Get Nodeb Design

1. Controller Layer

- Add Route:

  `GET("/v1/nodeb/:ranName", controller.GetNodeb)`
- Change name of requestController to **nodebController**:

```
controller := controllers.NewNodebController(logger, rmrService)
```

- New Controller method:

  `func (rc RequestController) GetNodeb (writer http.ResponseWriter, request *http.Request, params httprouter.Params)`

- **GetNb Business Logic**

Code	Comment
`ranName := params.ByName("ranName")`	
`respondingNode, err := GetRnibReader().GetNodeb(ranName)`	

`if err {`
`  writer.WriteHeader(http.StatusNotFound)`
`  return`
 	
`}`
	OR `writer.WriteHeader(http. StatusInternalServerError)` , depends whether we haven't found this key or we're having issues with DB/Server

`m := jsonpb.Marshaler{}`	
`result, err := m.MarshalToString (respondingNode)`	Convert the unmarshalled proto to JSON. Handle error
`w.Header().Set("Content-Type", "application/json")`	
`w.Write(result)`	Respond with 200OK and responding node JSON representation. (Concern: JSON size limitation of Go in case of maximal number of entities).

2. Entities

```
message RespondingNode{

    ConnectionStatus connectionStatus = 1;

    string ip = 2;

    uint32 port = 3;

    Node.Type nodeType = 4;

    oneof Nodes{

        ENB enb = 5;

        GNB gnb = 6;

    }

}


enum ConnectionStatus{

    UNKNOWN_ConnectionStatus = 0;

    Connected = 1;
```

```
    NotConnected =2;
}


message Node{
    enum Type {
        UNKNOWN = 0;
        ENB = 1;
        GNB = 2;
    }
}
```

3. RnibReader

- New package: **rnibReade**

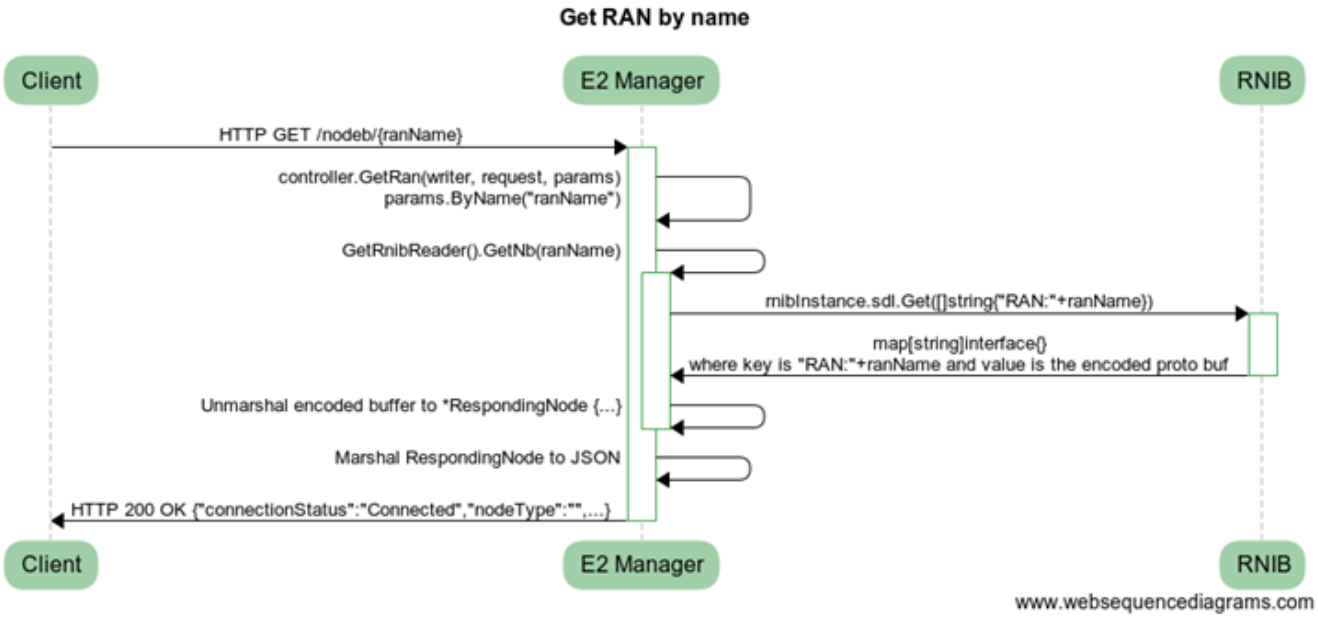
```
RnibReader interface {
            GetNb(key string) (*RespondingNode, error)
}


// Get may also create an instance
func GetRnibReader() RnibReader {
         return pool.Get().(RnibReader)
}
```

- Init RNIB instances pool (concern: separate the pool from rnibWriter layer)
- **GetNodeb Business Logic**

Code	Comment
`keyToCheck := "RAN:"+ranName`	
`defer pool.Put(rnibInstance)`	
`m, err := rnibInstance.sdl.Get([]string{ keyToCheck })`	USE SDL to fetch data from REDIS. RETURNS: (map[string]interface{}, error)
`data``, ok := m[keyToCheck]`	Get the value of the requested key. This will be a **byte array**
`pb := &RespondingNode{}`	Create a new pointer to the responding node
`proto.Unmarshal(data, pb)`	Unmarshal parses the protocol buffer representation in buf and places the decoded result in pb
`return pb`	

4. Sequence Diagram


Get RAN by name
Client
E2 Manager
RNIB
HTTP GET /nodeb/{ranName}
controller.GetRan(writer, request, params)
params.ByName("ranName")
GetRnibReader().GetNb(ranName)
rnibInstance.sdl.Get([]string{"RAN:"+ranName})
map[string]interface{}
where key is "RAN:"+ranName and value is the encoded proto buf
Unmarshal encoded buffer to *RespondingNode {...}
Marshal RespondingNode to JSON
HTTP 200 OK {"connectionStatus":"Connected","nodeType":"",...}
Client
E2 Manager
RNIB
www.websequencediagrams.com