Securing the application

## Securing the database:

To secure the database, my approach will be in four steps.

Step 1: Use a firewall to control the a. protocol b. port c. the source address.

Step 2: Encrypt the data with a passphrase using Secure Socket Layer Certification.

Step 3: Create roles and provide authentication based on the assigned roles.

Step 4: Set an authentication timeout for the DB.

## Securing the REST API:

1. The API can be secured by using https and SSL.
2. A session attribute can be added to the application.
3. Authentication can be implemented using Django OAuth kit / Django REST framework OAuth.

## Securing the angular application:

1. Prevent script injection using Sanitization in the angular application.
2. Use authentication cookies and angular’s build in HttpClient to prevent Cross-Site Reference Forgery.

class employee:

def \_\_init\_\_(id,salary,name,position,SIN):

this.salary = salary

this.name = name

def getSalary():

return this.salary

def getName():

return this.name

def getPosition():

return this.position

def getSIN():

return this.SIN

def getId(){

return this.id

}

class data\_engineer(employee):

def \_\_init\_\_(database\_credentials, database\_access\_level):

this.data

def getDatabaseCredentials():

return database\_credentials

def getDatabaseAccessLevel():

return database\_access\_level

class software\_developer(employee):

def \_\_init\_\_(docker\_registry\_account):

def getDockerRegistryAccount():

return docker\_registry\_account

class getEmployeeDetails:

employeeList