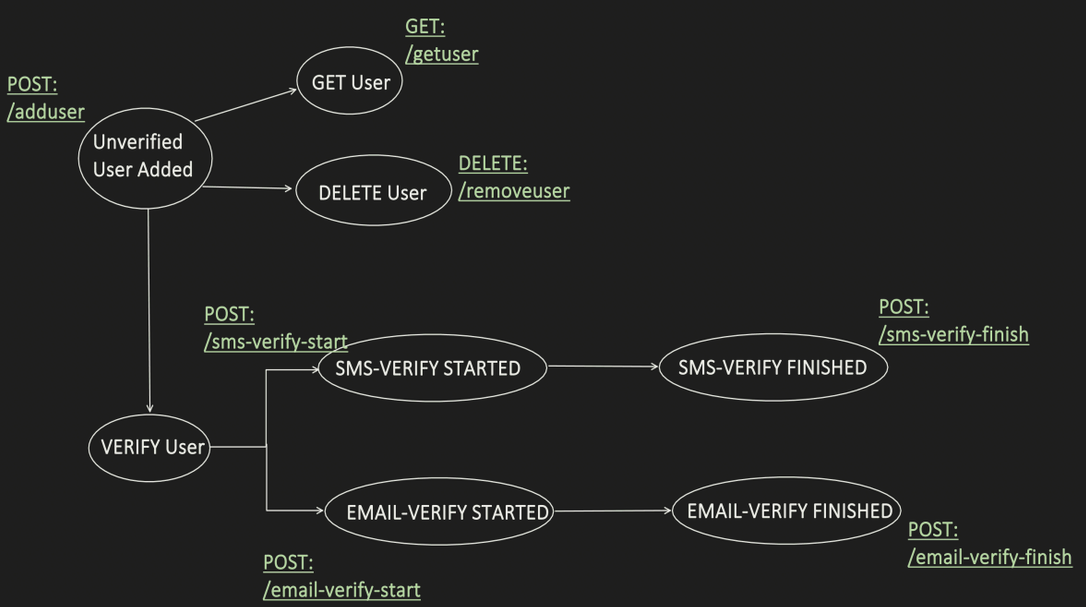
HATEOAS

Hypermedia As The Engine Of Application State

It tells the client, what the next action you can perform on particular item.

For example:



API Response, after *"Unverified User Added"* POST: /adduser API

Without *HATEOAS* link

```
{
    "userID": "123456",
    "name": "SJ",
    "verifyStatus": "UNVERIFIED"
}
```

With *HATEOAS* link

```
{
    "userID": "123456",
    "name": "SJ",
    "verifyStatus": "UNVERIFIED",
    "links": [
        {
            "rel": "self",
            "href": "http://localhost:8080/api/getUser/123456",
            "type": "GET"
        }
    ]
}
```

Before understanding HOW to do this, lets understand WHEN to use and WHY to use HATEOS link?

2 Major Purpose of using HATEOAS link is to achieve

- "LOOSE COUPLING" and
- "API DISCOVERY"

To achieve above, server provides the next set of APIs (actions) in the Response itself, which client can take. So that client have less business logic around APIs (which API to invoke, when to invoke, how to invoke etc....)

But, Adding all next set of ACTIONS can make our API Response Bloat up and has several disadvantages:

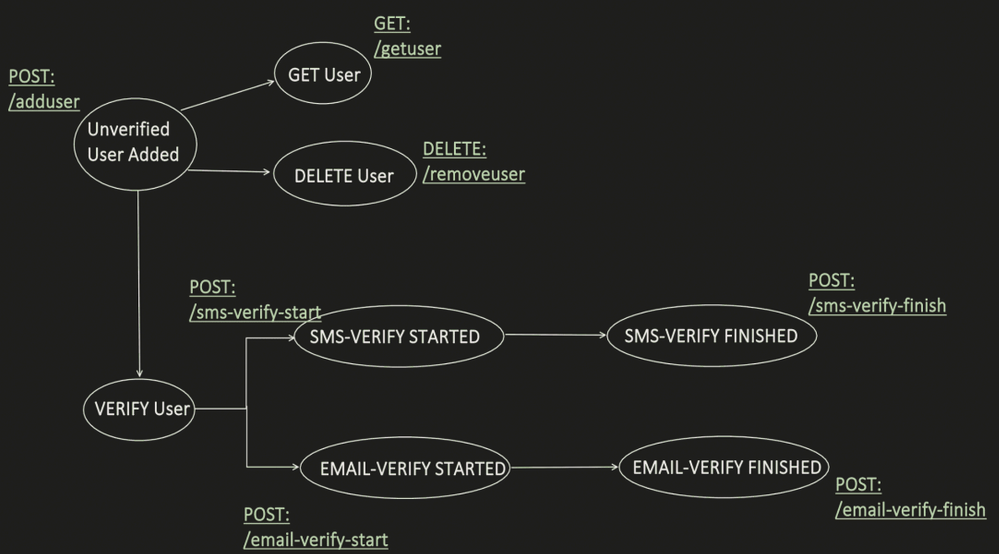
- Increase complexity at server side.
- Latency impact.
- Increase Payload size.


```
{
    "userID": "123456",
    "name": "SJ",
    "verifyStatus": "UNVERIFIED",
    "links": [
        {
            "rel": "self",
            "href": "http://localhost:8080/api/getUser/123456",
            "type": "GET"
        },
        {
            "rel": "remove",
            "href": "http://localhost:8080/api/removeuser/123456",
            "type": "DELETE"
        },
        {
            "rel": "update",
            "href": "http://localhost:8080/api/updateuser/123456",
            "type": "PATCH"
        },
        {
            "rel": "verify-start",
            "href": "http://localhost:8080/api/sms-verify-start/123456",
            "type": "POST"
        },
        {
            "rel": "verify-finish",
            "href": "http://localhost:8080/api/sms-verify-finish/123456",
            "type": "POST"
        }
    ]
}
```


Never Ever add all possible next set of actions (API) just like that.

So, proper analysis need to be done, what actually will help us to achieve LOOSE COUPLING

Now, if we see the above diagram again



I see, the tight coupling lies during VERIFY process.

Client need some info, before it can decide which Verify API to invoke. For example:

```
{
    "userID": "123456",
    "name": "SJ",
    "verifyStatus": "UNVERIFIED",
    "verifyType": "SMS",
    "verifyState": "NOT_YET_STARTED"
}
```

Client need to put business logic, that

```
if(VerifyStatus == "UNVERIFIED")
{
    if(verifyType== "SMS")
      {
          if(verifyState== "NOTE_YET_STARTED")
          {
                Call POST: /sms-verify-start
          }
          Else if (verifyState == "STARTED")
          {
                Call POST: /sms-verify-finish
          }
      }
      Else if(verifyType == "EMAIL")
      {
          if(verifyState== "NOTE_YET_STARTED")
          {
                Call POST: /email-verify-start
          }
          Else if (verifyState == "STARTED")
          {
                Call POST: /email-verify-finish
          }
      }
```

This dependency, can be removed by HATEOAS link

```
{
    "userID": "123456",
    "name": "SJ",
    "verifyStatus": "UNVERIFIED",
    "links": [
        {
            "rel": "verify",
            "href": "http://localhost:8080/api/sms-verify-finish/123456",
            "type": "POST"
        }
    ]
}
```

Now, we have achieved LOOSE COUPLING and Client code looks like this.

```
if(verifyStatus == "UNVERIFIED") {

    //invoke the verify URI, given in HATEOAS link

}
```

Dependency Required:

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-hateoas</artifactId>
<version>2.6.4</version>
</dependency>
```

```
@RestController
@RequestMapping(value = "/api/")
public class UserController {

    @Autowired
    User user;

    @PostMapping(path = "/adduser")
    public ResponseEntity<UserResponse> addUser() {

        UserResponse response = user.getUser();

        //our business logic to determine which Verify API need to be invoked.
        Link verifyLink =  WebMvcLinkBuilder.linkTo(UserController.class) WebMvcLinkBuilder
                .slash( object: "sms-verify-finish")
                .slash(response.getUserID())
                .withRel("verify") Link
                .withType("POST");

        response.addLink(verifyLink);

        return new ResponseEntity<>(response, HttpStatus.OK);
    }
}
```

```
public class HateoasLinks {

    private List<Link> links = new ArrayList<>();

    public void addLink(Link link) {
        links.add(link);
    }
}
```

```
public class UserResponse extends HateosLinks {

    private String userID;
    private String name;
    private String verifyStatus;

    //getters and setters here
}
```

Other way to create Link:

```
Link verifyLink =  Link.of( href: "/api/sms-verify-finish/" + response.getUserID())
        .withRel( relation: "verify")
        .withType("POST");
```


```
{
    "userID": "123456",
    "name": "SJ",
    "verifyStatus": "UNVERIFIED",
    "links": [
        {
            "rel": "verify",
            "href": "http://localhost:8080/api/sms-verify-finish/123456",
            "type": "POST"
        }
    ]
}
```