سینا فیضی

امیر حسین پورحیدر

موضوع: Online Exam

خلاصه کوتاهی از پروژه:

این پروژه یک **سامانه آزمون آنلاین** با Django است که دو نوع کاربر دارد : استاد و دانشجو

هسته کارمون اینه که استاد آزمون و سوال می‌سازه دانشجو آزمون را می‌دهد و پاسخ‌ها ذخیره می‌شوند. و کنار این‌ها ورود/خروج، نقش‌ها، لاگ‌گیری, تایید ایمیل/پیامک هم اضافه شده.

اجزای اصلی پروژه:

۱) Accounts
- مدل کاربر سفارشی بر اساس abstractuser
- تعیین نقش: استاد یا دانشجو
- ثبت نام و ورودو خروج
- محدودسازی دسترسی بر اساس نقش

۲) Exams
- تعریف مدل(زمان شروع – نمره و ...)
- صفحات لیست ازمونا بهمراه جزییات
- استاد میتواند ازمون طرح/ویرایش/حدف کند

۳) Questions
- تعریف مدل question با نوع سوال: تشریحی/فایلی/تستی
- تعریف مدل answer برای ذخیره جواب دانشجو
- صفحه take_exam که دانشجو سوال‌ها را می‌بیند و جواب‌ها را ثبت می‌کند

امکانات سیستمی

Session management

- بعد از login ، کاربر با session شناخته می‌شود(`request.user`)
- دسترسی‌ها با `decorator` و شرط نقش کنترل می‌شوند

Logging

- لاگ مرکزی در `settings.py`
- یک middleware که همه‌request ها را log می‌کند (آدرس، کاربر، مدت زمان، status)
- لاگ‌های مهم در view ها (ساخت آزمون، ورود به آزمون، ...)

DataBase

برای پایگاه داده از postgresql استفاده کردیم. ابزار مورد استفادمون pgadmin4 بود که باهاش دیتابیس پروژمونو ساختیم و از طریق settings.py متصلش کردیم به پروژه(name, user, password, host, port) با پر کردن این اطلاعات تونستیم وصلش کنیم.

جزییات پروژه:

بخشای accounts

- **admin.py:**

دکوراتور `@admin.register(User)` مدل `User` را در پنل ادمین Djangoثبت می‌کند تا در بخش مدیریت قابل مشاهده و مدیریت باشد.

کلاس `CustomUserAdmin(UserAdmin)` برای سفارشی‌سازی نمایش و مدیریت کاربران در پنل ادمین استفاده شده است. چون این کلاس از `UserAdmin` ارث بری می‌کند، امکانات پیش‌فرض Django برای مدیریت کاربران (مثل فرم ویرایش کاربر، لیست کاربران، مجوزها و ...) حفظ می‌شود و فقط بخش‌های موردنیاز پروژه به اون اضافه میشه.

گزینه‌ی `list_display` تعیین می‌کند در صفحه‌ی لیست کاربران(جدول کاربران در admin) چه ستون‌هایی نمایش داده شوند. در این پروژه علاوه بر فیلدهای اصلی مثل

username و email، فیلدهای سفارشی مثل phone و role و همچنین وضعیت‌های مربوط به تأیید ایمیل/پیامک و سطح دسترسی (is_email_verified, is_sms_verified, is_active, is_staff) نمایش داده می‌شوند تا مدیر سیستم بتواند سریع وضعیت هر کاربر را بررسی کند.

گزینه‌ی fieldsets نحوه نمایش فیلدها در صفحه‌ی جزئیات/ویرایش کاربر را گروه‌بندی می‌کند. در اینجا ابتدا UserAdmin.fieldsets (بخش‌های پیش‌فرض Django) حفظ شده و سپس یک بخش جدید با عنوان **Role & Verification** به آن اضافه شده است که شامل فیلدهای سفارشی پروژه مثل role, phone, is_email_verified, is_sms_verified است. این کار باعث می‌شود فیلدهای مربوط به نقش و تأیید کاربر در پنل مدیریت به‌صورت مرتب و واضح نمایش داده شوند.

- **Apps.py:**

در اینجا کلاس AccountsConfig از AppConfig ارث‌بری کرده و دو تنظیم اصلی انجام می‌دهد:

- مقدار name = 'accounts' نام اپ را مشخص می‌کند Django با این نام اپ را شناسایی می‌کند و از آن برای ثبت اپ در INSTALLED_APPS، پیدا کردن مدل‌ها، فایل‌های template ، سیگنال‌ها و سایر بخش‌های مرتبط استفاده می‌کند.
- مقدار default_auto_field = 'django.db.models.BigAutoField' تعیین می‌کند اگر برای مدل‌ها کلید اصلی (Primary Key) به‌صورت دستی تعریف نشده باشد، Django به طور پیش‌فرض از نوع BigAutoField استفاده کند. این نوع فیلد برای id های بزرگ‌تر مناسب است و باعث می‌شود در آینده با افزایش تعداد رکوردها محدودیت عددی نداشته باشیم.

- **Decorators.py:**

تابع email_verified_required(view_func)
این تابع یک **decorator** می‌سازد که هدف آن جلوگیری از دسترسی کاربران به بعضی صفحات تا زمانی است که **ایمیل خود را تایید نکرده‌اند** .وقتی این دکوراتور روی یک view قرار می‌گیرد، قبل از اجرای view بررسی می‌کند آیا کاربر لاگین است و آیا فیلد is_email_verified او برابر False است یا نه. اگر کاربر وارد شده باشد ولی ایمیلش تایید نشده باشد، یک پیام هشدار با messages.warning نمایش داده می‌شود و کاربر به مسیر verify_sent هدایت می‌شود تا مراحل تایید ایمیل را انجام دهد. اگر کاربر تایید شده باشد یا اصلاً لاگین نکرده باشد، view اصلی بدون مشکل اجرا می‌شود.
استفاده از @wraps(view_func) هم باعث می‌شود نام و مشخصات اصلی view حفظ شود و Django در routing و debug به مشکل نخورد.

تابع role_required(role: str)

این تابع یک دکوراتور پارامتردار تولید می‌کند تا بتوانیم دسترسی به view ها را بر اساس نقش کاربر کنترل کنیم. ورودی این تابع مقدار نقش موردنیاز است مثلاً(student یا instructor) سپس یک decorator برمی‌گرداند که روی view اعمال می‌شود. داخل wrapped_ اول بررسی می‌شود آیا کاربر وارد سیستم شده است یا نه (.is_authenticated)اگر وارد نشده باشد به صفحه login هدایت می‌شود. اگر وارد شده باشد ولی مقدار request.user.role با نقش موردنیاز برابر نباشد، یک پیام خطا نمایش داده می‌شود و کاربر به صفحه لیست آزمون‌ها (exams:exam_list) منتقل می‌شود. اگر نقش درست باشد، view اصلی اجرا می‌گردد. این ساختار باعث می‌شود کنترل دسترسی‌ها یکپارچه و قابل استفاده مجدد باشد و لازم نباشد در تک تک view ها شرط‌های تکراری نوشته شود.

متغیر instructor_required

این متغیر نتیجه فراخوانی role_required('instructor') است و یک دکوراتور آماده می‌سازد که فقط به کاربران با نقش استاد اجازه دسترسی می‌دهد. بنابراین هر viewکه با @instructor_required تزیین شود، فقط توسط استادها قابل مشاهده/اجرا خواهد بود و کاربران دیگر اجازه ورود ندارند.

متغیر student_required

این متغیر نتیجه‌ی فراخوانی role_required('student') است و یک دکوراتور آماده می‌سازد که فقط به کاربران با نقش دانشجو اجازه دسترسی می‌دهد. بنابراین هر view که با @student_required تزیین شود، فقط برای دانشجوها قابل استفاده است و سایر نقش‌ها دسترسی نخواهند داشت.

- **forms.py:**

کلاس SignUpForm(UserCreationForm)

این فرم برای **ثبت‌نام کاربر جدید** ساخته شده و از) UserCreationFormفرم آماده‌ی Djangoبرای ساخت کاربر با (password1/password2 ارث‌بری می‌کند. با این کار، منطق بررسی رمزها (تکرار رمز، استاندارد بودن رمز و ...) به‌صورت آماده در اختیار ماست و فقط فیلدهای موردنیاز پروژه را به آن اضافه می‌کنیم. در این فرم علاوه بر فیلدهای پیش‌فرض) مثل username و دو فیلد رمز(، سه فیلد جدید هم داریمemail :، phoneو roleتا کاربر هنگام ثبت‌نام اطلاعات کامل‌تری بدهد و نقش او مشخص شود.

فیلد email = forms.EmailField(required=True)

این فیلد برای گرفتن ایمیل کاربر است و چون required=Trueدارد، کاربر بدون وارد کردن ایمیل نمی‌تواند ثبت‌نام را تکمیل کند .EmailField به صورت خودکار فرمت ایمیل را هم تا حد خوبی اعتبارسنجی می‌کند.

فیلد phone = forms.CharField(required=True, help_text="")

این فیلد شماره موبایل را به‌صورت متنی دریافت می‌کند help_text .فقط برای راهنمایی کاربر در صفحه فرم نمایش داده می‌شود تا بداند فرمت درست شماره چیست. این فیلد بعداً توسط تابع clean_phone اعتبارسنجی دقیق تر می‌شود.

فیلد role = forms.ChoiceField(choices=User.ROLE_CHOICES)

این فیلد برای انتخاب نقش کاربر است (دانشجو یا استاد). گزینه‌ها از User.ROLE_CHOICESگرفته می‌شود تا نقش‌ها فقط محدود به مقادیر معتبر تعریف‌شده در مدل باشند و مقدار اشتباه ذخیره نشود.

متد __init__(self, *args, **kwargs)

این بخش برای **زیباتر کردن فرم** و هماهنگ شدن با Bootstrap نوشته شده است. در ابتدا با super()فرم اصلی ساخته می‌شود، سپس روی تمام فیلدهای فرم حلقه می‌زند و برای هر فیلد یک کلاس CSS اضافه می‌کند:

- به طور پیش‌فرض برای همه‌ی فیلدها form-control گذاشته می‌شود تا ظاهر Bootstrapداشته باشند.
- برای فیلد role به جای آن form-select گذاشته می‌شود چون در Bootstrap برای <select> کلاس مناسب‌تر همین است.

همچنین اگر قبلاً کلاس CSS دیگری روی widget وجود داشته باشد، با existing نگه داشته می‌شود تا کلاس‌های قبلی از بین نروند و در نهایت کلاس جدید به آن اضافه می‌شود.

متد clean_phone(self)

این متد برای **اعتبارسنجی شماره موبایل** است و Django آن را به طور خودکار هنگام form.is_valid()اجرا می‌کند. کاری که انجام می‌دهد:

۱. شماره را از cleaned_data می‌گیرد و فاصله‌های اضافی را حذف می‌کند.
۲. اگر شماره با 98+ شروع شود آن را به فرمت داخلی تبدیل می‌کند.
۳. سپس بررسی می‌کند شماره حتماً با 09 شروع شود، دقیقاً ۱۱ رقم داشته باشد و تمام کاراکترها عدد باشند.
۴. اگر این شروط برقرار نباشد، با ValidationError جلوی ثبت‌نام گرفته می‌شود و پیام خطا به کاربر نشان داده می‌شود.
۵. اگر درست باشد، همان شماره استاندارد شده را return می‌کند تا در دیتابیس ذخیره شود.

کلاس داخلی Meta

این قسمت مشخص می‌کند این فرم به کدام مدل وصل است و چه فیلدهایی از مدل قرار است در این فرم نمایش داده/ذخیره شود:

- model = User یعنی این فرم برای ساخت رکورد از مدل User استفاده می‌کند.

- `fields = ("username", "email", "phone", "role")`
  یعنی در فرم ثبت‌نام، علاوه بر فیلدهای رمز که از `UserCreationForm` می‌آیند، این چهار فیلد هم نمایش داده می‌شود و در نهایت در ساخت کاربر استفاده خواهد شد.

- **:models.py**

کلاس `User(AbstractUser)`
این کلاس مدل کاربر سفارشی پروژه است و از `AbstractUser` ارث بری می‌کند تا تمام قابلیت‌های آماده‌ی Djangoبرای کاربران (مثل `username`، رمز عبور، ورود/خروج، سطح دسترسی‌ها و ...) **حفظ** شود. سپس فیلد های اختصاصی پروژه به آن اضافه شدند تا بتوانیم **نقش کاربر (دانشجو/استاد)** و وضعیت **تأیید ایمیل و تأیید پیامک** را مدیریت کنیم. این مدل پایه اصلی تشخیص نوع کاربر و محدودسازی دسترسی در کل سیستم است.

`ROLE_CHOICES`و فیلد `role`
`ROLE_CHOICES` لیست نقش های مجاز در سیستم را تعریف می‌کند (دانشجو و استاد). فیلد `role` از نوع `CharField` است و نقش هر کاربر را ذخیره میکند. وجود `choices=ROLE_CHOICES`باعث می‌شود مقدار role فقط یکی از گزینه های معتبر باشد و مقدار اشتباه در دیتابیس ذخیره نشود. همچنین `default='student'` تعیین می‌کند اگر هنگام ساخت کاربر نقشی مشخص نشود، به طور پیش فرض کاربر دانشجو در نظر گرفته شود. این فیلد در بخش های مختلف پروژه برای کنترل دسترسی ها استفاده میشود (مثلاً اجازه ساخت آزمون فقط برای استاد).

فیلد `is_email_verified`
این فیلد یک `BooleanField` است که وضعیت تأیید ایمیل کاربر را نگه میدارد. مقدار پیش فرض آن `False` است، یعنی کاربر هنوز ایمیل خود را تایید نکرده است. پس از انجام فرآیند تأیید ایمیل، مقدار این فیلد `True` میشود. این فیلد معمولاً برای محدود کردن دسترسی کاربران تا زمان تایید ایمیل به کار میرود.

فیلد `phone`
این فیلد شماره موبایل کاربر را ذخیره میکند `blank=True` .اجازه می‌دهد در فرم‌ها خالی باشد و `null=True` اجازه میدهد در دیتابیس مقدار NULL داشته باشد .
`max_length=20`هم باعث می‌شود هم شماره‌های داخلی (`09...`) و هم فرمت‌های طولانی‌تر مثل `+98...`قابل ذخیره باشند. این شماره برای ارسال کد یکبارمصرف پیامکی استفاده میشود.

فیلد `is_sms_verified`
این فیلد وضعیت تأیید پیامکی کاربر را مشخص میکند. مقدار پیش فرض `False` است و زمانی `True` می‌شود که کاربر کد پیامکی صحیح (OTP) را وارد کند و شماره موبایل او تأیید شود. این فیلد برای کنترل اینکه کاربر تایید پیامکی شده یا نه به کار می‌رود.

فیلد `sms_code`
این فیلد کد یکبارمصرف پیامکی (OTP) را ذخیره می‌کند. طول آن ۶ کاراکتر در نظر گرفته

شده تا برای کدهای ۶ رقمی مناسب باشد blank=True و null=True باعث می‌شود در حالت عادی یا بعد از تایید شدن کاربر بتوان آن را خالی کرد. این کد هنگام ارسال پیامک تولید و ذخیره می‌شود و سپس با ورودی کاربر مقایسه می‌گردد.

فیلد sms_code_created_at
این فیلد زمان تولید کد OTP را ذخیره می‌کند تا بتوانیم برای کد پیامکی محدودیت زمانی (انقضا) تعریف کنیم؛ مثلاً کد فقط چند دقیقه معتبر باشد. با استفاده از این زمان می‌توان جلوی استفاده از کدهای قدیمی را گرفت و امنیت فرآیند تأیید پیامکی را افزایش داد.

- **Sms.py:**

تابع send_otp_sms(phone: str, code: str)
این تابع برای **ارسال کد یکبارمصرف پیامکی (OTP)** به شماره موبایل کاربر استفاده می‌شود. ورودی‌های تابع شامل phone(شماره گیرنده) و code(کد تایید) هستند و خروجی آن معمولاً پاسخ وب‌سرویس کاوه‌نگار است. این تابع طوری نوشته شده که هم در حالت واقعی (ارسال پیامک با کاوه‌نگار) کار کند و هم در حالت تست/توسعه بتواند به جای ارسال واقعی، کد را در ترمینال چاپ کند.

بررسی نوع ارائه‌دهنده پیامک(SMS_PROVIDER)
در ابتدای تابع بررسی می‌شود که آیا پروژه روی حالت kavenegar تنظیم شده است یا نه:
اگر مقدار SMS_PROVIDER برابر kavenegar نباشد، تابع به جای ارسال پیامک واقعی، کد را در ترمینال چاپ می‌کند و None برمی‌گرداند. این حالت برای توسعه و تست مناسب است، چون بدون نیاز به API و هزینه پیامک، می‌توان روند OTP را تست کرد.

خواندن settings

- KAVENEGAR_API_KEY برای احراز هویت و استفاده از API کاوه‌نگار
- KAVENEGAR_SENDER برای تعیین **شماره/خط ارسال‌کننده** که باید در پنل کاوه‌نگار معتبر و فعال باشد.

این مقادیر معمولاً از فایل env. خوانده می‌شوند و در settings.pyتنظیم می‌گردند.

اعتبارسنجی وجود API Key وSender
بعد از خواندن تنظیمات، تابع بررسی می‌کند که هر دو مقدار خالی نباشند. اگر api_key تنظیم نشده باشد، خطای ValueError با پیام مشخص داده می‌شود. همینطور اگر senderتنظیم نشده باشد، خطای جداگانه داده می‌شود. این کار باعث می‌شود اگر تنظیمات ناقص باشد، برنامه سریع و واضح خطا بدهد و مشکل ساده‌تر پیدا شود.

ساخت شیء API و آماده‌سازی پارامترها
در ادامه یک شیء از کلاس KavenegarAPIساخته می‌شود تا بتوانیم به وب‌سرویس کاوه‌نگار درخواست ارسال پیامک بزنیم. سپس دیکشنری paramsساخته می‌شود که شامل اطلاعات لازم برای ارسال پیامک است:

- `Sender` خط ارسال‌کننده معتبر
- `Receptor` شماره گیرنده (همان phone)
- `Message` متن پیام که در اینجا شامل کد تایید است

ارسال پیامک و مدیریت خطاها
در بخش `try`، پیامک با `api.sms_send(params)` ارسال می‌شود و نتیجه آن `return`می‌گردد. اگر وب‌سرویس یا شبکه خطا داشته باشد، کتابخانه کاوه‌نگار معمولاً خطاهایی از نوع `APIException` یا `HTTPException` می‌دهد. این خطاها گرفته می‌شوند و به یک خطای واضح‌تر (`RuntimeError`) تبدیل می‌شوند.

- **Tests.py:**

دکوراتور `@override_settings(...)`
این دکوراتور برای این است که **تنظیمات پروژه فقط در همین فایل تست (و فقط هنگام اجرای این تست‌ها)** به صورت موقت تغییر کند.

کلاس `EmailAndSmsVerificationTests(TestCase)`
این کلاس مجموعه تست‌های مربوط به فرآیند **ثبت‌نام و تأیید ایمیل/پیامک** را پوشش می‌دهد و از `TestCase` ارث بری می‌کند تا Django یک دیتابیس تست موقت بسازد، هر تست را در یک محیط ایزوله اجرا کند و بعد از پایان تست دیتابیس را پاک کند. هدف این کلاس این است که مطمئن شویم جریان ثبت‌نام و فعال‌سازی حساب دقیقاً طبق منطق پروژه انجام می‌شود.

متد
`test_signup_creates_inactive_user_and_sends_email`
این تست بررسی می‌کند که وقتی کاربر ثبت‌نام می‌کند:

۱. درخواست POST به آدرس `signup` با اطلاعات کاربر ارسال می‌شود(username ، email، phone، role، و دو رمز عبور.)
۲. انتظار داریم بعد از ثبت‌نام، پاسخ `302` باشد که معمولاً نشان‌دهنده redirect است (یعنی ثبت‌نام موفق بوده و کاربر به صفحه دیگری هدایت شده).
۳. سپس کاربر ساخته‌شده از دیتابیس گرفته می‌شود و چند شرط مهم چک می‌شود:
   - `user.is_active`باید `False` باشد: یعنی کاربر تا قبل از تأیید ایمیل نباید فعال شود.
   - `user.is_email_verified` باید `False`باشد: یعنی هنوز ایمیل تایید نشده است.
   - `user.is_sms_verified` باید `False`باشد: یعنی هنوز پیامک تایید نشده است.
   - `user.sms_code`نباید `None` باشد: یعنی هنگام ثبت‌نام یک کد OTP تولید و برای تایید پیامکی ذخیره شده است.
   - `len(mail.outbox) == 1` یعنی دقیقاً یک ایمیل در سیستم تست ایجاد شده و "ارسال ایمیل" از نظر منطقی انجام شده است.

این تست تضمین می‌کند که ثبت‌نام درست عمل می‌کند، کاربر تا تایید ایمیل غیر فعال می‌ماند و ارسال ایمیل نیز انجام می‌شود

متد test_verify_link_activates_user
این تست بررسی می‌کند که لینک تایید ایمیل واقعاً حساب کاربر را فعال می‌کند:

۱. یک کاربر به صورت دستی در دیتابیس ساخته می‌شود که `is_active=False` و `is_email_verified=False` دارد تا دقیقاً شبیه حالت بعد از ثبت‌نام باشد.
۲. سپس برای ساخت لینک تایید، دو مقدار تولید می‌شود:
   - `uid` که با `urlsafe_base64_encode(force_bytes(user.pk))` ساخته می‌شود و شناسه کاربر را به شکل امن داخل URL قرار می‌دهد.
   - `token` که با `default_token_generator.make_token(user)` ساخته می‌شود و یک توکن امن و زمان‌دار برای تایید ایمیل است.
۳. بعد یک درخواست GET به مسیر `verify_email` ارسال می‌شود (با `uidb64` و `token`)
۴. انتظار داریم پاسخ `200` باشد یعنی صفحه تایید با موفقیت باز شده و پردازش انجام شده است.
۵. سپس با `user.refresh_from_db()` داده‌ها دوباره از دیتابیس خوانده می‌شوند و بررسی می‌شود که:
   - `user.is_active` تبدیل به `True` شده باشد (حساب فعال شده)
   - `user.is_email_verified` تبدیل به `True` شده باشد (ایمیل تایید شده)

این تست تضمین می‌کند که مکانیزم لینک تایید ایمیل درست پیاده‌سازی شده و کاربر بعد از تایید، فعال می‌شود.

- **urls.py:**

لیست urlpatterns در فایل accounts/urls.py
این بخش مسیرهای مربوط به اپ Accounts را تعریف می‌کند. یعنی هر آدرسی که وارد سایت می‌شود و به Accounts مربوط است، از طریق این لیست به view مناسب وصل می‌شود. این URLها چرخه کامل مدیریت کاربر را پوشش می‌دهند: ورود، خروج، ثبت‌نام، و مراحل تأیید ایمیل و تأیید پیامکی.

مسیر login/
این مسیر صفحه ورود را نمایش می‌دهد و عملیات لاگین را انجام می‌دهد. از `LoginView`آماده Django استفاده شده تا منطق ورود) بررسی رمز و ساخت (session استاندارد و امن باشد. همچنین `template_name='accounts/login.html'`مشخص می‌کند صفحه‌ی HTMLورود از کدام قالب لود شود. نام `login`هم برای استفاده در `reverse`و `{% url %}`به کار می‌رود.

مسیر logout/
این مسیر برای خروج کاربر است و از LogoutView آماده Django استفاده می‌کند. با خروج، sessionکاربر پاک می‌شود و کاربر از حالت لاگین خارج می‌گردد. گزینه next_page='/accounts/login/'تعیین می‌کند بعد از logout کاربر به صفحه ورود هدایت شود. نام logoutهم برای لینک دادن و ریدایرکت استفاده می‌شود.

مسیر signup/
این مسیر فرم ثبتنام را به view سفارشی پروژه (signup_view) وصل می‌کند. چون ثبت‌نام در این پروژه شامل منطق اضافه مثل دریافتrole ، تولید کد پیامک و (در صورت نیاز) فعال نشدن کاربر تا تایید است، از view اختصاصی استفاده شده است. نام signup برای فراخوانی مسیر در قالب‌ها و تست‌ها کاربرد دارد.

مسیر verify-sent/
این مسیر معمولاً یک صفحه اطلاع‌رسانی است که بعد از ثبت‌نام به کاربر نشان می‌دهد "لینک تایید ایمیل ارسال شد" یا مرحله‌ی بعدی را توضیح می‌دهد. از این صفحه زمانی استفاده می‌شود که کاربر هنوز ایمیلش را تایید نکرده است. نام مسیر verify_sent است.

مسیر verify/<uidb64>/<token>/(تأیید ایمیل)
این مسیر برای تأیید ایمیل کاربر استفاده می‌شود و معمولاً داخل لینکی قرار می‌گیرد که به ایمیل کاربر ارسال می‌شود .uidb64 شناسه کاربر است که به صورت Base64 داخل URL قرار داده می‌شود و tokenیک توکن امنیتی زمان‌دار است تا کسی نتواند لینک را حدس بزند. در verify_email_view این اطلاعات بررسی می‌شود و اگر معتبر باشد، حساب فعال و ایمیل تایید می‌گردد. نام مسیر verify_email است.

مسیر verify-pending/<uidb64>/
این مسیر صفحه/مرحله‌ای را نشان می‌دهد که کاربر هنوز در حالت "در انتظار تکمیل تایید" است. برای مثال ممکن است کاربر ایمیل را تایید کرده باشد ولی هنوز مرحله پیامک باقی مانده باشد. پارامتر uidb64برای شناسایی کاربر استفاده می‌شود. نام مسیر verify_pendingاست.

مسیر verify-sms/<uidb64>/(تأیید پیامک)
این مسیر صفحه‌ی وارد کردن کد OTP پیامک را نمایش می‌دهد. کاربر کد ارسال‌شده را وارد می‌کند و verify_sms_view صحت کد و معتبر بودن آن را بررسی می‌کند. اگر درست باشد، is_sms_verified برای کاربر True می‌شود و مرحله تایید پیامکی کامل می‌گردد. نام مسیر verify_sms است.

مسیر resend-sms/<uidb64>/ (ارسال مجدد پیامک)
این مسیر برای ارسال مجدد کد پیامکی است؛ زمانی که کاربر پیامک را دریافت نکرده یا کد منقضی شده است. در resend_sms_view معمولاً یک کد جدید تولید می‌شود و دوباره ارسال پیامک انجام می‌گیرد. نام مسیر resend_sms است.

- **Views.py:**

۱) logger = logging.getLogger("app")

- یک logger با نام "app" می‌سازد/دریافت می‌کند تا رویدادها (مثل ورود دانشجو به آزمون) در لاگ ثبت شوند.

- بعداً با logger.info(...) پیام‌های سطح info ثبت می‌شوند.

۲) تابع question_list(request)

- نمایش لیست همه سوال‌ها.

کارش:

- از مدل Question همه سوال‌ها را می‌گیرد و بر اساس id مرتب می‌کند.

- با select_related('exam') اطلاعات آزمون مربوط به هر سوال را با JOIN می‌گیرد تا تعداد کوئری‌ها کم شود.

- با prefetch_related('choices') گزینه‌های مربوط به هر سوال را نیز به شکل بهینه واکشی می‌کند تا N+1 Query رخ ندهد.

- سپس قالب questions/question_list.html را رندر می‌کند و لیست سوال‌ها را با کلید questions به قالب می‌فرستد.

خروجی میشه صفحه question_list.html با متغیر questions.

۳) تابع question_detail(request, question_id)

- نمایش جزئیات یک سوال خاص + گزینه‌های آن.

کارش:

- با get_object_or_404 سوال را از روی id پیدا می‌کند؛ اگر وجود نداشت ۴۰۴ می‌دهد.

- سپس از طریق رابطه question.choices.all() همه گزینه‌های مربوط به سوال را می‌گیرد.

- در نهایت قالب questions/question_detail.html را رندر می‌کند و question و choices را به قالب می‌فرستد.

خروجی میشه صفحه question_detail.html با متغیرهای question و choices.

۴) تابع question_create(request)

دسترسی:

- فقط کاربر لاگین کرده
- ایمیل تایید شده
- نقش استاد (instructor_required)

هدف:

- ساخت (Create) یک سوال جدید توسط استاد.

کارش:

- اگر درخواست POST باشد یعنی فرم ارسال شده:
- فرم QuestionForm با داده‌های POST و همچنین user=request.user ساخته می‌شود.
- اگر فرم معتبر باشد:
- form.save() سوال را ذخیره می‌کند.
- پیام موفقیت نمایش داده می‌شود.
- سپس کاربر به صفحه جزئیات همان سوال redirect می‌شود.
- اگر درخواست GET باشد:
- فرم خالی QuestionForm به کاربر نمایش داده می‌شود.
- در پایان قالب questions/question_form.html با متغیر form رندر می‌شود.

۵) تابع question_create_for_exam(request, exam_id)

دسترسی:

- فقط استاد لاگین کرده + ایمیل تایید شده

هدف:

- ساخت سوال جدید برای یک آزمون مشخص (exam_id).

کارش:

- ابتدا با get_object_or_404 آزمون را می‌گیرد به شرط اینکه instructor آن همان request.user باشد.

(یعنی استاد فقط برای آزمون‌های خودش می‌تواند سوال اضافه کند.)

- اگر POST باشد:

- فرم ساخته می‌شود.

- اگر فرم معتبر باشد:

- با form.save(commit=False) سوال را موقت می‌سازد (بدون ذخیره در DB).

- سپس q.exam = exam را به صورت اجباری تنظیم می‌کند (force exam).

- بعد q.save() را انجام می‌دهد تا سوال ذخیره شود.

- پیام موفقیت نمایش می‌دهد و به صفحه جزئیات آزمون redirect می‌کند.

- اگر GET باشد:

- فرم با initial={'exam': exam} نمایش داده می‌شود تا آزمون پیش‌فرض پر باشد.

- در پایان قالب questions/question_form.html رندر می‌شود.

۶) تابع take_exam(request, exam_id)

دسترسی:

- فقط دانشجو لاگین کرده + ایمیل تایید شده (student_required)

هدف:

- دانشجو آزمون را باز کند، پاسخ‌ها را وارد کند و ارسال کند.

- پاسخ‌ها در جدول Answer ذخیره یا آپدیت می‌شوند.

مراحل اجرا:

الف) شروع:

- با logger.info ورود دانشجو به آزمون و exam_id ثبت می‌شود.
- آزمون از روی exam_id گرفته می‌شود.
- سوال‌های آزمون از طریق exam.questions.all() گرفته و بر اساس id مرتب می‌شوند.

ب) حالت POST (ارسال پاسخ‌ها):
- برای هر سوال یک نام فیلد می‌سازد: "question_<id> = field_name"
(مثلاً question_5)
- بسته به نوع سوال، پاسخ را از request.POST یا request.FILES می‌خواند:

۱) اگر نوع سوال 'mcq' (چندگزینه‌ای) باشد:
- selected_choice_id از POST گرفته می‌شود.
- اگر آی‌دی وجود داشت، تلاش می‌کند Choice مربوط به همان سوال را پیدا کند.
- اگر گزینه نامعتبر باشد (وجود ندارد یا برای این سوال نیست) choice = None می‌شود.
- سپس با Answer.objects.update_or_create:
- اگر قبلاً پاسخ این دانشجو برای این سوال ثبت شده باشد، آپدیت می‌کند.
- اگر نباشد، یک Answer جدید می‌سازد.
- در defaults مقدار selected_choice تنظیم می‌شود و text و file_upload خالی می‌شوند.

۲) اگر نوع سوال 'short' (پاسخ کوتاه) باشد:
- متن پاسخ از POST گرفته می‌شود و strip می‌شود.
- فقط اگر متن خالی نباشد، update_or_create انجام می‌دهد:
- text را ذخیره می‌کند
- selected_choice و file_upload را None می‌گذارد.

۳) اگر نوع سوال 'file' (پاسخ فایل) باشد:
- فایل از request.FILES گرفته می‌شود.

- اگر فایل موجود بود، update_or_create انجام می‌دهد:

- file_upload را ذخیره می‌کند

- text را خالی و selected_choice را None می‌گذارد.

ج) بعد از ذخیره همه پاسخ‌ها:

- پیام موفقیت "پاسخ‌ها ارسال شد." نمایش داده می‌شود.

- صفحه questions/exam_submitted.html رندر می‌شود (یعنی بعد از ارسال پاسخ‌ها دیگر به take_exam برنمی‌گردد).

د) حالت GET (اولین ورود به صفحه آزمون):

- قالب questions/take_exam.html رندر می‌شود و exam و questions به قالب داده می‌شود

- قالب باید فیلدهای ورودی را با نام question_<id> بسازد تا در POST قابل خواندن باشد.

- **Templates\accounts:**

templates/accounts/login.html

کارش اینه که صفحه ورود کاربر رو نمایش بده.

کاری که انجام میدهد اینه که فرم ورود (form) رو با csrf_token رندر می‌کنه، دکمه ورود می‌ذاره و یک لینک به صفحه ثبت‌نام هم می‌ده.

templates/accounts/signup.html

کارش اینه که صفحه ثبت‌نام کاربر جدید رو نمایش بده.

کاری که انجام میدهد اینه که فرم ثبت‌نام (form) رو با csrf_token رندر می‌کنه، دکمه ثبت‌نام می‌ذاره و یک لینک به صفحه ورود هم می‌ده.

templates/accounts/verify_email.txt

کارش اینه که متن ایمیل فعال‌سازی حساب رو بسازه.

کاری که انجام میدهد اینه که نام کاربری (user.username) و لینک فعال‌سازی (verify_link) رو داخل متن قرار می‌ده تا برای کاربر ایمیل شود.

templates/accounts/verify_sent.html

کارش اینه که بعد از ارسال لینک فعال‌سازی، به کاربر پیام بده که ایمیل ارسال شده.

کاری که انجام میدهد اینه که یک پیام ساده نمایش می‌ده که لینک فعال‌سازی ارسال شده و کاربر باید ایمیلش رو چک کنه.

templates/accounts/verify_pending.html

کارش اینه که صفحه انتخاب روش فعال‌سازی حساب رو نشان بده (ایمیل یا پیامک).

کاری که انجام میدهد اینه که وضعیت "یک قدم تا فعال‌سازی" رو نمایش می‌ده، توضیح می‌ده لینک ایمیل ارسال شده، و برای روش پیامکی دو لینک می‌سازه:

یکی برای وارد کردن کد پیامک با آدرس verify_sms و یکی برای ارسال مجدد کد با آدرس resend_sms.

همچنین زمان اعتبار کد را با ttl (به دقیقه) نشان می‌دهد و uidb64 را در لینک‌ها استفاده می‌کند.

templates/accounts/verify_sms.html

کارش اینه که صفحه وارد کردن کد پیامکی (OTP) رو نمایش بده.

کاری که انجام میدهد اینه که یک فرم POST برای وارد کردن کد ۶ رقمی می‌سازه (name="code")، csrf_token دارد، ttl را برای اعتبار کد نمایش می‌دهد، لینک برگشت به verify_pending می‌دهد و یک گزینه ارسال مجدد کد هم دارد (resend_sms) که از uidb64 استفاده می‌کند.

templates/accounts/verify_success.html

کارش اینه که بعد از تایید موفق، پیام موفقیت فعال‌سازی رو نمایش بده.

کاری که انجامدهد اینه که اعلام می‌کنه حساب فعال شده و یک لینک مستقیم به صفحه ورود (login) می‌گذارد.

templates/accounts/verify_failed.html

کارش اینه که اگر لینک فعال‌سازی نامعتبر یا منقضی بود، پیام خطا بده.

کاری که انجام میدهد اینه که پیام "لینک معتبر نیست" نمایش می‌دهد و یک لینک برای ساخت حساب جدید (signup) می‌گذارد.

بخشای exams

- **admin.py**

کلاس ExamAdmin

کارش اینه که نحوه نمایش و مدیریت مدل Exam رو داخل پنل ادمین Django مشخص کنه.

کاری که انجام میدهد اینه که:

- با @admin.register(Exam) مدل Exam رو در پنل ادمین ثبت می‌کند تا ادمین بتواند آزمون‌ها را مدیریت کند.

- list_display تعیین می‌کند که در لیست آزمون‌ها داخل ادمین، ستون‌های عنوان آزمون، موضوع، نمره کل، زمان شروع، مدت زمان آزمون (به دقیقه) و استاد برگزارکننده نمایش داده شوند.

- search_fields امکان جستجو در لیست آزمون‌ها را فراهم می‌کند؛ جستجو می‌تواند بر اساس عنوان آزمون، موضوع آزمون یا نام کاربری استاد انجام شود.

- list_filter یک فیلتر کناری در پنل ادمین ایجاد می‌کند که اجازه می‌دهد آزمون‌ها بر اساس موضوع (subject) فیلتر شوند.

- **Apps.py:**

ExamsConfig

کارش اینه که تنظیمات اپلیکیشن exams رو برای Django مشخص کنه.

کاری که انجام میدهد اینه که نوع پیش‌فرض فیلد کلید اصلی مدل‌ها رو BigAutoField قرار می‌دهد تا id ها به صورت عددی بزرگ ساخته شوند.

همچنین نام اپلیکیشن را برابر با 'exams' تعریف می‌کند تا Django این اپ را با این نام بشناسد.

- Forms.py:

ExamForm

کارش اینه که فرم ساخت و ویرایش آزمون رو بر اساس مدل Exam ایجاد کنه.

کاری که انجام میدهد اینه که از ModelForm استفاده می‌کند تا فیلدهای فرم مستقیماً از مدل Exam ساخته شوند.

در بخش Meta:

کارش اینه که مشخص کند این فرم به کدام مدل وصل است و چه فیلدهایی باید در فرم نمایش داده شوند.

مدل فرم Exam است و فیلدهای عنوان آزمون، موضوع، نمره کل، زمان شروع و مدت زمان آزمون در فرم قرار می‌گیرند.

:widgets

کارش اینه که ظاهر و نوع ورودی هر فیلد در قالب HTML کنترل شود.

برای title و subject از TextInput با کلاس form-control و placeholder مناسب استفاده شده تا ظاهر فرم زیباتر شود.

برای total_score از NumberInput استفاده شده و حداقل مقدار آن صفر در نظر گرفته شده است.

برای start_time از DateTimeInput با نوع datetime-local استفاده شده تا کاربر بتواند تاریخ و زمان را با انتخاب‌گر مرورگر وارد کند.

برای duration_minutes از NumberInput با حداقل مقدار ۱ استفاده شده تا مدت زمان آزمون معتبر باشد.

- **:Models.py**

ExamQuerySet

کارش اینه که مجموعه‌ای از فیلترها و کوئری‌های سفارشی برای مدل Exam تعریف کنه.

کاری که انجام میدهد اینه که منطق فیلتر کردن آزمون‌ها را از خود Viewها جدا می‌کند و در سطح QuerySet قرار می‌دهد.

متد by_instructor

کارش اینه که فقط آزمون‌هایی را برگرداند که توسط یک استاد مشخص ساخته شده‌اند.

متد upcoming

کارش اینه که آزمون‌هایی را برگرداند که زمان شروع آن‌ها هنوز نرسیده است.

این کار با مقایسه start_time با زمان فعلی سیستم انجام می‌شود و خروجی بر اساس زمان شروع به صورت صعودی مرتب می‌شود.

متد past

کارش اینه که آزمون‌هایی را برگرداند که زمان شروع آن‌ها گذشته است.

این آزمون‌ها بر اساس زمان شروع به صورت نزولی مرتب می‌شوند.

ExamManager

کارش اینه که Manager سفارشی برای مدل Exam فراهم کند تا متدهای QuerySet از طریق Exam.objects قابل استفاده باشند.

متد get_queryset

کارش اینه که به جای QuerySet پیش‌فرض Django، یک ExamQuerySet برگرداند تا متدهای سفارشی روی آن در دسترس باشند.

متد by_instructor

کارش اینه که متد by_instructor تعریف‌شده در ExamQuerySet را از طریق Manager در اختیار قرار دهد.

متد upcoming

کارش اینه که متد upcoming مربوط به ExamQuerySet را در سطح Manager قابل استفاده کند.

متد past

کارش اینه که متد past مربوط به ExamQuerySet را در سطح Manager قابل استفاده کند.

مدل Exam

کارش اینه که ساختار جدول آزمون‌ها را در پایگاه داده تعریف کند.

فیلد title

کارش اینه که عنوان آزمون را ذخیره کند.

فیلد subject

کارش اینه که موضوع آزمون را ذخیره کند.

فیلد total_score

کارش اینه که نمره کل آزمون را ذخیره کند.

فیلد start_time

کارش اینه که زمان شروع آزمون را ذخیره کند.

فیلد duration_minutes

کارش اینه که مدت زمان آزمون را بر حسب دقیقه ذخیره کند.

فیلد instructor

کارش اینه که استاد برگزارکننده آزمون را مشخص کند.

این فیلد به مدل کاربر متصل است و اگر استاد حذف شود، آزمون‌های او هم حذف می‌شوند.

related_name='exams_created' اجازه می‌دهد از سمت کاربر به آزمون‌های ساخته‌شده توسط او دسترسی داشته باشیم.

objects = ExamManager()

کارش اینه که به جای Manager پیش‌فرض، از Manager سفارشی استفاده شود تا متدهای by_instructor، upcoming و past در دسترس باشند.

متد __str__

کارش اینه که نمایش متنی هر آزمون (مثلاً در پنل ادمین) برابر با عنوان آزمون باشد.

- **Tests.py:**

ExamPermissionTests

کارش اینه که مجوزهای دسترسی و رفتارهای مربوط به بخش آزمون‌ها رو با تست‌های خودکار بررسی کنه.

این کلاس از TestCase ارث‌بری می‌کند تا تست‌ها روی دیتابیس تستی Django اجرا شوند.

متد setUp

کارش اینه که قبل از اجرای هر تست، داده‌های اولیه مورد نیاز را بسازد.

در این متد دو کاربر ایجاد می‌شود:

یک کاربر با نقش instructor که فعال است و ایمیلش تایید شده.

یک کاربر با نقش student که فعال است و ایمیلش تایید شده.

این کاربران در تست‌های بعدی برای بررسی سطح دسترسی استفاده می‌شوند.

test_instructor_can_access_exam_create

کارش اینه که بررسی کند استاد اجازه دسترسی به صفحه ساخت آزمون را دارد.

کاری که انجام میدهد اینه که با اطلاعات استاد وارد سیستم می‌شود،

سپس آدرس exam_create را درخواست می‌دهد

و انتظار دارد کد وضعیت پاسخ ۲۰۰ باشد که یعنی دسترسی مجاز است.

test_student_cannot_access_exam_create

کارش اینه که بررسی کند دانشجو اجازه دسترسی به صفحه ساخت آزمون را ندارد.

کاری که انجام میدهد اینه که با اطلاعات دانشجو وارد سیستم می‌شود،

سپس همان آدرس exam_create را درخواست می‌دهد

و انتظار دارد کد وضعیت پاسخ ۳۰۲ باشد که یعنی کاربر ریدایرکت شده و دسترسی مستقیم ندارد.

test_exam_manager_by_instructor

کارش اینه که عملکرد متد by_instructor در ExamManager را تست کند.

کاری که انجام میدهد اینه که دو آزمون ایجاد می‌کند:

یکی با استاد اول

یکی با کاربر دانشجو

سپس بررسی می‌کند که Exam.objects.by_instructor(self.instructor)

فقط یک آزمون را برگرداند که مربوط به همان استاد باشد.

- **Urls.py:**

این فایل مربوط به تنظیم مسیرهای (URL patterns) اپلیکیشن exams است.

'app_name = 'exams

کارش اینه که یک namespace برای آدرس‌های این اپلیکیشن تعریف کند.

این کار باعث می‌شود هنگام reverse کردن URLها بتوان از exams:exam_list، exams:exam_create و ... استفاده کرد

و تداخلی با URLهای اپلیکیشن‌های دیگر ایجاد نشود.

path('', views.exam_list, name='exam_list')

کارش اینه که آدرس اصلی بخش آزمون‌ها را به تابع exam_list وصل کند.

وقتی کاربر وارد مسیر /exams/ می‌شود، لیست آزمون‌ها به او نمایش داده می‌شود.

path('create/', views.exam_create, name='exam_create')

کارش اینه که مسیر ساخت آزمون جدید را تعریف کند.

وقتی کاربر (استاد) وارد /exams/create/ می‌شود، صفحه ایجاد آزمون نمایش داده می‌شود.

path('<int:exam_id>/', views.exam_detail, name='exam_detail')

کارش اینه که مسیر نمایش جزئیات یک آزمون خاص را مشخص کند.

عدد exam_id از URL گرفته می‌شود و برای پیدا کردن آزمون موردنظر در تابع exam_detail استفاده می‌شود.

- **Views.py:**

logger = logging.getLogger("app")

کارش اینه که یک logger با نام app بسازد تا رویدادهای مهم مربوط به آزمون‌ها ثبت شوند.

تابع exam_create

کارش اینه که امکان ساخت آزمون جدید را برای استاد فراهم کند.

این تابع فقط برای کاربری قابل دسترسی است که لاگین کرده، ایمیلش تایید شده و نقش استاد دارد.

کاری که انجام میدهد اینه که:

اگر درخواست از نوع POST باشد یعنی فرم ساخت آزمون ارسال شده است.

در این حالت یک لاگ ثبت می‌شود که نشان می‌دهد استاد وارد مرحله ساخت آزمون شده است.

سپس فرم ExamForm با داده‌های ارسال‌شده ساخته می‌شود.

اگر فرم معتبر باشد، با save(commit=False) یک شیء آزمون بدون ذخیره در پایگاه داده ساخته می‌شود.

بعد استاد جاری سیستم به عنوان instructor روی آزمون ست می‌شود.

سپس آزمون ذخیره می‌شود و کاربر به صفحه لیست آزمون‌ها هدایت می‌شود.

اگر درخواست از نوع GET باشد:

فرم خالی ExamForm ساخته می‌شود تا استاد بتواند اطلاعات آزمون را وارد کند

در نهایت قالب exams/exam_form.html با متغیر form رندر می‌شود.

تابع exam_list

کارش اینه که لیست همه آزمون‌ها را نمایش دهد.

کاری که انجام میدهد اینه که:

همه آزمون‌ها از پایگاه داده گرفته می‌شوند و بر اساس زمان شروع به صورت نزولی مرتب می‌شوند

یعنی آزمون‌های جدیدتر در بالای لیست قرار می‌گیرند.

سپس یک لاگ ثبت می‌شود که نشان می‌دهد چه کاربری (یا کاربر ناشناس) صفحه لیست آزمون‌ها را مشاهده کرده است.

در نهایت قالب exams/exam_list.html با متغیر exams رندر می‌شود.

تابع exam_detail

کارش اینه که جزئیات یک آزمون خاص را نمایش دهد.

کاری که انجام میدهد اینه که:

آزمون موردنظر با استفاده از exam_id گرفته می‌شود و اگر وجود نداشته باشد خطای ۴۰۴ برگردانده می‌شود.

سپس همه سوال‌های مربوط به آن آزمون از طریق رابطه exam.questions گرفته می‌شوند

و بر اساس id مرتب می‌شوند.

در نهایت قالب exams/exam_detail.html با متغیرهای exam و questions رندر می‌شود.

- **Templates\exams:**

templates/exams/exam_form.html

کارش اینه که فرم ساخت آزمون جدید رو به استاد نمایش بده.

کاری که انجام میدهد اینه که فرم ExamForm رو رندر می‌کند، فیلدهای عنوان، موضوع، نمره کل، زمان شروع و مدت زمان آزمون را نشان می‌دهد،

csrf_token دارد تا ارسال فرم امن باشد

و با ارسال فرم به صورت POST، اطلاعات آزمون برای ذخیره شدن به view مربوطه فرستاده می‌شود.

templates/exams/exam_list.html

کارش اینه که لیست همه آزمون‌ها رو نمایش بده.

کاری که انجام میدهد اینه که روی متغیر exams که از view گرفته شده حلقه می‌زند

و برای هر آزمون اطلاعاتی مثل عنوان، موضوع، زمان شروع و استاد برگزارکننده را نمایش می‌دهد.

معمولاً لینک ورود به صفحه جزئیات هر آزمون هم در این لیست قرار می‌گیرد.

templates/exams/exam_detail.html

کارش اینه که جزئیات یک آزمون خاص را نمایش بده.

کاری که انجام میدهد اینه که اطلاعات آزمون انتخاب‌شده (مثل عنوان، موضوع، زمان و مدت) را نشان می‌دهد

و سپس لیست سوال‌های مربوط به آن آزمون را از متغیر questions نمایش می‌دهد.

این صفحه معمولاً نقطه اتصال به بخش سوال‌ها یا شرکت در آزمون است.

بخشای questions

- **Apps.py**

QuestionsConfig

کارش اینه که تنظیمات اپلیکیشن questions رو برای Django مشخص کنه.

کاری که انجام میدهد اینه که نوع پیش‌فرض فیلد کلید اصلی مدل‌های این اپلیکیشن را BigAutoField قرار می‌دهد

تا شناسه‌ها به صورت عددی بزرگ ساخته شوند.

همچنین نام اپلیکیشن را برابر با 'questions' تعریف می‌کند

تا Django این اپ را با این نام بشناسد و بارگذاری کند.

- **Forms.py**

QuestionForm

کارش اینه که فرم ساخت و ویرایش سؤال را بر اساس مدل Question ایجاد کند.

متد __init__

کارش اینه که هنگام ساخته شدن فرم، رفتار آن را بر اساس کاربر تغییر دهد.

کاری که انجام میدهد اینه که اگر کاربر ارسال شده باشد

و نقش کاربر instructor باشد،

لیست آزمون‌های قابل انتخاب در فیلد exam را محدود می‌کند
به آزمون‌هایی که همان استاد ساخته است.
این کار باعث می‌شود استاد نتواند برای آزمون‌های دیگران سؤال ثبت کند.

بخش Meta در QuestionForm
کارش اینه که مشخص کند فرم به کدام مدل متصل است
و چه فیلدهایی باید در فرم نمایش داده شوند.
فیلدهای exam، متن سؤال، نوع سؤال و نمره سؤال در فرم قرار می‌گیرند.

widgets در QuestionForm
کارش اینه که نوع و ظاهر ورودی‌های فرم در HTML مشخص شود.
فیلد exam به صورت لیست انتخابی (select) نمایش داده می‌شود.
فیلد text به صورت textarea با چند سطر برای وارد کردن متن سؤال نمایش داده می‌شود.
فیلد question_type به صورت select برای انتخاب نوع سؤال نمایش داده می‌شود.
فیلد score به صورت ورودی عددی با حداقل مقدار ۱ نمایش داده می‌شود.

ChoiceForm
کارش اینه که فرم ساخت و ویرایش گزینه‌های سؤال را بر اساس مدل Choice ایجاد کند.

بخش Meta در ChoiceForm
کارش اینه که مشخص کند فرم به کدام مدل متصل است
و چه فیلدهایی باید در فرم نمایش داده شوند.
فیلدهای سؤال مربوطه، متن گزینه و صحیح یا غلط بودن گزینه در فرم قرار می‌گیرند.

widgets در ChoiceForm
کارش اینه که ظاهر فیلدهای فرم گزینه را مشخص کند.
فیلد question به صورت select نمایش داده می‌شود.

فیلد text به صورت ورودی متنی نمایش داده می‌شود.

فیلد is_correct به صورت checkbox برای مشخص کردن گزینه صحیح استفاده می‌شود.

- **Models.py**

QuestionQuerySet

کارش اینه که فیلترها و کوئری‌های پرکاربرد مربوط به سوال‌ها را یک‌جا تعریف کند تا در کل پروژه قابل استفاده باشند.

متد for_exam

کارش اینه که فقط سوال‌هایی را برگرداند که مربوط به یک آزمون مشخص هستند.

متد mcq

کارش اینه که فقط سوال‌های چندگزینه‌ای را برگرداند.

متد short

کارش اینه که فقط سوال‌های پاسخ کوتاه را برگرداند.

متد file

کارش اینه که فقط سوال‌های فایل‌دار را برگرداند.

QuestionManager

کارش اینه که متدهای QuestionQuerySet را از طریق Question.objects در دسترس قرار دهد.

متد get_queryset

کارش اینه که به جای QuerySet پیش‌فرض Django، یک QuestionQuerySet برگرداند تا متدهای سفارشی روی آن وجود داشته باشند.

متد for_exam

کارش اینه که متد for_exam را از طریق Manager قابل استفاده کند.

متد mcq

کارش اینه که متد mcq را از طریق Manager قابل استفاده کند.

نکته مهم اینه که برای short و file در Manager متدی تعریف نشده و فقط در QuerySet هست.

AnswerQuerySet

کارش اینه که فیلترها و کوئری‌های پرکاربرد مربوط به پاسخ‌ها را یک‌جا تعریف کند.

متد for_student

کارش اینه که فقط پاسخ‌های یک دانشجوی مشخص را برگرداند.

متد for_exam

کارش اینه که پاسخ‌هایی را برگرداند که مربوط به یک آزمون مشخص هستند.

این کار با مسیر question__exam انجام می‌شود یعنی از Answer به Question و بعد به Exam می‌رود.

AnswerManager

کارش اینه که متدهای AnswerQuerySet را از طریق Answer.objects در دسترس قرار دهد.

متد get_queryset

کارش اینه که به جای QuerySet پیش‌فرض Django، یک AnswerQuerySet برگرداند.

متد for_student

کارش اینه که متد for_student را در سطح Manager قابل استفاده کند.

متد for_exam

کارش اینه که متد for_exam را در سطح Manager قابل استفاده کند.

مدل Question

کارش اینه که اطلاعات هر سوال را در دیتابیس ذخیره کند.

QUESTION_TYPES

کارش اینه که نوع‌های مجاز برای سوال را تعریف کند.

سه نوع سوال دارد: پاسخ کوتاه، چندگزینه‌ای و فایل‌دار.

فیلد exam

کارش اینه که مشخص کند هر سوال متعلق به کدام آزمون است.

با related_name='questions' می‌توان از سمت آزمون به سوال‌ها با exam.questions دسترسی داشت.

اگر آزمون حذف شود سوال‌هایش هم حذف می‌شوند.

فیلد text

کارش اینه که متن سوال را ذخیره کند.

فیلد question_type

کارش اینه که نوع سوال را مشخص کند و فقط یکی از گزینه‌های QUESTION_TYPES را می‌پذیرد.

فیلد score

کارش اینه که نمره هر سوال را نگه دارد و مقدار پیش‌فرض آن ۱ است.

objects = QuestionManager()

کارش اینه که Manager سفارشی فعال شود تا متدهای for_exam و mcq در دسترس باشند.

متد __str__ در Question

کارش اینه که نمایش متنی سوال را کوتاه و قابل فهم کند و ۵۰ کاراکتر اول متن سوال را نشان دهد.

مدل Choice

کارش اینه که گزینه‌های مربوط به سوال‌های چندگزینه‌ای را ذخیره کند.

فیلد question

کارش اینه که هر گزینه را به یک سوال وصل کند.

با related_name='choices' می‌توان از سمت سوال به گزینه‌ها با question.choices دسترسی داشت.

اگر سوال حذف شود گزینه‌هایش هم حذف می‌شوند.

فیلد text

کارش اینه که متن گزینه را ذخیره کند.

فیلد is_correct

کارش اینه که مشخص کند این گزینه جواب درست است یا نه.

پیش‌فرض آن False است.

متد __str__ در Choice

کارش اینه که نمایش متنی گزینه را همان متن گزینه قرار دهد.

مدل Answer

کارش اینه که پاسخ دانشجو به هر سوال را ذخیره کند.

این مدل طوری طراحی شده که هر دانشجو برای هر سوال فقط یک پاسخ داشته باشد و اگر دوباره ارسال کند آپدیت شود.

فیلد question

کارش اینه که پاسخ را به سوال مربوطه وصل کند.

با related_name='answers' می‌توان از سمت سوال به همه پاسخ‌ها با question.answers دسترسی داشت.

فیلد student

کارش اینه که پاسخ را به کاربری که دانشجو است وصل کند.

با related_name='answers' می‌توان از سمت کاربر به پاسخ‌ها با user.answers دسترسی داشت.

اگر کاربر حذف شود پاسخ‌هایش هم حذف می‌شوند.

فیلد text

کارش اینه که متن پاسخ را برای سوال‌های پاسخ کوتاه نگه دارد.

blank=True و null=True یعنی می‌تواند خالی باشد.

فیلد file_upload

کارش اینه که فایل پاسخ را برای سوال‌های فایل‌دار نگه دارد.

upload_to='answers/' یعنی فایل‌ها داخل پوشه answers ذخیره می‌شوند.

blank=True و null=True یعنی می‌تواند خالی باشد.

فیلد selected_choice

کارش اینه که گزینه انتخاب‌شده دانشجو را برای سوال‌های چندگزینه‌ای نگه دارد.

on_delete=models.SET_NULL یعنی اگر گزینه حذف شود مقدار این فیلد null می‌شود و خود Answer حذف نمی‌شود.

blank=True و null=True یعنی می‌تواند خالی باشد.

فیلد submitted_at

کارش اینه که زمان ثبت پاسخ را ذخیره کند و هنگام ایجاد رکورد به صورت خودکار پر می‌شود.

objects = AnswerManager()

کارش اینه که Manager سفارشی فعال شود تا بتوان پاسخ‌ها را بر اساس دانشجو یا آزمون فیلتر کرد.

Meta unique_together = ('question', 'student')

کارش اینه که تضمین کند هر دانشجو برای هر سوال فقط یک پاسخ داشته باشد.

یعنی در دیتابیس اجازه نمی‌دهد دو Answer با question یکسان و student یکسان ساخته شود.

متد __str__ در Answer

کارش اینه که نمایش متنی پاسخ را قابل فهم کند و نشان دهد این پاسخ مربوط به کدام دانشجو و کدام سوال است.

property به نام auto_score

کارش اینه که نمره‌دهی خودکار را فقط برای سوال‌های چندگزینه‌ای انجام دهد.

اگر نوع سوال mcq باشد و دانشجو یک گزینه انتخاب کرده باشد:

اگر گزینه درست باشد نمره کامل سوال برگردانده می‌شود.

اگر گزینه غلط باشد نمره صفر برگردانده می‌شود.

برای بقیه نوع سوال‌ها همیشه صفر برمی‌گرداند.

متد clean

کارش اینه که اعتبارسنجی سطح مدل را انجام دهد تا پاسخ با نوع سوال سازگار باشد.

برای سوال mcq باید selected_choice حتماً پر باشد وگرنه خطا می‌دهد.

برای سوال short باید text حتماً پر باشد وگرنه خطا می‌دهد.

برای سوال file باید file_upload حتماً پر باشد وگرنه خطا می‌دهد.

این اعتبارسنجی باعث می‌شود پاسخ ناقص یا اشتباه در دیتابیس ذخیره نشود.

- **Tests.py**

QuestionAndTakeExamTests

کارش اینه که چند بخش مهم از اپ questions را با تست خودکار بررسی کند.

این تست‌ها هم دسترسی به شرکت در آزمون را چک می‌کنند و هم عملکرد Manager سوال‌ها را.

متد setUp

کارش اینه که قبل از اجرای هر تست، داده‌های اولیه لازم را بسازد.

کاری که انجام میدهد اینه که:

یک کاربر استاد با نقش instructor می‌سازد که فعال است و ایمیلش تایید شده است.

یک کاربر دانشجو با نقش student می‌سازد که فعال است و ایمیلش تایید شده است.

یک آزمون ایجاد می‌کند که استاد آن همان instructor ساخته‌شده است.

یک سوال برای همان آزمون ایجاد می‌کند که نوع آن short است و نمره ۱ دارد.

test_student_can_take_exam

کارش اینه که بررسی کند دانشجو اجازه ورود به صفحه شرکت در آزمون را دارد.

کاری که انجام میدهد اینه که:

دانشجو لاگین می‌کند.

آدرس take_exam را با exam_id مربوط به آزمون ساخته‌شده درخواست می‌دهد.

انتظار دارد status_code برابر ۲۰۰ باشد یعنی صفحه برای دانشجو قابل دسترسی است.

test_instructor_cannot_take_exam

کارش اینه که بررسی کند استاد اجازه شرکت در آزمون (صفحه take_exam) را ندارد.

کاری که انجام میدهد اینه که:

استاد لاگین می‌کند.

همان آدرس take_exam را درخواست می‌دهد.

انتظار دارد status_code برابر ۳۰۲ باشد یعنی استاد ریدایرکت شده و دسترسی مستقیم ندارد.

test_question_manager_for_exam

کارش اینه که بررسی کند متد for_exam در QuestionManager درست کار می‌کند.

کاری که انجام میدهد اینه که:

با Question.objects.for_exam(self.exam) سوال‌های مربوط به همان آزمون را می‌گیرد

و انتظار دارد تعداد آن‌ها ۱ باشد چون در setUp فقط یک سوال برای این آزمون ساخته شده است.

- **Urls.py**

'app_name = 'questions

کارش اینه که یک namespace برای آدرس‌های این اپلیکیشن تعریف کند.

این کار باعث می‌شود هنگام reverse کردن URLها

از questions:question_list، questions:take_exam و ... استفاده شود

و تداخلی با URLهای اپلیکیشن‌های دیگر ایجاد نشود.

path('', views.question_list, name='question_list')

کارش اینه که مسیر اصلی بخش سوال‌ها را تعریف کند.

با ورود به مسیر /questions/ لیست همه سوال‌ها نمایش داده می‌شود.

path('create/', views.question_create, name='question_create')

کارش اینه که مسیر ساخت سوال جدید را مشخص کند.

این مسیر برای استاد استفاده می‌شود تا بتواند سوال جدید ایجاد کند.

path('create/<int:exam_id>/', views.question_create_for_exam, name='question_create_for_exam')

کارش اینه که مسیر ساخت سوال برای یک آزمون مشخص را تعریف کند.

exam_id از آدرس گرفته می‌شود و سوال مستقیماً به همان آزمون متصل می‌شود.

path('take/<int:exam_id>/', views.take_exam, name='take_exam')

کارش اینه که مسیر شرکت دانشجو در آزمون را مشخص کند.

دانشجو با ورود به این مسیر می‌تواند سوال‌های آزمون را ببیند و پاسخ‌ها را ارسال کند.

path('detail/<int:question_id>/', views.question_detail, name='question_detail')

کارش اینه که مسیر نمایش جزئیات یک سوال خاص را تعریف کند.

question_id از آدرس گرفته می‌شود و اطلاعات همان سوال نمایش داده می‌شود.

- **Views.py**

logger = logging.getLogger("app")

کارش اینه که یک logger با نام app بسازد تا رویدادهای مهم مربوط به بخش سوال‌ها و آزمون ثبت شوند.

تابع question_list

کارش اینه که لیست همه سوال‌ها را نمایش دهد.

کاری که انجام میدهد اینه که:

همه سوال‌ها را از دیتابیس می‌گیرد

اطلاعات آزمون مربوط به هر سوال را با select_related واکشی می‌کند

و گزینه‌های هر سوال را با prefetch_related می‌گیرد تا تعداد کوئری‌ها کم شود.

سوال‌ها بر اساس id مرتب می‌شوند

و سپس قالب questions/question_list.html با متغیر questions رندر می‌شود.

تابع question_detail

کارش اینه که جزئیات یک سوال خاص را نمایش دهد.

کاری که انجام میدهد اینه که:

سوال را با استفاده از question_id می‌گیرد و اگر وجود نداشته باشد ۴۰۴ برمی‌گرداند.

سپس همه گزینه‌های مربوط به آن سوال را می‌گیرد.

در نهایت قالب questions/question_detail.html با متغیرهای question و choices رندر می‌شود.

تابع question_create

کارش اینه که استاد بتواند سوال جدید ایجاد کند.

این تابع فقط برای کاربری قابل دسترسی است که لاگین کرده، ایمیلش تایید شده و نقش instructor دارد.

کاری که انجام میدهد اینه که:

اگر درخواست POST باشد یعنی فرم ارسال شده است.

فرم QuestionForm با داده‌های ارسال‌شده و کاربر جاری ساخته می‌شود.

اگر فرم معتبر باشد، سوال ذخیره می‌شود

پیام موفقیت نمایش داده می‌شود

و کاربر به صفحه جزئیات همان سوال هدایت می‌شود.

اگر درخواست GET باشد:

فرم خالی QuestionForm (وابسته به کاربر) ساخته می‌شود.

در نهایت قالب questions/question_form.html با متغیر form رندر می‌شود.

تابع question_create_for_exam

کارش اینه که استاد بتواند برای یک آزمون مشخص سوال اضافه کند.

این تابع فقط برای استاد لاگین‌کرده و صاحب همان آزمون قابل دسترسی است.

کاری که انجام میدهد اینه که:

ابتدا آزمون با exam_id گرفته می‌شود

و بررسی می‌شود که استاد همان request.user باشد.

اگر درخواست POST باشد:

فرم QuestionForm ساخته می‌شود

و اگر معتبر باشد، سوال به صورت موقت ساخته می‌شود

سپس آزمون به صورت اجباری روی سوال ست می‌شود

و سوال ذخیره می‌شود.

بعد از ذخیره پیام موفقیت نمایش داده می‌شود

و کاربر به صفحه جزئیات آزمون هدایت می‌شود.

اگر درخواست GET باشد:

فرم QuestionForm با مقدار اولیه exam نمایش داده می‌شود.

در نهایت قالب questions/question_form.html رندر می‌شود.

تابع take_exam

کارش اینه که دانشجو در یک آزمون شرکت کند و پاسخ‌ها را ارسال کند.

این تابع فقط برای کاربری قابل دسترسی است که لاگین کرده، ایمیلش تایید شده و نقش student دارد.

کاری که انجام میدهد اینه که:

در ابتدای ورود دانشجو به آزمون، یک لاگ ثبت می‌شود.

سپس آزمون و سوال‌های مربوط به آن گرفته می‌شوند و بر اساس id مرتب می‌شوند.

اگر درخواست GET باشد:

قالب questions/take_exam.html رندر می‌شود

و آزمون و سوال‌ها برای نمایش به قالب ارسال می‌شوند.

اگر درخواست POST باشد:

برای هر سوال یک نام فیلد به صورت question_<id> ساخته می‌شود.

بسته به نوع سوال، پاسخ به شکل مناسب پردازش می‌شود.

برای سوال‌های چندگزینه‌ای:

گزینه انتخاب‌شده از POST گرفته می‌شود

اگر گزینه معتبر باشد ذخیره می‌شود

و با update_or_create پاسخ دانشجو ثبت یا به‌روزرسانی می‌شود.

برای سوال‌های پاسخ کوتاه:

متن پاسخ از POST گرفته می‌شود

اگر خالی نباشد با update_or_create ذخیره یا آپدیت می‌شود.

برای سوال‌های فایل‌دار:

فایل پاسخ از request.FILES گرفته می‌شود

و اگر وجود داشته باشد با update_or_create ذخیره یا آپدیت می‌شود.

بعد از ثبت همه پاسخ‌ها:

پیام موفقیت نمایش داده می‌شود

و قالب questions/exam_submitted.html نمایش داده می‌شود

که نشان می‌دهد آزمون با موفقیت ارسال شده است.

- **templates/questions**

templates/questions/question_list.html

کارش اینه که لیست همه سوال‌های موجود در سیستم را نمایش دهد.

کاری که انجام میدهد اینه که روی متغیر questions که از view گرفته شده حلقه می‌زند
و برای هر سوال اطلاعاتی مثل متن سوال، آزمون مربوطه و نوع سوال را نمایش می‌دهد.
معمولاً لینک رفتن به جزئیات هر سوال هم در این صفحه قرار دارد.

templates/questions/question_detail.html
کارش اینه که جزئیات یک سوال خاص را نمایش دهد.
کاری که انجام میدهد اینه که متن کامل سوال را نشان می‌دهد
و اگر سوال چندگزینه‌ای باشد، گزینه‌های مربوط به آن را نمایش می‌دهد.
این صفحه برای مشاهده دقیق یک سوال استفاده می‌شود.

templates/questions/question_form.html
کارش اینه که فرم ساخت سوال جدید یا ویرایش سوال را نمایش دهد.
کاری که انجام میدهد اینه که فرم QuestionForm را رندر می‌کند
و فیلدهایی مثل آزمون، متن سوال، نوع سوال و نمره را به کاربر (استاد) نشان می‌دهد.
این فرم با متد POST ارسال می‌شود و csrf_token دارد تا ارسال امن باشد.

templates/questions/take_exam.html
کارش اینه که صفحه شرکت دانشجو در آزمون را نمایش دهد.
کاری که انجام میدهد اینه که لیست سوال‌های آزمون را به ترتیب نمایش می‌دهد
و بسته به نوع سوال، ورودی مناسب ایجاد می‌کند:
برای سوال چندگزینه‌ای گزینه‌ها به صورت radio button
برای سوال پاسخ کوتاه ورودی متنی
و برای سوال فایل‌دار ورودی فایل.
نام فیلدها به صورت question_<id> ساخته می‌شود تا در view قابل پردازش باشد.

templates/questions/exam_submitted.html
کارش اینه که بعد از ارسال پاسخ‌های آزمون به دانشجو پیام موفقیت نمایش دهد.

کاری که انجام میدهد اینه که اعلام می‌کند پاسخ‌ها با موفقیت ثبت شده‌اند

و معمولاً اطلاعات کلی آزمون یا لینک بازگشت به لیست آزمون‌ها را نمایش می‌دهد.

بخشای online_exam

- **Asgi.py:**

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'online_exam.settings')

کارش اینه که به Django بگه تنظیمات اصلی پروژه از کدوم فایل باید خوانده شود.

اگر متغیر محیطی DJANGO_SETTINGS_MODULE از قبل تنظیم نشده باشد،

آن را برابر با online_exam.settings قرار می‌دهد تا Django بتواند تنظیمات پروژه را لود کند.

application = get_asgi_application()

کارش اینه که شیء اصلی ASGI پروژه را بسازد.

این شیء نقطه ورود پروژه برای سرورهای ASGI است

و برای اجرای پروژه با ASGI (مثلاً برای WebSocket یا سرورهای async) استفاده می‌شود.

- **Middleware.py:**

req_logger = logging.getLogger("requests")

کارش اینه که یک logger مخصوص ثبت لاگ درخواست‌های HTTP بسازد.

این لاگر برای ثبت اطلاعات مربوط به هر درخواست و پاسخ استفاده می‌شود.

RequestLogMiddleware

کارش اینه که به عنوان یک middleware تمام درخواست‌های ورودی به پروژه را بررسی و لاگ‌گیری کند.

متد __init__

کارش اینه که تابع get_response را دریافت و ذخیره کند.

get_response همان تابعی است که درخواست را به view بعدی می‌فرستد.

متد __call__

کارش اینه که روی هر درخواست HTTP اجرا شود.

کاری که انجام میدهد اینه که:

در ابتدای درخواست، زمان شروع پردازش را ذخیره می‌کند.

سپس درخواست را به بقیه‌ی زنجیره پردازش (view یا middlewareهای بعدی) می‌فرستد و response را می‌گیرد.

بعد از دریافت response، مدت زمان پردازش درخواست را محاسبه می‌کند و آن را به میلی‌ثانیه تبدیل می‌کند.

سپس اطلاعات کاربر را بررسی می‌کند:

اگر کاربر لاگین کرده باشد، نام کاربری او را می‌گیرد.

اگر کاربر لاگین نکرده باشد، مقدار "anonymous" قرار می‌دهد.

در نهایت با req_logger.info یک لاگ ثبت می‌کند که شامل موارد زیر است:

نوع درخواست (GET یا POST)

مسیر کامل درخواست

کد وضعیت پاسخ (status code)

نام کاربری یا anonymous

مدت زمان پردازش درخواست به میلی‌ثانیه

در پایان response بدون تغییر برگردانده می‌شود.

- **Settings.py:**

DATABASES

کارش اینه که مشخص کند پروژه از چه پایگاه داده‌ای استفاده می‌کند.

در این پروژه پایگاه داده PostgreSQL تنظیم شده که طبق صورت پروژه الزامی است.

تنظیمات شامل نام دیتابیس، نام کاربری، رمز عبور، هاست و پورت است

و امکان خواندن این مقادیر از متغیرهای محیطی هم وجود دارد.

این بخش باعث می‌شود داده‌های آزمون، سوال‌ها، کاربران و پاسخ‌ها در PostgreSQL ذخیره شوند.

AUTH_USER_MODEL

کارش اینه که پروژه از مدل کاربر سفارشی استفاده کند.

این مدل دارای نقش کاربر (student / instructor) و وضعیت تایید ایمیل است

و تمام بخش‌های احراز هویت بر اساس این مدل انجام می‌شود.

INSTALLED_APPS

کارش اینه که اپلیکیشن‌های اصلی پروژه فعال شوند.

اپ‌های accounts، exams و questions برای پیاده‌سازی ثبت‌نام، آزمون و سوال‌ها اضافه شده‌اند

و اپ‌های پیش‌فرض Django برای ادمین، احراز هویت و sessionها فعال هستند.

MIDDLEWARE

کارش اینه که پردازش درخواست‌ها را مدیریت کند.

علاوه بر middlewareهای پیش‌فرض Django،

middleware سفارشی ثبت لاگ درخواست‌ها (RequestLogMiddleware) اضافه شده

تا اطلاعات هر درخواست مثل مسیر، کاربر و زمان پاسخ ثبت شود.

LOGGING

کارش اینه که ثبت لاگ‌های سیستمی را مدیریت کند.

لاگر app برای رویدادهای مهم پروژه

و لاگر requests برای ثبت درخواست‌های HTTP استفاده می‌شود.

TEMPLATES

کارش اینه که محل بارگذاری قالب‌های HTML را مشخص کند.

قالب‌های مربوط به accounts، exams و questions از پوشه templates لود می‌شوند

و context processorهای پیش‌فرض Django فعال هستند.

LOGIN_URL و LOGIN_REDIRECT_URL

کارش اینه که مسیر ورود کاربران و مسیر بعد از لاگین مشخص شود.

کاربران بدون لاگین به صفحه ورود هدایت می‌شوند

و بعد از ورود موفق به صفحه اصلی مناسب منتقل می‌شوند.

EMAIL SETTINGS

کارش اینه که ارسال ایمیل فعال‌سازی حساب کاربری انجام شود.

این تنظیمات برای تایید ایمیل کاربران در فرآیند ثبت‌نام استفاده می‌شوند.

TIME_ZONE و USE_TZ

کارش اینه که مدیریت زمان در پروژه استاندارد باشد.

این تنظیمات روی زمان‌بندی آزمون‌ها و تشخیص آزمون‌های گذشته و آینده تاثیر مستقیم دارند.

- **Urls.py:**

path('admin/', admin.site.urls)

کارش اینه که مسیر دسترسی به پنل ادمین Django را فعال کند.

با رفتن به آدرس /admin/ مدیر سیستم می‌تواند مدل‌ها و داده‌ها را مدیریت کند.

path('accounts/', include('accounts.urls'))

کارش اینه که تمام مسیرهای مربوط به ثبت‌نام، ورود، خروج و تایید کاربر را به اپلیکیشن accounts واگذار کند.

تمام URLهای مربوط به حساب کاربری با پیشوند /accounts/ در دسترس هستند.

path('', include(('exams.urls', 'exams'), namespace='exams'))

کارش اینه که اپلیکیشن exams را به عنوان مسیر اصلی سایت قرار دهد.

به این معنی که صفحه اصلی سایت لیست آزمون‌ها است

و تمام URLهای مربوط به آزمون‌ها بدون پیشوند اضافی در دسترس هستند.

path('questions/', include(('questions.urls', 'questions'), namespace='questions'))

کارش اینه که مسیرهای مربوط به سوال‌ها را به اپلیکیشن questions واگذار کند.

تمام URLهای مربوط به سوال‌ها با پیشوند /questions/ در دسترس هستند.

- **Wsgi.py:**

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'online_exam.settings')

کارش اینه که به Django بگه فایل تنظیمات اصلی پروژه کدام است.

اگر متغیر محیطی DJANGO_SETTINGS_MODULE از قبل تنظیم نشده باشد، آن را برابر با online_exam.settings قرار می‌دهد تا تنظیمات پروژه بارگذاری شوند.

application = get_wsgi_application()

کارش اینه که شیء اصلی WSGI پروژه را بسازد.

قسمتای دیگه پروژه:

Template

- Base.html:

base.html

کارش اینه که قالب پایه کل وب‌سایت را تعریف کند و همه templateهای دیگر از آن استفاده کنند.

کاری که انجام میدهد اینه که:
ساختار اصلی HTML صفحه را با زبان فارسی و جهت راست‌به‌چپ (rtl) مشخص می‌کند
و تنظیمات متای پایه مثل charset و viewport را قرار می‌دهد.

عنوان صفحه با استفاده از block title تعریف شده
تا هر صفحه بتواند عنوان مخصوص خودش را جایگزین کند
و اگر جایگزین نشد عنوان پیش‌فرض Online Exam نمایش داده می‌شود.

Bootstrap نسخه راست‌به‌چپ از طریق CDN لود می‌شود
و با استایل‌های ساده داخلی ظاهر کلی سایت یک‌دست می‌شود
مثل فونت، پس‌زمینه، عرض کانتینر و ظاهر کارت‌ها و نوار ناوبری.

block head تعریف شده تا هر صفحه در صورت نیاز
بتواند فایل CSS یا تنظیمات اضافه‌ای را به بخش head اضافه کند.

نوار ناوبری بالای صفحه ساخته شده
که لینک صفحه اصلی آزمون‌ها و صفحه سوالات را نمایش می‌دهد.

در نوار ناوبری وضعیت ورود کاربر بررسی می‌شود:
اگر کاربر لاگین کرده باشد، نام کاربری و نقش او نمایش داده می‌شود
و یک دکمه خروج که با متد POST و csrf_token کار می‌کند نشان داده می‌شود.
اگر کاربر لاگین نکرده باشد، دکمه‌های ورود و ثبت‌نام نمایش داده می‌شوند.

بخش main محتوای اصلی هر صفحه را در یک کانتینر محدود قرار می‌دهد
تا ظاهر سایت مرتب و خوانا باشد.

پیام‌های سیستمی Django (messages) در بالای محتوا نمایش داده می‌شوند

و نوع پیام با توجه به message.tags به شکل alert مناسب Bootstrap نمایش داده می‌شود.

block content محل قرار گرفتن محتوای اختصاصی هر صفحه است
و تمام templateهای دیگر محتوای خودشان را داخل این بلاک قرار می‌دهند.

در انتها فایل JavaScript مربوط به Bootstrap لود می‌شود
تا اجزای تعاملی مثل منوی موبایل درست کار کنند.

- **.env**

بخش Database
کارش اینه که تنظیمات اتصال به پایگاه داده PostgreSQL را مشخص کند.
USE_POSTGRES=1 مشخص می‌کند که پروژه باید از PostgreSQL استفاده کند.
POSTGRES_DB نام دیتابیس پروژه را تعیین می‌کند.
POSTGRES_USER نام کاربری دیتابیس را مشخص می‌کند.
POSTGRES_PASSWORD رمز عبور دیتابیس را مشخص می‌کند.
POSTGRES_HOST آدرس سرور دیتابیس را تعیین می‌کند که در اینجا روی localhost قرار دارد.
POSTGRES_PORT پورت اتصال به دیتابیس PostgreSQL را مشخص می‌کند.

این متغیرها باعث می‌شوند تنظیمات دیتابیس بدون تغییر مستقیم در settings.py
و به شکل امن‌تر و قابل تغییر در محیط‌های مختلف انجام شود.

بخش SMS
کارش اینه که تنظیمات ارسال پیامک را برای پروژه مشخص کند.
SMS_PROVIDER تعیین می‌کند از چه سرویس‌دهنده‌ای برای ارسال پیامک استفاده می‌شود
که در این پروژه kavenegar انتخاب شده است.

KAVENEGAR_API_KEY کلید دسترسی به سرویس کاوه‌نگار است

که برای احراز هویت درخواست‌های ارسال پیامک استفاده می‌شود.

KAVENEGAR_SENDER شماره ارسال‌کننده پیامک را مشخص می‌کند
که پیامک‌های تایید حساب یا کد فعال‌سازی از طریق آن ارسال می‌شوند.

این تنظیمات برای قابلیت تایید کاربر با پیامک
که جزو بخش‌های امتیازی پروژه است استفاده می‌شوند.

- **Manage.py**

تابع main

کارش اینه که دستورات مدیریتی Django را اجرا کند.

این تابع نقطه شروع اجرای دستوراتی مثل runserver، migrate، createsuperuser و test است.

کاری که انجام میدهد اینه که:

ابتدا متغیر محیطی DJANGO_SETTINGS_MODULE را برابر با online_exam.settings قرار می‌دهد

تا Django بداند تنظیمات پروژه از کدام فایل باید بارگذاری شود.

سپس تلاش می‌کند تابع execute_from_command_line را از Django ایمپورت کند.

اگر Django نصب نباشد یا محیط مجازی فعال نشده باشد،

یک خطای واضح و راهنما نمایش داده می‌شود تا مشکل مشخص شود.

در ادامه execute_from_command_line(sys.argv) اجرا می‌شود

که دستورات واردشده در خط فرمان را پردازش و اجرا می‌کند.

بخش اخر برای اینکه اطمینان حاصل کند تابع main فقط زمانی اجرا شود که این فایل مستقیماً اجرا شده باشد و نه زمانی که از جای دیگری ایمپورت می‌شود.

- **Requirements.txt**

نسخه های استفاده شده و در کل چیزایی که برای اجرای پرژه نیاز هست توش نوشته شده.

Logs

کارش اینه که رویدادها و رفتارهای مهم سیستم را ثبت کند
تا بتوان عملکرد پروژه را بررسی کرد، خطاها را راحت‌تر پیدا کرد
و فعالیت کاربران و درخواست‌ها را مانیتور کرد.

App.log

کارش اینه که رویدادهای مهم داخل منطق برنامه ثبت شوند.
مثلاً وقتی استاد آزمون جدید می‌سازد
یا دانشجو وارد آزمون می‌شود
یا پاسخ‌ها ارسال می‌شوند.
این لاگ‌ها معمولاً داخل viewها با logger.info استفاده می‌شوند.

Requests.log

کارش اینه که اطلاعات مربوط به هر درخواست HTTP را ثبت کند.
این لاگر در middleware سفارشی RequestLogMiddleware استفاده شده است.
برای هر درخواست موارد زیر ثبت می‌شود:
متد درخواست (GET یا POST)
مسیر کامل درخواست
کد وضعیت پاسخ
نام کاربری (یا anonymous اگر کاربر لاگین نکرده باشد)
مدت زمان پردازش درخواست به میلی‌ثانیه.

errors.log

کارش اینه که خطاها و مشکلات جدی پروژه را ثبت کند.

LOGGING در settings.py

کارش اینه که مشخص کند لاگ‌ها کجا و با چه سطحی ثبت شوند.

با این تنظیمات لاگ‌ها می‌توانند در کنسول نمایش داده شوند

یا در فایل ذخیره شوند (بسته به تنظیمات پروژه).

این بخش باعث می‌شود لاگرهای app و requests به درستی کار کنند.