Providing the ability to build application through configuration.
* Providing boilerplate code for cross cutting concerns

**Nimbus Architecture**:



**Content**:

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

**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

**Command**: The command is the instruction that the framework understands to execute and come back with an output. It is like 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

**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.

* Two main technologies in Nimbus

The framework is primarily having 2 main components

**FrontEnd** : Built using Angular 4, RxJs, SASS

**BackEnd** : Built using Spring framework components, Activiti, Drools, Query DSL, RxJava

**Spring framework Components**: Below are few many spring components used

**Spring Batch Integration**: We have two approaches to build Spring batch

* Step by step:

This Approach will be used to any simple business logic like reading input file and triggering/doing some actions

* Junk process:

This Approach will be useful for doing multiple actions in single flow like reading data from file/DB and inserting to DB/file and it has approach has many features like how many records to be fetch for single call, etc

**Spring Data Rest**: By using spring its easy way to implement the REST services using @RestController and it will provide the same method functionalities

**E.g.**: GET, POST, PUT, ect

**Spring Boot**: With the spring boot it will reduce so much code to write and Spring Boot has below four major compoments

* Spring Boot Starter
* It will help to configuring the multiple dependencies
* Spring Boot Auto Configure
* It will help to no need to create the spring configuration file. It will help managing the beans automatically
* Spring Boot CLI
* With this feature the Spring boot application we can run through command prompt
* Spring Boot Activator
* This feature will help to manage the beans life cycle

**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
* Actions
* \_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 criterion
* \_process: Executes assigned workflow process or custom handlers

* **How @Config annotations are used in the framework with examples**
* **@Config annotation:**

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.

In the example shown above, when the button is clicked, the control will be navigated to the specified url.  
*nav* is the http call for navigation

**Attributes**:

**url** takes String value

**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

If we have multiple url’s needs to be configured then we have @Configs annotation. It has attribute ‘values’ and it takes value as ‘Config[]’

Example on Config Annotation:

@Config(url="/pageAddEditGoal/tileEditGoal/sectionEditGoal/goalDetailsForm/\_nav?pageId=pageCarePlanSummary")

@Button(type = Button.Type.PLAIN)

Private String cancel;

Example with Configs Annotation:

@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('<!/.m/patientReferred/firstName!>').and(cmcase.patientReferred.lastName.eq('<!/.m/patientReferred/lastName!>'))"),

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

         })