

# Technical Report

Access Review Team

Sponsored by Coupa Software

Point of Contact: Phillip Cox

Advisor: Patrick Tague

Team member: Kaiyu Liu, Rundong Liu, Haoran Liu

# **Table of Contents**

- 1. Introduction**
- 2. Motivation**
- 3. Related work**
- 4. System design**
- 5. System implementation**
- 6. Experiments and analysis**
- 7. Conclusions and future work**

## 1.Introduction

### Project Overview:

Least privilege is the basic security principle when dealing with resource management. A user is only allowed to access the resources he needs, in order to deal with his job but not other unrelated ones. This principle is important to almost all companies to ensure data privacy and preventing unintended information leakage.

Access control plays an essential role in implementing least privilege principle. By giving an access control list, a company can supply different permissions on different resource based on user's role and responsibility. In order to ensure that the access control works properly, access review is a significant process to review the correctness of access control list and it needs to be paid more attention to when managing sensitive data and resources.

Nowadays, more and more web applications and services are used by many companies. Most of the companies require performing user access reviews to prevent that none of left employees still have company accounts, and guarantee that all the other users should have the correct access privileges at least four times a year. In most companies, it is done manually and requires a lot of burdensome human work.

## 2 Motivation

Coupa software has many services for internal use. For example they have AWS, RightScale, DNS and so on. There are also lots of employees in coupa with different roles. They are supposed to access and operate different applications and data within. For example most normal users would have read permissions of DNS data because they need to do domain lookup. But administrators of the DNS server will need permission to read and modify DNS data. So an efficient access review tool should be used to manage all restricted resources and applications and make sure there is no permission abuse in the real implementation.

### Goals and Solution:

In this Access Review project, we need to come up with a solution to make access review process easier to manage and realize better visualization. This solution has to keep permission control more reliable and can greatly relieve the work of administrators

Our solution is to build a web application to handle role based access review process. It is a basic applications as a service. The web application can provide access review service to enterprise users more efficiently, because it is distributed and maintained for all users at a single point – in the web server side.

All the modification and review operation are facilitated by providing different interface for different users implemented on the cloud service platform. Integrated with a variety of automated tools, account management, permission review can be implemented together in the application. In this way management and administration are more controllable and efficient too.

### **3 Related Work**

#### **3.1 Clarify Customer Needs**

For the first step of the project, we need to make clear the requirements of our customer.

We visit Coupa Software in their HeadQuarter and talk about the project in detail.

We get to know the real needs of their company and discuss the idea with them and make an agreement towards how to do this.

#### **3.2 Research on Access Control**

Access control systems have authorization, identification, authentication, access approval and some other abilities. By using login credentials including passwords, personal identification numbers, biometric scans, and physical or electronic keys, it can realize all mentioned abilities.

Access control is very useful in the real-world life. In some place physical access control is needed, but in our project we mean the logic access control and especially deal with the role-based access control (RBAC).

RBAC can make us able to handle lots of authorized tasks dynamically. We can manage the actions based on the flexible functions, relationships, and constraints.

Traditional access control would grant or revoke user access which is different from RBAC. In RBAC, roles can be easily created, changed, or deleted if the enterprise evolves. These can be done without having to individually update the privileges for every user.

### 3.3 Research on Web Development Framework

Play, Django and Ruby on rails are three of the most popular MVC (Model–view–controller) web development framework. These three popular web framework all has its own features , it is very hard to say which one is better than the others.

	Play	Django	Rails
Language	Java/Scala	Python	Ruby
design pattern	MVC	MVC	MVC
Main feature	It aims to optimize developer productivity by using convention over configuration, hot code reloading and display of errors in the browser.	Django's primary goal is to ease the creation of complex, database-driven websites	It encourages and facilitates the use of web standards such as JSON or XML for data transfer, and HTML, CSS and JavaScript for display and user interfacing.

We compare their features and our own project goal. Finally we decides to choose Django as our development framework because it is more friendly to database-driven websites and that is exactly what we need to store massive access control matrix.

Django is a high-level MVC Python Web framework that make web development quick and tidy. It is also free and open source and helps developers to create complicated, database-driven websites.

It can help us to finish building applications from scratch very fast. Django takes security seriously and can avoid many common security mistakes.

Its tutorial is all available online and easy to follow. That speeds up the developing process and makes sure deliverables of the project would be on time.

### 3.4 Research on Database

MySQL, PostgreSQL and SQLite are three of the most popular free open source database in web development.

They all have their advantages and disadvantages. So we compare them with our project goals in detail.

MySQL	PostgreSQL	SQLite
Easy to work with	Strong community	Good for developing and testing
secure and speedy to use	Strong third-party support	file-based
Scalable and powerful	Extensible	no user management
has some known issues	Performance is not good as MySQL	Lack of possibility to tinker with for additional performance

Because we want to make our application have reliability and data integrity, so we choose PostgreSQL as our final system database. In this way, we can ensure the integration of the system.

But during development process, we do not deal with massive amount of data. We tend to use SQLite as our development database, because it is lightweight small and easy to migrate.

### 3.5 Make Plans

#### Division of Work

After we are clear about project in details, we begin to divide of work. Every one of us is assigned with specific responsibilities.

- Kaiyu Liu is mainly responsible for the backend part.
- Rundong Liu is responsible for the design and iterative revision part of the project.
- Haoran Liu is responsible for the frontend part of the project.

We also have a lot of cooperation through the whole developing process. So, besides working on our separate part of the project, we set regular meeting time to talk about progress we make and ideas come up with.

In this way, we try to keep the project in a healthy state and make progress in a good manner.

#### Progress Schedule

In order to make the whole team well-organized, we work out a schedule:

### *1st iteration*

- 1.1 Design database storage Schema
- 1.2 Design system workflow
- 1.3 Parse csv file, extract data and save into database
- 1.4 Design basic wireframe for the web interface

### *2nd iteration*

- 2.1 Design database storage Schema
- 2.2 Design system workflow
- 2.3 Parse csv file, extract data and save into database
- 2.4 Design basic wireframe for the web interface

### *3rd iteration*

- 3.1 Improve the functions to show the review pages for auditors
- 3.2 Refine the front end view to make it user-friendly
- 3.3 Refine database schema for better data normalization
- 3.4 Design the functions for administrator to assign the applications
- 3.5 Test for the application of this step

### *4th iteration*

- 4.1 Design the PDF format of the review report to download
- 4.2 Change the SQLite database into Postgresql database
- 4.3 Test the whole application for the usability and refine it
- 4.4 Test the whole application for integration and refine it

## **4 System Design**

### **System Functions Outline**

Our goal is to build web application for access review. The review should provide all the functions below:

- Visualize access control list to facilitate permission review and audit. The userlist and their permissions of these services are supplied in csv file format. We need to parse these data, store it in the backend database and display content in a good manner.
- Provide authentication and session management for different roles. When users log into the system, the dashboard should display different content based on user's role.
- An administrator is needed for the system. The system should provide interfaces for admin to manage the whole system. An admin can perform operations like create app, upload permission file, assign permission for managers and auditors and so on.

- The system should provide a way to generate reports after review process is completed. PDF is a widely used format for report, so we plan to implement a pdf generator for the system to dynamically output report.
- Session management and authentication are still required for the system for security reason.

## Roles Outline

In our apps, there are five kinds of entities Admin, Auditor, Manager, Users and Applications.

- Admin is the one that controls the whole app, he can decide who is the manager of what application, and who is the auditor. If the identity of someone is changed, the admin would change its permission in the application immediately. There is only one admin in the application
- User means the normal user in the app that do not have any privilege to the applications and cannot change the userlist. They can only review the applications that assigned to them.
- Auditor is the one that audit all the applications, he can verify the process and make sure everything is going right. Auditor can only view the access permissions for all services.
- Manager is the one that reviews applications assigned to him, he can change the permission if he/she finds something is not right in the list. And he can approve or deny the review list.
- Application means the services involved in the company, for example: AWS and Rightscale. When users log into the system, the dashboard would display the applications that are related to themselves.

## Relation

There are several relations in this system:

- App-User-Permission

This is the real access control list. Given an app and a user id, the matrix will show which permission the user has for this application.

All the access review process are related with this relation table and it is the most important part of the project.



- App-Manager

This relation shows the permission of managers. This is assigned by administrator and performs the access control for managers of the system.

- App-Auditor

This relation shows the permission of auditors. This is assigned by administrator and performs the access control for auditors of the system.

### Data Extraction

To get the real access control matrix, we have to parse our raw data to a data structure which can be stored in the back end database.

For this part, we build data extraction module to parse csv files supplied by our customer and build a user-app-permission matrix and store it in the database.

## 5 System implementation

### Model:

For the table for the Admin, Manager, and Auditor, we use OneToOneField(User) because we wish to store information related to default User in Django. This one-to-one model can store non-auth related information about a site user. And also we customize these table with two more features: last name and first name. As for the normal user, we just give them two column in the table: last name and first name because all they can do is to log in to the system and read something, they cannot change the data stored at our database.

The application table is to store the services the company used, like AWS, so we just need one column i.e. application name.

There are two tables store the information of the relationships. One of them is relations between application and manger, we would have the application and manager columns that use the foreign key reference to the application and manager table. The application-auditor table is just the same as the application-manager table.

There is another table called application-permission table, this would has five columns including application, regular-user, manager, status, reviewed\_by. The status would be a choice between "Read and Write " and "Read only".

This model design can handle all the situation and relationship in this application.

```

class Admin(models.Model):
    user = models.OneToOneField(User)
    last_name = models.CharField(max_length=30)
    first_name = models.CharField(max_length=30)

'''
Manager
The one that reviews applications assigned to him
'''
class Manager(models.Model):
    user = models.OneToOneField(User)
    last_name = models.CharField(max_length=30)
    first_name = models.CharField(max_length=30)

    def __unicode__(self):
        return self.first_name + " " + self.last_name

'''
Auditor
The one that audit all the applications
'''
class Auditor(models.Model):
    user = models.OneToOneField(User)
    last_name = models.CharField(max_length=30)
    first_name = models.CharField(max_length=30)

'''
User
The normal user in the app
'''
class RegularUser(models.Model):
    last_name = models.CharField(max_length=30)
    first_name = models.CharField(max_length=30)
    #user_name = models.CharField(max_length=30)

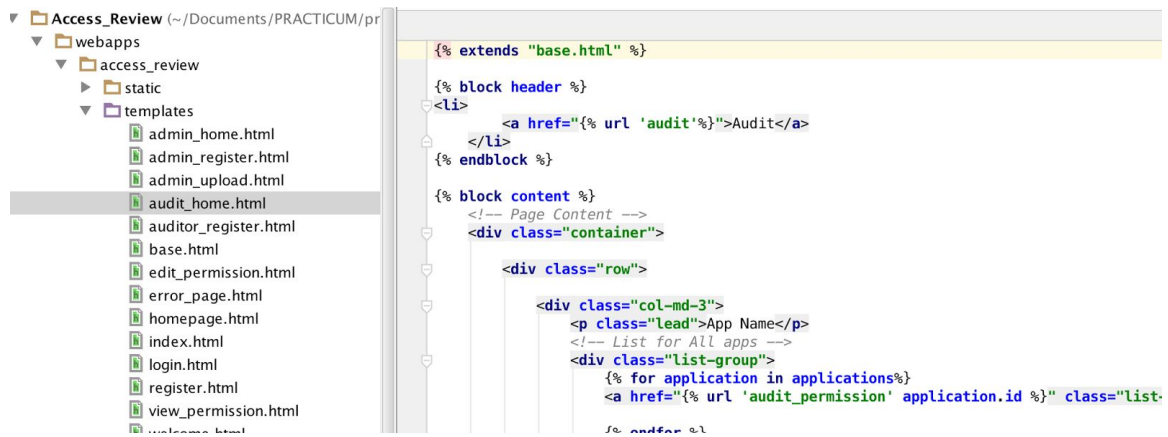
'''
Application
id is autoincremented
'''

```

## View:

We have several template pages to handle the view part. All the data presented in the templates in web display.

There are register and login template which is the basic need of the system. When people register for this website, he/she can choose to register as a manager/auditor/normal users, the system would ask them to type in the email address, first name, last name and password. The user can dive into the system only if they have registered. And the admin would assign these user with different applications. We maintained several template pages including view\_permissions and edit\_permissions. The view\_permissions is designed for the normal users, this page would use for loop to show the lists in the database. And edit\_permissions page would give managers a chance to change the permission for the users, and we would save the changes into database.



## Controller:

All the controllers are implemented in `view.py` file.

This part have the access to the database and use the post, get method to load the html template page and all the actions taken in the frontend would be handled here. All the data in and out database would be controlled . For example: the manager want to change the permission of Dena, the according data in the table in the column would be changed via our controller.

```
context['form'] = LoginForm()
return render(request, 'login.html', context)

form = LoginForm(request.POST)
context['form'] = form
if not form.is_valid():
    print '???'
    return render(request, 'login.html', context)

try:
    user = authenticate(username=form.cleaned_data['username'],
                        password=form.cleaned_data['password'])
    if user == None:
        context = {'form': form,
                    'message': 'The username or password is wrong. Please try again.'}
        return render(request, 'login.html', context)
    login(request, user)
    #need to be updated
except(AttributeError):
    return redirect(reverse('login'))

return filter(request)

@transaction.atomic
def manager_register(request):
    context = {}

    # Just display the registration form if this is a GET request
    if request.method == 'GET':
        context['form'] = ManagerRegistrationForm()
        return render(request, 'register.html', context)

    errors = []
    context['errors'] = errors

    form = ManagerRegistrationForm(request.POST)
    context['form'] = form
    if not form.is_valid():
```

## Frontend:

We use Bootstrap to format the table layout of the tables, buttons and navigations.

## Multithread:

Performance and responsiveness are very essential for our application. Users may be annoyed if the web application regularly appears freeze or responses slowly. Even though a transaction may take very little time to complete, when throughput becomes larger, average waiting time for every user may increase dramatically.

In order to deal with this problem, we make our web application multithreading and can handle multiple requests concurrently.

We use library from Django to implement the multithreading function.

When implementing concurrency we add mechanism to ensure database transaction atomicity. Thus if more than one user modify the data at the same time, the data integrity will not be broken. This function is important for our system to support visiting simultaneously.

## Security:

Since our application plays an administrator's role in the whole access control in the entire company, any security issue may lead to severe damage to resource management. So we do take security seriously.

## Basic Authentication

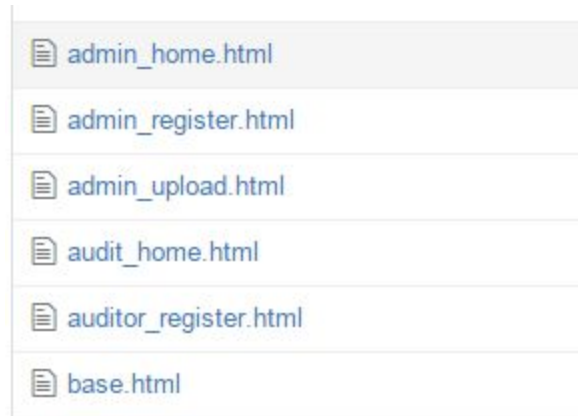
We implemented secure login authentication. Every user is required to input username and password in order to access our system.

In the backend, we realize role based access control. Every user is assigned with a role, and content will be displayed based on user's role. This mechanism prevents user to access data they do not have permission on.

```
# check whether user in customer group
def in_manager(function=None):
    actual_decorator = user_passes_test(
        lambda u: u.is_authenticated() and u.groups.filter(name='Manager').exists()
    )
    return actual_decorator(function)
```

## Web Security Protection

By using Django's template as the display method, our system is not vulnerable to XSS attack. Django's templates escape characters which are particularly dangerous to web interface. This mechanism protects user from malicious input.



By adding anti-csrf token in forms and requests, we ensure our system is safe from CSRF attack. Every time a sensitive action is requested, our backend will check if the CSRF token is the same as we generate before. This makes sure that requests are submitted by the authenticated user.

```
<form id="view_form" class="form-horizontal" name="update-form" method="post" action="/permission/1">
  <input type="hidden" value="WxsPAxdvKWu2VqPvp0UUeEBW4Ds6EUKf" name="csrfmiddlewaretoken"></input>
  <div class="table-responsive"></div>
  <div class="form-actions"></div>
```

#### One-click import access control list:

We need to create a method to convert csv file to abstract access control list and insert these record to the backend database by just clicking on the button. The data would be parsed first and then will be written into the database.

#### Generate PDF Report :

We use ReportLab python package to implement this function  
Reportlab is a python package used to generate customized pdf file  
We use it to generate access review report in a pdf format for users to download and check the list offline.

## 6 Experiments and analysis

We do lots of tests during the developing process. It is mainly focus on the correctness and performance of the whole system.

At the very beginning, we create some data and store them into database, we use that to test our application and make sure everything work well. After the system works fine with some data, we import large data set supplied by our customer and do fully real test cases.

Here is a go through of our system

Firstly, we register as Admin, Manager, Auditor and Normal User.

# Administrator

Username:

admin

First name:

Last name:

Email:

Password:

Confirm password:

REGISTER

# Auditor

Username:

auditor1

First name:

haoran

Last name:

liu

Rundong Liu

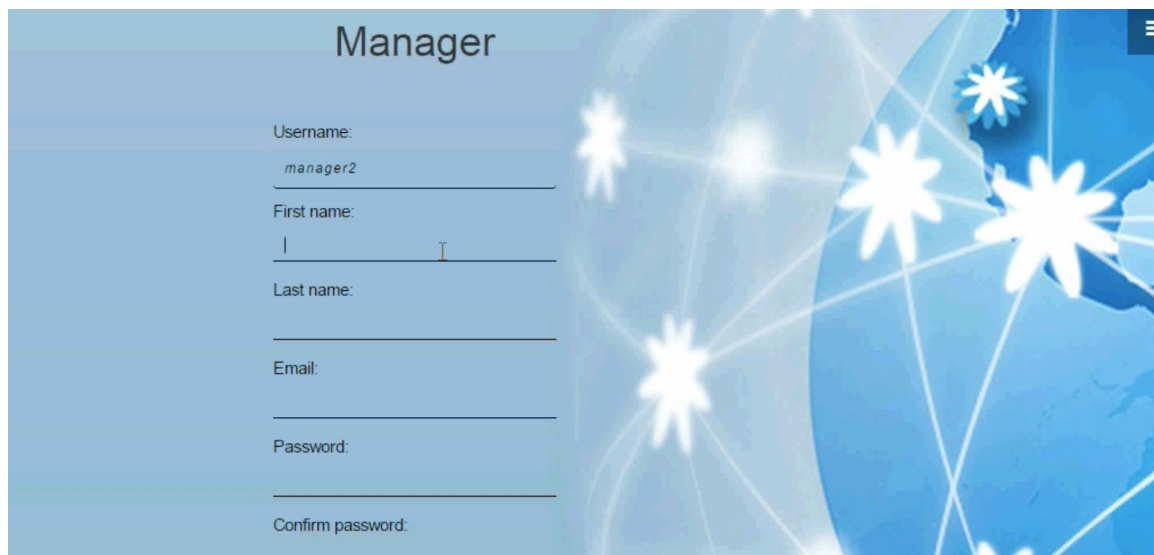
liu

Chrome自动填充设置...

Password:

Confirm password:

REGISTER

A login form titled "Manager" with a background image of a globe and network connections. The form contains the following fields: Username (with "manager2" entered), First name (with "I" entered), Last name, Email, Password, and Confirm password.

Manager

Username:  
manager2

First name:  
I

Last name:

Email:

Password:

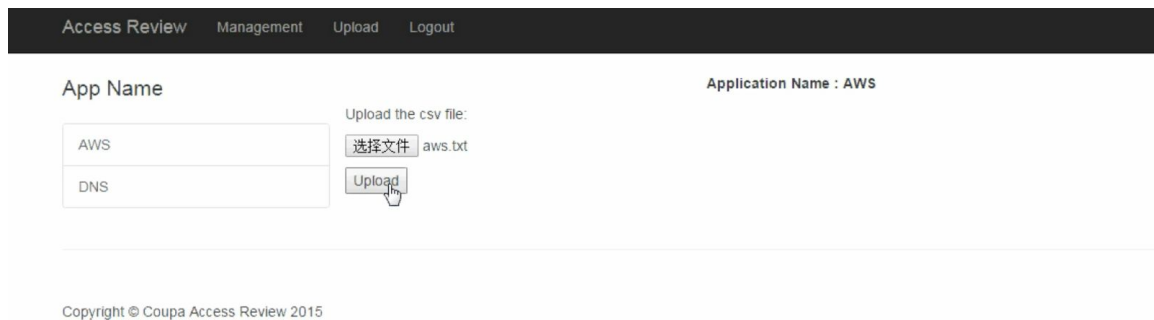
Confirm password:

We log in the system as Admin, Manager, Auditor to test the whole system.

Admin:

As admin, we try to manager to whole system.

Firstly, the admin can create some services and upload the csv files for them.

A screenshot of the "Manager" interface. At the top is a navigation bar with "Access Review", "Management", "Upload", and "Logout". Below the bar, on the left, is a table with "App Name" as the header and two rows: "AWS" and "DNS". On the right, there is a section titled "Upload the csv file:" with a text input containing "aws.txt", a "选择文件" (Select File) button, and an "Upload" button. A mouse cursor is pointing at the "Upload" button. The text "Application Name : AWS" is displayed at the top right of the main content area. At the bottom, a copyright notice reads "Copyright © Coupa Access Review 2015".

Access Review Management Upload Logout

App Name

App Name
AWS
DNS

Upload the csv file:

选择文件 aws.txt

Upload

Application Name : AWS

Copyright © Coupa Access Review 2015

The admin also has the right to assign permission for managers and auditors. Notice that when a manager or auditor is newly registered, he is not able to review or audit any applications. At this time, the administrator can assign an application to him. After that, the manager or auditor can see the contents which he is assigned to.



Access ReviewManagementUploadLogout

Please add application first.

App Name

App Name

Create App

Manager

Approved Manager

Manager's First Name

Manager's Last Name

Denied Manager

Manager's First Name

Manager's Last Name

Auditor

Approved Auditor


Auditor's First Name

Auditor's Last Name

Denied Auditor

Auditor's First Name

Auditor's Last Name



Access ReviewManagementUploadLogout

App Name

AWS

DNS

Create App

Manager

Approved Manager

Manager's First Name

Manager's Last Name

Rundong

Liu

Remove

Denied Manager

Manager's First Name

Manager's Last Name

kaiyu

liu

Assign

Auditor

Approved Auditor

Auditor's First Name

Auditor's Last Name

When logged in as a manager, there would be a dashboard on the left side that can lead manager to all the application that he can do the access review. When the manager clicks on the service name, there would be the whole access review list in the web interface, the manager can review this list and change the permissions if he thinks he needs to do so. For example: he can change the permission from “read and write” into “read only” for Bob, and all the changes would be saved in the database.

16



Access Review
Review
Logout

App Name

AWS

Application	User's First Name	User's Last Name	Reviewed By	Permission
AWS	Venkat	Tummalapalli	None	Read <div>Edit</div>
AWS	Victoria	Alvarez	None	Read and Write <div>Edit</div>
AWS	Cym	Sanguinet	None	Read and Write <div>Edit</div>
AWS	Michael	Baldwin	None	Read and Write <div>Edit</div>
AWS	Linh	Dieu	None	Read <div>Edit</div>
AWS	Daniel	Mark	None	Read and Write <div>Edit</div>
AWS	Martin	Lienhard	None	Read <div>Edit</div>
AWS	Samata	Yariagadda	None	Read and Write <div>Edit</div>
AWS	Sivapriya	Ramachandran	None	Read <div>Edit</div>
AWS	Amit	Suryavanshi	None	Read <div>Edit</div>
AWS	Anshuman	Nene	None	Read and Write <div>Edit</div>

Access Review
Review
Logout

## Permission Update

Application	User's First Name	User's Last Name	Reviewed By	Permission
AWS	Venkat	Tummalapalli	None	Read and Write

Update

Copyright © Coupa Access Review 2015

The auditor can only review the userlist he is related to. For example, if Bob can only access the AWS service, then he can only see the user list of the AWS and cannot see the others.

Access Review    Audit    Logout					
App Name	Application	User's First Name	User's Last Name	Reviewed By	Permission
<div> <div>AWS</div> <div>DNS</div> </div>	DNS	Matthew	Lawrence	kaiyu liu	Read and Write
	DNS	Richard	Anderson	None	Read and Write
	DNS	Chris	Turner	None	Read and Write
	DNS	Stephen	Cussen	None	Read and Write
	DNS	Elizabeth	Simpson	None	Read
	DNS	Jessie	Womble	None	Read and Write
	DNS	Kim	Dickson	None	Read
	DNS	Joseph	Freitag	None	Read and Write
	DNS	Michelle	Martinez	None	Read
	DNS	Brandon	Henderson	None	Read
	DNS	John	Zimmerman	None	Read and Write
	DNS	Harish	Reddy	None	Read
	DNS	Penelope	Foster	None	Read and Write
	DNS	William	McMullen	None	Read

After verification of the system, we have done the experiments to download the access review list as a PDF format file. This is easy for people to download and print.

Access Review Report for AWS				
User ID	First Name	Last Name	Permission	Reviewed By
1	Venkat	Tummalapalli	Read-write	Rundong Liu
2	Victoria	Alvarez	Read	Rundong Liu
3	Cym	Sanguinet	Read-write	None
4	Michael	Baldwin	Read-write	None
5	Linh	Dieu	Read	None
6	Daniel	Mark	Read-write	None
7	Martin	Lienhard	Read	None
8	Samata	Yarlagadda	Read-write	None
9	Sivapriya	Ramachandran	Read	None
10	Amit	Suryavanshi	Read	None
11	Anshuman	Nene	Read-write	None
12	Brannon	Adlesh	Read	None
13	Camille	Falor	Read-write	None
14	Jonathan	Weiss	Read	None
15	Sabaji	Naik	Read-write	None
16	Vikas	Parwar	Read-write	None
17	Roneel	Prasad	Read-write	None
18	David	Chambers	Read	None
19	Jalधि	Valia	Read-write	None
20	Virginia	Synnot	Read-write	None
21	Gilles	Declercq	Read	None

The system has good concurrency control mechanism, works well in the multithreaded environment. If two user try to modify the data at the same time, the data would not crash, and the transactions has good atomicity.

## 7 Conclusions and Future Work

### Conclusions:

The design of the access review system for Coupa Software has been discussed above. With data supplied by Coupa Software, we design schema to parse the

data and get useful information into database. After comparing among the framework and the databases, we choose Python/Django framework and PostgreSQL for our project. We use the agile development and revise the code during the develop process and finally implement all the functions we plan to deliver.

The system can visualize access control list, make access review easy to manage and can help insure security for companies.

#### Future Work:

- Our design is based on relative straightforward scenario given by Coupa Software. But in real life, situation may be much more complicated, we may need to keep profile for every user in the system, provide instant chat function in the system, supply account management and recovery function and so on. We may need to refine our database schema and add more features to make a stable and strong product.
- Even though all the functions and logic are implemented correctly, the interfaces provided in the system now are relatively unadorned. We still need to refine the front-end part to make pages more beautiful, for example we can use more frontend framework just as the Bootstrap to format the pages.
- At this stage, we do not have access to the Coupa Software's database, if we can access to that, we would combine system with their authentication for the real-world access control.
- We can still add more features in the system to make it more stable and strong. For instance, we can add instant chat function to help users to communicate each other. An manager can send message to administrator to require permission to an application. We can add account recovery method to deal with other security issue.

## REFERENCES :

Sandhu, Ravi S., and Pierangela Samarati. "Access control: principle and practice." Communications Magazine, IEEE 32.9 (1994): 40-48.

Sandhu, Ravi S., et al. "Role-based access control models." Computer 2 (1996): 38-47.

Repost from Richard Zorza's Access to Justice Blog: New York Making Big Progress on Multi-Lingual Court Orders

<http://accessreview.ca/2015/10/19/repost-from-richard-zorzass-access-to-justice-blog-new-york-making-big-progress-on-multi-lingual-court-orders/>

<https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>

<http://jerel.co/blog/2015/07/an-introduction-to-emberjs-for-django-developers>

<http://www.postgresql.org/>

[https://en.wikipedia.org/wiki/Evaluation\\_approaches](https://en.wikipedia.org/wiki/Evaluation_approaches)

<http://csrc.nist.gov/publications/nistbul/csl95-12.txt>

<https://www.djangoproject.com>

<http://searchsecurity.techtarget.com/definition/access-control>

<http://searchsecurity.techtarget.com/definition/role-based-access-control-RBAC>

<http://www.digonunes.com/blog/rails-vs-django-vs-play-frameworks/>

<http://codemonkeyism.com/playing-play-framework-java/>

[https://en.wikipedia.org/wiki/Ruby\\_on\\_Rails](https://en.wikipedia.org/wiki/Ruby_on_Rails)

[https://en.wikipedia.org/wiki/Django\\_\(web\\_framework\)](https://en.wikipedia.org/wiki/Django_(web_framework))

<http://searchsecurity.techtarget.com/definition/biometrics>

<https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>

## Appendix:

### Github

Related resource and introduction can be found on the public github  
[https://github.com/KearneyLiu/Access\\_Review](https://github.com/KearneyLiu/Access_Review)

### Contact Information

Kaiyu Liu: [kaiyu.liu@west.cmu.edu](mailto:kaiyu.liu@west.cmu.edu)  
Rundong Liu: [rundong.liu@west.cmu.edu](mailto:rundong.liu@west.cmu.edu)  
Haoran Liu: [haoran.liu@west.cmu.edu](mailto:haoran.liu@west.cmu.edu)