1. Architecture of the Nimbus framework, with a specific focus on any two of the technologies used.

**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.

**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 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.

**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.

**Configuration**

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

**Technologies used in this framework:**

**FrontEnd** - Built using Angular 4, RxJs, SASS.

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

**About QueryDSL**:

Querydsl provides a typesafe querying layer on top of JPA, JDO, JDBC and other backends.

For adding Query DSL in our framework we need to add the below dependencies as part of pom.

<dependency>

  <groupId>com.mysema.querydsl</groupId>

  <artifactId>querydsl-apt</artifactId>

  <version>1.8.2</version>

  <scope>provided</scope>

</dependency>

<dependency>

  <groupId>com.mysema.querydsl</groupId>

  <artifactId>querydsl-hql</artifactId>

  <version>1.8.2</version>

</dependency>

<dependency>

  <groupId>org.slf4j</groupId>

  <artifactId>slf4j-log4j12</artifactId>

  <version>1.5.2</version>

</dependency>

**if we use JPA**:

The JPAAnnotationProcessor finds domain types annotated with the javax.persistence.Entity annotation and generates query types for them

**If we use Hibernate**:

If you use Hibernate annotations in your domain types you should use the APT processor *com.mysema.query.apt.hibernate.HibernateAnnotationProcessor* instead

**End to End Process of persist or save or update a new entity in framework.**

Create core entity.

@Domain(value="owner", includeListeners={ListenerType.persistence, ListenerType.update})

@Repo(value=Database.rep\_mongodb, cache=Cache.rep\_device)

@Getter @Setter @ToString(callSuper=true)

public class Owner extends IdString {

    private static final long serialVersionUID = 1L;

    private String firstName;

    private String lastName;

    private String address;

    private String city;

    private String telephone;

}

Create view route for above entity,

Getter @Setter

public class VROwnerLanding {

@Page(defaultPage=true, layout="home")

private VPOwners vpOwners;

}

In Above class there is dependency with VPOwners. Like complex type.

So we have to write class as below.

@Getter @Setter

public class VROwnerLanding {

@Page(defaultPage=true, layout="home")

private VPOwners vpOwners;

@Model

@Getter @Setter

public static class VPOwners {

@Tile(title = "OWNERS", imgSrc = "resources/icons/task.svg#Layer\_1", size = Tile.Size.Medium)

private VTOwners vtOwners;

}

}

Againg VPOwners has a relation with VTOwners;

@Getter @Setter

public class VROwnerLanding {

@Page(defaultPage=true, layout="home")

private VPOwners vpOwners;

@Model

@Getter @Setter

public static class VPOwners {

@Tile(title = "OWNERS", imgSrc = "resources/icons/task.svg#Layer\_1", size = Tile.Size.Medium)

private VTOwners vtOwners;

}

@Model

@Getter @Setter

public static class VTOwners {

@Section

private VSSearchOwnerCriteria vsSearchOwnerCriteria;

@Section

private VSOwners vsOwners;

}

}

Once all the entity and view relation mappings done run the drools to show or hide the buttons which are to be display.

And restart the application to see the changes.