데이타 베이스 프로젝트 – 통합 CCTV 관리 사이트

최종보고서 – 최종 결과 보고서 및 구현 내용

팀명 : 유혜사

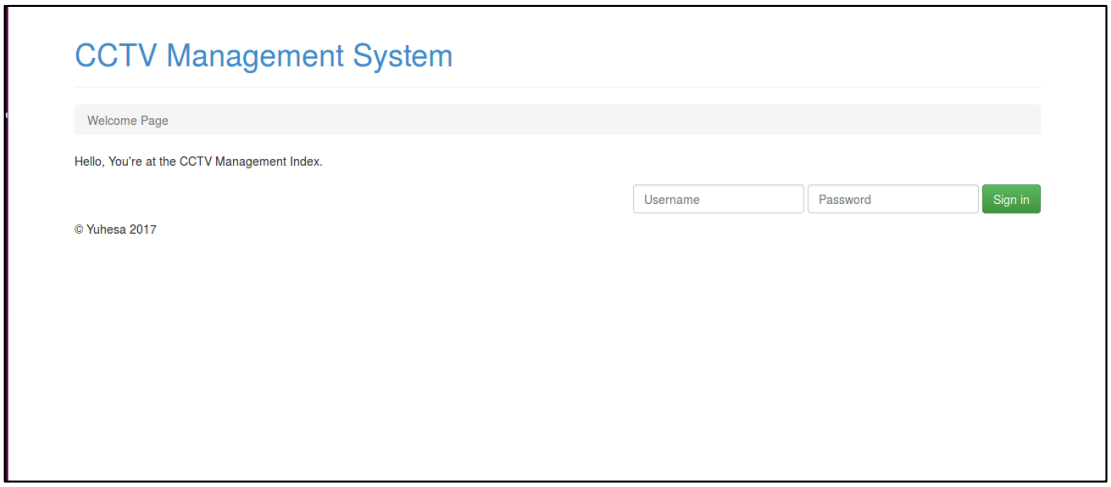
팀원 : WANG GENGYU (2014147565)
멜리사 (2015147516)
방이휘 (2015147545)

1. STEP BY STEP PROJECT PROGRESSION

DATE	CONTENT
05.10.2017	Decision on using Django as the web framework and Python
10-26.10.2017	Study on Django and Python
02.11.2017	Construct ER Diagram and Relational Diagram
05.11.2017	Complete Report 1 (ER-Diagram and Mapping ERD to Relational Schema)
09-15.11.2017	Server environment preferences
16.11.2017	DDL Creation, Database table creation, System design
20.11.2017	Complete Report 2 (System Design Statement)
22.11.2017	UI design and configuration
24-30.11.2017	DML Creation, Database connection, Function configuration
02-05.12.2017	Testing and debugging
05-06.12.2017	Report

2. IMPLEMENTATION DETAILS BY SPECIFIC FUNCTION & RESULT

2.1. At the main page of the CCTV Management System, user can only log in with the super user username and password for the very first time or log in with the username and password that is created by the super user. After logging out, user will return to this page.

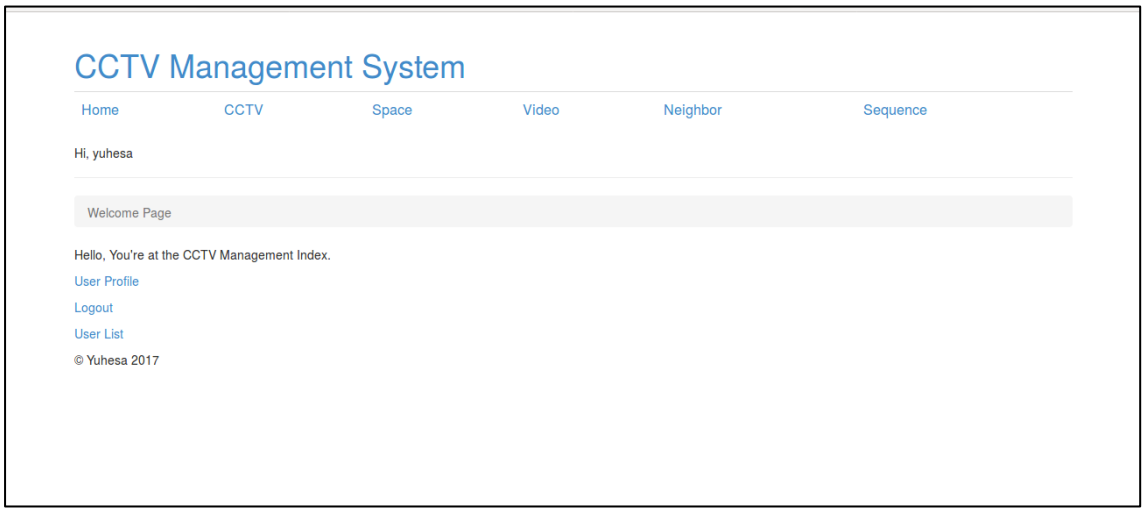


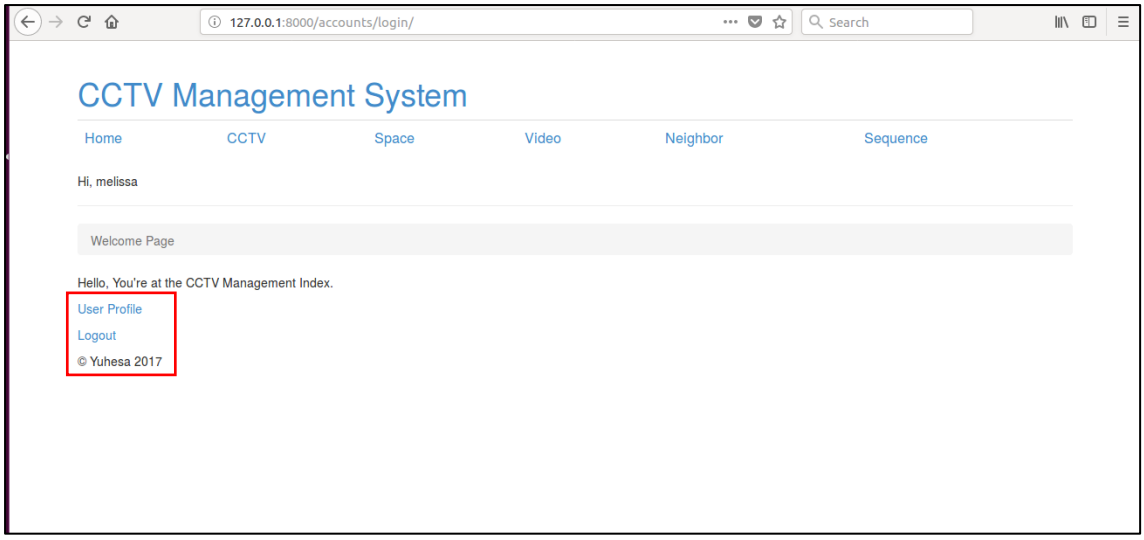
```python
def SQLQuery(SQL, commit = None, parameter = None, lastInsertID = 0):
    db = DBConnect()
    cursor = db.cursor()
    res = None
    if parameter:
        cursor.execute(SQL, parameter)
    else:
        cursor.execute(SQL)
    if commit:
        db.commit()
        if lastInsertID:
            cursor.execute('SELECT LAST_INSERT_ID()')
            res = cursor.fetchall()
        db.close()
        return res
    else:
        res = cursor.fetchall()
        db.close()
        return res

def index(request):
    return render(request, 'index.html')
```

```python
@csrf_protect
def login(request):
    if request.method == 'POST':
        username = request.POST.get('username', '')
        pw = request.POST.get('password', '')

        SQL = 'SELECT id, first_name, last_name, is_superuser, phone_number '
        SQL +='FROM auth_user '
        SQL +='WHERE username=%s AND password=%s'
        res = SQLQuery(SQL,0, [username, pw])

        if res:
            request.session['id'] = res[0][0]
            request.session['username'] = username
            request.session['is_authenticated']= 1
            request.session['is_superuser'] = res[0][3]

        return render(request, 'index.html')
```

```python
def logout(request):
    if request.session['is_authenticated']:
        request.session['is_authenticated'] = 0
        request.session['is_superuser'] = 0
        del request.session['id']
        del request.session['username']
    return render(request, 'index.html')
```

2.2. **HOME:** After logging in, *(yuhesa is the super user)*, the welcome page of the CCTV Management System shows header for *CCTV*, *Space*, *Video*, *Neighbor* and *Sequence*, which are the available in any pages as all pages are extended from *base.html*. Also, user options like *User Profile*, *User List* and *Logout* can be seen in the home page. In this case, <u>only Super User can see the *User List*.</u>

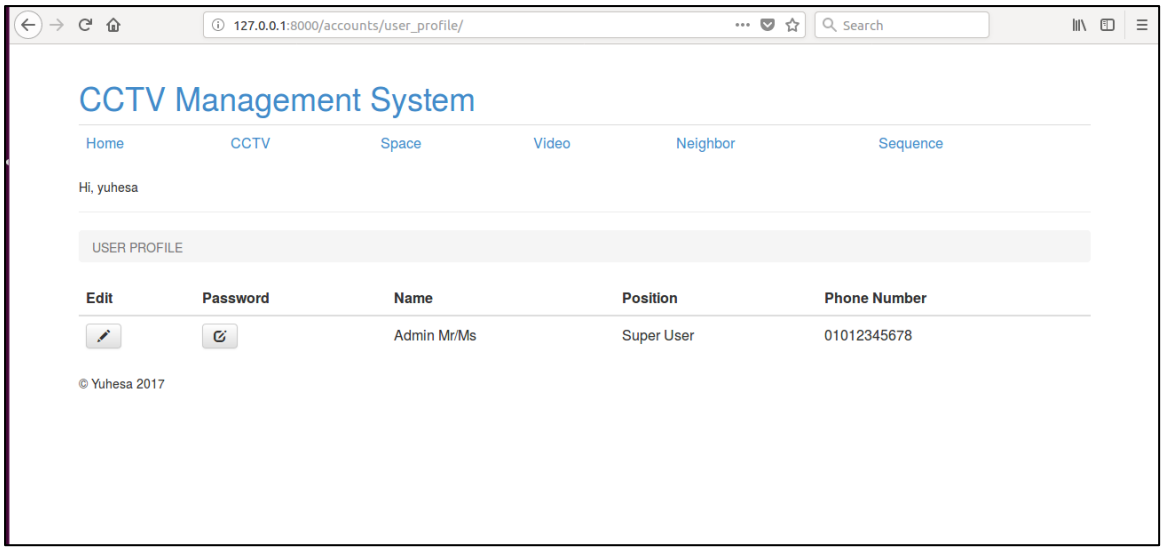





```html
<html>
    <head>
        <title>CCTV Management System</title>
        <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
        <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
        <link href='//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext' rel='stylesheet' type='text/css'>
        <link rel="stylesheet" href="{% static 'css/blog.css' %}">
    </head>
    <body>
        <div class="container">
            <div class="page-header">
                <h1><a href="/">CCTV Management System</a></h1>
                {% if request.session.is_authenticated %}
                    <table class="table">
                    <tbody>
                        <tr>
                          <td><a href="/">Home</a></td>
                          <td><a href="/cctv_list">CCTV</a></td>
                          <td><a href="/space_list">Space</a></td>
                          <td><a href="/video_list">Video</a></td>
                          <td><a href="/neighbor_list">Neighbor</a></td>
                          <td><a href="/sequence_list">Sequence</a></td>
                        </tr>
                    </tbody>
                    </table>
                    <p>Hi, {{ request.session.username }}</p>
                {% endif %}
            </div>
        </div>

        <div class="content container">
            {% block content %}
            {% endblock %}
        </div>

        <div class="container">
            <footer class="footer">
                <p>&copy; Yuhesa 2017</p>
            </footer>
        </div>
    </body>
</html>
```

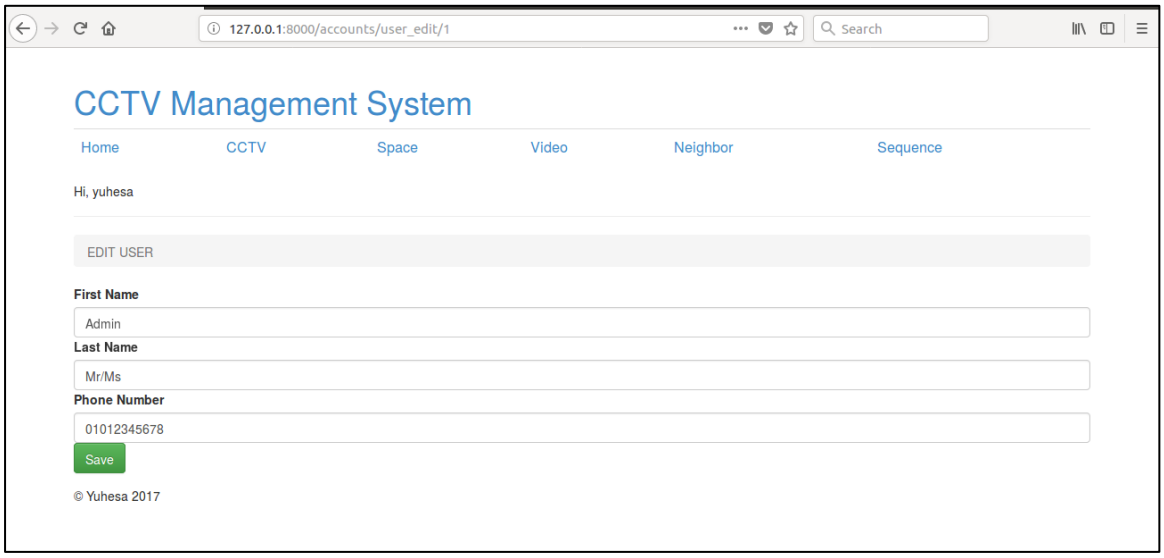
2.3. **USER:**

2.3.1. In the user_profile page, all user can choose to edit his details or change password, and check position, whether is the super user or normal user, and phone number.



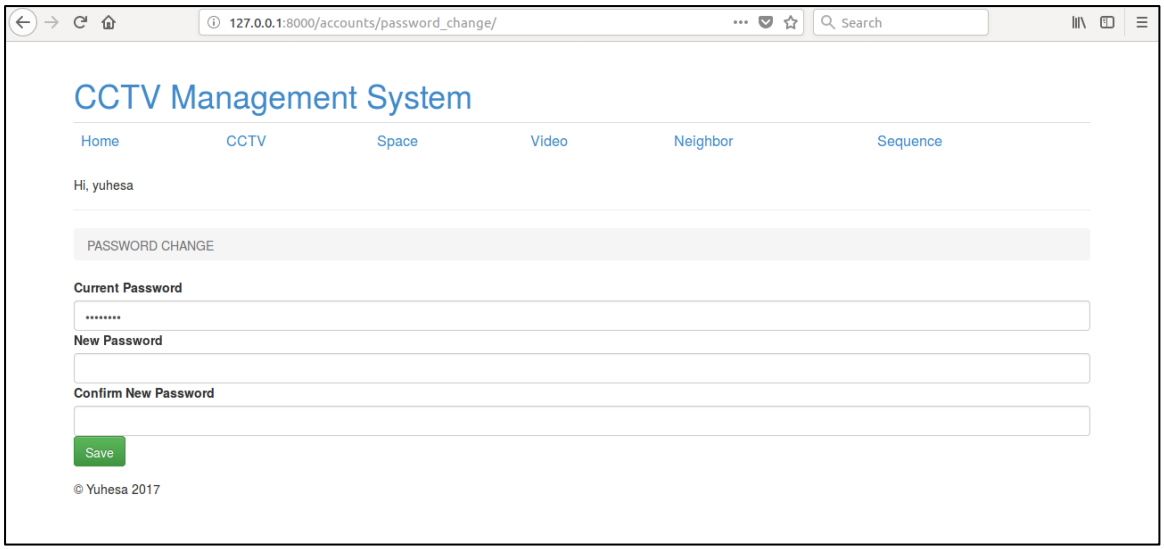

```
def user_profile(request):
    if request.session['is_superuser']:
        position = 'Super User'
    else:
        position = 'Manager'

    SQL = 'SELECT id, first_name, last_name, phone_number '
    SQL += 'FROM auth_user '
    SQL += 'WHERE id =' + str(request.session['id'])
    res = SQLQuery(SQL)
    user = {'id':res[0][0], 'name':res[0][1]+' '+res[0][2], 'phone_number':res[0][3]}
    return render(request, 'management/user_profile.html',{'user':user,'position':position})
```

2.3.2. In user_edit page, every user can edit his details (First Name, Last Name and Phone number) in user_edit page. In this case, user cannot change his username, but can change password in password_change page.



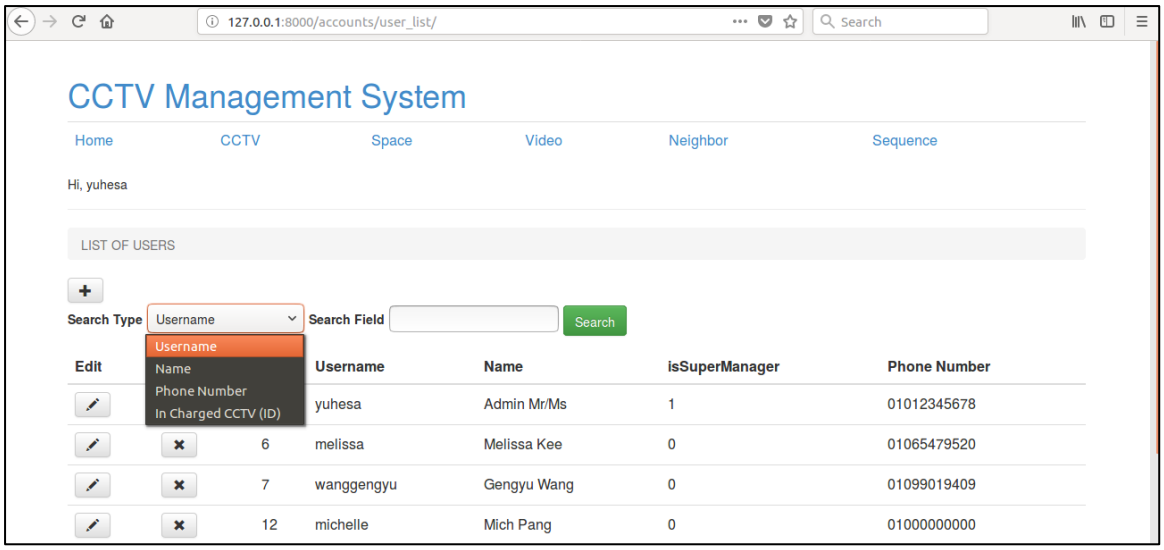
```python
def user_edit(request,pk):
    if request.method == "POST":
        first_name = request.POST.get('first_name', '')
        last_name = request.POST.get('last_name', '')
        phone_number = request.POST.get('phone_number', '')
        SQL = 'UPDATE auth_user '
        SQL += 'SET first_name=%s, last_name=%s, phone_number=%s '
        SQL += 'WHERE id='+str(pk)
        SQLQuery(SQL,1,[first_name,last_name,phone_number])
        if request.session['is_superuser']:
            return redirect('/accounts/user_list') #render(request, 'management/user_list.html')
        else:
            return redirect('/accounts/user_profile')

    else:
        SQL = 'SELECT first_name, last_name, phone_number '
        SQL += 'FROM auth_user '
        SQL += 'WHERE id = '+str(pk)
        result = SQLQuery(SQL)
        user = {'id':pk,'first_name':result[0][0],'last_name':result[0][1],'phone_number':result[0][2]}
        return render(request, 'management/user_edit.html', {'user': user})
```

2.3.3. In password_change page, normal user can change his password that is previously given by super user; if the new password does not match in the second confirmation column, the password would not be changed successfully.




```python
def password_change(request):
    if request.method == "POST":
        current_password = request.POST.get('current_password', '')
        new_password1 = request.POST.get('new_password1', '')
        new_password2 = request.POST.get('new_password2', '')
        if new_password1 != new_password2:
            return render(request, 'management/user_passwordchange.html',{'alert2':1})
        SQL = 'SELECT password '
        SQL += 'FROM auth_user '
        SQL += 'WHERE id = '+str(request.session['id'])
        password = SQLQuery(SQL)[0][0]
        if password == current_password:
            SQL = 'UPDATE auth_user '
            SQL += 'SET password=%s '
            SQL += 'WHERE id='+str(request.session['id'])
            SQLQuery(SQL,1,[new_password1])
            return redirect('/accounts/profile')
        else:
            return render(request, 'management/user_passwordchange.html',{'alert1':1})
    return render(request, 'management/user_passwordchange.html')
```

2.3.4. In user_list page, only super user can create new user, edit the details or delete the user. Also, only super user able to search the normal user based on username, name, phone number or his in charge CCTV ID.

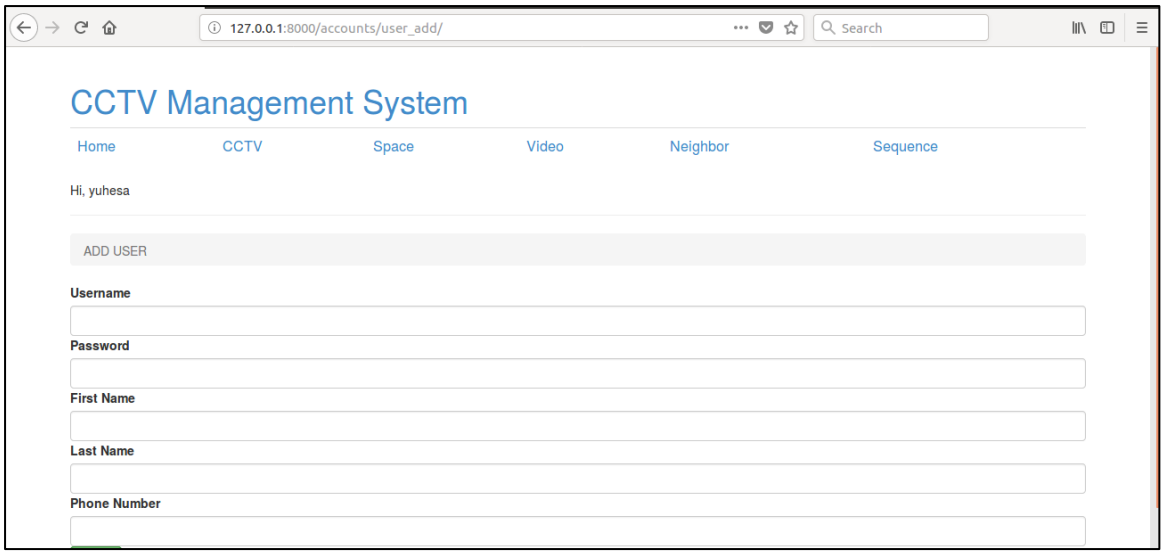



```python
def user_list(request):
    SQL = 'SELECT id, username, first_name, last_name, is_superuser, phone_number '
    SQL +='FROM auth_user'
    res = SQLQuery(SQL)

    users = []
    for re in res:
        users.append(
            {'id':re[0],
            'username':re[1],
            'name':re[2]+' '+re[3],
            'is_superuser':re[4],
            'phone_number':re[5]
            })
    return render(request, 'management/user_list.html',{'users': users})
```

```python
def user_search(request):
    if request.method == "GET":
        SQL = 'SELECT DISTINCT auth_user_id, username, first_name, last_name, is_superuser, phone_number '
        SQL +='FROM (SELECT auth_user.id as auth_user_id, management_cctv.id as management_cctv_id, username, first_name, last_name, is_superuser, phone_number
        SQL +='FROM management_cctv RIGHT JOIN auth_user ON management_cctv.in_charge_user_id = auth_user.id) as u_c WHERE '#'WHERE u.id = c.in_charge_user_id '
        search_type = request.GET.get('search_type','')
        search_field = request.GET.get('search_field','')
        search_field = '%'+search_field+'%'
        if search_type == 'name':
            SQL += '(u_c.first_name LIKE %s OR u_c.last_name LIKE %s)'
            res = SQLQuery(SQL,0,[search_field,search_field])
        elif search_type =='in_charged_cctv':
            SQL += 'u_c.management_cctv_id LIKE \"'+search_field+'\"'
            res = SQLQuery(SQL)
        else:
            SQL += 'u_c.'+search_type+' LIKE \"'+search_field+'\"'
            res = SQLQuery(SQL)
        users = []
        not_exist = 1
        for re in res:
            users.append(
                {'id':re[0],
                'username':re[1],
                'name':re[2]+' '+re[3],
                'is_superuser':re[4],
                'phone_number':re[5]
                })
        return render(request, 'management/user_list.html', {'users': users,'search_type':search_type,'search_field':search_field,'notexist':not_exist})
    return redirect('accounts/user_list')
```

```python
def user_delete(request,pk):
    SQL = 'DELETE FROM auth_user '
    SQL += 'WHERE id = '+str(pk)
    SQLQuery(SQL,1)

    return redirect('user_list')
```

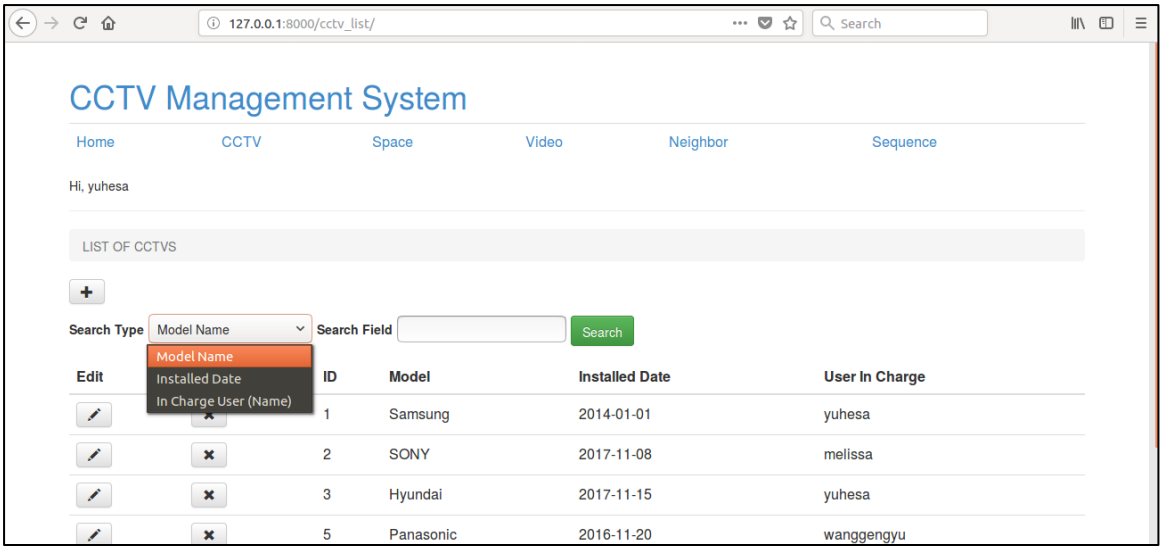
2.3.5. In user_add page, only super user can add new user for the system, with details like username, password, first name, last name and phone number. The added information will then be shown in the user list.

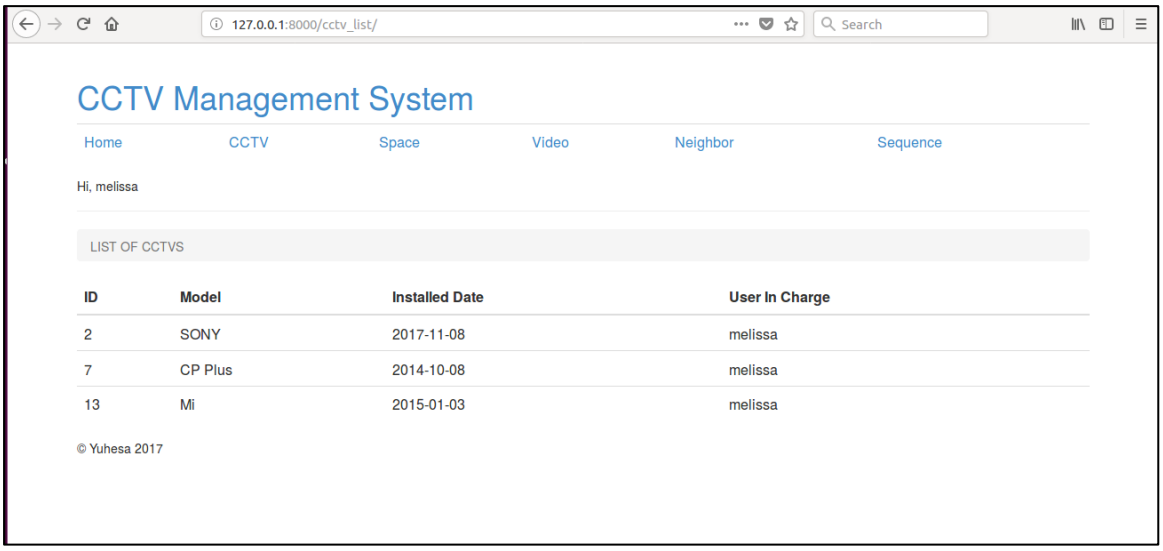



```
def user_add(request):
    if request.method == "POST":
        username = request.POST.get('username', '')
        password =  request.POST.get('password', '')
        first_name = request.POST.get('first_name', '')
        last_name = request.POST.get('last_name', '')
        phone_number = request.POST.get('phone_number', '')
        #e_mail = 'mail@email.com'
        SQL = "INSERT INTO auth_user(username,password,first_name,last_name,phone_number) "
        SQL += "VALUES(%s, %s, %s, %s, %s)"
        SQLQuery(SQL,1,[username,password,first_name,last_name,phone_number])
        return redirect('/accounts/user_list')
    else:
        return render(request, 'management/user_add.html')
```

2.4. CCTV:

2.4.1. In cctv_list page, only super user can add, edit, delete CCTV and search CCTV based on the Model Name, Installed Date or User in Charge's name. While for normal user, he can only see the details of the CCTV that he in charge with details of CCTV ID, Model Name, Installed date and his name. He cannot edit, delete and search the CCTV.



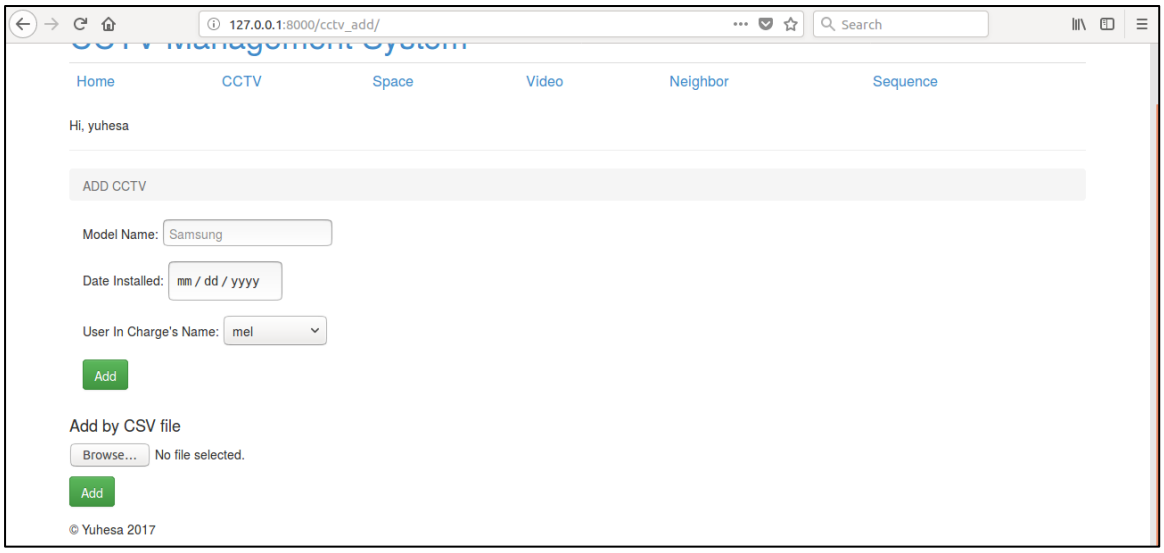


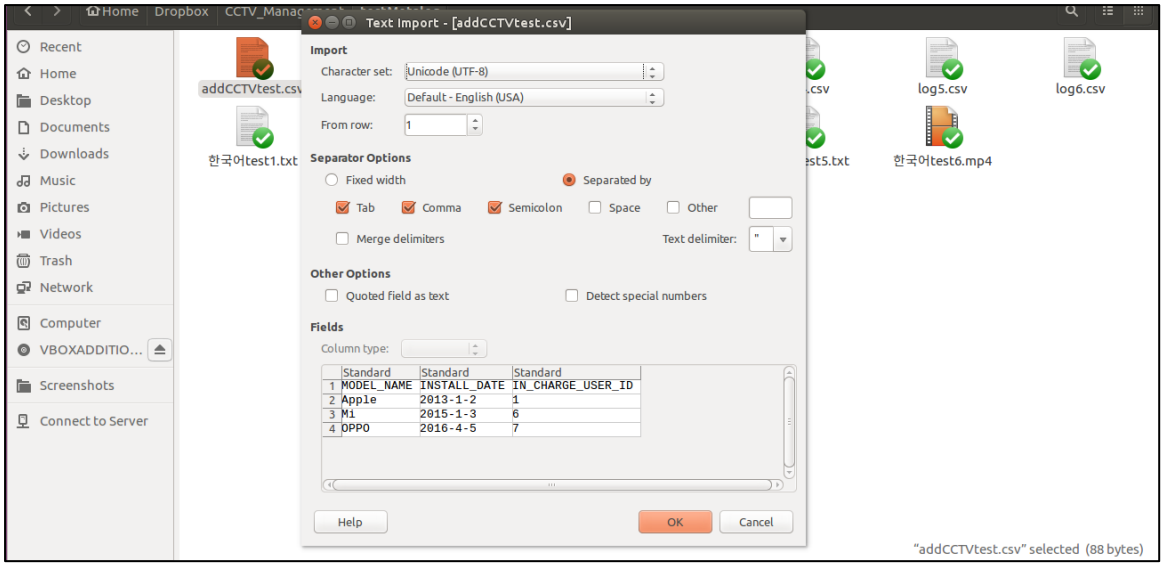

```
def cctv_list(request):
    if request.session['is_superuser']:
        SQL = 'SELECT DISTINCT c.id, c.model_name, c.install_date, u.username '
        SQL +='FROM management_cctv AS c, auth_user AS u '
        SQL +='WHERE u.id = c.in_charge_user_id '
        SQL +='ORDER BY c.id ASC'
    else:
        SQL = 'SELECT DISTINCT c.id, c.model_name, c.install_date, u.username '
        SQL +='FROM management_cctv AS c, auth_user AS u '
        SQL +='WHERE u.id = c.in_charge_user_id AND u.id =' + str(request.session['id']) + ' '
        SQL +='ORDER BY c. id ASC'
    res = SQLQuery(SQL)

    cctvs = []
    for re in res:
        cctvs.append(
            {'id':re[0],
            'model_name':re[1],
            'install_date':str(re[2]),
            'in_charge_user':re[3]
            })

    return render(request, 'management/cctv_list.html', {'cctvs': cctvs})
```

```
def cctv_search(request):
    if request.method == "GET":
        SQL = 'SELECT c.id, c.model_name, c.install_date, u.username '
        SQL +='FROM management_cctv AS c, auth_user AS u '
        SQL +='WHERE u.id = c.in_charge_user_id '
        search_type = request.GET.get('search_type','')
        search_field = request.GET.get('search_field','')
        search_field = '%'+search_field+'%'

        if search_type == 'name':
            SQL += 'AND (u.first_name LIKE %s OR u.last_name LIKE %s)'
            res = SQLQuery(SQL,0,[search_field,search_field])
        else:
            SQL += 'AND c.'+search_type+' LIKE \"'+search_field+'\"'
            res = SQLQuery(SQL)

        cctvs = []
        not_exist = 1
        for re in res:
            not_exist = 0
            cctvs.append(
                {'id':re[0],
                'model_name':re[1],
                'install_date':str(re[2]),
                'in_charge_user':re[3]
                })

        return render(request, 'management/cctv_list.html',{'cctvs': cctvs, 'search_type': search_type, 'search_field': search_field, 'notexist': not_exist})
    return redirect('/cctv_list')
```

```
def cctv_delete(request,pk):
    SQL = 'DELETE FROM management_cctv '
    SQL += 'WHERE id = '+str(pk)
    SQLQuery(SQL,1)
    return redirect('cctv_list')
```

2.4.2. In cctv_add page, <u>only</u> super user can add new CCTV with model name, date installed and the user in charge's name. Or user can add a list of CCTVs through CSV file. Example with 3 CCTV are shown below.



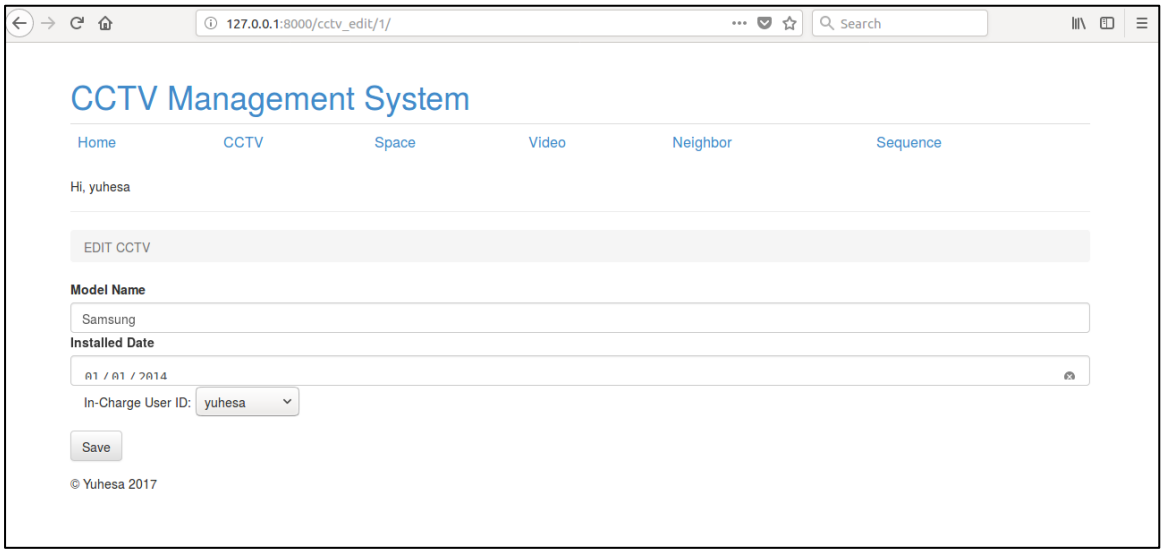


Edit	Delete	ID	Model	Installed Date	User In Charge
		1	Samsung	2014-01-01	yuhesa
		2	SONY	2017-11-08	melissa
		3	Hyundai	2017-11-15	yuhesa
		5	Panasonic	2016-11-20	wanggengyu
		7	CP Plus	2014-10-08	melissa
		8	Netatmo	2001-12-13	wanggengyu
		12	Apple	2013-01-02	yuhesa
		13	Mi	2015-01-03	melissa
		14	OPPO	2016-04-05	wanggengyu

© Yuhesa 2017

```
@csrf_protect
def cctv_add(request):
    if request.method == 'POST':
        if request.POST.get('model_name','')!='':  # POST: process registration of cctv
            model_name = request.POST.get('model_name')
            install_date = request.POST.get('install_date')
            in_charge_user_id = request.POST.get('in_charge_user')

            SQL = 'INSERT INTO management_cctv(MODEL_NAME, INSTALL_DATE, IN_CHARGE_USER_ID) '
            SQL +='VALUES(%s, %s, %s)'
            SQLQuery(SQL, 1, [model_name, install_date, in_charge_user_id])
        else:
            fileCCTV = request.FILES['fileCCTV']
            FileSystemStorage(location=settings.MEDIA_ROOT+'/CCTV_Add/').save(fileCCTV.name, fileCCTV)
            readfile = open(settings.MEDIA_ROOT+'/CCTV_Add/'+fileCCTV.name,'r')
            reader = csv.reader(readfile, delimiter = ',')
            next(reader)
            for row in reader:
                SQL = 'INSERT INTO management_cctv(MODEL_NAME, INSTALL_DATE, IN_CHARGE_USER_ID) '
                SQL += 'VALUES(%s,%s,%s)'
                SQLQuery(SQL,1,[row[0], datetime.strptime(row[1], "%Y-%m-%d").date(), int(row[2])])
            readfile.close()
        return redirect('cctv_list')
    else: # GET: show register form
        SQL = 'SELECT username, id '
        SQL +='FROM auth_user '
        res = SQLQuery(SQL)

        users = []
        for re in res:
            users.append(
            {'username':re[0],
             'id': re[1]
            })

        return render(request, 'management/cctv_add.html', {'users':users})
```

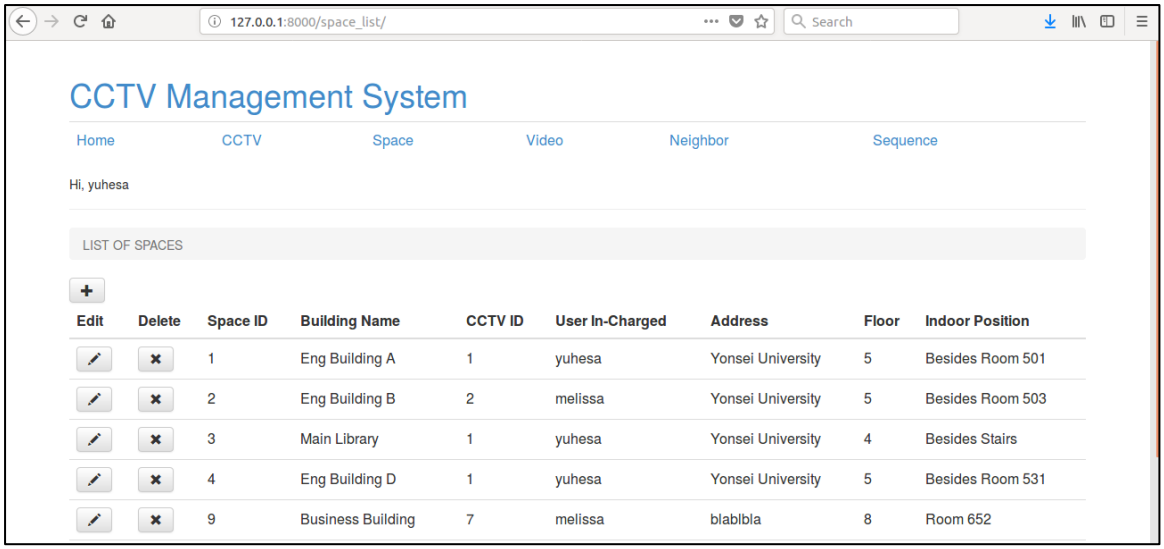
2.4.3. In cctv_edit page, only super user can edit the details of CCTV with model name, date installed and the user in charge's name.



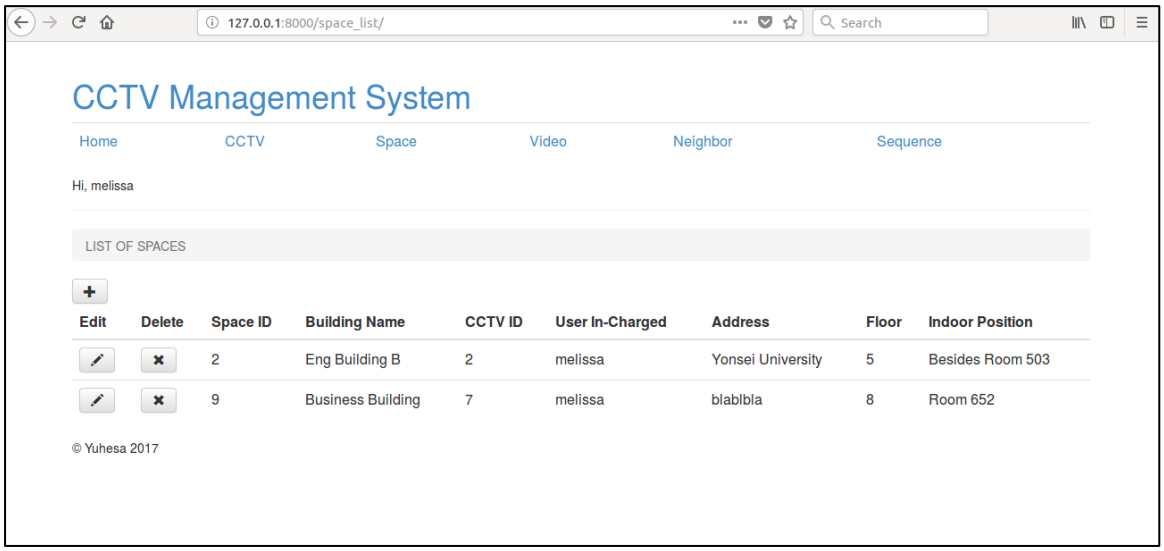

```
def cctv_edit(request, pk):
    if request.method == "POST":
        model_name = request.POST.get('model_name', '')
        install_date = request.POST.get('install_date', '')
        in_charge_user_id = request.POST.get('in_charge_user', '')

        SQL = 'UPDATE management_cctv '
        SQL += 'SET model_name=%s, install_date=%s, in_charge_user_id=%s '
        SQL += 'WHERE id='+str(pk)
        SQLQuery(SQL, 1, [model_name, install_date, in_charge_user_id])

        return redirect('cctv_list')
    else:
        SQL = 'SELECT model_name, install_date, in_charge_user_id '
        SQL += 'FROM management_cctv '
        SQL += 'WHERE id = '+str(pk)
        res = SQLQuery(SQL)
        cctv = {'id':pk, 'model_name':res[0][0], 'install_date':res[0][1], 'in_charge_user_id':res[0][2]}

        SQL = 'SELECT username, id '
        SQL +='FROM auth_user '
        res = SQLQuery(SQL)

        users = []
        for re in res:
            users.append(
            {'username':re[0],
             'id': re[1]
            })

        return render(request, 'management/cctv_edit.html', {'cctv': cctv, 'users': users})
```

2.5. SPACE:

2.5.1. In space_list page, only the super user can see, edit and delete the details of all the added space by all users, like Building Name, CCTV ID, User In Charge, Address, Floor and Indoor Position. While for normal user, he can only see, edit and delete the space details that he in charge according to his in charge CCTV. He cannot see, edit and delete the details of other user's in charge space.



Edit	Delete	Space ID	Building Name	CCTV ID	User In-Charged	Address	Floor	Indoor Position
		1	Eng Building A	1	yuhesa	Yonsei University	5	Besides Room 501
		2	Eng Building B	2	melissa	Yonsei University	5	Besides Room 503
		3	Main Library	1	yuhesa	Yonsei University	4	Besides Stairs
		4	Eng Building D	1	yuhesa	Yonsei University	5	Besides Room 531
		9	Business Building	7	melissa	blablbla	8	Room 652



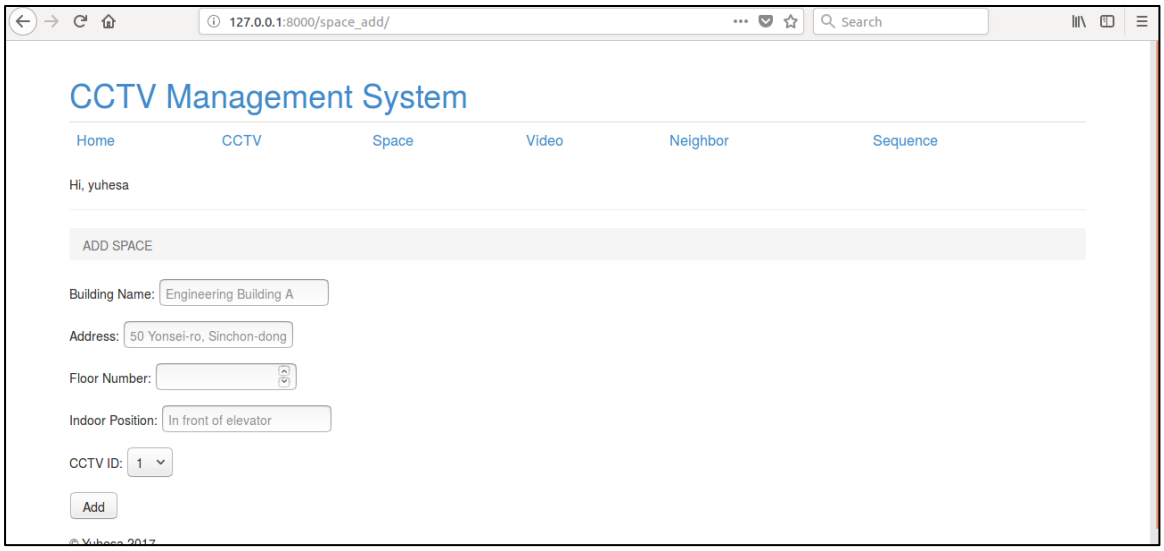
Edit	Delete	Space ID	Building Name	CCTV ID	User In-Charged	Address	Floor	Indoor Position
		2	Eng Building B	2	melissa	Yonsei University	5	Besides Room 503
		9	Business Building	7	melissa	blablbla	8	Room 652

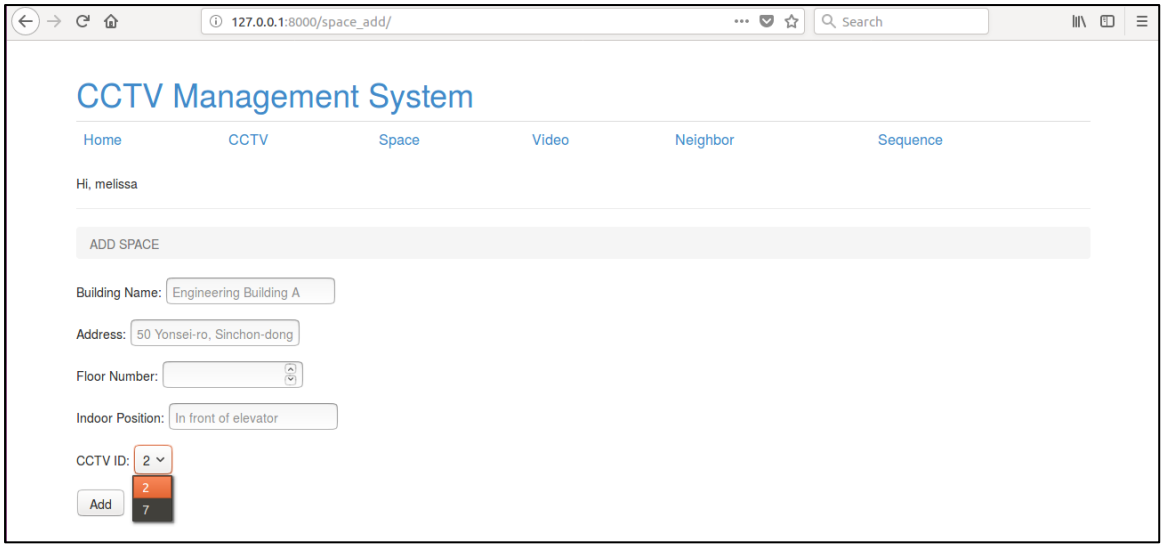
© Yuhesa 2017

```
def space_list(request):
    db = DBConnect()
    cursor = db.cursor()
    if request.session['is_superuser']:
        SQL = 'SELECT s.id, s.cctv_id, s.building_name, s.address, s.floor, s.inroom_position, u.username '
        SQL +='FROM management_space AS s, auth_user AS u, management_cctv AS c '
        SQL +='WHERE u.id = c.in_charge_user_id AND s.cctv_id = c.id '
    else:
        SQL = 'SELECT s.id, s.cctv_id, s.building_name, s.address, s.floor, s.inroom_position, u.username '
        SQL +='FROM management_space AS s, auth_user AS u, management_cctv AS c '
        SQL +='WHERE u.id = c.in_charge_user_id AND s.cctv_id = c.id AND u.id =' + str(request.session['id'])
    cursor.execute(SQL)
    res = cursor.fetchall()
    db.close()

    spaces = []
    for re in res:
        spaces.append(
            {'id':re[0],
            'cctv_id':re[1],
            'building_name':re[2],
            'address':re[3],
            'floor':str(re[4]),
            'inroom_position':re[5],
            'in_charge_user':re[6],
            })

    return render(request, 'management/space_list.html', {'spaces': spaces})
```

```
def space_delete(request,pk):
    SQL = 'DELETE FROM management_space '
    SQL += 'WHERE id = '+str(pk)
    SQLQuery(SQL,1)
    return redirect('space_list')
```

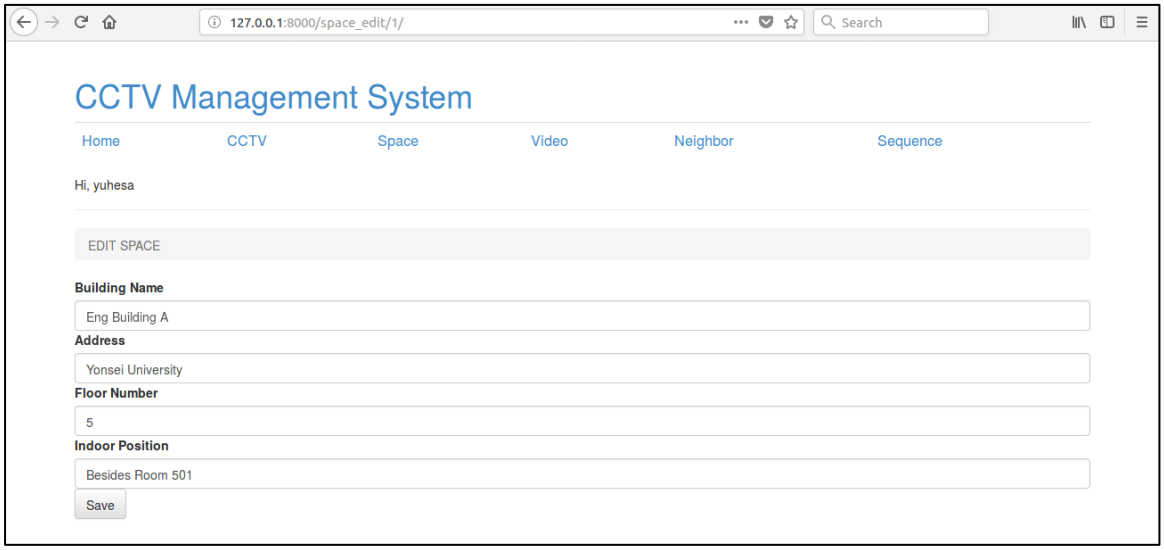
2.5.2. In space_add page, all user can add new space with building name, address, floor number, indoor position, and choose the CCTV ID that he in charge. In this case, only super user can add space for all the user.





```python
@csrf_protect
def space_add(request):
    if request.method == 'POST': # POST: add new space
        id = request.POST.get('id')
        building_name = request.POST.get('building_name')
        address = request.POST.get('address')
        floor = request.POST.get('floor')
        inroom_position = request.POST.get('inroom_position')
        cctv_id = request.POST.get('cctv_id')

        SQL = 'INSERT INTO management_space(ID, BUILDING_NAME, ADDRESS, FLOOR, INROOM_POSITION, CCTV_ID) '
        SQL +='VALUES(%s, %s, %s, %s, %s, %s)'
        SQLQuery(SQL,1, [id, building_name, address, floor, inroom_position, cctv_id])

        return redirect('space_list')
    else: # GET: show register form
        if request.session['is_superuser']:
            SQL = 'SELECT id '
            SQL +='FROM management_cctv '
            SQL +='ORDER BY id ASC'
        else:
            SQL = 'SELECT c.id '
            SQL +='FROM management_cctv AS c, auth_user as u '
            SQL +='WHERE u.id = c.in_charge_user_id AND u.id ='+str(request.session['id'])+' '
            SQL +='ORDER BY c.id ASC '
        res = SQLQuery(SQL)

        cctvs = []
        for re in res:
            cctvs.append(
            {'id':re[0]
            })

        return render(request, 'management/space_add.html', {'cctvs':cctvs})
```

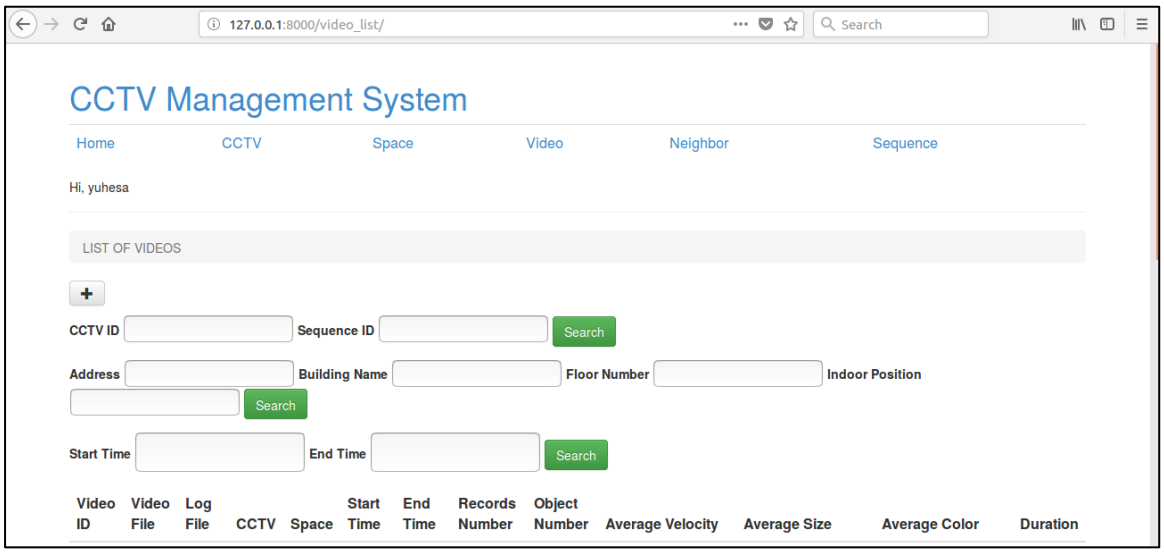
2.5.3. In space_edit page, user can edit the details of space with building name, address, floor number and indoor position.

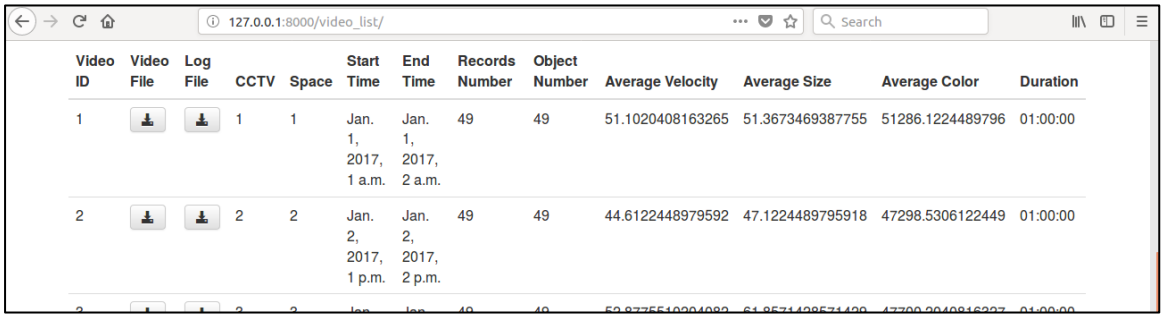



```python
def space_edit(request, pk):
    if request.method == "POST":
        building_name = request.POST.get('building_name')
        address = request.POST.get('address')
        floor = request.POST.get('floor')
        inroom_position = request.POST.get('inroom_position')
        SQL = 'UPDATE management_space '
        SQL += 'SET building_name=%s, address=%s, floor=%s, inroom_position=%s '
        SQL += 'WHERE id='+str(pk)
        SQLQuery(SQL, 1, [building_name, address, floor, inroom_position])
        return redirect('space_list')

    else:
        SQL = 'SELECT building_name, address, floor, inroom_position '
        SQL += 'FROM management_space '
        SQL += 'WHERE id = '+str(pk)
        result = SQLQuery(SQL)
        space = {'id':pk, 'building_name':result[0][0], 'address':result[0][1], 'floor':result[0][2], 'inroom_position':result[0][3]}
        return render(request, 'management/space_edit.html', {'space': space})
```

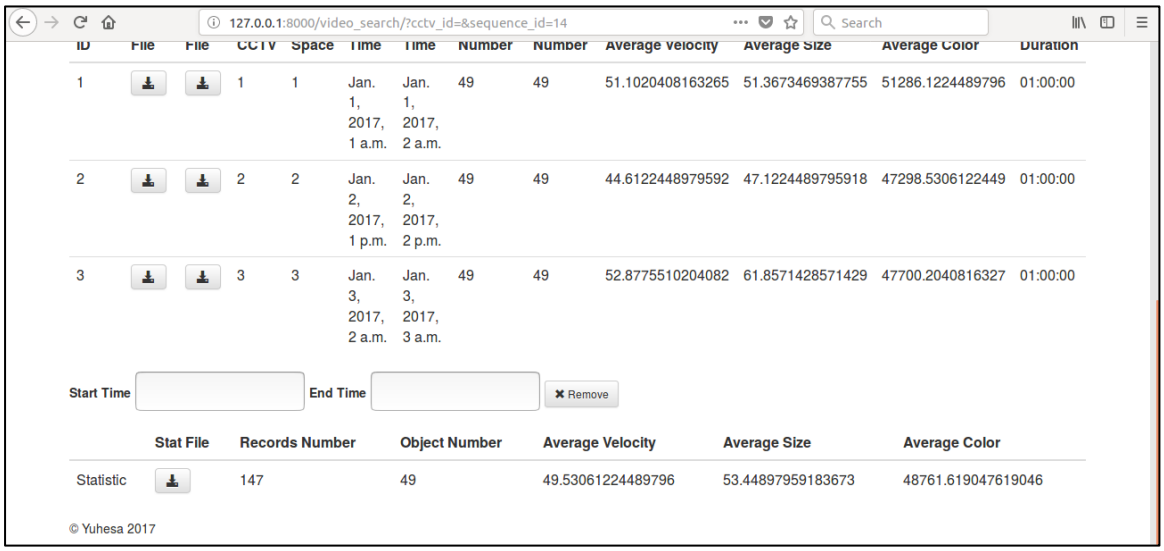
2.6. VIDEO:

2.6.1. In video_list page, all user can download any of the uploaded video file and log file. Also, user can see the metalog statistics of the videos. Users also can search the video file and metalog file based on CCTV ID, Sequence ID, starting time or end time, or through space details like address, building name, floor number and indoor position. Then, user can download the statistics file from the result. The example shown is the result from searching sequence ID of 16. Other than that, user can delete the video with the start time and end time.





Example:

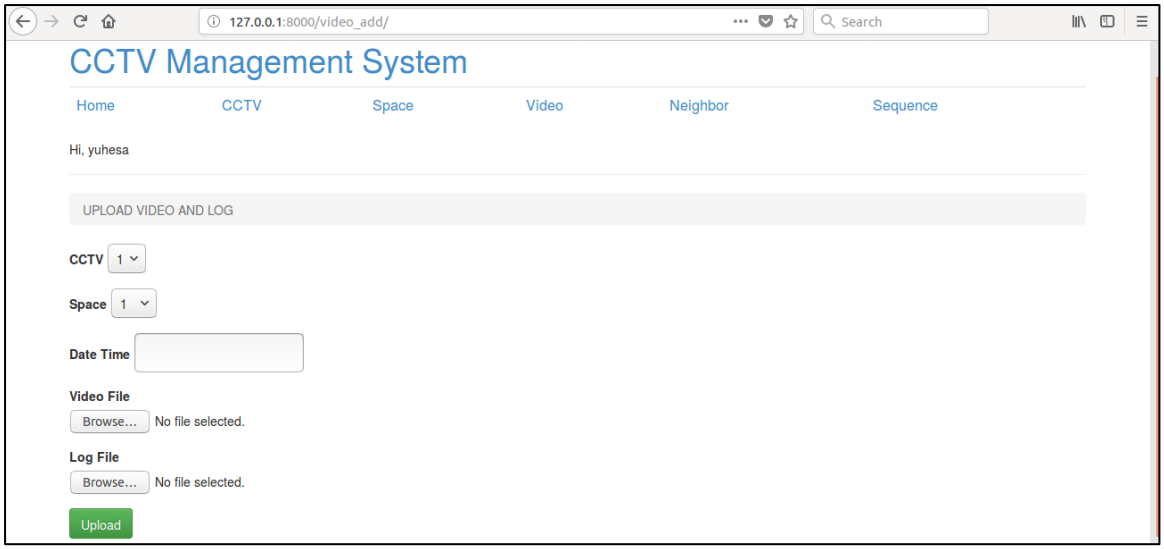


```python
def video_list(request):
    SQL = 'SELECT id, video_file, log_file, cctv_id, space_id, records_number, obj_number, avg_velocity, avg_size, avg_color, start_time, end_time, duration '
    SQL +='FROM management_video'
    res = SQLQuery(SQL)
    videos = []
    for re in res:
        videos.append(
            {'id':re[0],
            'video_file':re[1],
            'log_file':re[2],
            'cctv_id':re[3],
            'space_id':re[4],
            'records_number':re[5], 'obj_number':re[6], 'avg_velocity':re[7], 'avg_size':re[8], 'avg_color':re[9], 'start_time':re[10], 'end_time':re[11],
            'duration':time.strftime('%H:%M:%S', time.gmtime(int(re[12])))
            })
    return render(request, 'management/video_list.html', {'videos': videos})
```

```python
def video_search(request):
    res = []
    if request.method == 'GET' and 'cctv_id' in request.GET:
        SQL = 'SELECT DISTINCT id, video_file, log_file, cctv_id, space_id, records_number, obj_number, avg_velocity, avg_size, avg_color, start_time,
        end_time, duration '
        SQL +='FROM management_video '
        cctv_id = request.GET.get('cctv_id','')
        if cctv_id != '':
            SQL += 'WHERE cctv_id ='+str(cctv_id)
            res = SQLQuery(SQL)
    if request.method == 'GET' and 'sequence_id' in request.GET:
        sequence_id = request.GET.get('sequence_id','')
        if sequence_id != '':
            SQL = 'SELECT DISTINCT v.id, video_file, log_file, cctv_id, space_id, records_number, obj_number, avg_velocity, avg_size, avg_color, start_time,
            end_time, duration '
            SQL += 'FROM management_video as v, management_neighbor as n, management_sequence as s '
            SQL += 'WHERE (n.space_1_id = v.space_id or n.space_2_id = v.space_id) and (s.neighbor_1_id = n.id or s.neighbor_2_id = n.id) '
            SQL += 'AND s.id ='+str(sequence_id)
            res = SQLQuery(SQL)
    if request.method == 'GET' and ('address' in request.GET or 'building_name' in request.GET or 'floor' in request.GET or 'inroom_position' in request.GET):
        address = request.GET.get('address','')
        building_name = request.GET.get('building_name','')
        floor = request.GET.get('floor','')
        inroom_position = request.GET.get('inroom_position','')
        SQL = 'SELECT DISTINCT v.id, video_file, log_file, v.cctv_id, space_id, records_number, obj_number, avg_velocity, avg_size, avg_color, start_time,
        end_time, duration '
        SQL += 'FROM management_video as v, management_space as s '
        SQL += 'WHERE v.space_id = s.id AND '
        addAND = 0
        if address != '':
            SQL += ' s.address LIKE \"%'+address+'%\"'
            addAND = 1
        if building_name != '':
            if addAND == 1: SQL += ' AND '
            SQL += ' s.building_name LIKE \"%'+building_name+'%\"'
            addAND = 1
        if floor != '':
            if addAND == 1: SQL += ' AND '
            SQL += ' s.floor LIKE \"%'+floor+'%\"'
            addAND = 1
        if inroom_position != '':
            if addAND == 1: SQL += ' AND '
            SQL += ' s.inroom_position LIKE \"%'+inroom_position+'%\"'
        res = SQLQuery(SQL)
    if request.method == 'GET' and 'start_time' in request.GET and 'end_time' in request.GET:
        start_time = datetime.strptime(request.GET['start_time'],"%Y-%m-%dT%H:%M")
        end_time = datetime.strptime(request.GET['end_time'],"%Y-%m-%dT%H:%M")
        if start_time != '' and end_time != '':
            SQL = 'SELECT DISTINCT id, video_file, log_file, cctv_id, space_id, records_number, obj_number, avg_velocity, avg_size, avg_color, start_time,
            end_time, duration '
            SQL += 'FROM management_video '
            SQL += 'WHERE start_time >= %s AND end_time <= %s'
            res = SQLQuery(SQL,0,[start_time,end_time])
    videos = []
    videoID = []
    #generate file
    writefile = open(settings.MEDIA_ROOT+'/statistic/statistic.csv', "w")
    #writefile = FileSystemStorage(location=settings.MEDIA_ROOT+'/statistic/').open('statistic.csv', "w")
    writefile.write('VideoID,CCTVID,SpaceID,StartTime,EndTime,RecordsNumber,ObjectNumber,AverageVelocity,AverageSize,AverageColor,Duration\n')
    for re in res:
        writefile.write('%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s\n' % (re[0],re[3],re[4],re[5],re[6],re[7],re[8],re[9],re[10],re[11],re[12]))
        videoID.append(re[0])
        videos.append(
            {'id':re[0],
            'video_file':re[1],
            'log_file':re[2],
            'cctv_id':re[3],
            'space_id':re[4],
            'records_number':re[5], 'obj_number':re[6], 'avg_velocity':re[7], 'avg_size':re[8], 'avg_color':re[9], 'start_time':re[10], 'end_time':re[11],
            'duration':time.strftime('%H:%M:%S', time.gmtime(int(re[12])))
            })
    if len(videoID) != 0: #statistic
        format_strings = ','.join(['%s'] * len(videoID))
        db = DBConnect()
        cursor = db.cursor()
        SQL = 'SELECT count(id), count(DISTINCT object_id), avg(velocity), avg(size), avg(color) '
        SQL += 'FROM management_metalog '
        cursor.execute(SQL+"WHERE video_id IN (%s)" % format_strings,tuple(videoID))
        re = cursor.fetchall()[0]
        db.close()
        stat = {'records_number':re[0], 'obj_number':re[1], 'avg_velocity':re[2], 'avg_size':re[3], 'avg_color':re[4],'statFile':
        '/media/statistic/statistic.csv'}
        writefile.write('%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s\n' % (' ',' ',' ',' ',' ',re[0],re[1],re[2],re[3],re[4],' '))
    writefile.close()

    search = 1
    return render(request, 'management/video_list.html', {'videos': videos, 'search':1, 'stat':stat})
```

```
@csrf_protect
def video_delete(request):
    if request.method == 'POST' and 'start_time' in request.POST and 'end_time' in request.POST:
        start_time = datetime.strptime(request.POST['start_time'],"%Y-%m-%dT%H:%M")
        end_time = datetime.strptime(request.POST['end_time'],"%Y-%m-%dT%H:%M")
        if start_time != '' and end_time != '':
            SQL = 'SELECT DISTINCT id '
            SQL += 'FROM management_video '
            SQL += 'WHERE start_time >= %s AND end_time <= %s'
            res = SQLQuery(SQL,0,[start_time,end_time])

            for re in res:
                video_id = re[0]
                SQL = 'DELETE FROM management_metalog '
                SQL += 'WHERE video_id=%s'
                SQLQuery(SQL,1,[video_id])

                SQL = 'DELETE FROM management_video '
                SQL += 'WHERE id=%s'
                SQLQuery(SQL,1,[video_id])
    return redirect('video_list')
```

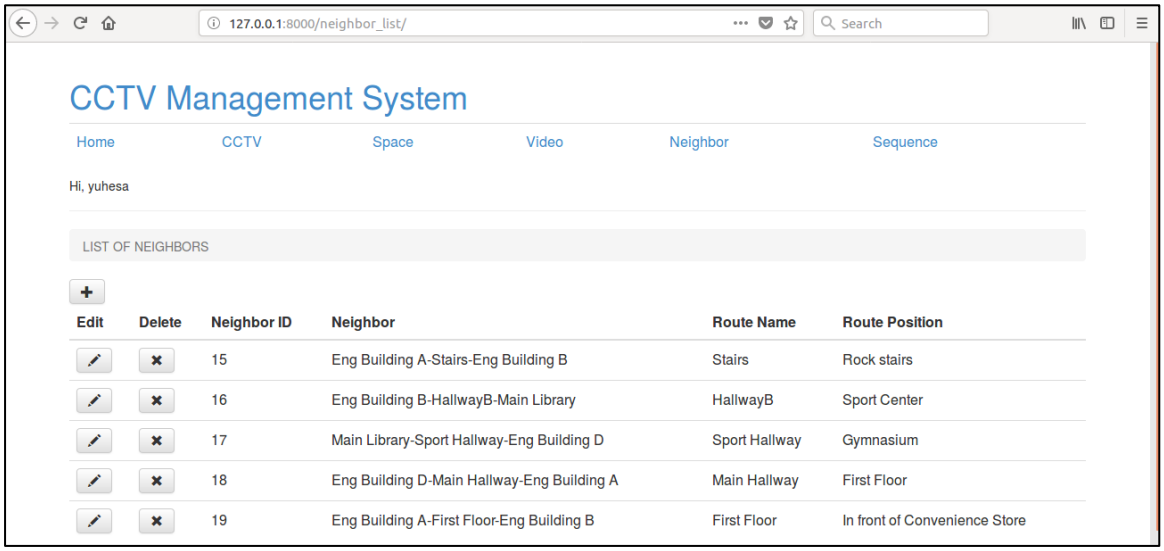
2.6.2. In video_add page, all user can upload new video file and log file with related CCTV ID and space ID, and starting date time of the video that he in charge. In this case, <u>only</u> super user can add video for <u>all</u> the user.

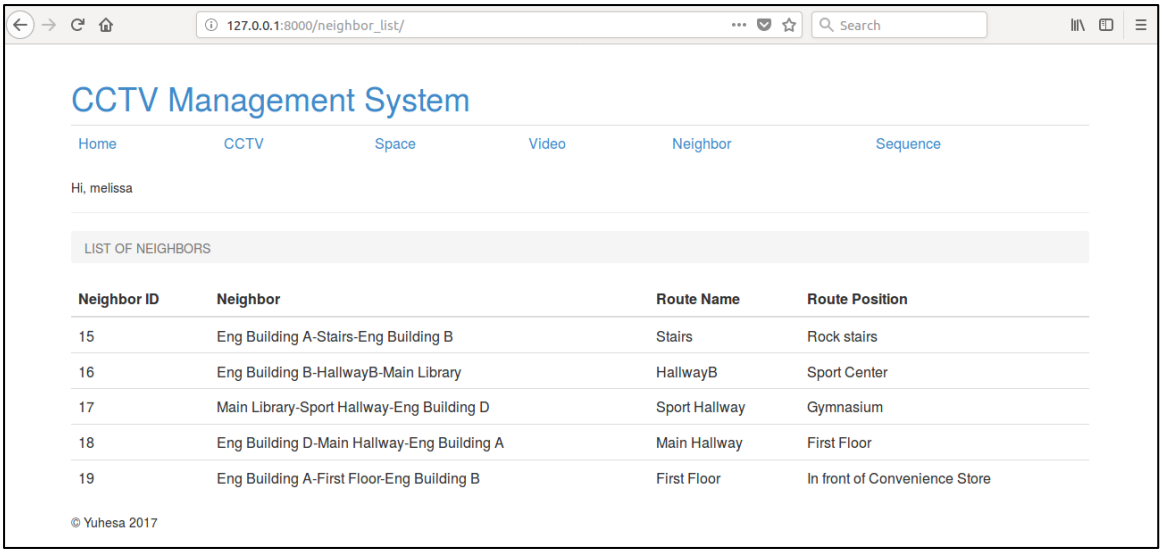


```python
def video_add(request):
    if request.method == 'POST':
        cctv_id = request.POST['cctv_id']
        space_id = request.POST['space_id']
        start_time = datetime.strptime(request.POST['datetime'],"%Y-%m-%dT%H:%M")
        end_time = start_time + timedelta(hours=1)
        duration = (end_time-start_time).total_seconds()
        fileVideo = request.FILES['fileVideo']
        fileLog = request.FILES['fileLog']

        URLvideo = FileSystemStorage(base_url=settings.MEDIA_URL+'video/'+str(space_id)+'/').url(FileSystemStorage(location=settings.MEDIA_ROOT+'/video/'+str(
        space_id)+'/').save(fileVideo.name, fileVideo))
        URLlog = FileSystemStorage(base_url=settings.MEDIA_URL+'log/'+str(space_id)+'/').url(FileSystemStorage(location=settings.MEDIA_ROOT+'/log/'+str(
        space_id)+'/').save(fileLog.name, fileLog))
        SQL = 'INSERT INTO management_video(video_file,log_file,cctv_id,space_id,start_time,end_time,duration) '
        SQL += 'VALUES(%s,%s,%s,%s,%s,%s,%s)'
        videoID = SQLQuery(SQL,1,[URLvideo, URLlog, cctv_id, space_id, start_time, end_time, duration],1)[0][0]
        with FileSystemStorage(location=settings.MEDIA_ROOT+'/log/'+str(space_id)+'/').open(fileLog.name, "rt") as csvfile:
            reader = csv.reader(csvfile, delimiter = ',')
            next(reader)
            for row in reader:
                SQL = 'INSERT INTO management_metalog(video_id,object_id,x_position,y_position,timestamp,size,velocity,color) '
                SQL += 'VALUES(%s,%s,%s,%s,%s,%s,%s,%s)'
                SQLQuery(SQL,1,[videoID, int(row[1]), float(row[2]), float(row[3]), datetime.fromtimestamp(int(row[0])), row[4], row[5],row[6]])
        #fetch meta statistic info from metalog
        SQL = 'SELECT count(id), count(DISTINCT object_id), avg(velocity), avg(size), avg(color) '
        SQL += 'FROM management_metalog '
        SQL += 'WHERE video_id='+str(videoID)
        stat = SQLQuery(SQL)[0]
        #update meta statistic info to video
        SQL = 'UPDATE management_video '
        SQL += 'SET records_number=%s, obj_number=%s, avg_velocity=%s, avg_size=%s, avg_color=%s '
        SQL += 'WHERE id='+str(videoID)
        SQLQuery(SQL,1,[stat[0],stat[1],stat[2],stat[3],stat[4]])
        return redirect('/video_list')
    else:
        if request.session['is_superuser']:
            SQL1 = 'SELECT id '
            SQL1 +='FROM management_cctv '
            SQL1 +='ORDER BY id ASC'
            SQL2 = 'SELECT id '
            SQL2 +='FROM management_space '
            SQL2 +='ORDER BY id ASC'
        else:
            SQL1 = 'SELECT c.id '
            SQL1 +='FROM management_cctv AS c, auth_user as u '
            SQL1 +='WHERE u.id = c.in_charge_user_id AND u.id ='+str(request.session['id'])+' '
            SQL1 +='ORDER BY c.id ASC '
            SQL2 = 'SELECT s.id '
            SQL2 +='FROM management_space AS s, auth_user AS u, management_cctv AS c '
            SQL2 +='WHERE u.id = c.in_charge_user_id AND s.cctv_id = c.id AND u.id =' + str(request.session['id'])+' '
            SQL2 +='ORDER BY s.id ASC '
        res1 = SQLQuery(SQL1)
        res2 = SQLQuery(SQL2)
        cctvs = []
        for re in res1:
            cctvs.append({'id':re[0]})
        spaces = []
        for re in res2:
            spaces.append({'id':re[0]})
        return render(request, 'management/video_add.html',{'cctvs':cctvs, 'spaces':spaces} )
```

2.7. NEIGHBOR:

2.7.1. In neighbor_list page, only the super user can see, edit and delete neighbor space. Neighbor ID is generated by auto numerical system, while neighbor name is generated by the first space, route name and the second space. Also route name and position are also shown in this page. Normal user can only see, but not edit and delete any contents of neighbor.

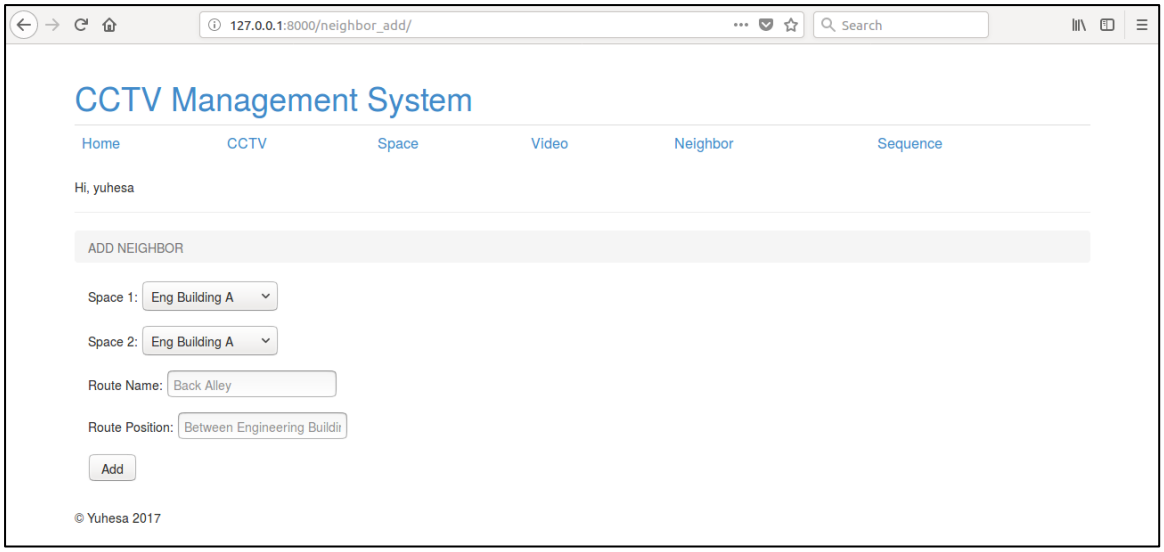





```python
def neighbor_delete(request,pk):
    SQL = 'DELETE FROM management_neighbor '
    SQL += 'WHERE id = '+str(pk)
    SQLQuery(SQL,1)
    return redirect('neighbor_list')
```

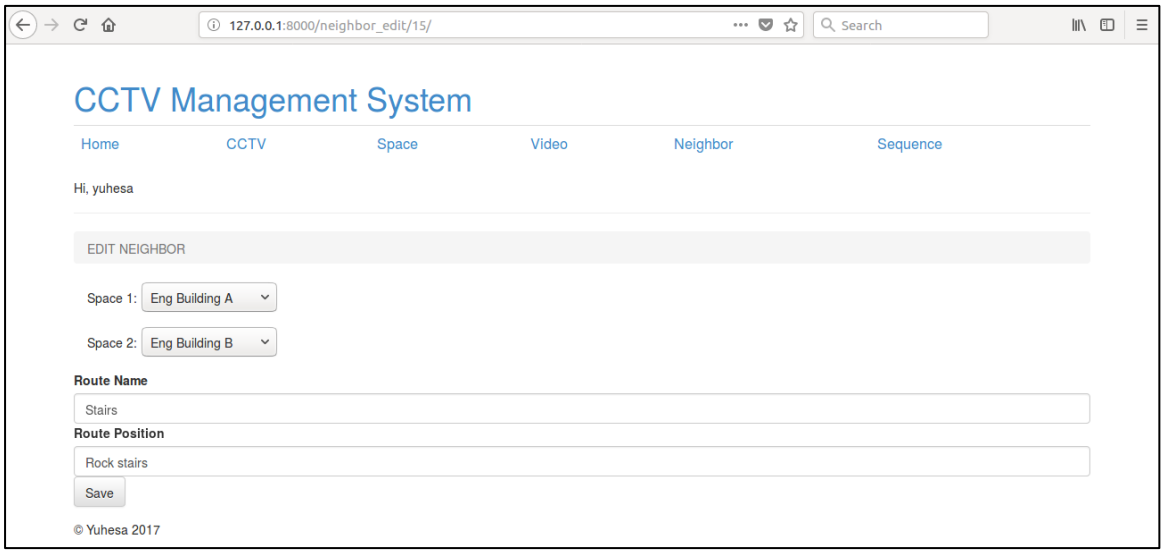
```python
def neighbor_list(request):
    db = DBConnect()
    cursor = db.cursor()
    SQL = 'SELECT n.id, s1.building_name, s2.building_name, route_name, route_position '
    SQL +='FROM management_neighbor AS n '
    SQL +='JOIN management_space AS s1 ON n.space_1_id = s1.id '
    SQL +='JOIN management_space AS s2 ON n.space_2_id = s2.id '
    cursor.execute(SQL)
    res = cursor.fetchall()
    db.close()

    neighbors = []
    for re in res:
        neighbors.append(
            {'id':re[0],
            'space_1':re[1],
            'space_2':re[2],
            'route_name':re[3],
            'route_position':re[4]
            })
    return render(request, 'management/neighbor_list.html', {'neighbors': neighbors})
```

2.7.2. In neighbor_add page, <u>only</u> super user can add new neighbor with the space 1, space 2 with spaces that added previously, route name and route position. Neighbor space is defined as the route between two different spaces, thus, if both space 1 and space 2 are selected with same spaces, neighbor space query will not be added successfully.




```
@csrf_protect
def neighbor_add(request):
    if request.method == 'POST': # POST: add new neighbor
        id = request.POST.get('id')
        space_1_id = request.POST.get('space_1')
        space_2_id = request.POST.get('space_2')
        route_name = request.POST.get('route_name')
        route_position = request.POST.get('route_position')
        if(space_1_id != space_2_id):
            SQL = 'INSERT INTO management_neighbor(ID, SPACE_1_ID, SPACE_2_ID, ROUTE_NAME, ROUTE_POSITION) '
            SQL +='VALUES(%s, %s, %s, %s, %s) '
            SQLQuery(SQL, 1, [id, space_1_id, space_2_id, route_name, route_position])

        return redirect("neighbor_list")
    else: # GET: show register form
        SQL = 'SELECT id, building_name '
        SQL +='FROM management_space '
        SQL +='ORDER BY id ASC'
        res = SQLQuery(SQL)

        spaces = []
        for re in res:
            spaces.append(
            {'id':re[0],
             'building_name':re[1]
            })
        return render(request, 'management/neighbor_add.html', {'spaces':spaces})
```

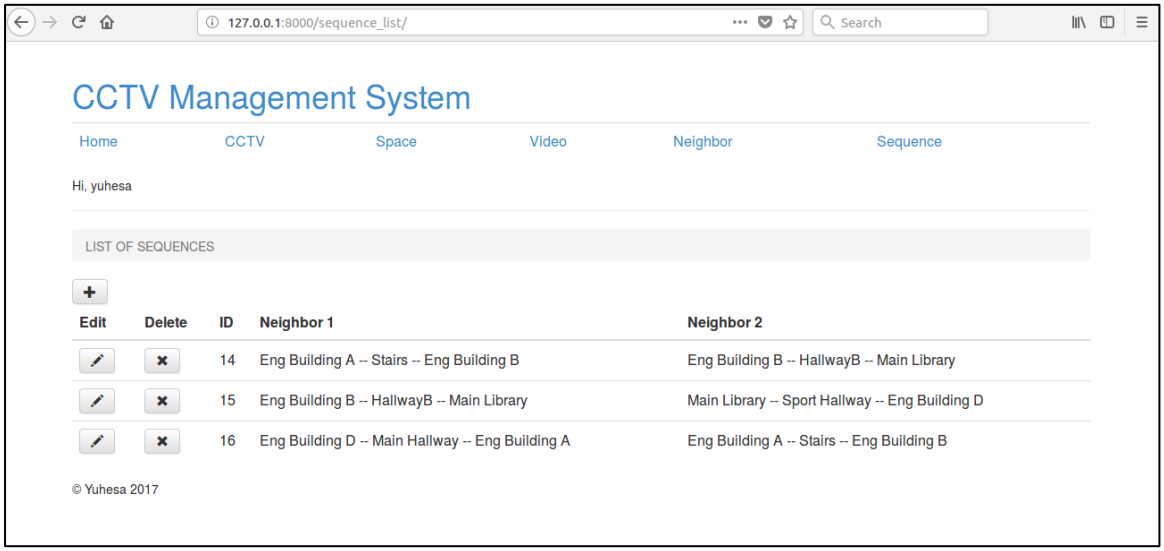
2.7.3. In neighbor_edit page, <u>only</u> super user can edit neighbor with the space 1, space 2 with spaces that added previously, route name and route position.

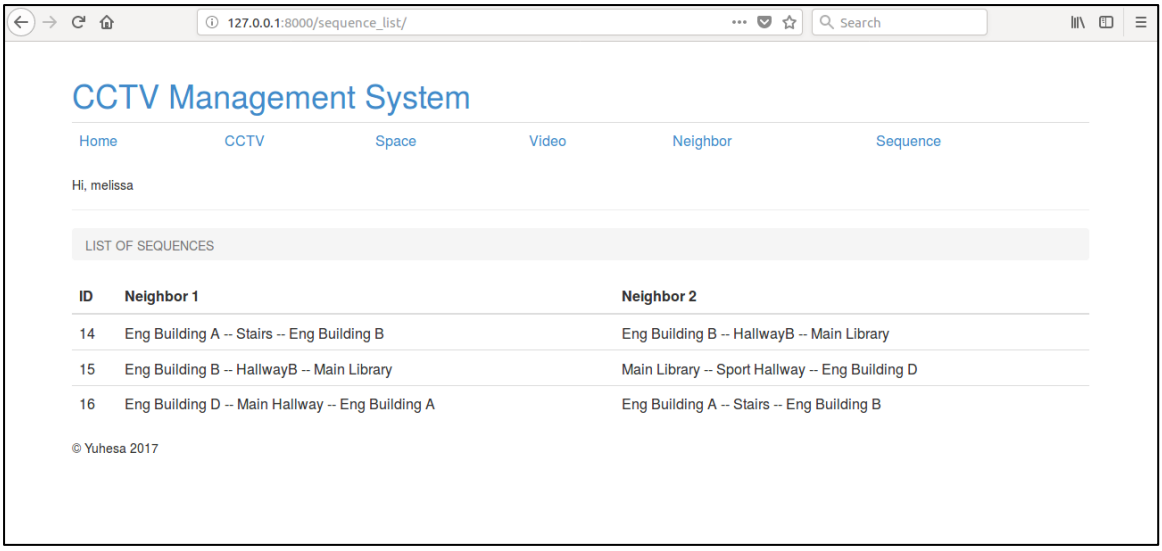


```python
def neighbor_edit(request, pk):
    if request.method == "POST":
        space_1_id = request.POST.get('space_1')
        space_2_id = request.POST.get('space_2')
        route_name = request.POST.get('route_name')
        route_position = request.POST.get('route_position')
        if(space_1_id != space_2_id):
            SQL = 'UPDATE management_neighbor '
            SQL += 'SET space_1_id=%s, space_2_id=%s, route_name=%s, route_position=%s '
            SQL += 'WHERE id='+str(pk)
            SQLQuery(SQL, 1, [space_1_id, space_2_id, route_name, route_position])
        return redirect('neighbor_list')

    else:
        SQL = 'SELECT space_1_id, space_2_id, route_name, route_position '
        SQL += 'FROM management_neighbor '
        SQL += 'WHERE id = '+str(pk)
        result = SQLQuery(SQL)
        neighbor = {'id':pk, 'space_1_id':result[0][0], 'space_2_id':result[0][1], 'route_name':result[0][2], 'route_position':result[0][3]}

        SQL = 'SELECT id, building_name '
        SQL +='FROM management_space '
        res = SQLQuery(SQL)

        spaces = []
        for re in res:
            spaces.append(
            {'id':re[0],
             'building_name': re[1]
            })

        return render(request, 'management/neighbor_edit.html', {'neighbor': neighbor, 'spaces': spaces})
```

2.8. SEQUENCE:

2.8.1. In sequence_list page, only the super user can see, edit and delete sequence. Sequence ID is generated by auto numerical system; neighbor 1 and neighbor 2 are generated according to the neighbor name in neighbor_list page. Normal user can only see, but not edit and delete any contents of sequence.

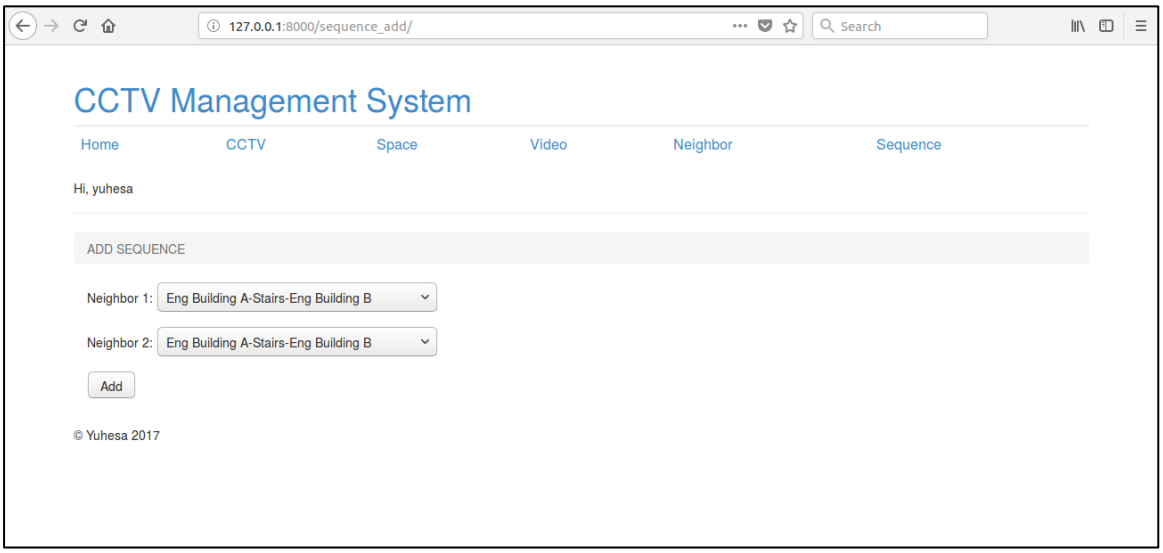


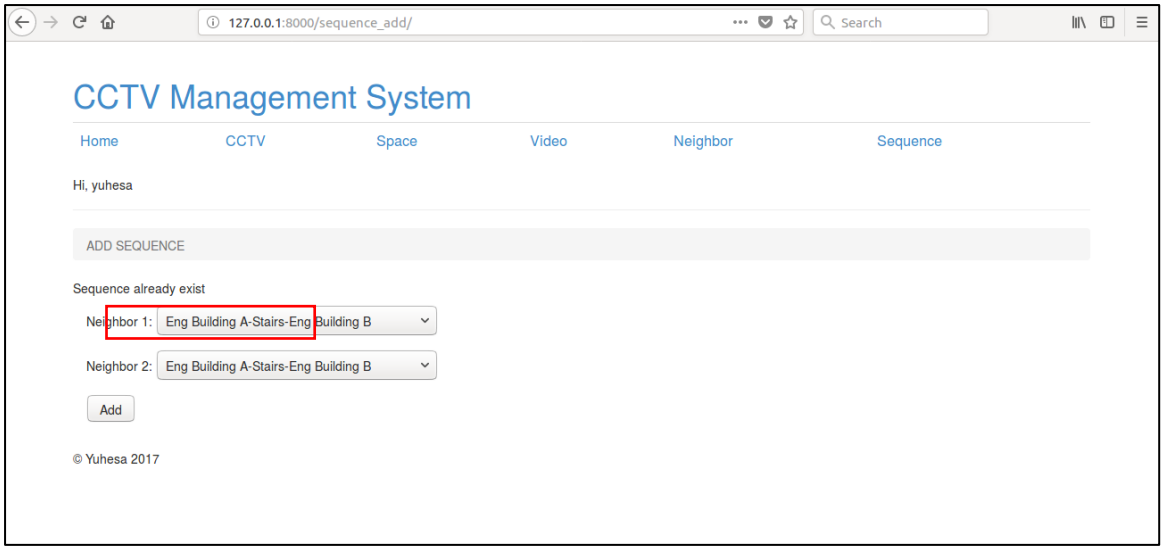



```
def sequence_list(request):
    SQL = 'SELECT s.id, n1.route_name, n2.route_name, n1s1.building_name, n1s2.building_name, n2s1.building_name, n2s2.building_name '
    SQL +='FROM management_sequence AS s '
    SQL +='JOIN management_neighbor AS n1 ON s.neighbor_1_id = n1.id '
    SQL +='JOIN management_neighbor AS n2 ON s.neighbor_2_id = n2.id '
    SQL +='JOIN management_space AS n1s1 ON n1.space_1_id = n1s1.id '
    SQL +='JOIN management_space AS n1s2 ON n1.space_2_id = n1s2.id '
    SQL +='JOIN management_space AS n2s1 ON n2.space_1_id = n2s1.id '
    SQL +='JOIN management_space AS n2s2 ON n2.space_2_id = n2s2.id '
    SQL +='ORDER BY s.id '
    res = SQLQuery(SQL)

    sequences = []
    for re in res:
        sequences.append(
            {'id':re[0],
            'neighbor_1':re[1],
            'neighbor_2':re[2],
            'neighbor1_space1':re[3],
            'neighbor1_space2':re[4],
            'neighbor2_space1':re[5],
            'neighbor2_space2':re[6],
            })
    return render(request, 'management/sequence_list.html', {'sequences': sequences})
```

```
def sequence_delete(request,pk):
    SQL = 'DELETE FROM management_sequence '
    SQL += 'WHERE id = '+str(pk)
    SQLQuery(SQL,1)
    return redirect('sequence_list')
```

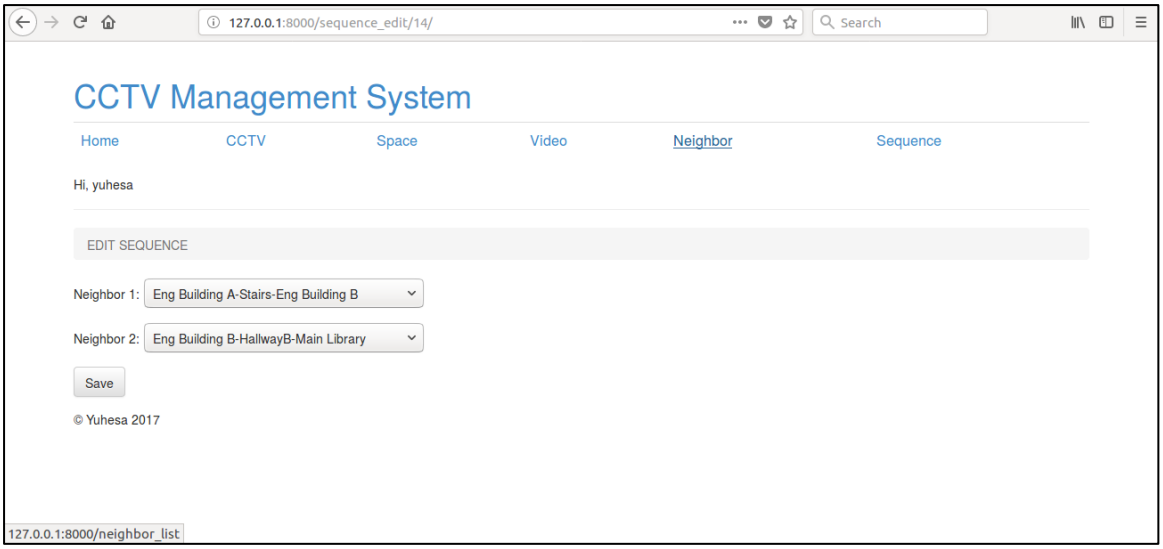
2.8.2. In sequence_add page, <u>only</u> super user can add new sequence with the neighbor 1 and neighbor 2 with neighbors that added previously. Sequence is defined by connectable neighbor space, thus, if the space 1 in neighbor 2 is different with the space 2 in neighbor 1, sequence will not be added successfully. Also, if same sequence is inserted, error message *'Sequence already exist'* will be shown.






```
@csrf_protect
def sequence_add(request):
    if request.method == 'POST': # POST: add new sequence
        id = request.POST.get('id')
        neighbor_1_id = request.POST.get('neighbor_1')
        neighbor_2_id = request.POST.get('neighbor_2')

        SQL = 'SELECT s.building_name '
        SQL +='FROM management_neighbor AS n, management_space AS s '
        SQL +='WHERE n.id='+str(neighbor_1_id)+' AND n.space_2_id = s.id '
        s2_building_name = SQLQuery(SQL)[0][0]

        SQL = 'SELECT s.building_name '
        SQL +='FROM management_neighbor AS n, management_space AS s '
        SQL +='WHERE n.id='+str(neighbor_2_id)+' AND n.space_1_id = s.id '
        s1_building_name = SQLQuery(SQL)[0][0]

        if ((neighbor_1_id != neighbor_2_id) & (s2_building_name == s1_building_name)):
            try:
                SQL = 'INSERT INTO management_sequence(ID, NEIGHBOR_1_ID, NEIGHBOR_2_ID) '
                SQL +='VALUES(%s, %s, %s) '
                SQLQuery(SQL, 1, [id, neighbor_1_id, neighbor_2_id])
            except MySQLdb.IntegrityError as e:
                SQL = 'SELECT n.id, s1.building_name, s2.building_name, route_name '
                SQL +='FROM management_neighbor AS n '
                SQL +='JOIN management_space AS s1 ON n.space_1_id = s1.id '
                SQL +='JOIN management_space AS s2 ON n.space_2_id = s2.id '
                res = SQLQuery(SQL)

                neighbors = []
                for re in res:
                    neighbors.append(
                    {'id':re[0],
                     's1_building_name':re[1],
                     's2_building_name':re[2],
                     'route_name':re[3]
                    })
                return render(request, 'management/sequence_add.html', {'neighbors':neighbors, 'error':str(e.__cause__)})

        return redirect("sequence_list")

    else: # GET: show register form
        SQL = 'SELECT n.id, s1.building_name, s2.building_name, route_name '
        SQL +='FROM management_neighbor AS n '
        SQL +='JOIN management_space AS s1 ON n.space_1_id = s1.id '
        SQL +='JOIN management_space AS s2 ON n.space_2_id = s2.id '
        res = SQLQuery(SQL)

        neighbors = []
        for re in res:
            neighbors.append(
            {'id':re[0],
             's1_building_name':re[1],
             's2_building_name':re[2],
             'route_name':re[3]
            })
        return render(request, 'management/sequence_add.html', {'neighbors':neighbors})
```

2.8.3. In sequence_edit page, <u>only</u> super user can edit sequence with the neighbor 1 and neighbor 2 with neighbors that added previously.




```
def sequence_edit(request, pk):
    if request.method == "POST":
        neighbor_1_id = request.POST.get('neighbor_1')
        neighbor_2_id = request.POST.get('neighbor_2')

        SQL = 'SELECT s.building_name '
        SQL +='FROM management_neighbor AS n, management_space AS s '
        SQL +='WHERE n.id='+str(neighbor_1_id)+' AND n.space_2_id = s.id '
        s2_building_name = SQLQuery(SQL)[0][0]

        SQL = 'SELECT s.building_name '
        SQL +='FROM management_neighbor AS n, management_space AS s '
        SQL +='WHERE n.id='+str(neighbor_2_id)+' AND n.space_1_id = s.id '
        s1_building_name = SQLQuery(SQL)[0][0]

        if ((neighbor_1_id != neighbor_2_id) & (s2_building_name == s1_building_name)):
            try:
                SQL = 'UPDATE management_sequence '
                SQL += 'SET neighbor_1_id=%s, neighbor_2_id=%s '
                SQL += 'WHERE id='+str(pk)
                SQLQuery(SQL, 1, [neighbor_1_id, neighbor_2_id])
            except MySQLdb.IntegrityError as e:
                SQL = 'SELECT neighbor_1_id, neighbor_2_id '
                SQL += 'FROM management_sequence '
                SQL += 'WHERE id = '+str(pk)
                result = SQLQuery(SQL)
                sequence = {'id':pk, 'neighbor_1_id':result[0][0], 'neighbor_2_id':result[0][1]}

                SQL = 'SELECT n.id, s1.building_name, s2.building_name, route_name '
                SQL +='FROM management_neighbor AS n '
                SQL +='JOIN management_space AS s1 ON n.space_1_id = s1.id '
                SQL +='JOIN management_space AS s2 ON n.space_2_id = s2.id '
                res = SQLQuery(SQL)

                neighbors = []
                for re in res:
                    neighbors.append(
                    {'id':re[0],
                     's1_building_name':re[1],
                     's2_building_name':re[2],
                     'route_name':re[3]
                    })
                return render(request, 'management/sequence_edit.html', {'sequence': sequence, 'neighbors':neighbors, 'error':str(e.__cause__)})
        return redirect('sequence_list')

    else:
        SQL = 'SELECT neighbor_1_id, neighbor_2_id '
        SQL += 'FROM management_sequence '
        SQL += 'WHERE id = '+str(pk)
        result = SQLQuery(SQL)
        sequence = {'id':pk, 'neighbor_1_id':result[0][0], 'neighbor_2_id':result[0][1]}

        SQL = 'SELECT n.id, s1.building_name, s2.building_name, route_name '
        SQL +='FROM management_neighbor AS n '
        SQL +='JOIN management_space AS s1 ON n.space_1_id = s1.id '
        SQL +='JOIN management_space AS s2 ON n.space_2_id = s2.id '
        res = SQLQuery(SQL)

        neighbors = []
        for re in res:
            neighbors.append(
            {'id':re[0],
             's1_building_name':re[1],
             's2_building_name':re[2],
             'route_name':re[3]
            })
```

3. CHANGES AND ADDITION

3.1. Some modification are made to the MySQL script. Changes include:

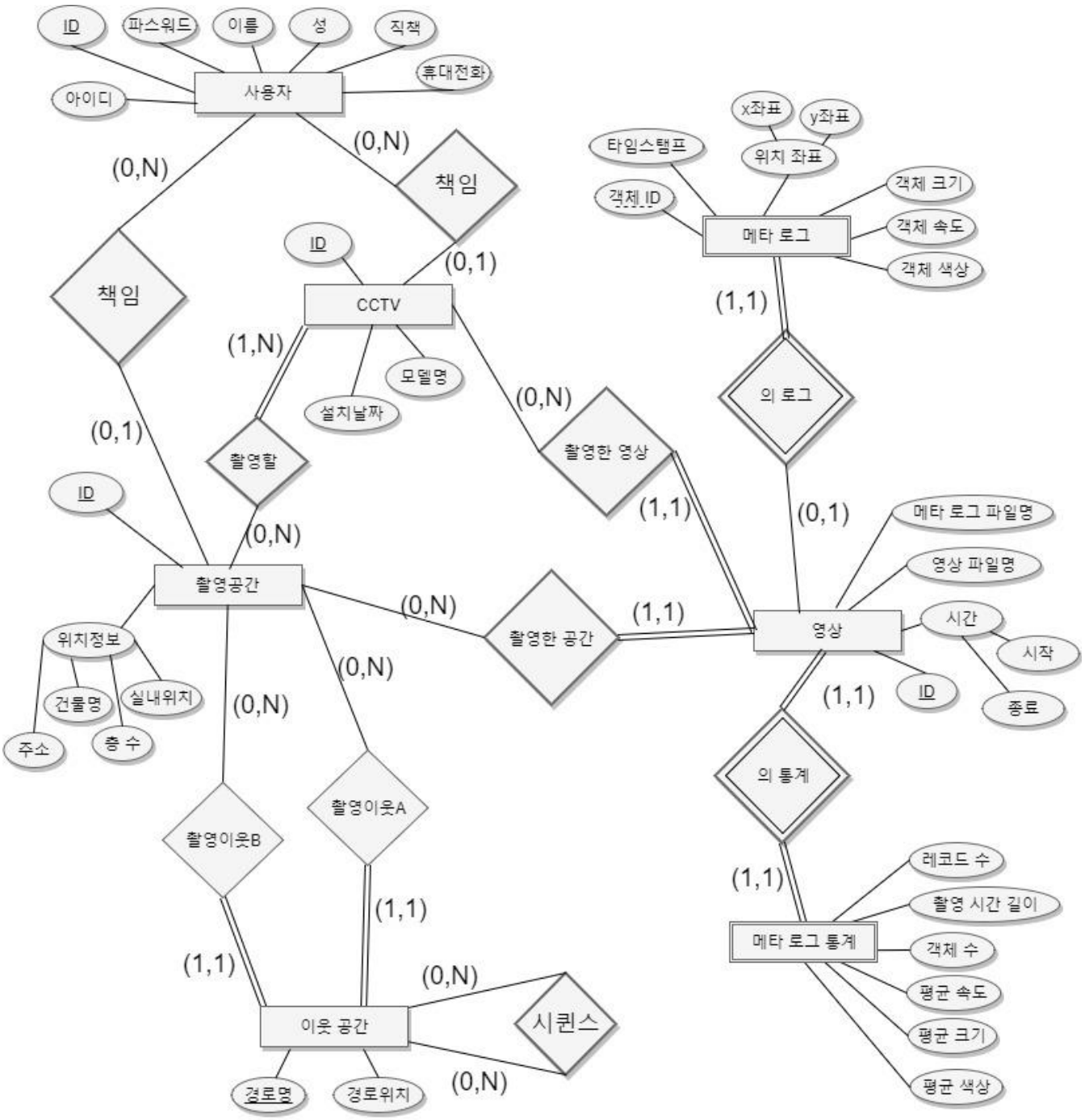
3.1.1. Lengthen the maximum number of character for some attributes to avoid the value is truncated to fit.

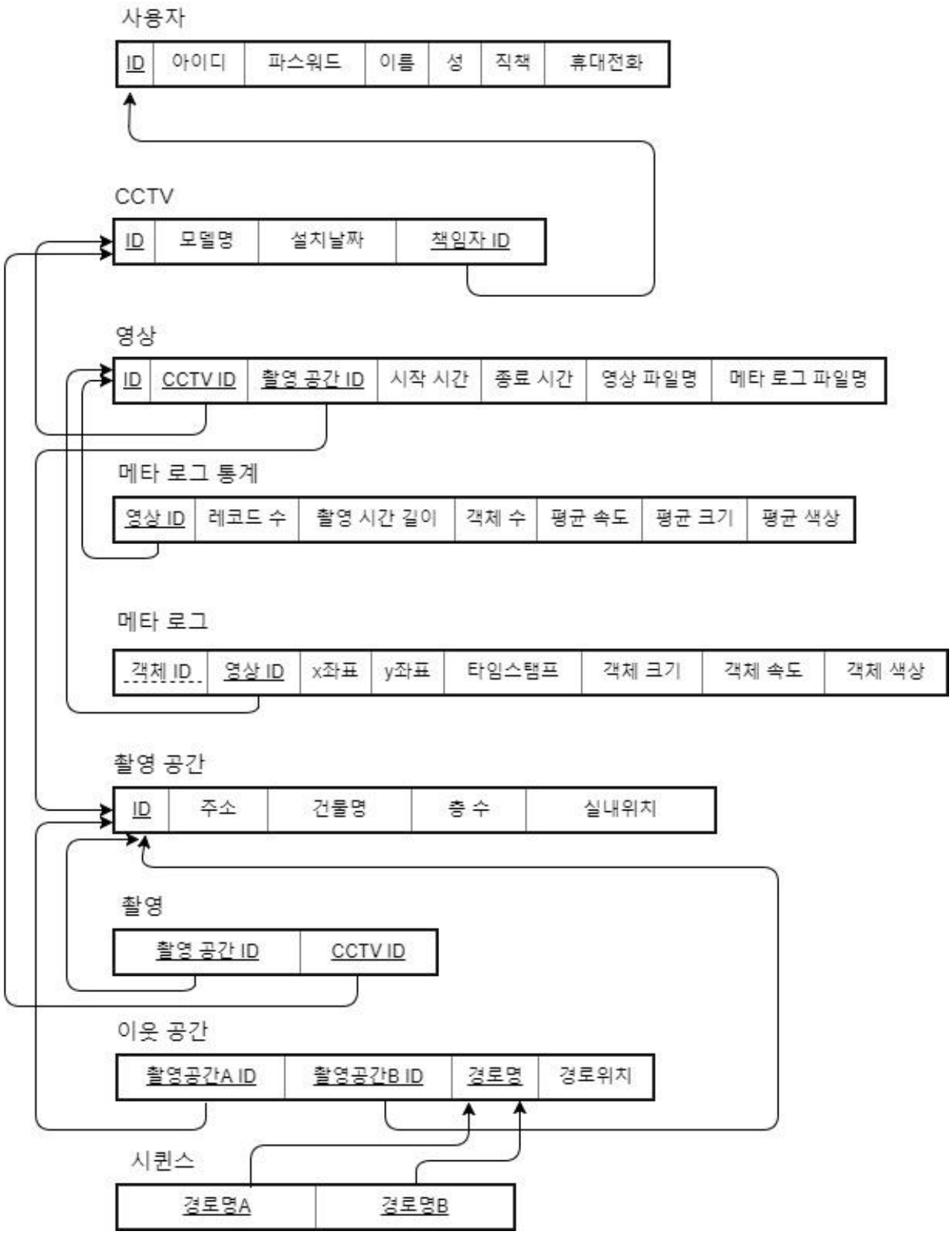
```
/*CHANGES ON LENGTH*/

TABLE `auth_user`
`password`      VARCHAR(30)              -> VARCHAR(128)
`phone_number`  VARCHAR(11)              -> VARCHAR(20)

TABLE `management_cctv`
`model_name`    VARCHAR(10)              -> VARCHAR(30)

TABLE `management_space`
`address`           VARCHAR(50)          -> VARCHAR(100)
`building_name`     VARCHAR(10)          -> VARCHAR(50)
`inroom_position`   VARCHAR(20)          -> VARCHAR(30)

TABLE `management_neighbor`
`route_name`     VARCHAR(20)             -> VARCHAR(100)
`route_position` VARCHAR(20)             -> VARCHAR(100)

TABLE `management_video`
`video_file` VARCHAR(100)                -> VARCHAR(500)
`log_file` VARCHAR(100)                  -> VARCHAR(500)
`end_time` DATETIME                      -> DATETIME(6)
`start_time` DATETIME                    -> DATETIME(6)

TABLE `management_metalog`
`timestamp` DATETIME                     -> DATETIME(6)
```

3.1.2. Other changes as in auth_user table attribute name is changed to first name last name for better standardized result, position is changed to Boolean ‘is_superuser’ as there is only super user and normal user, and username attribute is added to differentiate between name. In CCTV table, instead of date and time, only date is needed. In metalog table, in order to count the number of object, object ID is added for identification. Besides, foreign keys are added as table are all mutual related.

```
TABLE `auth_user`
`name`          VARCHAR(30)              -> `first_name` VARCHAR(30) NOT NULL,`last_name` VARCHAR(30) NOT NULL,
`position`      VARCHAR(30)  NOT NULL    -> `is_superuser` TINYINT(1) NOT NULL DEFAULT '0',
`username`      VARCHAR(150) NOT NULL    (add new attribute)

TABLE `management_cctv`
`install_date`  DATETIME                 -> DATE
add foreign keys on update cascade

TABLE `management_space`
(drop `in_charge_user_id` attribute) reason: duplicate = redundant data
add foreign keys on update cascade

TABLE `management_neighbor`
add foreign keys on update cascade on delete cascade

TABLE `management_sequence`
add foreign keys on update cascade on delete cascade

TABLE `management_video`
add foreign keys on update cascade

TABLE `management_metalog`
`object_id` INT(11)                              (add new attribute)
add foreign keys on update cascade
```

3.2. As attribute of some entities are modified, new version of ERD Diagram and Relational Schema are attached.




사용자
ID
아이디
파스워드
이름
성
직책
휴대전화
CCTV
ID
모델명
설치날짜
책임자 ID
영상
ID
CCTV ID
촬영 공간 ID
시작 시간
종료 시간
영상 파일명
메타 로그 파일명
메타 로그 통계
영상 ID
레코드 수
촬영 시간 길이
객체 수
평균 속도
평균 크기
평균 색상
메타 로그
객체 ID
영상 ID
x좌표
y좌표
타임스탬프
객체 크기
객체 속도
객체 색상
촬영 공간
ID
주소
건물명
층 수
실내위치
촬영
촬영 공간 ID
CCTV ID
이웃 공간
촬영공간A ID
촬영공간B ID
경로명
경로위치
시퀀스
경로명A
경로명B