شروع توضیحات کد app.py (برای ارائه):

1. Import های اولیه:

Python

```
import os
import secrets
from flask import (Flask, render_template, request, redirect, url_for, jsonify, flash, abort, make_response, g)
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime, timedelta, timezone
from functools import wraps
from urllib.parse import urlparse, urljoin
```

- os: برای کار با مسیرهای فایل سیستم (مثلاً برای آدرس دیتابیس).
- secrets: برای تولید شناسه‌های نشست (Session ID) امن و تصادفی.
- flask: کتابخانه اصلی فلسک و ماژول‌های مورد نیاز آن:
  - Flask: کلاس اصلی برای ایجاد اپلیکیشن.
  - render_template: برای نمایش فایل‌های HTML.
  - request: برای دسترسی به اطلاعات درخواست‌های ورودی (مثل فرم‌ها، کوکی‌ها، هدرها).
  - redirect: برای هدایت کاربر به URL دیگر.
  - url_for: برای تولید URL بر اساس نام تابع (view function).
  - jsonify: برای تبدیل دیکشنری پایتون به پاسخ JSON (مفید برای API ها و درخواست‌های AJAX).
  - flash: برای نمایش پیام‌های موقت به کاربر (مثل "ورود با موفقیت انجام شد").
  - abort: برای متوقف کردن درخواست با یک کد خطای HTTP (مثل 404 یا 403).
  - make_response: برای ساخت یک شیء پاسخ کامل (مفید برای تنظیم کوکی‌ها).
  - g: یک آبجکت "گلوبال" مخصوص هر درخواست. از آن برای ذخیره اطلاعاتی مثل کاربر لاگین شده (g.user) در طول یک چرخه درخواست-پاسخ استفاده می‌کنیم.
- flask_sqlalchemy: برای کار با دیتابیس به روش ORM (Object-Relational Mapper).
- werkzeug.security: برای هش کردن امن رمزهای عبور (generate_password_hash) و بررسی صحت آن‌ها (check_password_hash).

- datetime, timedelta, timezone: برای کار با تاریخ و زمان، محاسبه فاصله زمانی (برای انقضای نشست) و مدیریت مناطق زمانی (استفاده از UTC برای یکنواختی).
- functools.wraps: برای ساخت دکوراتورها به طوری که اطلاعات تابع اصلی (مثل نام و docstring) حفظ شود.
- urllib.parse.urlparse, urljoin: برای تجزیه و تحلیل URL ها و بررسی امنیت آن‌ها (در تابع is_safe_url).

2. پیکربندی برنامه Flask (app):

Python

```
app = Flask(__name__)
app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY',
'a-very-secure-dev-secret-key-manual')
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(minutes=30)

basedir = os.path.abspath(os.path.dirname(__file__))
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///' + os.path.join(basedir, 'app.db')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
```

- app = Flask(__name__): ایجاد یک نمونه از اپلیکیشن فلسک. __name__ به فلسک کمک می‌کند تا مسیر فایل‌های استاتیک و تمپلیت‌ها را پیدا کند.
- app.config['SECRET_KEY']: یک کلید مخفی و پیچیده که برای امضای کوکی‌های نشست و سایر موارد امنیتی استفاده می‌شود. **خیلی مهم است که در محیط پروداکشن یک مقدار امن و غیرقابل حدس باشد.**
- app.config['PERMANENT_SESSION_LIFETIME']: حداکثر طول عمر پیش‌فرض برای نشست‌های سرور (و کوکی‌ها) در صورتی که گزینه "مرا به خاطر بسپار" فعال نباشد. اینجا 30 دقیقه تنظیم شده.
- app.config['SQLALCHEMY_DATABASE_URI']: رشته اتصال به دیتابیس. در اینجا از SQLite استفاده شده که فایل دیتابیس (app.db) را در کنار همین فایل app.py ایجاد می‌کند.
- app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False: غیرفعال کردن قابلیت ردیابی تغییرات SQLAlchemy که در اکثر موارد نیاز نیست و منابع مصرف می‌کند.
- db = SQLAlchemy(app): ایجاد یک نمونه از SQLAlchemy و اتصال آن به اپلیکیشن فلسک. از این آبجکت db برای تعریف مدل‌ها و کار با دیتابیس استفاده می‌شود.

3. کنترل کش مرورگر (@app.after_request):

```Python
@app.after_request
def set_response_headers(response):
    # ... (کد تابع) ...
```

- این دکوراتور باعث می‌شود تابع set_response_headers **بعد از هر درخواست** و قبل از ارسال پاسخ به کلاینت اجرا شود.
- هدف این تابع، افزودن هدرهای HTTP به پاسخ است که به مرورگر دستور می‌دهند صفحات HTML را کش نکند (Cache-Control: no-cache, no-store, must-revalidate, Pragma: no-cache, Expires: 0).
- این کار مهم است تا مطمئن شویم کاربر همیشه آخرین نسخه صفحات را می‌بیند، مخصوصاً بعد از لاگین، لاگ‌اوت یا تغییر اطلاعات حساس.

4. مدل‌های دیتابیس (User, Post, ServerSession):

- **class User(db.Model):**
  - id: کلید اصلی (Primary Key) و یکتا برای هر کاربر.
  - username: نام کاربری، باید یکتا باشد.
  - password_hash: هش رمز عبور کاربر. **هرگز خود رمز عبور را ذخیره نمی‌کنیم.**
  - role: نقش کاربر ('user' یا 'admin').
  - posts: رابطه یک-به-چند با مدل Post. نشان می‌دهد هر کاربر می‌تواند چندین پست داشته باشد. "cascade="all, delete-orphan یعنی با حذف یک کاربر، تمام پست‌های او نیز به طور خودکار حذف می‌شوند.
  - server_sessions: رابطه یک-به-چند با مدل ServerSession. با حذف کاربر، نشست‌های فعال او هم حذف می‌شوند.
  - set_password(self, password): متدی برای هش کردن رمز عبور ورودی و ذخیره آن در password_hash.
  - check_password(self, password): متدی برای بررسی اینکه آیا رمز عبور ورودی با هش ذخیره شده مطابقت دارد یا خیر.
- **class Post(db.Model):**
  - id: کلید اصلی برای هر پست.
  - text: متن پست.
  - timestamp: زمان ایجاد پست (به طور پیش‌فرض با منطقه زمانی UTC).
  - user_id: کلید خارجی (Foreign Key) که به id کاربر نویسنده پست اشاره دارد.
- **class ServerSession(db.Model):**
  - این مدل برای **پیاده‌سازی دستی مدیریت نشست سمت سرور** ایجاد شده است.

- id: کلید اصلی برای هر رکورد نشست.
- session_id: یک رشته تصادفی و یکتا که به عنوان شناسه نشست عمل می‌کند. این شناسه در کوکی مرورگر کاربر ذخیره می‌شود.
- user_id: کلید خارجی به id کاربری که این نشست متعلق به اوست.
- remembered: یک فیلد boolean که نشان می‌دهد آیا گزینه "مرا به خاطر بسپار" هنگام ایجاد این نشست فعال بوده یا خیر. این فیلد برای تعیین طول عمر نشست در سمت سرور استفاده می‌شود.
- created_at: زمان ایجاد نشست.
- last_seen: آخرین باری که کاربر با این نشست فعالیتی داشته است. برای مدیریت انقضای نشست‌های فعال استفاده می‌شود.
- user_agent: اطلاعات مرورگر کاربر (اختیاری، برای امنیت یا تحلیل).
- ip_address: آدرس IP کاربر در زمان ایجاد نشست (اختیاری).

5. بارگذاری اطلاعات کاربر قبل از هر درخواست (@app.before_request):

Python

```
@app.before_request
def load_user_from_server_session():
    # ... (کد تابع) ...
```

- این تابع **قبل از اجرای هر تابع view (روت)** اجرا می‌شود.
- **وظیفه اصلی:** بررسی کوکی server_session_id، پیدا کردن نشست متناظر در جدول ServerSession، بررسی انقضای آن (با در نظر گرفتن وضعیت remembered)، و در صورت معتبر بودن، بارگذاری اطلاعات کاربر (User) و خود نشست (ServerSession) در آبجکت g (یعنی g.user و g.current_session).
- اگر نشست منقضی شده باشد یا session_id نامعتبر باشد، g.user برابر None باقی می‌ماند.
- همچنین last_seen نشست را به‌روز می‌کند تا انقضای آن بر اساس آخرین فعالیت باشد.
- در صورت وجود خطای AttributeError برای last_seen (که نباید رخ دهد)، نشست مشکل‌دار را حذف می‌کند.

6. توابع کمکی و دکوراتورها:

- **def login_required_server_session(f):**
  - یک **دکوراتور** است که برای محافظت از روت‌ها استفاده می‌شود.
  - بررسی می‌کند آیا g.user (که توسط load_user_from_server_session تنظیم شده) وجود دارد یا نه.
  - اگر g.user وجود نداشته باشد (یعنی کاربر لاگین نیست)، کاربر را به صفحه لاگین (url_for('login')) هدایت می‌کند و URL درخواستی فعلی را به عنوان پارامتر next به صفحه لاگین می‌فرستد تا پس از لاگین موفق، کاربر به همان صفحه بازگردد.

- **def admin_required(func):**
  - یک **دکوراتور** دیگر که دسترسی به یک روت را فقط به کاربرانی با نقش admin محدود می‌کند.
  - بررسی می‌کند آیا g.user وجود دارد و آیا g.user.role برابر با 'admin' است.
  - اگر شرایط برقرار نباشد، خطای 403 (Forbidden) با یک پیام مناسب به کاربر نمایش داده می‌شود.
- **def is_safe_url(target):**
  - یک تابع کمکی برای **جلوگیری از حملات Open Redirect**.
  - وقتی کاربر پس از لاگین به پارامتر next (URL قبلی) هدایت می‌شود، این تابع بررسی می‌کند که آیا آن URL متعلق به همین اپلیکیشن (همان دامنه) است یا یک URL خارجی مخرب.

7. روت‌ها (Routes / View Functions):

- **@app.route('/') (تابع index):**
  - صفحه اصلی برنامه که لیست پست‌ها را نمایش می‌دهد.
  - نیاز به لاگین دارد (@login_required_server_session).
  - قابلیت جستجو در پست‌ها بر اساس متن ('request.args.get('q')).
  - اگر کاربر ادمین باشد، تمام پست‌ها را می‌بیند (مگر اینکه جستجو کرده باشد). کاربران عادی به طور پیش‌فرض 30 پست اخیر را می‌بینند.
- **@app.route('/my-posts') (تابع my_posts):**
  - صفحه‌ای برای نمایش پست‌های خود کاربر لاگین شده.
  - نیاز به لاگین دارد.
- **@app.route('/admin') (تابع admin_dashboard):**
  - داشبورد مدیریتی.
  - نیاز به لاگین و دسترسی ادمین دارد (@admin_required).
  - لیست تمام کاربران سیستم را نمایش می‌دهد و فرم‌هایی برای مدیریت آن‌ها (ایجاد کاربر، تغییر نقش، حذف کاربر) دارد.
- **@app.route('/admin/create_user', methods=['POST']) (تابع admin_create_user):**
  - روت برای پردازش فرم ایجاد کاربر جدید توسط ادمین.
  - داده‌ها را از فرم (request.form) می‌خواند، اعتبار سنجی می‌کند، کاربر جدید را با نقش مشخص شده ایجاد و در دیتابیس ذخیره می‌کند.
  - از flash برای نمایش پیام موفقیت یا خطا استفاده می‌کند.
- **@app.route('/admin/user/<int:user_id>/set_role', methods=['POST']) (تابع admin_set_user_role):**
  - روت برای تغییر نقش یک کاربر توسط ادمین.
  - user_id کاربری که قرار است نقشش تغییر کند از URL گرفته می‌شود.
  - رمز عبور ادمین فعلی برای تأیید عملیات از فرم خوانده می‌شود.
  - نقش جدید را در دیتابیس برای کاربر مورد نظر ذخیره می‌کند.
- **@app.route('/admin/user/<int:user_id>/delete', methods=['DELETE']) (تابع admin_delete_user):**
  - روت برای حذف یک کاربر توسط ادمین، **با استفاده از متد DELETE**.

- user_id کاربر از URL گرفته می‌شود.
- رمز عبور ادمین فعلی برای تأیید از بدنه درخواست JSON (request.get_json()) خوانده می‌شود (چون کلاینت با JavaScript درخواست DELETE را ارسال می‌کند).
- کاربر مورد نظر را از دیتابیس حذف می‌کند (پست‌ها و نشست‌هایش هم به دلیل cascade در مدل‌ها حذف می‌شوند).
- یک پاسخ JSON برای کلاینت برمی‌گرداند (که موفقیت یا شکست عملیات را نشان می‌دهد).

- **@app.route('/profile', methods=['GET', 'POST']) (تابع profile):**
  - صفحه پروفایل کاربر لاگین شده.
  - در متد GET، فرم تغییر رمز عبور و گزینه حذف حساب را نمایش می‌دهد.
  - در متد POST (فقط برای فرم تغییر رمز عبور)، اطلاعات فرم را پردازش می‌کند، رمز عبور فعلی را بررسی می‌کند و در صورت صحت، رمز جدید را تنظیم می‌کند.
  - **مهم:** هنگام تغییر رمز عبور، تمام نشست‌های دیگر این کاربر (به جز نشست فعلی) از دیتابیس حذف می‌شوند تا امنیت افزایش یابد (کاربر از دستگاه‌های دیگر لاگ‌اوت می‌شود).
- **@app.route('/profile/delete', methods=['DELETE']) (تابع delete_profile):**
  - روت برای حذف حساب کاربری توسط خود کاربر، **با استفاده از متد DELETE**.
  - رمز عبور کاربر برای تأیید از بدنه درخواست JSON خوانده می‌شود.
  - پس از بررسی‌ها، کاربر فعلی (g.user) را از دیتابیس حذف می‌کند.
  - یک پاسخ JSON (شامل URL برای ریدایرکت به صفحه لاگین) به کلاینت برمی‌گرداند.
- **@app.route('/login', methods=['GET', 'POST']) (تابع login):**
  - صفحه ورود کاربران.
  - اگر کاربر از قبل لاگین باشد، او را به صفحه اصلی هدایت می‌کند.
  - در متد GET، فرم لاگین را نمایش می‌دهد.
  - در متد POST:
    - نام کاربری و رمز عبور را از فرم می‌خواند.
    - وضعیت چک‌باکس "مرا به خاطر بسپار" (remember_me) را می‌خواند.
    - کاربر را در دیتابیس جستجو و رمز عبور را بررسی می‌کند.
    - اگر معتبر بود:
    - یک session_id تصادفی و امن با secrets.token_urlsafe() تولید می‌کند.
    - یک رکورد جدید در جدول ServerSession با user_id، session_id و وضعیت remembered ایجاد و ذخیره می‌کند.
    - یک کوکی به نام server_session_id با مقدار session_id تولید شده در مرورگر کاربر تنظیم می‌کند.
    - max_age (طول عمر) کوکی بر اساس وضعیت remember_me تنظیم می‌شود (1 روز اگر remember_me فعال باشد، در غیر این صورت مقدار PERMANENT_SESSION_LIFETIME یعنی 30 دقیقه).
    - کوکی به صورت httponly تنظیم می‌شود تا از طریق JavaScript قابل دسترسی نباشد (برای امنیت).
    - کاربر را به صفحه مناسب (صفحه next، داشبورد ادمین، یا صفحه اصلی) هدایت می‌کند.
- **@app.route('/register', methods=['GET', 'POST']) (تابع register):**

- صفحه ثبت نام کاربران جدید.
- اگر کاربر از قبل لاگین باشد، او را به صفحه اصلی هدایت می‌کند.
- در متد GET، فرم ثبت نام را نمایش می‌دهد.
- در متد POST، نام کاربری و رمز عبور را از فرم می‌خواند، بررسی می‌کند که نام کاربری تکراری نباشد، کاربر جدید را با نقش پیش‌فرض 'user' ایجاد می‌کند (رمز عبور هش می‌شود) و در دیتابیس ذخیره می‌کند. سپس کاربر را به صفحه لاگین هدایت می‌کند.

- **@app.route('/logout', methods=['POST']) (تابع logout):**
  - روت برای خروج کاربر. فقط با متد POST قابل دسترسی است (برای امنیت، چون عملیات تغییر وضعیت است).
  - نیاز به لاگین دارد.
  - رکورد نشست فعلی کاربر (g.current_session) را از جدول ServerSession در دیتابیس حذف می‌کند.
  - کوکی server_session_id را از مرورگر کاربر پاک می‌کند (با تنظیم تاریخ انقضا در گذشته).
  - کاربر را به صفحه لاگین هدایت می‌کند.
- **@app.route('/posts', methods=['POST']) (تابع submit_post):**
  - روت برای پردازش فرم ارسال پست جدید.
  - متن پست را از فرم می‌خواند و یک پست جدید برای کاربر لاگین شده (g.user) در دیتابیس ایجاد می‌کند.
  - کاربر را به صفحه اصلی (ایندکس) هدایت می‌کند.
- **@app.route('/post/<int:post_id>/delete', methods=['DELETE']) (تابع delete_post):**
  - روت برای حذف یک پست توسط نویسنده آن، **با استفاده از متد DELETE**.
  - post_id پستی که باید حذف شود از URL گرفته می‌شود.
  - بررسی می‌کند که آیا کاربر لاگین شده نویسنده پست است یا خیر.
  - پست را از دیتابیس حذف می‌کند.
  - یک پاسخ JSON به کلاینت برمی‌گرداند.
- **@app.route('/admin/post/<int:post_id>/delete', methods=['DELETE']) (تابع admin_delete_post):**
  - روت برای حذف هر پستی توسط ادمین (در کد فعلی شما، فقط ادمین با ID=1 این اجازه را دارد، که می‌توانید این شرط if g.user.id != 1: را بردارید اگر می‌خواهید همه ادمین‌ها این قابلیت را داشته باشند)، **با استفاده از متد DELETE**.
  - پست را از دیتابیس حذف می‌کند.
  - یک پاسخ JSON به کلاینت برمی‌گرداند.

8. مدیریت خطاها (@app.errorhandler):

- **@app.errorhandler(404) (تابع page_not_found):**
  - یک صفحه سفارشی (404.html) را برای خطای "صفحه یافت نشد" نمایش می‌دهد.
- **@app.errorhandler(403) (تابع forbidden):**
  - یک صفحه سفارشی (unauthorized.html) را برای خطای "دسترسی غیرمجاز" نمایش می‌دهد. پیامی که

توسط abort(403, description=...) ارسال شده، در این صفحه قابل نمایش است.

9. تابع init_db() (برای ایجاد اولیه دیتابیس):

```
Python
```

```
def init_db():
    # ... (کد تابع) ...
```

- این تابع برای ایجاد اولیه تمام جداول تعریف شده در مدل‌ها (User, Post, ServerSession) در دیتابیس استفاده می‌شود.
- همچنین یک کاربر ادمین پیش‌فرض با نام کاربری admin و رمز عبور adminpass (که **باید در محیط پروداکشن تغییر کند**) ایجاد می‌کند، اگر از قبل وجود نداشته باشد.
- این تابع به طور مستقیم در اجرای اصلی برنامه صدا زده نمی‌شود، اما در بلوک if __name__ == '__main__': کدی مشابه برای اطمینان از وجود جداول و ادمین پیش‌فرض قرار داده شده است. برای اجرای دستی، می‌توان از طریق flask shell اقدام کرد.

10. اجرای سرور (if __name__ == '__main__':)

```
Python
```

```
if __name__ == '__main__':
    with app.app_context():
        db.create_all()
        if not User.query.filter_by(username='admin').first():
            # ... (ایجاد ادمین پیش‌فرض) ...
    app.run(host='0.0.0.0', port=5000, debug=True)
```

- این بلوک کد فقط زمانی اجرا می‌شود که فایل app.py مستقیماً اجرا شود (نه وقتی که به عنوان یک ماژول import می‌شود).
- with app.app_context():: یک context اپلیکیشن ایجاد می‌کند که برای اجرای دستورات دیتابیس (db.create_all()) لازم است.
- db.create_all(): جداول را در صورت عدم وجود ایجاد می‌کند.
- بخش ایجاد ادمین پیش‌فرض نیز در اینجا تکرار شده تا در هر بار اجرای سرور توسعه، از وجود آن اطمینان حاصل شود (مناسب برای محیط توسعه).

- app.run(host='0.0.0.0', port=5000, debug=True): سرور توسعه داخلی فلسک را اجرا می‌کند.
  - host='0.0.0.0': باعث می‌شود سرور روی تمام IPهای موجود سیستم گوش دهد و از طریق شبکه محلی قابل دسترس باشد.
  - port=5000: پورتی که سرور روی آن اجرا می‌شود.
  - debug=True: حالت دیباگ را فعال می‌کند. در این حالت، تغییرات در کد به طور خودکار بارگذاری مجدد می‌شوند و خطاهای دقیق در مرورگر نمایش داده می‌شوند. **این گزینه باید در محیط پروداکشن False باشد.**

نکات کلیدی برای ارائه:

- **تفاوت با Flask-Login**: توضیح دهید که چرا تصمیم گرفتید احراز هویت و مدیریت نشست را دستی پیاده‌سازی کنید (مثلاً برای یادگیری عمیق‌تر یا نیاز به کنترل بیشتر). اشاره کنید که Flask-Login بسیاری از این کارها را ساده‌تر می‌کند اما در این پروژه هدف، پیاده‌سازی از پایه بوده است.
- **مدیریت نشست سمت سرور**: تاکید کنید که شناسه‌های نشست در کوکی کاربر ذخیره می‌شوند، اما اطلاعات اصلی نشست (مثل user_id، وضعیت remembered) در دیتابیس سمت سرور نگهداری می‌شود. این روش امن‌تر از ذخیره تمام اطلاعات در کوکی است.
- **کوکی httponly**: توضیح دهید که چرا کوکی نشست به صورت httponly تنظیم شده (برای جلوگیری از دسترسی JavaScript و کاهش خطر حملات XSS).
- **Remember Me**: نحوه عملکرد "مرا به خاطر بسپار" را توضیح دهید: هم طول عمر کوکی در مرورگر و هم طول عمر نشست در سرور بر اساس انتخاب کاربر تغییر می‌کند.
- **استفاده از متد DELETE**: توضیح دهید که چرا برای عملیات حذف از متد DELETE به جای POST استفاده شده و این کار چگونه با اصول RESTful سازگارتر است. همچنین به چالش ارسال درخواست DELETE از فرم‌های HTML و نیاز به JavaScript اشاره کنید.
- **g آبجکت**: نقش g را در نگهداری اطلاعات کاربر لاگین شده برای هر درخواست توضیح دهید.
- **دکوراتورها**: کاربرد دکوراتورهای @login_required_server_session و @admin_required را برای کنترل دسترسی به روت‌ها شرح دهید.
- **امنیت رمز عبور**: تاکید کنید که رمزهای عبور هرگز به صورت متن ساده ذخیره نمی‌شوند و از توابع هش (generate_password_hash, check_password_hash) استفاده می‌شود.
- **جلوگیری از Open Redirect**: اهمیت تابع is_safe_url را توضیح دهید.