

Cookie and Session

HTTP(Hypertext Transfer Protocol)

클라이언트와 서버 간에 데이터를 주고 받기 위한 프로토콜

1. 비 연결 지향

- 서버는 요청에 대한 응답을 보낸 후 연결을 끊음

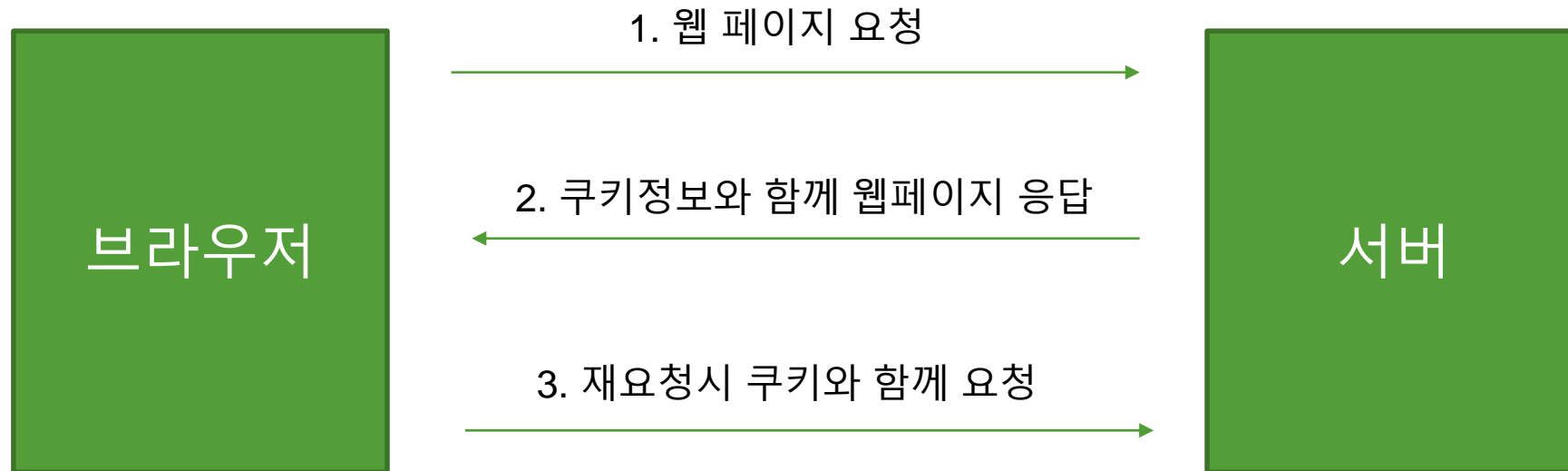
2. 무상태

- 연결이 끊어지면 클라이언트와 서버 간의 통신이 끝나며 정보가 유지되지 않음
- ex) 로그인 상태가 유지될 수 없음 / 장바구니가 유지되지 않음

Cookie

웹 사이트가 사용자의 브라우저에 저장하는 작은 데이터 파일

=> 사용자의 세션 정보, 개인 설정 및 상태 유지 등에 사용됨



Cookie 사용 원리

1. 클라이언트는 쿠키를 Key-Value 형태로 저장
2. 클라이언트 화면에 필요한 데이터가 쿠키에 저장되어 있다면, 해당 쿠키로 화면을 랜더링
3. 동일한 서버에 요청을 다시 보낼 경우 저장된 쿠키를 함께 전송
 - 로그인 정보를 쿠키에 넣고 관리함으로써 로그인이 유지됨

=> 쿠팡 장바구니 예시 확인하기

Cookie 사용 목적

1. 세션 관리

- 로그인, 아이디 자동완성, 팝업 체크, 장바구니 등의 정보관리

2. 개인화

- 언어 설정, 테마 설정, 자동로그인

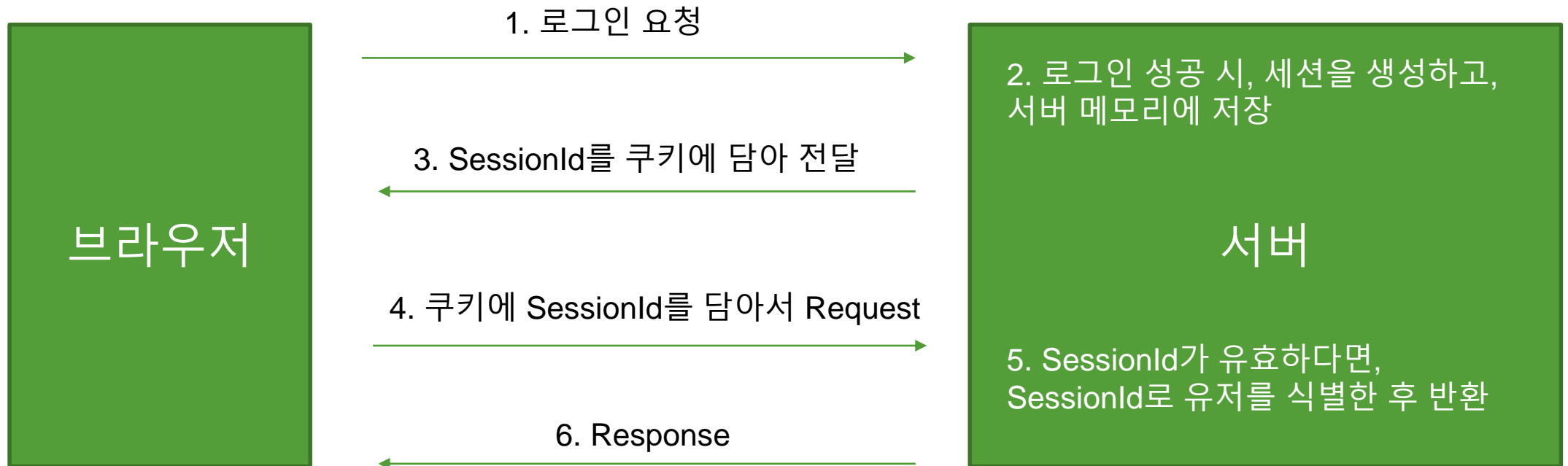
3. 트래킹

- 방문 페이지에 따른 광고, 관련성이 높은 상품 추천

Session

서버와 클라이언트 간의 상태를 유지하기 위해 서버에서 생성하는
데이터 저장 방식

=> 쿠키에 세션 데이터를 저장하여 매 요청 시 세션 데이터를 보냄



Session 종류

1. Session Cookie

- 현재 세션이 종료되면 삭제
- 브라우저 종료와 함께 세션이 삭제

2. Persistent Cookies

- Expires 속성에 지정된 날짜 or Max-age 속성에 지정된 기간이 지나면 삭제

Django Auth

Django Auth 시작 전 준비사항

accounts app 생성 및 app url 경로 세팅하기

=> Django 에서는 auth와 관련된 내용을 accounts로 사용하기 때문에 가급적이면

accounts를 그대로 사용하는 것을 권장

```
from django.urls import path
from . import views

# accounts/urls.py
app_name = 'accounts'
urlpatterns = [
    path('index/', views.index, name='index'),
]
```

```
# projects/urls.py
urlpatterns = [
    path('accounts/', include('accounts.urls')),
    ...
]
```

Django Custom User Model

기존에 장고가 지원해주는 User Model은 개발자가 직접 수정 불가능

- Django.contrib.auth에서 지원하는 user model code ([코드 링크](#))

```
# Application definition
# my_project/settings.py
INSTALLED_APPS = [
    'articles',
    'accounts',
    'django_extensions',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

```
auth_user
└─ columns
    ├── id int
    ├── password varchar(128)
    ├── last_login datetime(6)
    ├── is_superuser tinyint(1)
    ├── username varchar(150)
    ├── first_name varchar(150)
    ├── last_name varchar(150)
    ├── email varchar(254)
    ├── is_staff tinyint(1)
    ├── is_active tinyint(1)
    └── date_joined datetime(6)
```

```
class AbstractUser(AbstractBaseUser, PermissionsMixin):
    """
    An abstract base class implementing a fully featured User model
    admin-compliant permissions.

    Username and password are required. Other fields are optional.
    """

    username_validator = UnicodeUsernameValidator()

    username = models.CharField(
        _("username"),
        max_length=150,
        unique=True,
        help_text=(
            "Required. 150 characters or fewer. Letters, digits and "
            "underscores only. Beginning and ending with a letter or digit."
        ),
        validators=[username_validator],
        error_messages={
            "unique": _("A user with that username already exists."),
        },
    )
    first_name = models.CharField(_("first name"), max_length=150, blank=True)
    last_name = models.CharField(_("last name"), max_length=150, blank=True)
    email = models.EmailField(_("email address"), blank=True)
```

Django Custom User Model

프로젝트에서 바로 User Model 을 사용하지 않더라도 초기에 설정 필수

- 모델끼리 내부적으로 의존성이 강하게 얹혀 있어서 이후에 user model 변경 불가
- 이미 프로젝트가 진행된 경우에는 데이터베이스를 초기화한 후에 진행해야 함
- 공식문서에서도 프로젝트를 시작할 때 초기 설정하는 것을 강력하게 권장

Using a custom user model when starting a project 📄

If you're starting a new project, it's highly recommended to set up a custom user model, even if the default **User** model is sufficient for you. This model behaves identically to the default user model, but you'll be able to customize it in the future if the need arises:

```
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
    pass
```

Django Custom User Model

1. 기존 유저 모델이 상속받은 AbstractUser를 이용해 Custom User Model 생성하기

- 기존 유저 모델과 완전히 같은 기능을 하지만 이후 커스텀이 가능

```
from django.db import models
from django.contrib.auth.models import AbstractUser

# accounts/models.py
class User(AbstractUser):
    # 바로 User Model을 작성하지 않더라도,
    # 이후 유저 모델 커스텀을 위해서 미리 작성
    pass
```

Django Custom User Model

2. Django 프로젝트의 기본 유저 모델을 (1)에서 생성한 모델로 변경

```
#MyProject/settings.py

# settings.py에 적혀있지는 않지만 Django 내부적으로
# 기초 모델은 'auth.User'로 설정되어 있음
# AUTH_USER_MODEL = 'auth.User'

# 내가 생성한 User Model로 기본 User Model 변경
AUTH_USER_MODEL = 'accounts.User'
```

Django Custom User Model

3. Admin site 출력을 위해 생성한 User Model을 등록

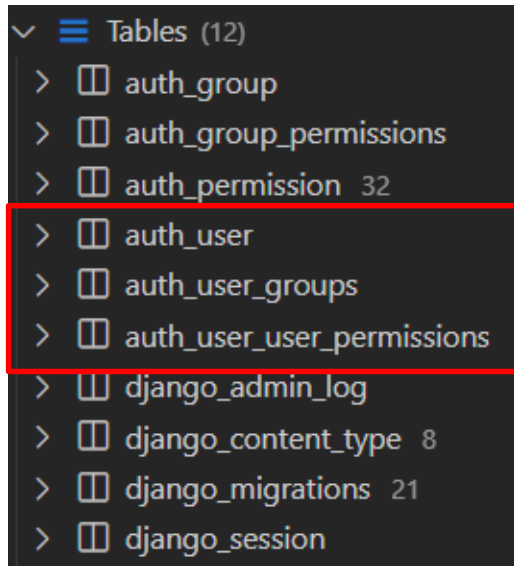
```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import User

# accounts/admin.py
admin.site.register(User, UserAdmin)
```

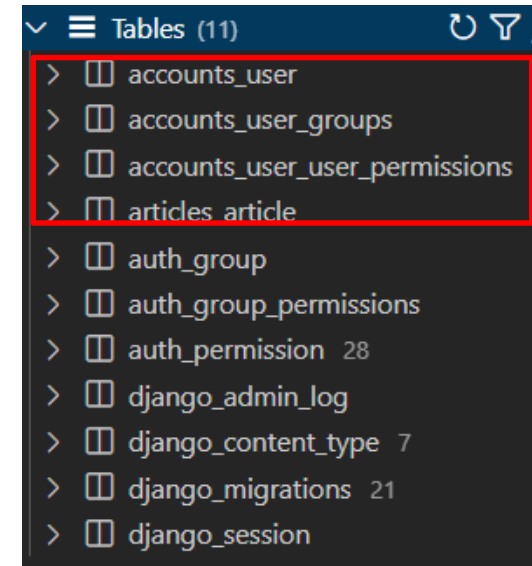
Django Custom User Model

4. DB 마이그레이션 진행 후 테이블 확인

- 이미 테이블이 생성되어 있다면 모두 삭제 후 진행
- vscode 테이블 목록에서 오른쪽 클릭 후 drop 진행 시 테이블 삭제



기존 User Model



Custom User Model

Django Login

로그인에 사용할 데이터를 입력 받는 **AuthenticationForm()**을 사용

- 로그인 페이지 불러오기

```
from django.urls import path
from . import views

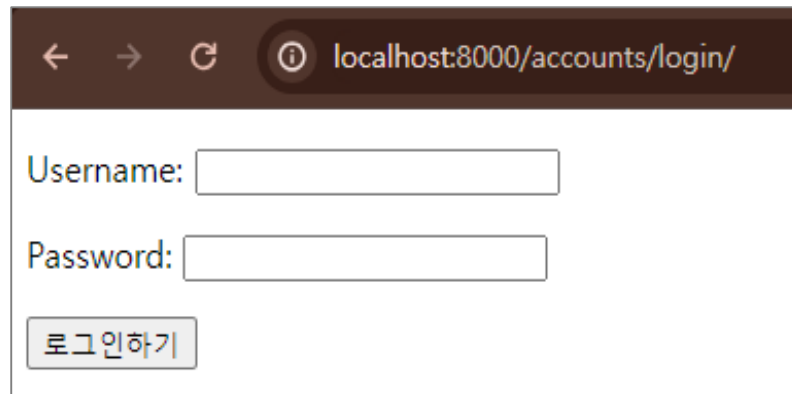
# accounts/urls.py
app_name = 'accounts'
urlpatterns = [
    path('login/', views.login, name='login')
]
```

```
from django.shortcuts import render
from django.contrib.auth.forms import AuthenticationForm

# accounts/views.py
def login(request):
    if request.method == "POST":
        pass
    else:
        form = AuthenticationForm()
        context = {
            "form": form
        }
    return render(request, "accounts/login.html", context)
```

```
<!-- accounts/login.html -->
{% extends "base.html" %}

{% block content %}
    <form action="{% url 'accounts:login' %}" method="POST">
        {% csrf_token %}
        {{ form.as_p }}
        <input type="submit" value="로그인하기">
    </form>
{% endblock content %}
```



← → ↻ ⓘ localhost:8000/accounts/login/

Username:

Password:

Django Login

로그인에 사용할 데이터를 입력 받는 **AuthenticationForm()**을 사용

- 로그인 기능 구현

1. login(request, user)

- AuthenticationForm으로 입력 받은 유저 정보
데이터로 로그인

2. AuthenticationForm.get_user()

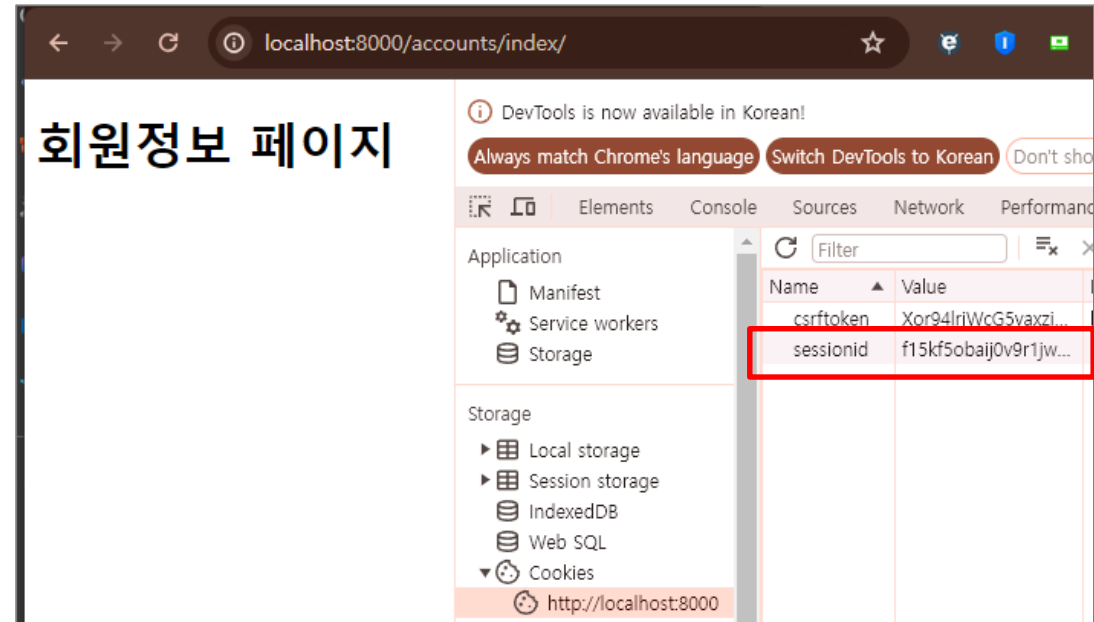
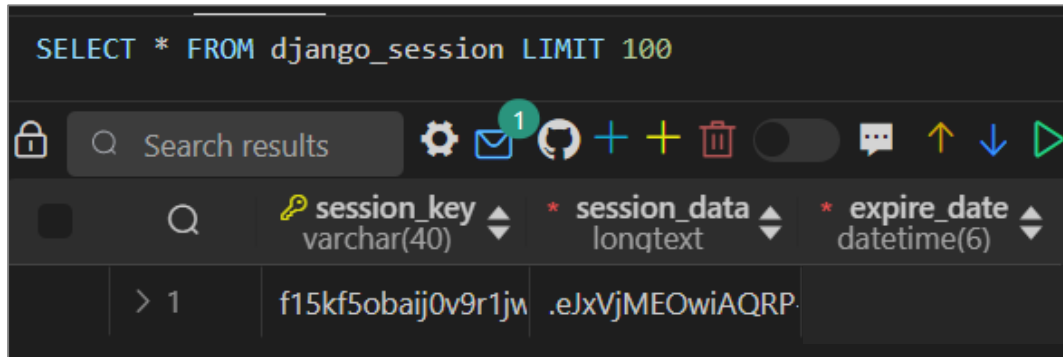
- 로그인 성공 시 로그인한 사용자 정보 반환

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login as auth_login

# accounts/views.py
def login(request):
    if request.method == "POST":
        form = AuthenticationForm(request, request.POST)
        if form.is_valid():
            # contrib.auth 패키지의 login 함수를 사용하는데
            # 현재 함수와 이름이 중복이므로 별명을 지어서 사용
            # login(request, form.get_user())
            1. auth_login(request, 2. form.get_user())
            return redirect('accounts:index')
        else:
            form = AuthenticationForm()
            context = {
                "form": form
            }
            return render(request, "accounts/login.html", context)
```

Django Login

슈퍼계정 생성 후 로그인 테스트 및 세션 데이터 확인

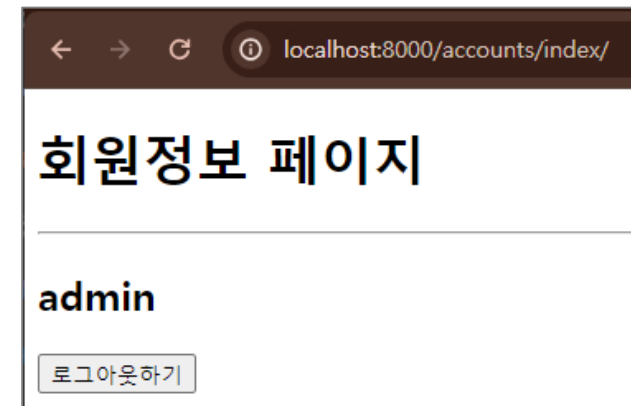


Django 유저 정보 출력

Django는 자주 사용하는 데이터를 미리 템플릿에 포함시키는데,
그 중 **user** 를 사용하면 context에 데이터를 전달하지 않아도 사용 가능

```
# MyProjects/settings.py
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]
```

```
<!-- accounts/index.html -->
{% extends "base.html" %}
{% block content %}
<h1>회원정보 페이지</h1>
<hr>
<h2>{{ user.username }}</h2>
<form action="{% url 'accounts:logout' %}" method="POST">
    {% csrf_token %}
    <input type="submit" value="로그아웃하기">
</form>
{% endblock content %}
```



Django Logout

Django.contrib.auth 패키지의 **logout** 사용

- 회원정보 페이지에서 로그아웃 버튼 작성 및 로그아웃 로직 작성

```
# accounts/urls.py
app_name = 'accounts'
urlpatterns = [
    path('index/', views.index, name='index'),
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout')
]
```

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login as auth_login
from django.contrib.auth import logout as auth_logout
```

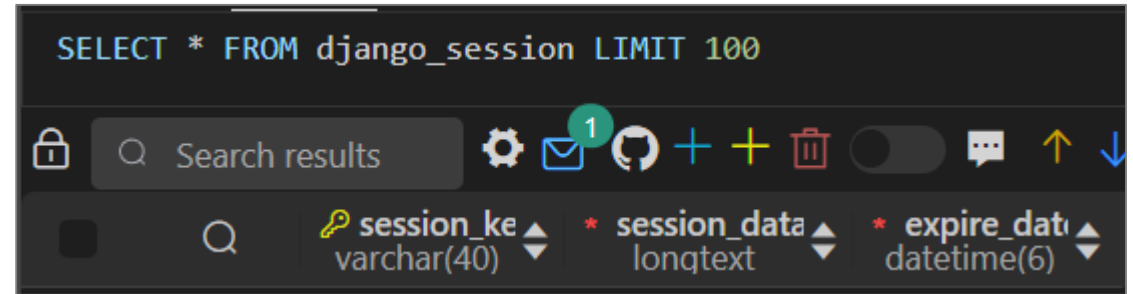
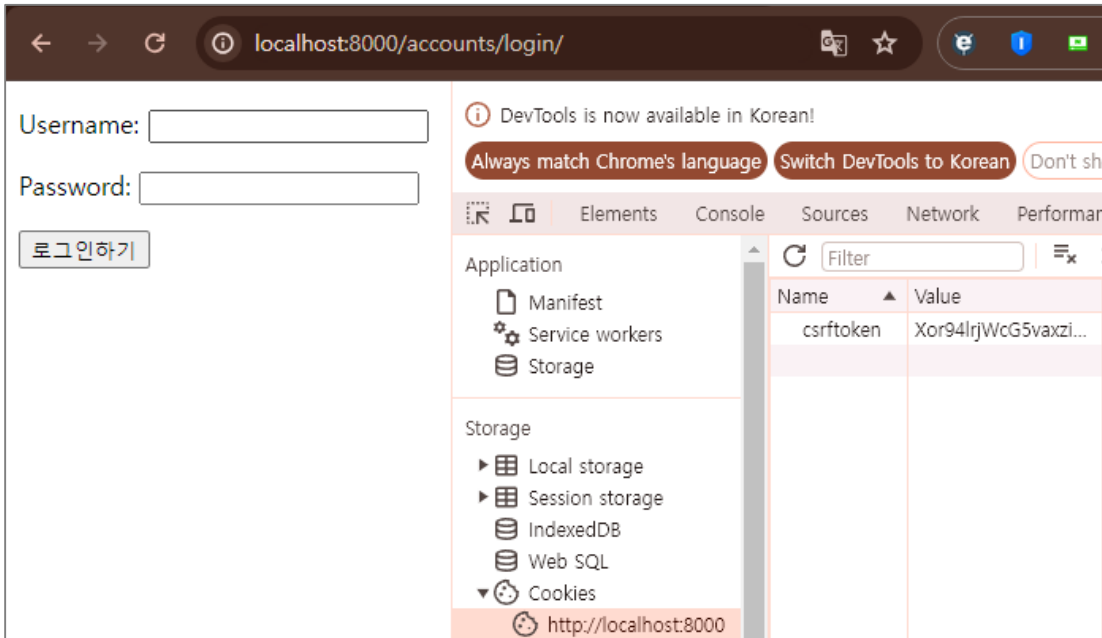
```
# accounts/views.py
def logout(request):
    auth_logout(request)
    return redirect("accounts:login")
```

```
<!-- accounts/index.html-->
{% extends "base.html" %}
{% block content %}
<h1>회원정보 페이지</h1>
<form action="{% url 'accounts:logout' %}" method="POST">
    {% csrf_token %}
    <input type="submit" value="로그아웃하기">
</form>
{% endblock content %}
```



Django Logout

- 클라이언트는 쿠키에서 session id를 삭제
- 서버는 session data를 DB에서 삭제



Django 회원가입

회원가입에 사용할 데이터를 입력 받는 **UserCreationForm()**을 사용

- 회원가입 페이지 작성

```
# accounts/urls.py
app_name = 'accounts'
urlpatterns = [
    path('index/', views.index, name='index'),
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout'),
    path('signup/', views.signup, name='signup')
]
```

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm
from django.contrib.auth import login as auth_login
from django.contrib.auth import logout as auth_logout

# accounts/views.py
def signup(request):
    if request.method == "POST":
        pass
    else:
        form = UserCreationForm()
    context = {
        'form': form
    }
    return render(request, 'accounts/signup.html', context)
```

```
<!-- accounts/signup.html -->
{% extends "base.html" %}

{% block content %}
    <form action="{% url 'accounts:signup' %}" method="POST">
        {% csrf_token %}
        {{ form.as_p }}
        <input type="submit" value="회원가입하기">
    </form>
{% endblock content %}
```

localhost:8000/accounts/signup/

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

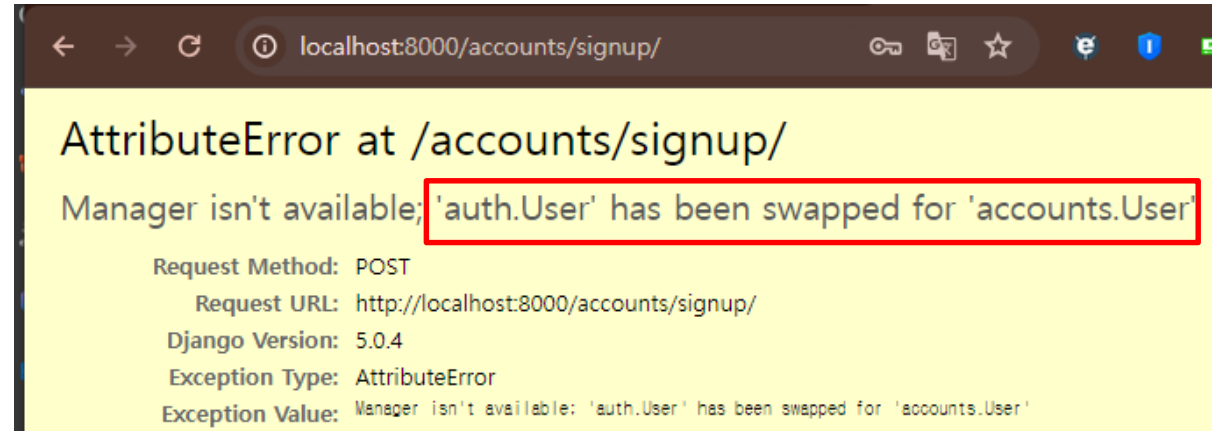
Django 회원가입

회원가입에 사용할 데이터를 입력 받는 **UserCreationForm()**을 사용

- 회원가입 로직 작성 (에러 발생)

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm
from django.contrib.auth import login as auth_login
from django.contrib.auth import logout as auth_logout

# accounts/views.py
def signup(request):
    if request.method == "POST":
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('accounts:login')
        else:
            form = UserCreationForm()
            context = {
                'form': form
            }
            return render(request, 'accounts/signup.html', context)
```



Django 회원가입

회원가입에 사용되는 UserCreationForm은 기존 유저 모델 기반 작성

- UserCreationForm 코드 참고 ([링크](#))

=> 우리가 생성한 User Model을 기반으로 UserCreationForm을 재작성하자!!!

```
class BaseUserCreationForm(SetPasswordMixin, forms.ModelForm):
    """
    A form that creates a user, with no privileges, from
    password.
    """

    password1, password2 = SetPasswordMixin.create_password_fields()
    usable_password = SetPasswordMixin.create_usable_password()

    class Meta:
        model = User
        fields = ("username",)
        field_classes = {"username": UsernameField}
```

localhost:8000/accounts/signup/

AttributeError at /accounts/signup/

Manager isn't available; 'auth.User' has been swapped for 'accounts.User'

Request Method: POST
Request URL: http://localhost:8000/accounts/signup/
Django Version: 5.0.4
Exception Type: AttributeError
Exception Value: Manager isn't available; 'auth.User' has been swapped for 'accounts.User'

Django 회원가입

기존 UserCreationForm을 그대로 상속받고, 오직 model 부분만 변경

- get_user_model
 - 현재 프로젝트에서 적용중인 유저 모델을 반환
 - 원활한 유지 보수(직접적인 모델명 변경 X) 를 위해 사용을 권장

```
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import get_user_model

# accounts/forms.py
class CustomUserCreationForm(UserCreationForm):
    class Meta(UserCreationForm.Meta):
        model = get_user_model()
```

Django 회원가입

새로 생성한 CustomUserCreationForm을 활용한 회원가입 로직 작성

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm
from django.contrib.auth import login as auth_login
from django.contrib.auth import logout as auth_logout
from .forms import CustomUserCreationForm

# accounts/views.py
def signup(request):
    if request.method == "POST":
        # form = UserCreationForm(request.POST)
        form = CustomUserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('accounts:login')
    else:
        # form = UserCreationForm()
        form = CustomUserCreationForm()
    context = {
        'form': form
    }
    return render(request, 'accounts/signup.html', context)
```

SELECT * FROM accounts_user LIMIT 100

	id	password	last_login	is_superuser	username
	bigint	varchar(128)	datetime	tinyint(1)	varchar(150)
> 1	1	pbkdf2_sha256\$7	2024-0	1	admin
> 2	2	pbkdf2_sha256\$7	(NULL)	0	root

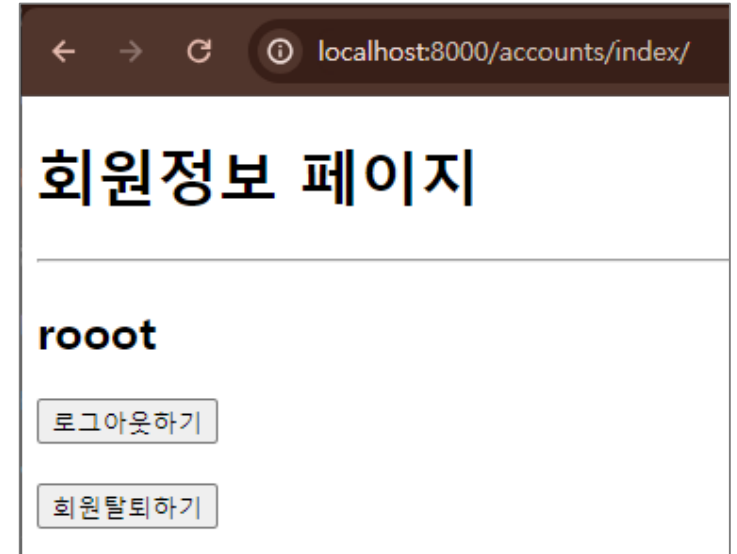
Django 회원탈퇴

request user 정보의 delete 메소드를 활용하여 삭제

```
# accounts/urls.py
app_name = 'accounts'
urlpatterns = [
    path('index/', views.index, name='index'),
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout'),
    path('signup/', views.signup, name='signup'),
    path('delete/', views.delete, name='delete')
]
```

```
# accounts/views.py
def delete(request):
    request.user.delete()
    return redirect('accounts:login')
```

```
<!-- accounts/index.html -->
{% extends "base.html" %}
{% block content %}
<h1>회원정보 페이지</h1>
<hr>
<h2>{{ user.username }}</h2>
<form action="{% url 'accounts:logout' %}" method="POST">
    {% csrf_token %}
    <input type="submit" value="로그아웃하기">
</form><br>
<form action="{% url 'accounts:delete' %}" method="POST">
    {% csrf_token %}
    <input type="submit" value="회원탈퇴하기">
</form>
{% endblock content %}
```



SELECT * FROM accounts_user LIMIT 100

id	password	last_login	is_superuser	username
1	pbkdf2_sha256\$720000\$8rt	2024-04-11 15:11:45.452210	1	admin

Django 회원정보 수정

정보수정에 데이터를 입력 받는 **UserChangeForm()**을 사용

- 정보수정 페이지 작성 및 **CustomUserChangeForm()** 작성

```
# accounts/urls.py
app_name = 'accounts'
urlpatterns = [
    path('index/', views.index, name='index'),
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout'),
    path('signup/', views.signup, name='signup'),
    path('delete/', views.delete, name='delete'),
    path('update/', views.update, name="update")
]
```

```
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from django.contrib.auth import get_user_model
```

```
# accounts/forms.py
class CustomUserChangeForm(UserChangeForm):
    class Meta(UserChangeForm.Meta):
        model = get_user_model()
```

```
from .forms import CustomUserCreationForm, CustomUserChangeForm

#accounts/views.py
def update(request):
    if request.method == "POST":
        pass
    else:
        form = CustomUserChangeForm(instance=request.user)
    context = {
        'form': form
    }
    return render(request, "accounts/update.html", context)
```

Django 회원정보 수정

정보수정에 데이터를 입력 받는 **UserChangeForm()**을 사용

- 정보수정 페이지 작성 및 **CustomUserChangeForm()** 작성

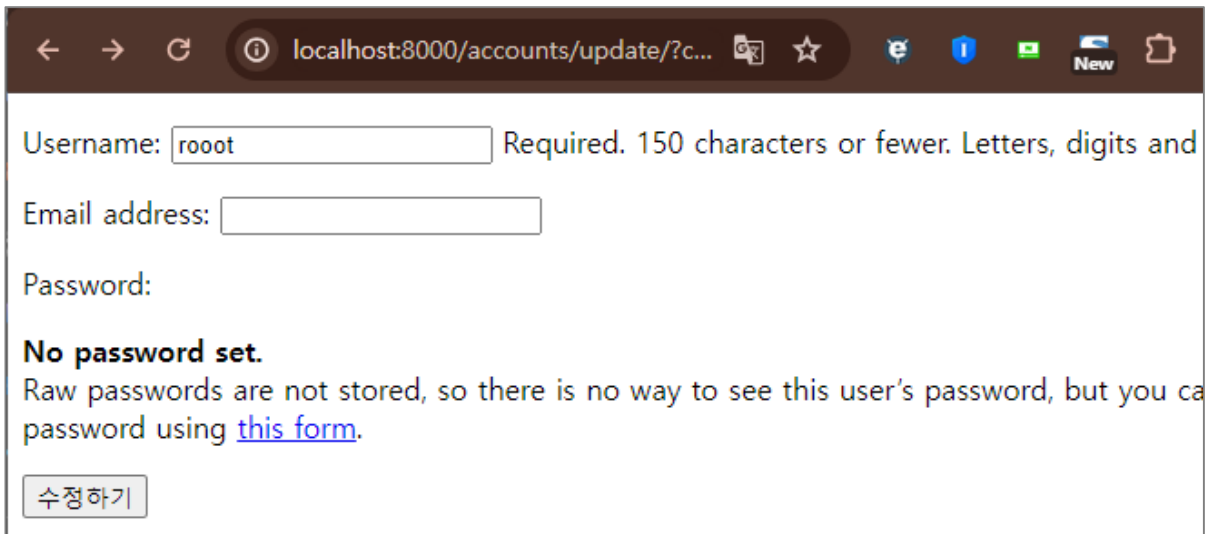
```
<!-- accounts/index.html -->
{% extends "base.html" %}
{% block content %}
<h1>회원정보 페이지</h1>
<hr>
<h2>{{ user.username }}</h2>
<form action="{% url 'accounts:update' %}" method="POST">
  {% csrf_token %}
  <input type="submit" value="회원정보 수정하기">
</form><br>
<form action="{% url 'accounts:logout' %}" method="POST">
  {% csrf_token %}
  <input type="submit" value="로그아웃하기">
</form><br>
<form action="{% url 'accounts:delete' %}" method="POST">
  {% csrf_token %}
  <input type="submit" value="회원탈퇴하기">
</form>
{% endblock content %}
```

```
<!-- accounts/update.html -->
{% extends "base.html" %}
{% block content %}
  <form action="{% url 'accounts:update' %}" method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="수정하기">
  </form>
{% endblock content %}
```

Django 회원정보 수정

원하는 회원정보는 나오도록 CustomUserChangeForm 수정

```
# accounts/forms.py
class CustomUserChangeForm(UserChangeForm):
    class Meta(UserChangeForm.Meta):
        model = get_user_model()
        fields = ('username', 'email')
```



← → ↻ ⓘ localhost:8000/accounts/update/?c... ☆ 🐙 ⓘ 🇺🇸 New 📁

Username: Required. 150 characters or fewer. Letters, digits and

Email address:

Password:

No password set.
Raw passwords are not stored, so there is no way to see this user's password, but you can reset it using [this form](#).

Django 회원정보 수정

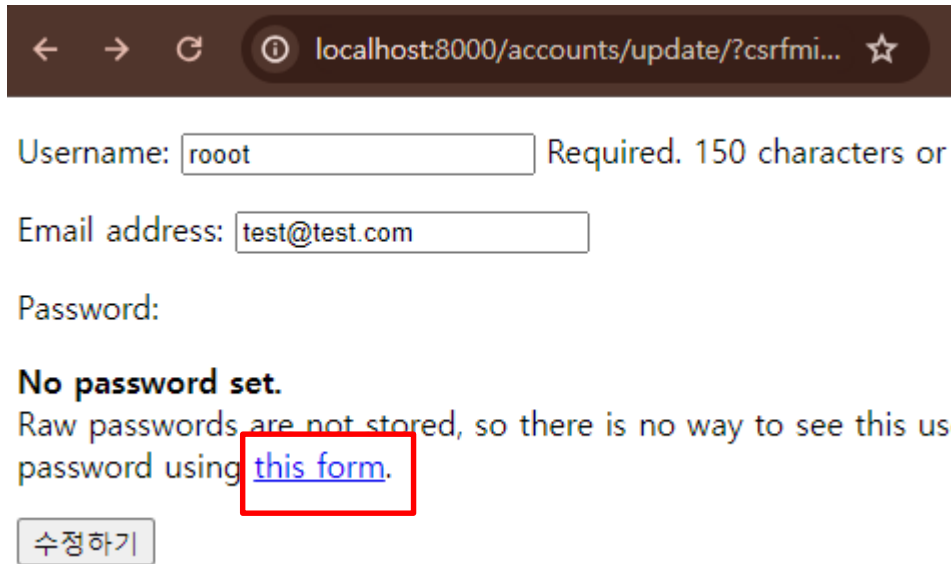
회원정보 수정 로직 작성

```
#accounts/views.py
def update(request):
    if request.method == "POST":
        form = CustomUserChangeForm(request.POST, instance=request.user)
        if form.is_valid():
            form.save()
            return redirect('accounts:index')
    else:
        form = CustomUserChangeForm(instance=request.user)
    context = {
        'form': form
    }
    return render(request, "accounts/update.html", context)
```

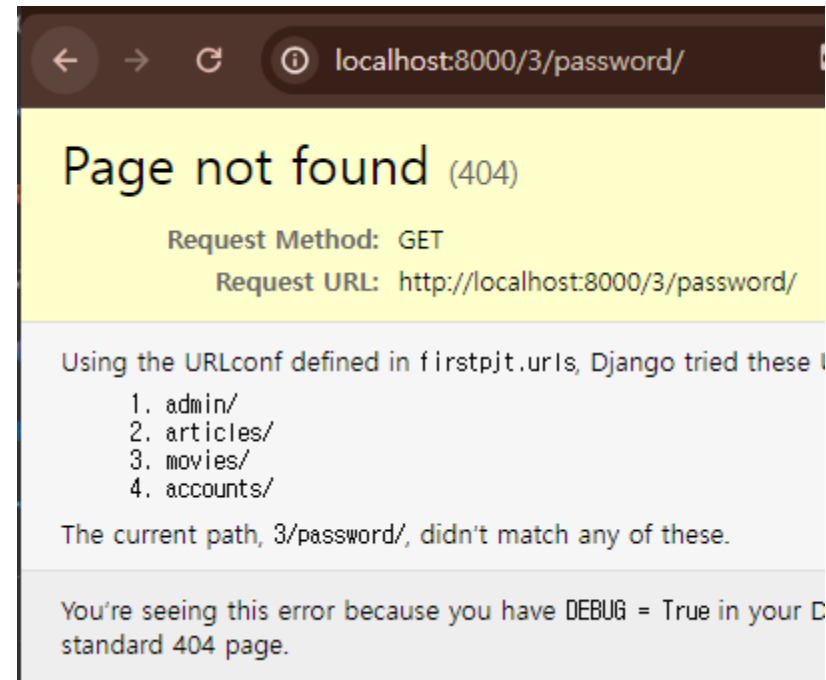
Django 비밀번호 변경

비밀번호 변경 시 입력 받는 **PasswordChangeForm()**을 사용

- Django는 수정페이지에서 비밀번호 변경 페이지로 이동하는 것을 지원



A screenshot of a web browser showing the Django password change form. The address bar shows 'localhost:8000/accounts/update/?csrfmi...'. The form has three input fields: 'Username:' with 'root', 'Email address:' with 'test@test.com', and 'Password:'. Below the fields, it says 'No password set. Raw passwords are not stored, so there is no way to see this us password using [this form.](#)' where 'this form.' is highlighted with a red box. At the bottom is a button labeled '수정하기'.



Django 비밀번호 변경

비밀번호 변경 페이지 생성

```
from django.contrib import admin
from django.urls import path, include
from accounts import views

# MyProejects/urls.py
urlpatterns = [
    path('admin/', admin.site.urls),
    path('articles/', include('articles.urls')),
    path('movies/', include('movies.urls')),
    path('accounts/', include('accounts.urls')),
    path('<int:user_id>/password/', views.change_password, name="change_password"),
]
```

```
from django.contrib.auth.forms import AuthenticationForm, UserCreationForm, PasswordChangeForm
from django.contrib.auth import login as auth_login
from django.contrib.auth import logout as auth_logout
from .forms import CustomUserCreationForm, CustomUserChangeForm

# accounts/views.py
def change_password(request, user_id):
    if request.method == "POST":
        pass
    else:
        form = PasswordChangeForm(request.user)
    context = {
        'form': form
    }
    return render(request, 'accounts/change_password.html', context)
```

```
<!-- accounts/change_password.html -->
{% extends "base.html" %}
{% block content %}
    <form action="{% url 'change_password' user_id=user.pk %}" method="POST">
        {% csrf_token %}
        {{ form.as_p }}
        <input type="submit" value="변경하기">
    </form>
{% endblock content %}
```

← → ↺ ⓘ localhost:8000/3/password/ ⓘ ☆ ⓘ ⓘ ⓘ

Old password:

New password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

New password confirmation:

Django 비밀번호 변경

비밀번호 변경 로직 작성

- update_session_auth_hash 함수를 이용해 비밀번호 변경 후에도 세션 유지

```
# accounts/views.py
def change_password(request, user_id):
    if request.method == "POST":
        form = PasswordChangeForm(request.user, request.POST)
        if form.is_valid():
            user = form.save()
            update_session_auth_hash(request, user)
            return redirect('accounts:index')
    else:
        form = PasswordChangeForm(request.user)
    context = {
        'form': form
    }
    return render(request, 'accounts/change_password.html', context)
```

Django 사용자 접근 제한

로그인/비로그인 사용자별 접근 제한이 가능

1. **is_authenticated** 속성

- User Model의 속성으로 로그인 유저는 True, 비로그인 유저는 False를 반환

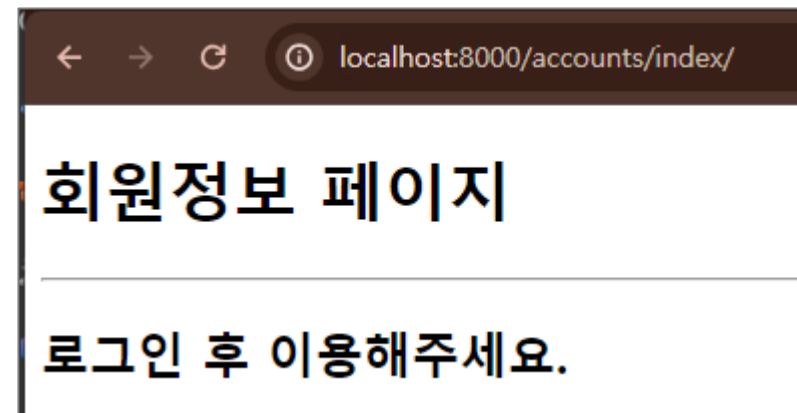
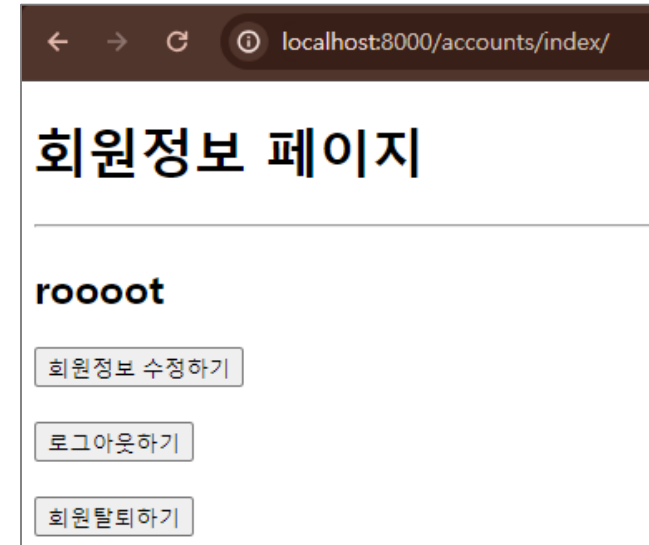
2. **login_required** 데코레이터

- 인증된 사용자에 대해서만 해당 데코레이터 아래 함수를 실행

Django 사용자 접근 제한

is_authenticated 속성 (template에 사용)

```
<!-- accounts/index.html -->
{% extends "base.html" %}
{% block content %}
<h1>회원정보 페이지</h1>
<hr>
{% if user.is_authenticated %}
<h2>{{ user.username }}</h2>
<form action="{% url 'accounts:update' %}" method="POST">
    {% csrf_token %}
    <input type="submit" value="회원정보 수정하기">
</form><br>
<form action="{% url 'accounts:logout' %}" method="POST">
    {% csrf_token %}
    <input type="submit" value="로그아웃하기">
</form><br>
<form action="{% url 'accounts:delete' %}" method="POST">
    {% csrf_token %}
    <input type="submit" value="회원탈퇴하기">
</form>
{% else %}
<h2>로그인 후 이용해주세요.</h2>
{% endif %}
{% endblock content %}
```



Django 사용자 접근 제한

is_authenticated 속성 (view에 사용)

```
# accounts/views.py
def signup(request):
    # 이미 로그인한 경우, 회원가입 로직 실행 막기
    if request.user.is_authenticated:
        return redirect("accounts:index")
```

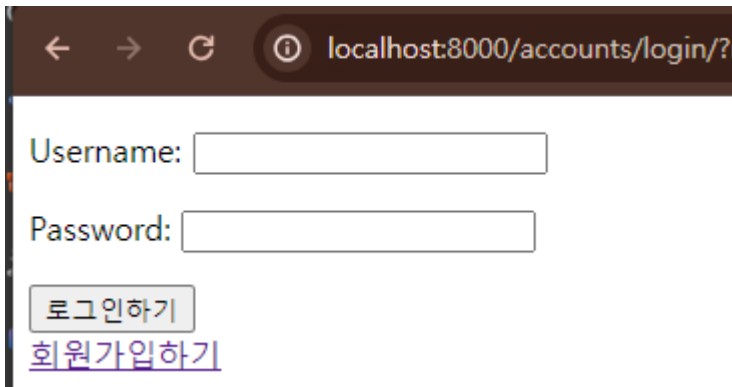
```
#accounts/views.py
def update(request):
    # 로그인하지 않은 경우, 정보수정 로직 수행 제한
    if not request.user.is_authenticated:
        return redirect("accounts:index")
```

Django 사용자 접근 제한

login_required 데코레이터 (view에 사용)

=> 잘못된 접근 시 무조건 /accounts/login/ 주소로 redirect 진행

```
#accounts/views.py
# 로그인하지 않은 경우, 정보수정 로직 수행 제한
@login_required
def update(request):
    if request.method == "POST":
```



A screenshot of a web browser window. The address bar shows 'localhost:8000/accounts/login/'. The page contains a login form with two input fields: 'Username:' and 'Password:'. Below the password field is a button labeled '로그인하기' (Login). At the bottom of the form is a link labeled '회원가입하기' (Sign Up).