Senior Backend Applicant Test

This test is meant to measure applicant skills regarding your:

- Ability to kickstart a project
- Ability to highly configurable implement state machines
- Ability to rapid-prototype using available plugins and framework
- *Excellence in selection of appropriate architecture and model depending on problem*

What is Provided:

- This document
- A **BIG** wish for success! We are sure that you will ace this test :)

Behavioral Specs:

Please consider a system with following behavioral specs:

- Visitor can create an Object A without registration.
- Visitor can create many *Object A(* A visitor holds a session)
- Visitor can register into our system and becomes user(very dummy login procedure should do it)
- Visitor does NOT necessarily needs to create *Object A* before he registers
- After registration,most recent *Object A* (if exists) needs to linked to user
- User/Visitor can Not remove *Object A*
- If *Object A* is missing, user should NOT be allowed to do further operations until he adds *Object A*
- User adds Object B,C,D. Each created object needs to linked to *Object A*.
- User can only add a single *Object B or C or D* at a time. (you can NOT add all of them at the same time)
- User can only add *Object B, Object C and Object D* in the following order *B, D, C*
- User can NOT add multiple *Object B or Object C or Object D* (at the end there must a single instance of each)
- User can NOT remove *Object B or Object C or Object D.*
- User can NOT proceed to the next step without adding all 3 objects (*B,C,D*)
- Right after User finished adding all objects System will add an Object R with an additional internal property called ***'foo'*** that a randomly chosen value(true/false). *Object R* needs to be linked to Object A.
- User can NOT remove *Object R.*
- User can query system. This shall return User's recent Object A(if exists) alongside with linked other objects(we want to see the status of user). Additionally, If *(Object R).foo* is **true** do nothing after returning status, If *(Object R).foo* is **false**, Remove all objects (*Object A,B,C,D,R*). However, User will remain registered and he can start over again(adding *Object A* so on...).

OPTIONAL:

- A an admin (Let's call him Superman) can delete only *Object R and C* from any User.
- Superman cannot remove *Object C* before *Object R* is removed.
- Another admin (Let's call him Batman) can do whatever he wants to do(add and remove and objects)
- Batman's actions should not cause any break-up in the systems(like removing C before R or removing A before the rest or removing B before C and D or adding R before B,C,D. You got the idea)

Object definitions:

Object A,B,C,D

```
{
id: 1, /*will be unique within its kind(for example Object A cannot have 2 instances with id = 1)*/
Type: 'A', /*B,C,D respectively*/
created_at: '01/01/1970' /*creation time must be dynamic*/
/*you can add more properties in case you need*/
}
```

Object R

```
{
id: 1, /*will be unique within its kind*/
Type: 'R', /*lovely car from Honda :) */
created_at: '01/01/1970', /*creation time must be dynamic*/
Foo: false
/*you can add more properties in case you need*/
}
```

Test procedure:

- Applicant must kickstart a Python, Java, Go, Javascript codebase(You can pick your favorite programming language, Framework etc. You can use available seed projects, please use GIT as source version control)
- Applicant implement required REST APIs to handle interactions(register, add A, add B, check status etc..)
- Applicant must implement a control structure(state machine) to handle rule-set defined above (behavioral specs)

Hints:

- Methodology for creating project structure, Object relations, etc. is totally up to developer(we would like to see how you approach to this problem.)

- Developer is free to use any 3rd party plugin to enhance app and accelerate development speed(plugin must be usable for commercial projects)

- Using advanced concepts(if it is permitted by language/framework you have chosen) like DI(Dependency Injection), AOP(aspect oriented programming) is a big plus

- You can optionally persist objects. You are free to use a Relational/non-relational DB

- Suppose that this rule-set (behavioral specs) and superuser permissions are prone to change by business people. Having state machine and permissions coded static is definitely not favourable. We need a configurable system. So this is the by far the most important concern while we evaluate your application. Try your best shot on that.

- You might want to draw a flowchart diagram before you start implementation. It helps you to understand machine states in an easier way. However, we don't need to see it. This is not even optional. We know what flowchart looks like. Just do it for your own convenience.

- ***Don't worry. You don't have to finish all of the task. Sometimes less is more. Just try to make your approach understandable by us by mentioning stuff you could do in code comments. Don't forget that we, as a fair company, still favour the candidate who completed implementation over the candidates who mentioned their approach in comments.***

Delivery method:

Via email: compressed file.
Via git url: github, bitbucket FTW!

Solution should include a README documentation that explains how to set-up and boot-up Solution(A small description about API usage as well. We will use Postman for testing application.).