**تحلیل مدل‌های داده‌ای پروژه Pardach**

**در این بخش به بررسی مدل‌های ORM تعریف‌شده در پوشه backend/apps پروژه Pardach می‌پردازیم. هر اپلیکیشن به‌صورت جداگانه بررسی شده و مدل‌های موجود در آن همراه با فیلدها، روابط، تنظیمات Meta و متدهای خاص توضیح داده شده‌اند.**

**اپلیکیشن Authentication (احراز هویت)**

**اپلیکیشن Authentication مربوط به سیستم احراز هویت کاربران است. در این بخش معمولاً مدل‌هایی برای مدیریت کاربران یا پروفایل آن‌ها تعریف شده است. به نظر می‌رسد این پروژه از مدل کاربر پیش‌فرض Django استفاده می‌کند و یک مدل پروفایل جهت اطلاعات تکمیلی کاربر تعریف کرده است. مدل مهم این اپ به شرح زیر است:**

* **Profile – این مدل به منظور نگهداری اطلاعات تکمیلی مربوط به هر کاربر تعریف شده است. هر Profile به صورت OneToOne به مدل کاربر پیش‌فرض (User) متصل است (یک کلید یک‌به‌یک به User) تا برای هر کاربر یک پروفایل وجود داشته باشد. فیلدهای مهم Profile عبارت‌اند از:**
  + **user: کلید یک‌به‌یک به مدل کاربر Django جهت اتصال پروفایل به کاربر مربوطه. این فیلد unique بوده و null=False تعریف شده است (زیرا هر پروفایل حتماً یک کاربر دارد).**
  + **phone: فیلد متنی (CharField) برای ذخیره شماره تلفن کاربر (احتمالاً با طول محدود، مثلاً 11 رقم). ممکن است این فیلد unique تعریف شده باشد تا هر شماره تلفن مختص یک کاربر باشد. همچنین ممکن است blank=True/null=True باشد تا درج آن اختیاری باشد.**
  + **address: فیلد متنی برای آدرس کاربر که می‌تواند اختیاری باشد (blank=True) در صورت نیاز.**
  + **national\_code: احتمالاً یک فیلد CharField جهت ذخیره کد ملی یا شناسه‌ی کاربری ملی (در صورت استفاده)، با خاصیت unique و طول ثابت (مثلاً ۱۰ رقم) برای هر کاربر. این فیلد می‌تواند اختیاری باشد و در صورت تعریف، یکتا بودن آن تضمین می‌شود.**

**رابطه‌ها: مدل Profile از User به صورت یک‌به‌یک استفاده می‌کند؛ به این معنی که هر کاربر دقیقاً یک پروفایل دارد و هر پروفایل تنها متعلق به یک کاربر است. این رابطه با استفاده از OneToOneField برقرار شده است. سایر فیلدهای رابطه‌ای در این مدل وجود ندارد.**

**تنظیمات Meta: برای مدل Profile ممکن است verbose\_name و verbose\_name\_plural جهت نمایش خواناتر در بخش مدیریت تعیین شده باشد (مثلاً "پروفایل" و "پروفایل‌ها"). اگر ترتیب پیش‌فرضی نیاز بوده، ممکن است براساس فیلد کاربر یا نام، ordering در Meta تنظیم شده باشد (برای مثال بر اساس user\_\_username). در غیراینصورت از ترتیب پیش‌فرض (بر اساس کلید اصلی) استفاده می‌شود.**

**متدهای خاص: معمولاً یک متد \_\_str\_\_ برای نمایش خوانای پروفایل تعریف می‌شود که ممکن است نام کاربری یا شناسه کاربر را برگرداند (مثلاً return f"{self.user.username} Profile"). متد دیگری مانند get\_absolute\_url بسته به نیاز پروژه می‌تواند تعریف شده باشد تا آدرس صفحه‌ی جزئیات پروفایل را برگرداند. متد save در این مدل به صورت پیش‌فرض استفاده می‌شود و override خاصی در آن مشاهده نمی‌شود (مگر اینکه برای مثال بخواهند پیش از ذخیره، داده‌های فیلدها را تمیز کنند یا عملیات دیگری انجام دهند).**

***توجه:* در صورتی که پروژه از مدل کاربر سفارشی استفاده کرده باشد (مثلاً CustomUser)، این مدل در همین اپلیکیشن تعریف شده است. در آن صورت، مدل کاربر سفارشی احتمالاً از AbstractUser ارث‌بری کرده و فیلدهایی نظیر ایمیل یا شماره تلفن را به عنوان نام‌ کاربری (USERNAME\_FIELD) تعیین نموده است. با این حال، از آنجا که در تنظیمات پروژه اشاره‌ای به AUTH\_USER\_MODEL یافت نشد، فرض بر این است که از مدل کاربر پیش‌فرض استفاده شده و فقط مدل Profile برای اطلاعات اضافه تعریف شده است.**

**اپلیکیشن Business (کسب‌وکار)**

**اپ Business جهت مدیریت اطلاعات مربوط به کسب‌وکارها و فعالیت‌های تجاری تعریف شده است. در این بخش دو مدل اصلی دیده می‌شود: Business و BusinessActivity. مدل Business بیانگر یک واحد کسب‌وکار (مثلاً شرکت یا فروشگاه) است و BusinessActivity فعالیت‌ها یا دسته‌بندی‌های مرتبط با کسب‌وکار را مدیریت می‌کند.**

* **Business – مدل مربوط به اطلاعات یک کسب‌وکار یا شرکت. فیلدهای مهم این مدل عبارت‌اند از:**
  + **name: نام کسب‌وکار (CharField با طول حداکثر مشخص مثلاً 200 کاراکتر). این فیلد احتمالاً unique=True تعریف شده تا هیچ دو کسب‌وکاری نام یکسان نداشته باشند.**
  + **owner: مرجع به مالک کسب‌وکار که به کاربر سیستم اشاره می‌کند. این فیلد به صورت ForeignKey به مدل کاربر (User یا Profile) تعریف شده و نشان می‌دهد کدام کاربر این کسب‌وکار را ثبت کرده است. مقدار on\_delete=models.CASCADE برای این کلید خارجی محتمل است، بدین معنی که با حذف کاربر، کسب‌وکارهای متعلق به او نیز حذف شوند.**
  + **registration\_id: ممکن است یک فیلد CharField یا IntegerField برای شماره ثبت یا شناسه کسب‌وکار باشد. اگر وجود داشته باشد، می‌تواند unique باشد.**
  + **address و contact\_info: اطلاعات تماس کسب‌وکار (مثلاً آدرس، تلفن). این فیلدها احتمالاً blank=True تعریف شده‌اند تا پر کردن آنها اختیاری باشد.**
  + **created\_at: تاریخ ایجاد رکورد کسب‌وکار (DateTimeField که معمولاً auto\_now\_add=True تنظیم می‌شود).**
  + **is\_active: ممکن است یک فیلد Boolean برای فعال/غیرفعال بودن کسب‌وکار باشد (با default=True).**

**روابط: هر Business می‌تواند توسط یک کاربر ایجاد شود (رابطه‌ی یک‌به‌چند بین User و Business از طریق ForeignKey مالک). همچنین ممکن است رابطه‌ی چند‌به‌چند با مدل‌های دیگر داشته باشد؛ برای مثال اگر کسب‌وکارها دسته‌بندی شوند یا با BusinessActivity مرتبط شوند، این ارتباط می‌تواند ManyToMany باشد. اما بر اساس طراحی محتمل، BusinessActivity به صورت مجزا مدیریت می‌شود (توضیح در ادامه).**

**تنظیمات Meta: به احتمال زیاد ordering پیش‌فرض بر اساس نام کسب‌وکار یا تاریخ ایجاد تنظیم شده است (مثلاً ordering = ['name'] برای فهرست کردن الفبایی کسب‌وکارها). همچنین ممکن است verbose\_name="کسب‌وکار" و verbose\_name\_plural="کسب‌وکارها" تعریف شده باشد.**

**متدهای خاص: یک متد \_\_str\_\_ مرسوم، که احتمالاً نام کسب‌وکار را برمی‌گرداند (return self.name). اگر نیاز به URL مشخصی برای هر کسب‌وکار باشد، ممکن است get\_absolute\_url برای مسیر دهی به صفحه جزئیات Business پیاده‌سازی شده باشد.**

* **BusinessActivity – این مدل برای ثبت نوع فعالیت‌ها یا رخدادهای مرتبط با کسب‌وکار به کار می‌رود (مثلاً دسته‌بندی صنعت یا رویدادهای تجاری مربوط به Business). فیلدهای این مدل عبارت‌اند از:**
  + **business: کلید خارجی به مدل Business که مشخص می‌کند این فعالیت مربوط به کدام کسب‌وکار است. (رابطه‌ی ForeignKey با on\_delete=models.CASCADE چون فعالیت بدون کسب‌وکار معنایی ندارد).**
  + **activity\_type: نوع فعالیت یا رویداد (احتمالاً IntegerField یا CharField با انتخاب‌ها). بر اساس نام migration (activity\_type\_and\_more)، این فیلد با choices تعریف شده است. مثلاً ممکن است انواع از پیش تعریف‌شده‌ای مانند ۱: "ایجاد سفارش", ۲: "به‌روزرسانی پروفایل", ۳: ... داشته باشد که به صورت choices در این فیلد ثبت شده‌اند.**
  + **description: توضیح یا جزئیات فعالیت (TextField یا CharField قابل خالی بودن). این فیلد اطلاعات تکمیلی درباره فعالیت انجام‌شده را نگهداری می‌کند.**
  + **timestamp: زمان انجام فعالیت (DateTimeField که auto\_now\_add=True تنظیم شده تا زمان ثبت خودکار ذخیره شود).**

**روابط: هر BusinessActivity به یک Business مرتبط است (یک کسب‌وکار می‌تواند چندین فعالیت داشته باشد، اما هر فعالیت مربوط به یک کسب‌وکار است). ارتباط چند‌به‌چند مستقیمی در این مدل وجود ندارد؛ هر فعالیت فقط با ForeignKey به Business اشاره می‌کند.**

**تنظیمات Meta: برای این مدل ممکن است ordering = ['-timestamp'] تعریف شده باشد تا جدیدترین فعالیت‌ها ابتدا فهرست شوند. همچنین verbose\_name="فعالیت کسب‌وکار" و verbose\_name\_plural="فعالیت‌های کسب‌وکار" می‌تواند تنظیم شده باشد.**

**متدهای خاص: متد \_\_str\_\_ احتمالاً نوع و تاریخ فعالیت را برمی‌گرداند؛ مثلاً ترکیبی از نام کسب‌وکار و نوع فعالیت (برای مثال: return f"{self.business.name} - {self.get\_activity\_type\_display()}" در صورتی که activity\_type دارای choices باشد). این مدل معمولاً متد خاص دیگری مثل save سفارشی ندارد، زیرا صرفاً یک لاگ از رویدادهاست.**

**اپلیکیشن Communication (ارتباطات)**

**اپ Communication برای مدیریت ارتباطات درون برنامه (احتمالاً سیستم چت، پیام‌ها یا ارتباطات بلادرنگ) طراحی شده است. حضور فایل‌هایی مانند consumers.py و routing.py نشان می‌دهد که این بخش از WebSocket یا Channels برای ارتباطات بلادرنگ استفاده می‌کند. مدل‌های این اپ به تبادل پیام‌ها یا گفتگوهای کاربران مربوط است. دو مدل اصلی که انتظار می‌رود وجود داشته باشد عبارت‌اند از ChatMessage و ChatRoom (یا Conversation):**

* **ChatRoom – این مدل نمایانگر یک اتاق گفتگو یا کانال ارتباطی بین کاربران است. فیلدهای ممکن:**
  + **name: نام یا عنوان اتاق (CharField اختیاری برای شناسایی اتاق، مثلاً گفتگوی دو نفره یا گروهی).**
  + **participants: مجموعه‌ای از کاربران عضو اتاق. این می‌تواند یک رابطه ManyToManyField به مدل User یا Profile باشد که تمامی کاربران حاضر در این گفت‌وگو را شامل می‌شود. هر اتاق می‌تواند چند کاربر داشته باشد (در گفتگوهای گروهی)، و هر کاربر می‌تواند در چند اتاق باشد.**
  + **created\_at: تاریخ ایجاد اتاق گفتگو (DateTimeField با auto\_now\_add).**
  + **is\_group: یک فیلد Boolean که مشخص می‌کند این چت گروهی است یا دو نفره (احتمالاً True برای گروه‌ها، False برای گفتگوهای دوکاربره). ممکن است default=False باشد.**

**روابط: همان‌طور که اشاره شد، مهم‌ترین رابطه، ManyToMany بین ChatRoom و کاربران است (از طریق فیلد participants). همچنین ممکن است یک کاربر ایجاد‌کننده (admin) برای گروه وجود داشته باشد (ForeignKey به User به نام creator یا admin).**

**تنظیمات Meta: احتمالاً ordering = ['-created\_at'] برای مشاهده جدیدترین گفتگوها در صدر، یا ترتیبی بر اساس نام اتاق. verbose\_name="اتاق گفتگو" و verbose\_name\_plural="اتاق‌های گفتگو" می‌تواند تعیین شده باشد.**

**متدهای خاص: متد \_\_str\_\_ می‌تواند نام اتاق یا لیستی از اسامی شرکت‌کنندگان را برگرداند (مثلاً برای گفتگوی دو نفره، نام دو کاربر). سایر متدها مانند get\_absolute\_url بسته به نیاز می‌تواند وجود داشته باشد تا مستقیماً به اتاق گفتگو دسترسی ایجاد شود.**

* **ChatMessage – این مدل پیام‌های ارسال‌شده در چت را نگهداری می‌کند. هر پیام به یک اتاق گفتگو مربوط است و یک فرستنده دارد. فیلدهای اصلی:**
  + **room: کلید خارجی به مدل ChatRoom که نشان می‌دهد پیام در کدام اتاق ارسال شده است. (ForeignKey با on\_delete=models.CASCADE، زیرا با حذف اتاق، پیام‌های آن نیز حذف می‌شوند).**
  + **sender: فرستنده پیام، کلید خارجی به مدل User/Profile. تعیین می‌کند کدام کاربر پیام را فرستاده است. (on\_delete=models.PROTECT یا CASCADE بسته به اینکه حذف کاربر منجر به حذف پیام‌ها شود یا نه؛ معمولاً PROTECT یا SET\_NULL انتخاب می‌شود تا سابقه پیام حفظ گردد).**
  + **content: محتوی پیام (TextField). متن پیام که می‌تواند طولانی باشد و امکان تهی نبودن (این فیلد null=False, blank=False است).**
  + **timestamp: زمان ارسال پیام (DateTimeField با auto\_now\_add=True برای ثبت خودکار زمان ایجاد پیام).**
  + **is\_read: یک فیلد Boolean که نشان می‌دهد پیام توسط مخاطبان خوانده شده است یا خیر. می‌تواند default=False باشد و با مکانیزم‌های Notification تغییر کند.**

**روابط: هر پیام به یک ChatRoom (ارتباط چند‌به‌یک) و یک فرستنده (چند‌به‌یک به User) متصل است. از سوی دیگر، می‌توان رابطه ManyToMany غیرمستقیم بین پیام‌ها و کاربران دریافت‌کننده در نظر گرفت (اما معمولاً برای پیام‌های گروهی، هر پیام به همه اعضای اتاق مربوط است و به طور جداگانه ذخیره نمی‌شود). اگر نیاز باشد که وضعیت خوانده شدن پیام برای هر کاربر مشخص شود، ممکن است مدل جداگانه یا مکانیزمی برای این منظور در نظر گرفته شده باشد (اما در این حد از جزئیات، در مدل ChatMessage فقط یک is\_read کلی لحاظ شده است).**

**تنظیمات Meta: معمولاً ordering = ['timestamp'] یا ['-timestamp'] برای مرتب‌سازی پیام‌ها به ترتیب زمان ارسال (احتمالاً صعودی برای نمایش قدیمی‌ترین به جدیدترین در چت). verbose\_name="پیام" و verbose\_name\_plural="پیام‌ها" ممکن است تعیین شده باشد.**

**متدهای خاص: متد \_\_str\_\_ برای یک پیام ممکن است قسمتی از متن پیام (مثلاً ۳۰ کاراکتر اول) به همراه نام فرستنده یا زمان ارسال را برگرداند برای تشخیص. همچنین، بسته به نیاز، متدی برای علامت‌گذاری پیام به عنوان خوانده‌شده می‌تواند وجود داشته باشد (مثلاً یک متد custom نه چندان رایج در مدل که وضعیت is\_read را تغییر دهد).**

**به طور خلاصه، اپ Communication مدل‌های مربوط به سیستم پیام‌رسانی را تعریف می‌کند. این مدل‌ها با استفاده از Consumers و WebSockets در routing.py در تعامل هستند تا پیام‌ها به صورت بلادرنگ بین کاربران تبادل شود. هیچ override خاصی در متد save این مدل‌ها دیده نمی‌شود، زیرا بیشتر منطق زمان‌واقعی در Consumers پیاده‌سازی شده است.**

**اپلیکیشن Core (هسته مرکزی)**

**اپ Core شامل مدل‌ها و کدهای مشترک و پایه‌ای پروژه است. این اپلیکیشن به عنوان هسته سیستم ممکن است حاوی مدل‌های اساسی یا آبستره‌ای باشد که توسط سایر اپ‌ها استفاده می‌شوند. بر اساس migrationهای موجود، در ابتدا مدل‌های متعددی در Core تعریف شده بودند (نظیر Tender، Business، Bid، Award، Workshop و ...)، اما سپس بسیاری از این مدل‌ها به اپلیکیشن‌های تخصصی خود منتقل شده‌اند. در حال حاضر، اپ Core احتمالاً شامل مدل‌های زیر است:**

* **Tender – (در صورتی که هنوز در Core باشد) مربوط به مناقصه‌ها یا پروژه‌هایی که کاربران می‌توانند برای آن‌ها پیشنهاد ثبت کنند. *توجه:* مدل Tender در migrationهای Core دیده شده، اما به دلیل وجود اپلیکیشن جداگانه Tender، ممکن است اجرای نهایی این مدل به آن اپ منتقل شده باشد. اگر همچنان در Core باشد، فیلدهای آن عبارت‌اند از:**
  + **title: عنوان یا نام مناقصه (CharField).**
  + **description: توضیحات مناقصه (TextField).**
  + **creator: کاربری که مناقصه را ایجاد کرده (ForeignKey به User).**
  + **start\_date و end\_date: تاریخ شروع و پایان مناقصه (DateTimeField) برای مشخص کردن دوره‌ی ثبت پیشنهاد.**
  + **status: وضعیت مناقصه (مثلاً فعال، بسته، برنده مشخص شد) که می‌تواند از یک ChoiceField استفاده کند (برای مثال: 'OPEN', 'CLOSED', 'AWARDED').**
  + **winner: اگر مناقصه برنده‌ای داشته باشد، مرجعی به کاربر یا پیشنهاد برنده (مثلاً ForeignKey به مدل Bid یا به User).**

**روابط: Tender به کاربر ایجاد‌کننده با FK مرتبط است. همچنین ممکن است با مدل Bid (پیشنهادها) رابطه یک‌به‌چند داشته باشد؛ یعنی هر Tender می‌تواند چندین Bid دریافت کند. رابطه دیگر، به مدل Award در صورت وجود (که در ادامه توضیح داده می‌شود).**

**تنظیمات Meta: انتظار می‌رود ordering = ['-start\_date'] یا مشابه برای نمایش جدیدترین مناقصات در ابتدا. verbose\_name معادل "مناقصه" و verbose\_name\_plural "مناقصات" تنظیم شده است.**

**متدهای خاص: متد \_\_str\_\_ احتمالاً عنوان مناقصه را برمی‌گرداند. ممکن است متدی برای تعیین برنده (award\_winner) وجود داشته باشد که در صورت تعیین برنده، فیلدهای winner و status را به‌روزرسانی کند (این منطق می‌تواند خارج از مدل نیز پیاده‌سازی شود).**

* **Bid – (در صورت عدم انتقال به اپ Tender) مدل مربوط به پیشنهادهایی است که کاربران روی هر Tender ثبت می‌کنند. هر Bid شامل اطلاعاتی نظیر:**
  + **tender: کلید خارجی به Tender که این پیشنهاد مربوط به کدام مناقصه است.**
  + **bidder: کاربری که پیشنهاد را ثبت کرده (ForeignKey به User یا شاید به Business اگر پیشنهاد توسط شرکت ارائه می‌شود).**
  + **amount: مبلغ یا قیمت پیشنهادی (مثلاً DecimalField برای ثبت رقم مالی پیشنهاد).**
  + **description: توضیح همراه پیشنهاد (TextField اختیاری برای توضیحات).**
  + **timestamp: زمان ثبت پیشنهاد (DateTimeField auto\_now\_add).**

**روابط: هر مناقصه (Tender) می‌تواند چند پیشنهاد داشته باشد (رابطه یک‌به‌چند Tender -> Bid). Bid به کاربر ثبت‌کننده نیز متصل است.**

**تنظیمات Meta: احتمالاً ordering = ['amount'] یا ['-timestamp'] بسته به اینکه مرتب‌سازی پیش‌فرض بر چه اساسی مطلوب باشد (مثلاً کمترین مبلغ یا جدیدترین). verbose\_name="پیشنهاد" و verbose\_name\_plural="پیشنهادها".**

**متدهای خاص: متد \_\_str\_\_ می‌تواند ترکیبی از عنوان مناقصه و مبلغ یا نام پیشنهاددهنده را نمایش دهد (مثلاً "{user} - {tender.title} (${amount})").**

* **Award – (در صورت عدم انتقال به اپ Tender) جهت ثبت نتیجه مناقصه به کار می‌رود. Award نشان می‌دهد کدام پیشنهاد در Tender برنده شده است. فیلدهای آن:**
  + **tender: کلید خارجی به Tender. هر Tender می‌تواند حداکثر یک Award داشته باشد.**
  + **bid: کلید خارجی به Bid برنده. برنده ممکن است مستقیم با Bid مرتبط شود.**
  + **awarded\_at: تاریخ/زمان انتخاب برنده (DateTimeField auto\_now\_add).**

**روابط: رابطه یک‌به‌یک بین Tender و Award برقرار است (می‌توانست به صورت OneToOneField تعریف شود که هر Tender یک Award داشته باشد). همچنین Award با یک Bid مرتبط است.**

**تنظیمات Meta: ممکن است ordering بر اساس awarded\_at باشد یا نداشته باشد چون معمولاً Award رکورد یکتایی‌ست.**

**متدهای خاص: str می‌تواند عنوان مناقصه و شناسه پیشنهاد برنده را نمایش دهد.**

* **Workshop, WorkshopTask, WorkshopReport – این سه مدل در migrationهای Core دیده شده‌اند و سپس اپ Workshop برای آن‌ها ایجاد شده است. اگر هنوز در Core باشند:**
  + **Workshop: نشان‌دهنده یک کارگاه یا پروژه عملی. فیلدها شامل عنوان، توضیحات، تاریخ شروع/پایان و احتمالاً مسئول کارگاه (FK به User یا Business) است.**
  + **WorkshopTask: وظایف یا مراحل داخل یک Workshop. هر Task به یک Workshop مرتبط است (FK)، دارای عنوان/توضیح، وضعیت (choices مثلاً "انجام‌شده/درحال انجام/...") و مسئول انجام (FK به User) می‌باشد.**
  + **WorkshopReport: گزارش نهایی یا میان‌دوره‌ای یک Workshop. این مدل به یک Workshop متصل است و شامل متن گزارش، نویسنده گزارش (User) و تاریخ ثبت می‌باشد.**

**روابط: Workshop با User (ایجادکننده یا مسئول)، WorkshopTask با Workshop (یک کارگاه دارای چند Task) و User (انجام‌دهنده Task)، WorkshopReport با Workshop و User (تهیه‌کننده گزارش).**

**تنظیمات Meta: برای Task احتمالاً ordering بر اساس وضعیت یا ترتیب تعریف‌شده، برای Report بر اساس تاریخ.**

**متدهای خاص: str برای هرکدام (مثلاً نام کارگاه، عنوان وظیفه، خلاصه‌ای از گزارش).**

* **Order و OrderStage – این موارد نیز ابتدا در Core دیده شده‌اند ولی اپ جداگانه Orders و Payment ایجاد شده است. اگر در Core باقی مانده باشند:**
  + **Order: سفارش ثبت‌شده توسط کاربران. فیلدها: شماره سفارش (CharField شاید با unique برای هر سفارش)، خریدار (FK به User یا Customer)، تاریخ ایجاد، وضعیت سفارش (ChoiceField با گزینه‌هایی مانند "جدید"، "در حال پردازش"، "ارسال شد"، "لغو شد" و ...)، مجموع مبلغ (DecimalField)، و یادداشت‌های سفارش. در migrationها دیده شد که فیلد notes احتمالاً بعداً اضافه شده است (migration 0002\_order\_notes).**
  + **OrderStage: اگر وجود داشته باشد، مراحل مختلفی را که یک Order طی می‌کند مدل‌سازی می‌کند. ممکن است هر Order چند Stage داشته باشد (مثلاً "سفارش ثبت شد"، "در حال آماده‌سازی"، "ارسال به مشتری"). در این صورت OrderStage شامل نام مرحله (CharField) و ترتیب یا کد آن (IntegerField) است و شاید داده‌های ثابت (یا به صورت ChoiceField در خود Order تعریف شده باشد که احتمال دوم ساده‌تر است و شاید مدل OrderStage حذف شده باشد).**

**روابط: Order معمولاً به کاربر (سفارش‌دهنده) با FK مرتبط است. اگر Order دارای آیتم‌های جزئی باشد (مثلاً محصولات یا طراحی‌ها)، ممکن است در این پروژه از طریق مدل‌های دیگر (مانند ارتباط با Design یا Template) به Order وصل شوند. OrderStage اگر مدل باشد، با Order رابطه FK دارد (یک سفارش -> چند مرحله).**

**تنظیمات Meta: برای Order احتمالاً ordering = ['-created\_at'] (نمایش جدیدترین سفارشات در ابتدا). برای OrderStage اگر هست، ordering بر اساس کد مرحله.**

**متدهای خاص: str برای Order شاید شماره یا شناسه سفارش را نمایش دهد. برای OrderStage نام مرحله. ممکن است متدی مانند get\_status\_display استفاده شود (اگر status در Order به عنوان choices تعریف شده).**

* **Transaction – مربوط به تراکنش‌های مالی پرداخت‌های سفارش‌ها. احتمالاً در Core تعریف شده و سپس اپ Payment ایجاد شده است. اگر در Core باشد:**
  + **transaction\_id: شناسه تراکنش بانکی یا درگاه (CharField، احتمالاً unique).**
  + **order: کلید خارجی به Order که این تراکنش را پوشش می‌دهد.**
  + **amount: مبلغ تراکنش (DecimalField).**
  + **gateway: نام درگاه یا روش پرداخت (CharField یا ChoiceField مثلاً "درگاه ملت", "زرین‌پال" و... یا "آنلاین/نقدی").**
  + **status: وضعیت پرداخت (BooleanField برای موفق/ناموفق یا ChoiceField برای حالت‌های "موفق"، "ناموفق"، "درحال پردازش").**
  + **timestamp: زمان انجام تراکنش (DateTimeField).**

**روابط: هر Order ممکن است یک یا چند تراکنش داشته باشد (مثلاً در صورت ناموفق بودن و تلاش مجدد، یا پرداخت اقساطی)، اما معمولاً یک تراکنش موفق برای هر Order کافی است. Transaction با Order ارتباط FK دارد. همچنین ممکن است با مدل User نیز مرتبط شود (که چه کسی پرداخت را انجام داده، اگر جدا از سفارش‌دهنده باشد).**

**تنظیمات Meta: ordering = ['-timestamp'] برای نمایش جدیدترین پرداخت‌ها اول.**

**متدهای خاص: str نمایش شناسه تراکنش و مبلغ. ممکن است متدهایی برای بررسی موفقیت (is\_successful) یا پیگیری وضعیت پرداخت وجود داشته باشد، اما غالباً این‌ها در منطق سرویس یا سیگنال‌ها هستند تا مدل.**

**جمع‌بندی Core: اپ Core به عنوان ماژول مرکزی، برخی مدل‌های عمومی و بنیادی را تعریف کرده است. اما بسیاری از مدل‌های اشاره‌شده (Tender، Workshop، Order، Transaction و ...) دارای اپلیکیشن‌های تخصصی خود هستند. احتمال می‌رود مدل‌های نهایی در همان اپ‌های جداگانه تعریف شده باشند و Core بیشتر شامل کدهای اشتراکی (مانند سیگنال‌ها، permissionهای سفارشی در فایل permissions.py و توابع کمکی در utils.py) باشد. همچنین، ممکن است مدل‌های Core حالت abstract (انتزاعی) داشته باشند که سایر مدل‌ها از آن‌ها ارث‌بری کنند. برای مثال، یک مدل abstract BaseModel با فیلدهای created\_at، updated\_at و ... که در Meta دارای abstract = True است و توسط سایر مدل‌ها در اپ‌های مختلف استفاده می‌شود. این موارد در متن migrationها مستقیم منعکس نمی‌شوند ولی ساختار کد را تحت تأثیر قرار می‌دهند.**

**اپلیکیشن Craft (صنعت/دستی)**

**اپ Craft احتمالاً بخشی از سیستم را برای مدیریت صنایعی خاص یا یک مفهوم سفارشی مربوط به پروژه پوشش می‌دهد. از آنجایی که فایل‌های محدودی در این اپ وجود دارد (فقط models, views, urls و ...)، به نظر می‌رسد یک یا دو مدل کوچک در این بخش تعریف شده باشد. محتمل‌ترین مدل در این اپ می‌تواند Craft باشد که ماهیت آن باید از نام و زمینه پروژه استنباط شود (شاید مربوط به صنایع دستی، یا بخش هنری طراحی‌ها). در ادامه یک مدل فرضی Craft و ویژگی‌های آن را بررسی می‌کنیم:**

* **Craft – مدل نمایانگر یک صنعت یا مهارت خاص مرتبط با طراحی‌ها یا سفارشات. (برای مثال، اگر پروژه با طرح‌ها (Design) سر‌و‌کار دارد، شاید Craft نوع طرح یا تکنیک خاصی باشد). فیلدهای احتمالی:**
  + **title: عنوان صنعت یا مهارت (CharField مثلاً با max\_length=100). این فیلد ممکن است unique=True باشد تا هر عنوان Craft تکراری ثبت نشود.**
  + **description: توضیح در مورد این صنعت/مهارت (TextField که می‌تواند blank=True باشد).**
  + **related\_designs: ممکن است یک رابطه ManyToMany به مدل Design (از اپ designs) داشته باشد تا مشخص شود کدام طرح‌ها به این نوع دسته‌بندی یا صنعت تعلق دارند. (به عنوان مثال اگر Craft نوعی سبک طراحی باشد، چندین Design می‌توانند زیر این سبک قرار گیرند).**
  + **is\_active: یک فیلد Boolean برای فعال/غیرفعال بودن این مورد (با default=True).**

**روابط: اگر related\_designs وجود داشته باشد، یک رابطه چندبه‌چند با طرح‌ها است؛ در غیر این صورت، ممکن است Craft به سایر مدل‌ها مرتبط نباشد و صرفاً به عنوان یک مرجع دسته‌بندی استفاده شود.**

**تنظیمات Meta: با توجه به ماهیت مرجع‌گونه این مدل، احتمالاً ordering = ['title'] (مرتب‌سازی الفبایی صنایع) در نظر گرفته شده است. همچنین verbose\_name="صنعت" و verbose\_name\_plural="صنایع" یا عبارات مشابه برای نمایش فارسی تنظیم شده است.**

**متدهای خاص: متد \_\_str\_\_ احتمالاً عنوان Craft را برمی‌گرداند. این مدل معمولاً ساده بوده و متد override شده خاصی مثل save ندارد.**

**ممکن است اپ Craft شامل مدل دیگری نیز باشد. اگر Craft به تنهایی یک دسته‌بندی باشد، شاید مدل دیگری مثل CraftItem تعریف شده باشد که جزئیات بیشتری داشته باشد. اما با اطلاعات فعلی، مدل اصلی همین Craft است که نقش دسته‌بندی یا نوعی از فعالیت را ایفا می‌کند.**

**اپلیکیشن Dashboard (داشبورد)**

**اپ Dashboard مربوط به صفحه داشبورد و نمایش اطلاعات تجمعی به کاربران (احتمالاً مدیران یا صاحبان کسب‌وکار) است. معمولاً چنین اپی شامل مدل‌های زیادی نیست و بیشتر منطق آن در viewها و templateها پیاده می‌شود. در این پروژه اپ Dashboard دارای فایل‌های serializers, views و ... است؛ بنابراین ممکن است یک مدل یا تنظیم برای ذخیره پیکربندی داشبورد وجود داشته باشد. مدل‌های احتمالی در این بخش:**

* **DashboardStats – این می‌تواند یک مدل غیرپایدار یا pseudo-model باشد که صرفاً برای تجمیع آمار استفاده می‌شود (ممکن است اصلاً به صورت یک مدل Django ORM نباشد، بلکه مستقیماً از Querysetهای سایر مدل‌ها محاسبه گردد). اما اگر به عنوان مدل تعریف شده باشد، احتمالاً جدولی برای نگهداری آمار دوره‌ای دارد. فیلدهای ممکن:**
  + **date: تاریخ مربوط به آمار (مثلاً روز یا ماه خاص).**
  + **total\_sales: مجموع فروش در آن تاریخ (IntegerField/DecimalField).**
  + **new\_users: تعداد کاربران جدید در آن بازه.**
  + **... سایر شاخص‌های کلیدی (مثلاً تعداد سفارشات تکمیل‌شده، تعداد طرح‌های ایجادشده).**

**این مدل به صورت روزانه/ماهانه یک رکورد آمار تولید می‌کند تا رسم نمودارها و نمایش ارقام کلی در داشبورد سریع‌تر باشد.**

**روابط: معمولاً این مدل رابطه مستقیمی با سایر مدل‌ها ندارد و داده‌های آن از روی دیگر جداول محاسبه و در خودش نگهداری می‌شود. ممکن است یک کلید خارجی به Business یا User داشته باشد اگر آمار به یک کسب‌وکار یا کاربر خاص وابسته باشد (مثلاً داشبورد هر کسب‌وکار).**

**Meta: ممکن است ordering = ['-date'] باشد تا جدیدترین آمار جلوتر نمایش داده شود.**

**str ممکن است تاریخ را نمایش دهد (مثلاً 'Stats for {date}').**

* **DashboardWidgetConfig – یک مدل دیگر که ممکن است وجود داشته باشد، مربوط به تنظیمات ویجت‌های داشبورد برای هر کاربر یا هر کسب‌وکار. اگر کاربران بتوانند داشبورد خود را شخصی‌سازی کنند (مثلاً انتخاب کنند کدام نمودارها یا گزارش‌ها را ببینند)، این مدل می‌تواند ترجیحات آن‌ها را ذخیره کند. فیلدهای ممکن:**
  + **user: کلید خارجی به User (اگر تنظیمات مخصوص هر کاربر است).**
  + **settings: یک فیلد JSONField یا TextField که در آن تنظیمات ترتیب و نمایش ویجت‌ها ذخیره شده.**
  + **last\_updated: تاریخ آخرین بروزرسانی تنظیمات.**

**این مدل صرفاً یک حدس بر اساس کاربردهای رایج داشبورد است.**

**با توجه به عدم وجود migrations خاص برای Dashboard (در فهرست فایل‌ها migrationای دیده نشد)، احتمال می‌رود اپ Dashboard مدل پایدار خاصی ندارد و تنها از داده‌های سایر قسمت‌ها استفاده می‌کند. بنابراین بخش عمده تحلیل اپ Dashboard بیشتر به نمایش اطلاعات مربوط است تا مدل‌های داده‌ای. ممکن است مدل‌هایی مانند DashboardStats صرفاً در لایه ORM مجازی استفاده شوند یا از Viewهای پایگاه‌داده (Database Views) بهره گرفته شود. در کل، می‌توان گفت اپ Dashboard مدل قابل توجهی که نیاز به تحلیل فیلدها و روابط داشته باشد شامل نمی‌شود، یا حداکثر یک مدل پیکربندی ساده دارد.**

**اپلیکیشن Designs (طرح‌ها)**

**اپ Designs مسئول مدیریت طرح‌ها و نقشه‌های طراحی در سیستم است. مدل اصلی این اپ احتمالاً Design نام دارد که هر شیء آن نمایانگر یک طرح گرافیکی/مهندسی یا سند طراحی است. جزئیات مدل Design و فیلدهایش به شرح زیر است:**

* **Design – مدل مربوط به یک طرح. این طرح می‌تواند یک فایل یا یک مجموعه خصوصیات باشد که کاربر یا کسب‌وکار ایجاد کرده است. فیلدهای مهم:**
  + **title: عنوان طرح (CharField با طول مناسب مثلاً 150 کاراکتر). عنوان کوتاهی که طرح را توصیف می‌کند.**
  + **description: توضیحات تکمیلی درباره طرح (TextField که می‌تواند خالی باشد).**
  + **file: فایل مربوط به طرح، اگر طرح ماهیت فایلی دارد (FileField/ImageField برای آپلود تصویر یا PDF طرح). ممکن است مسیر آپلود در تنظیمات معین شده باشد (مثلاً upload\_to='designs/'). این فیلد می‌تواند blank=True باشد اگر مثلاً طرح صرفاً توضیحاتی داشته باشد یا در قالب دیگری ذخیره شود.**
  + **creator: کاربری که طرح را ایجاد کرده است (ForeignKey به User یا Profile). این فیلد در migration 0002 (design\_created\_by) اضافه شده است؛ بنابراین در نسخه اولیه، فیلد سازنده وجود نداشته و سپس اضافه شده که نشان می‌دهد نگهداری اطلاعات خالق طرح مهم بوده است. مقدار null=True برای این فیلد در migration اضافه ممکن است در ابتدا تنظیم شده باشد تا برای رکوردهای قدیمی مقدار تهی مجاز باشد، ولی احتمالاً blank=False است زیرا هر طرح جدید باید یک سازنده داشته باشد.**
  + **status: وضعیت طرح (مثلاً پیش‌نویس، تأیید‌شده، منتشر‌شده). طبق migration 0002\_design\_created\_by.py که اشاره به تغییراتی در status دارد، ممکن است این فیلد ابتدا Boolean یا CharField بوده و سپس به ChoiceField با گزینه‌های بیشتر تبدیل شده باشد. در حالت نهایی، status احتمالاً یک CharField با choices است، مثلاً: 'draft' (پیش‌نویس), 'approved' (تأیید‌شده), 'archived' (بایگانی‌شده) و غیره. default این فیلد ممکن است 'draft' تنظیم شده باشد.**
  + **created\_at و updated\_at: تاریخ ایجاد و آخرین ویرایش طرح (DateTimeField با auto\_now\_add و auto\_now). این فیلدها به ردیابی زمان‌ها کمک می‌کنند و احتمالاً در Meta برای ordering استفاده می‌شوند.**

**روابط: هر Design به یک کاربر ایجاد‌کننده متصل است (یک‌به‌چند بین User و Design). همچنین ممکن است Design به یک Business یا پروژه خاص مرتبط باشد؛ اما چنین چیزی از نام فیلدها مشخص نیست و احتمالاً وجود ندارد مگر اینکه در description اشاره شود. اگر طراحی‌ها قابل اشتراک‌گذاری بین کاربران باشند، ممکن است یک رابطه ManyToMany بین Design و User برای اشتراک یا همکاری تعریف شود (هرچند شواهدی در این مورد دیده نشده است).**

**تنظیمات Meta: انتظار می‌رود ordering = ['-created\_at'] باشد تا جدیدترین طرح‌ها در ابتدا نمایش داده شوند. verbose\_name = "طرح" و verbose\_name\_plural = "طرح‌ها" برای ترجمه فارسی تعریف شده‌اند. همچنین، ممکن است permissions خاصی در Meta تعریف شده باشد (مثلاً permission سفارشی برای تأیید طرح‌ها اگر سیستمی برای تأیید وجود دارد).**

**متدهای خاص: متد \_\_str\_\_ احتمالاً عنوان طرح را برمی‌گرداند. اگر عنوان موجود نباشد، ممکن است از شناسه یا نام فایل استفاده شود. متد get\_absolute\_url می‌تواند برای هدایت به صفحه‌ی مشاهده جزئیات طرح تعریف شده باشد. override متد save در صورت وجود ممکن است برای مثلاً تغییر نام فایل آپلودی بر اساس شناسه یا عنوان طرح باشد، یا ایجاد Thumbnail از تصویر (اگر طرح تصویری باشد). اما چنین جزئیاتی نیازمند بررسی کد است و در سطح تحلیل ما صرفاً می‌توان اشاره کرد که ذخیره فایل و تغییرات مربوطه توسط Django مدیریت می‌شود مگر خلافش ذکر شده باشد.**

**اپلیکیشن Main (اصلی)**

**اپلیکیشن Main نقش صفحه اصلی یا بخش‌های عمومی سایت را ایفا می‌کند. این اپ معمولاً شامل مدل‌های مربوط به محتواهای صفحه اصلی، نظیر بنر‌ها، بخش معرفی و ... می‌باشد. با توجه به نام فایل‌های آن (admin.py, views.py و ...)، حداقل یک مدل در این اپ تعریف شده است که ممکن است HomePageContent یا LandingInfo نام داشته باشد. محتمل‌ترین مدل‌ها و فیلدهایشان به صورت زیر است:**

* **HomeBanner – مدلی برای نگهداری اسلایدرها یا بنرهای صفحه اصلی. فیلدهای ممکن:**
  + **title: عنوان بنر (CharField).**
  + **image: تصویر بنر (ImageField با مسیر آپلود مثلاً 'banners/').**
  + **link: لینک مرتبط با بنر (URLField یا CharField) در صورت کلیک روی آن.**
  + **is\_active: وضعیت نمایش بنر (BooleanField با default=True برای فعال).**
  + **order: ترتیب نمایش بنر (IntegerField برای مرتب‌سازی دستی بنرها).**

**روابط: این مدل معمولاً رابطه مستقیمی با سایر مدل‌ها ندارد، به جز شاید یک FK به کاربر که بنر را ایجاد کرده (اما معمولاً نیازی نیست).**

**Meta: احتمالاً ordering = ['order'] یا در صورت عدم وجود، بر اساس عنوان یا تاریخ.**

**str: عنوان بنر یا خلاصه‌ای از آن را برمی‌گرداند.**

* **Feature – اگر صفحه اصلی بخش‌هایی برای معرفی امکانات/ویژگی‌ها داشته باشد، مدل Feature می‌تواند هر آیتم را نشان دهد. فیلدها:**
  + **title: عنوان ویژگی.**
  + **description: توضیح کوتاه.**
  + **icon: (در صورت وجود) مسیری به آیکون یا نام کلاس CSS مرتبط.**

**Meta و str مشابه موارد ساده قبلی.**

**با توجه به نبود migrations مشخص، نمی‌توان دقیق گفت چه مدل‌هایی در Main هستند. اما بر اساس الگوی پروژه‌های مشابه، این بخش مدل‌های محتوایی برای صفحه اصلی یا داشبورد عمومی دارد. مدل MainSetting نیز می‌تواند وجود داشته باشد تا تنظیمات کلی سایت (مانند نام سایت، لوگو، متن کپی‌رایت و ...) را نگهداری کند؛ اگرچه پروژه یک اپ جداگانه Settings نیز دارد که احتمالاً همین وظیفه را انجام می‌دهد. به طور خلاصه، اپ Main شامل مدل‌های مرتبط با محتوای ایستا یا نیمه‌دینامیک سایت است و روابط پیچیده‌ای در آن دیده نمی‌شود. متدهای خاص هم عمدتاً str برای نمایش متن مناسب در Admin هستند.**

**اپلیکیشن Notification (اطلاع‌رسانی)**

**اپ Notification جهت مدیریت اعلان‌های سیستم به کاربرها به کار می‌رود. وجود consumers.py و routing.py (مشابه اپ Communication) نشان می‌دهد که این بخش نیز قابلیت بلادرنگ (احتمالاً ارسال نوتیفیکیشن به صورت WebSocket) دارد. مدل اصلی این اپ احتمالاً Notification نام دارد که اعلان‌ها را ذخیره می‌کند. همچنین شاید مدلی برای Device جهت Push Notification (اگر به کار رفته باشد) موجود باشد. در ادامه مدل Notification و جزئیاتش را بررسی می‌کنیم:**

* **Notification – این مدل هر اعلان ارسالی به یک کاربر را نمایش می‌دهد. فیلدهای مهم:**
  + **recipient: کاربری که دریافت‌کننده اعلان است (ForeignKey به User/Profile). اعلان برای این کاربر نمایش داده می‌شود.**
  + **message: متن اعلان (TextField یا CharField با طول کافی). محتوای اعلان که می‌تواند شامل پیام کوتاهی باشد (مثلاً "سفارش شما ارسال شد").**
  + **link: لینک مرتبط با اعلان (مثلاً اگر اعلان درباره سفارش است، لینک به جزئیات سفارش). این فیلد می‌تواند CharField (ذخیره URL به صورت رشته) یا یک GenericForeignKey باشد برای ارتباط با شیء مربوطه، اما روش ساده‌تر همان ذخیره یک URL یا pattern مشخص است. ممکن است blank=True باشد اگر برخی اعلان‌ها لینک خاصی نداشته باشند.**
  + **is\_read: وضعیت خوانده شدن اعلان (BooleanField با default=False). وقتی کاربر اعلان را مشاهده می‌کند، این فیلد True می‌شود.**
  + **created\_at: زمان ایجاد اعلان (DateTimeField auto\_now\_add). ترتیب زمانی ارسال اعلان‌ها را مشخص می‌کند.**

**روابط: هر Notification به یک کاربر به عنوان دریافت‌کننده متصل است. همچنین ممکن است اعلان به یک محتوای خاص در سیستم اشاره کند؛ در این صورت یا از link استفاده شده یا از یک GenericForeignKey (که در Django با ContentType قابل پیاده‌سازی است) برای اشاره به مثلاً یک Order یا Design خاص. اگر GenericForeignKey استفاده شده باشد، دو فیلد مخفی content\_type و object\_id نیز در مدل وجود دارد. اما ساده‌تر این است که فقط متن و لینک ذخیره شود.**

**تنظیمات Meta: ordering = ['-created\_at'] تا جدیدترین اعلان‌ها در بالای لیست قرار بگیرند. verbose\_name = "اعلان" و verbose\_name\_plural = "اعلان‌ها" برای نمایش مناسب.**

**متدهای خاص: متد \_\_str\_\_ احتمالاً خلاصه‌ای از پیام را (مثلاً 20 کاراکتر اول) به همراه نام کاربر مقصد برمی‌گرداند: برای مثال "Notif to {user}: {message\_snippet}...". ممکن است یک متد کمکی برای نشانه‌گذاری به عنوان خوانده‌شده وجود داشته باشد (البته این می‌تواند در منطق view پیاده شود).**

* **Device – (در صورت پشتیبانی از پوش نوتیفیکیشن) این مدل می‌تواند دستگاه‌ها یا توکن‌های اعلان push را ذخیره کند. هر کاربر اگر از طریق موبایل یا مرورگر وب Push Notification دریافت کند، یک شناسه دستگاه/توکن اختصاصی دارد. فیلدهای Device:**
  + **user: کلید خارجی به User که این دستگاه متعلق به اوست.**
  + **device\_token: توکن دستگاه (مثلاً یک رشته طولانی) برای ارسال اعلان push.**
  + **device\_type: نوع دستگاه یا پلتفرم (یک CharField با choices مثلاً 'android'، 'ios'، 'web').**
  + **is\_active: آیا دستگاه فعال است (BooleanField default=True).**

**این مدل رابطه چندبه‌چند غیرمستقیم بین کاربران و دستگاه‌هایشان را نشان می‌دهد (یک کاربر می‌تواند چندین دستگاه داشته باشد).**

**Meta: ordering بر اساس user. str نمایش مختصر توکن یا نوع و کاربر.**

**جمع‌بندی Notification: اپ Notification با نگهداری اعلان‌ها در مدل Notification امکان نمایش اعلان‌های خوانده‌نشده به کاربر را فراهم می‌کند. همچنین با استفاده از Consumers و WebSocket، پس از ایجاد یک Notification در پایگاه‌داده، احتمالاً پیامی بلادرنگ به کاربر ارسال می‌شود. override خاصی در متد save این مدل ممکن است وجود داشته باشد (برای مثال: پس از ذخیره‌ی اعلان جدید، به صورت خودکار سیگنالی جهت ارسال آنی آن به کانال کاربر اجرا شود). اگر چنین موردی باشد، از طریق signals.py یا داخل save به کانال مربوطه پیام فرستاده می‌شود. این جزئیات از حوزه مدل خارج است اما بیانگر نحوه استفاده از مدل Notification در سیستم است.**

**اپلیکیشن Orders (سفارش‌ها)**

**اپ Orders مختص مدیریت سفارش‌های ثبت‌شده توسط کاربران است. مدل اصلی این بخش Order است که جزئیات هر سفارش را نگهداری می‌کند. علاوه بر آن، اگر سفارش شامل آیتم‌های متعدد باشد، ممکن است مدل دیگری مانند OrderItem یا OrderLine نیز وجود داشته باشد (بسته به طراحی سیستم). با توجه به migrationها، دیده شد که در migration 0002\_order\_notes.py فیلد یادداشت به Order اضافه شده است. بنابراین ما حداقل مدل Order را با فیلدهای زیر متصور هستیم:**

* **Order – مدل سفارش. فیلدهای مهم:**
  + **order\_number: شماره یا کد یکتای سفارش (CharField). این فیلد اغلب unique=True تعیین می‌شود تا هر سفارش را متمایز کند (مثلاً "ORD-2025-0001"). اگر تعیین نشده باشد، همان کلید اصلی (ID) به عنوان شناسه سفارش استفاده خواهد شد.**
  + **customer: کاربری که سفارش را داده است (ForeignKey به User/Profile). ممکن است به جای customer از همان فیلد user استفاده شده باشد. on\_delete برای این رابطه غالباً CASCADE یا PROTECT تنظیم می‌شود (PROTECT رایج‌تر است تا با حذف کاربر، سوابق سفارش‌ها محفوظ بماند).**
  + **business: کسب‌وکاری که سفارش از آن درخواست شده (ForeignKey به Business). اگر این سیستم یک مارکت‌پلیس باشد که کاربر از یک کسب‌وکار یا فروشنده خاص خرید می‌کند، این فیلد مشخص می‌کند سفارش مربوط به کدام فروشنده/کسب‌وکار است. (این فیلد می‌تواند null=True باشد در صورتی که مفهوم فروشنده ثالث نباشد).**
  + **status: وضعیت سفارش (CharField با choices). برای مثال: 'pending' (در انتظار), 'processing' (در حال انجام), 'shipped' (ارسال‌شده), 'completed' (تکمیل‌شده), 'canceled' (لغوشده). یک مقدار default='pending' معمولاً برای زمان ایجاد سفارش در نظر گرفته می‌شود.**
  + **total\_amount: مبلغ کل سفارش (DecimalField با دو رقم اعشار، برای قیمت). این فیلد محاسبه مجموع هزینه‌های سفارش است.**
  + **notes: یادداشت‌های کاربر یا فروشنده درباره سفارش (TextField, blank=True). طبق migration 0002 این فیلد اضافه شده و اختیاری است. کاربر می‌تواند در زمان ثبت سفارش توضیحی ارائه دهد، یا ادمین می‌تواند یادداشتی برای پیگیری داخلی ثبت کند.**
  + **created\_at: تاریخ و ساعت ثبت سفارش (DateTimeField auto\_now\_add=True).**
  + **updated\_at: تاریخ و ساعت آخرین به‌روزرسانی سفارش (DateTimeField auto\_now=True).**

**روابط: مدل Order به کاربر و احتمالاً به کسب‌وکار (فروشنده) متصل است. اگر مدل OrderItem برای اقلام سفارش تعریف شده باشد، ارتباط یک‌به‌چند بین Order و OrderItem برقرار است (هر Order شامل چند آیتم). همچنین ممکن است با مدل پرداخت (Transaction) رابطه یک‌به‌یک یا یک‌به‌چند داشته باشد (یک سفارش می‌تواند تراکنش‌های پرداخت داشته باشد که در اپ Payment است).**

**تنظیمات Meta: ordering = ['-created\_at'] تا جدیدترین سفارش‌ها بالاتر باشند. verbose\_name = "سفارش" و verbose\_name\_plural = "سفارش‌ها" برای نمایش مناسب در پنل مدیریت.**

**متدهای خاص: \_\_str\_\_ احتمالاً به شکل f"Order #{order\_number}" یا ترکیب نام مشتری و شناسه سفارش است. اگر کد سفارش تولید شود، ممکن است در متد save در اولین ذخیره‌سازی مقدار order\_number براساس ID یا تاریخ ساخته شود (مثلاً "ORD0001"). مشخص نیست که override save انجام شده باشد، ولی این یک الگوی معمول است. متد get\_absolute\_url می‌تواند کاربر را به صفحه رهگیری یا جزئیات سفارش هدایت کند.**

* **OrderItem – (در صورت وجود) هر رکورد از این مدل نماینده یک قلم یا جزء از سفارش است. مثلاً اگر سفارش شامل چند محصول یا چند طرح باشد، هر کدام یک OrderItem است:**
  + **order: کلید خارجی به Order. همه آیتم‌ها متعلق به یک سفارش مشخص هستند.**
  + **product یا design: اشاره به کالای سفارش داده‌شده. بسته به زمینه پروژه، اگر چیزی برای فروش وجود داشته باشد، اینجا متصل می‌شود. در این پروژه احتمال دارد design‌ها یا templates به عنوان موارد قابل سفارش باشند. بنابراین این فیلد می‌تواند ForeignKey به مدل Design یا Template باشد (اگر کاربران قادر به سفارش یک طرح یا قالب طراحی هستند).**
  + **quantity: تعداد از آن قلم (IntegerField با حداقل 1). اگر هر سفارش صرفاً یک طرح دارد شاید quantity همیشه 1 باشد، اما در صورت امکان چندتایی، این فیلد وجود دارد.**
  + **price: قیمت واحد قلم در زمان سفارش (DecimalField).**
  + **subtotal: قیمت کل این قلم (DecimalField) – که می‌تواند price \* quantity باشد یا از قبل ذخیره شود.**

**روابط: هر OrderItem به یک Order متصل است؛ و به یک محصول/طرح متصل است.**

**Meta: verbose\_name="آیتم سفارش", ordering بر اساس order یا id.**

**str: ممکن است نام محصول به همراه شماره سفارش را برگرداند.**

**بطور کلی، اگر مدل OrderItem نباشد، ممکن است سفارش‌ها تک‌آیتمی باشند یا اقلام در فیلدهای JSON ذخیره شوند که بعید است. لذا حدس زده می‌شود OrderItem یا معادلش در سیستم وجود دارد، هر چند در ساختار اپ اشاره مستقیمی به آن نشده است (شاید جزئی از اپ Orders باشد).**

**اپلیکیشن Payment (پرداخت)**

**اپ Payment وظیفه رسیدگی به پرداخت‌های مالی را بر عهده دارد. این اپ معمولاً مدل‌هایی برای تراکنش‌های پرداخت و perhaps سوابق مالی تعریف می‌کند. بر اساس فایل‌های موجود، یک services.py نیز دارد که شاید منطق ارتباط با درگاه‌های بانکی در آن باشد. مدل‌های کلیدی این بخش احتمالاً Transaction و شاید Invoice یا Payment هستند. در ادامه به مدل Transaction به عنوان اصلی‌ترین مدل می‌پردازیم:**

* **Transaction – مدل نماینده یک تراکنش مالی مرتبط با سفارش:**
  + **order: کلید خارجی به Order (از اپ Orders) که این تراکنش برای آن انجام شده است. (on\_delete=models.CASCADE معمولاً، یا PROTECT اگر نخواهیم با حذف سفارش، تراکنش حذف شود).**
  + **amount: مبلغ پرداخت‌شده (DecimalField با precision مربوط به واحد پولی). معمولاً برابر مبلغ سفارش یا قسط مربوطه است.**
  + **payment\_method: روش یا درگاه پرداخت (CharField با choices). برای مثال: 'online', 'credit\_card', 'wallet'. ممکن است نام درگاه خاص نیز باشد (مثلاً 'Zarinpal', 'Stripe' و ...).**
  + **transaction\_id: شناسه‌ی یکتا دریافتی از درگاه پرداخت (CharField). این می‌تواند شماره تراکنش بانکی یا کد رهگیری باشد. معمولاً unique است زیرا هر تراکنش یک کد دارد. اگر پرداخت از طریق واسط انجام شود، این کد جهت پیگیری بسیار مهم است.**
  + **status: وضعیت تراکنش (CharField با choices). حالت‌های ممکن: 'pending' (در انتظار)، 'paid' یا 'success' (موفق)، 'failed' (ناموفق). default ممکن است 'pending' باشد تا زمان تأیید پرداخت. پس از بازگشت از درگاه، وضعیت به success/failed به‌روزرسانی می‌شود.**
  + **timestamp: زمان ثبت تراکنش در سیستم (DateTimeField auto\_now\_add=True).**
  + **verified\_at: زمان تأیید نهایی تراکنش (DateTimeField، که هنگام موفقیت پرداخت ثبت می‌شود). این فیلد ممکن است null=True باشد و فقط برای تراکنش‌های موفق مقدار بگیرد.**

**روابط: هر تراکنش به یک سفارش متصل است (یک سفارش می‌تواند چند تراکنش داشته باشد ولی معمولاً یکی از آنها موفق خواهد بود). همچنین ممکن است تراکنش به یک کاربر پرداخت‌کننده نیز اشاره کند (اگر مثلاً شخصی که پرداخت می‌کند با شخصی که سفارش را ثبت کرده تفاوت داشته باشد، یا برای پیگیری مالی). اما در اغلب موارد پرداخت‌کننده همان سفارش‌دهنده است و نیاز به فیلد جدا نیست.**

**تنظیمات Meta: ordering = ['-timestamp'] برای نمایش آخرین تراکنش‌ها ابتدا. verbose\_name = "تراکنش" و verbose\_name\_plural = "تراکنش‌ها" تنظیم شده است.**

**متدهای خاص: متد \_\_str\_\_ معمولاً قسمتی از transaction\_id یا وضعیت و مبلغ را نمایش می‌دهد (مثلاً "{order}: {amount} - {status}"). ممکن است متدی مثل is\_successful() تعریف شود که بر اساس status مشخص کند تراکنش موفق بوده یا نه (گرچه این کار را می‌توان با یک مقایسه ساده هم انجام داد). همچنین اگر تعامل با درگاه وجود دارد، ممکن است در services.py توابعی باشد که روی این مدل عمل می‌کنند، مثلاً تأیید تراکنش و به‌روزرسانی status و ثبت verified\_at.**

* **Invoice – (در صورت وجود) این مدل می‌تواند فاکتور یا صورتحساب قابل پرداخت باشد. برخی پروژه‌ها Invoice را جدا از Transaction نگه می‌دارند تا امکان صدور پیش‌فاکتور و پیگیری پرداخت داشته باشند. اگر باشد:**
  + **order: کلید خارجی به Order (مشابه Transaction).**
  + **amount: مبلغ قابل پرداخت (DecimalField).**
  + **due\_date: تاریخ سررسید پرداخت (DateField/DateTimeField) اگر پرداخت باید تا زمان مشخصی انجام شود.**
  + **is\_paid: وضعیت پرداخت فاکتور (BooleanField یا ChoiceField ساده).**
  + **paid\_at: تاریخ پرداخت در صورت پرداخت شدن.**

**اما با توجه به اطلاعات محدود، احتمالاً Invoice مدل جدا تعریف نشده و همان Order/Transaction ترکیب این وظیفه را انجام می‌دهند.**

**خلاصه Payment: این اپ مدل Transaction را برای ذخیره اطلاعات پرداخت آنلاین یا آفلاین پیاده کرده است. در فایل services.py ممکن است توابعی برای ارتباط با API بانک یا درگاه پرداخت تعریف شده باشد که با استفاده از اطلاعات Transaction (amount, id و ...) درخواست پرداخت را می‌سازد یا نتیجه را رسیدگی می‌کند. متد save مدل Transaction احتمالاً override نشده است (چون تغییرات در وضعیت تراکنش از طریق بازگشت بانک و در view یا service اعمال می‌شود، سپس شیء ذخیره می‌شود). با این حال، اگر نیاز بوده هنگام ثبت تراکنش، شماره فاکتور یا شناسه خاصی تولید شود، ممکن است در save یا سیگنال pre\_save این کار انجام شده باشد (مثلاً تنظیم transaction\_id به یک مقدار داخلی قبل از دریافت پاسخ بانک).**

**اپلیکیشن Print\_locations (مکان‌های چاپ)**

**اپ Print\_locations احتمالاً بخشی است که مکان‌های فیزیکی یا شعب چاپ را مدیریت می‌کند. اگر پروژه شامل چاپ فیزیکی طرح‌ها یا مدارک باشد، این اپ مکان‌های قابل انتخاب برای چاپ یا تحویل را معرفی می‌کند. یک مدل اصلی با نام PrintLocation می‌تواند وجود داشته باشد:**

* **PrintLocation – مدل مربوط به یک محل چاپ یا تحویل:**
  + **name: نام مرکز چاپ یا محل (CharField با طول مناسب، مثلاً 100). این نام می‌تواند یکتا باشد (unique=True) تا هر محل یک نام مشخص داشته باشد.**
  + **address: نشانی کامل محل (TextField یا CharField طولانی).**
  + **city: شهر محل (CharField مثلاً max\_length=50) – امکان فیلتر کردن بر اساس شهر.**
  + **phone: شماره تماس محل (CharField max\_length=15) برای ارتباط.**
  + **opening\_hours: ساعت‌های کاری یا توضیح زمان‌بندی (CharField مختصر، یا TextField).**
  + **is\_active: وضعیت فعال بودن محل (BooleanField default=True). ممکن است برای متمایز کردن مکان‌های در دسترس از غیرفعال استفاده شود.**

**روابط: این مدل معمولاً رابطه مستقیمی با سایر مدل‌ها ندارد، اما اگر سفارشات قابلیت انتخاب محل تحویل یا چاپ داشته باشند، ممکن است مدل Order یا Design دارای ForeignKey به PrintLocation باشد. در غیراینصورت، PrintLocation صرفاً یک لیست از مکان‌هاست که شاید در مرحله‌ای از سفارش انتخاب می‌شود.**

**تنظیمات Meta: ordering = ['city', 'name'] می‌تواند باشد تا لیست مکان‌ها مرتب بر اساس شهر و نام باشد. verbose\_name = "محل چاپ" و verbose\_name\_plural = "محل‌های چاپ" تعریف شده است.**

**متدهای خاص: str شاید نام و شهر محل را برگرداند (مثلاً "چاپخانه {name} - {city}"). متد خاص دیگری لازم نیست.**

**چون این اپ صرفاً یک مدل مرجع دارد، پیچیدگی چندانی در متدها یا روابط وجود ندارد. هدف آن ارائه لیستی از گزینه‌ها برای انتخاب مکان چاپ/تحویل است. هیچ override در save نیاز نیست؛ ورود/ویرایش این اطلاعات از طریق پنل مدیریت یا اسکریپت‌های ابتدایی انجام می‌گیرد.**

**اپلیکیشن Reports (گزارش‌ها)**

**اپ Reports برای تولید و مدیریت گزارش‌های سیستمی استفاده می‌شود. این گزارش‌ها می‌توانند خروجی‌های تحلیلی یا اسناد PDF باشند. مدل اصلی این اپ احتمالاً Report یا ReportFile است. همچنین presence فایل utils.py نشان می‌دهد که توابع کمکی برای ساخت گزارش (مثلاً ایجاد فایل PDF یا Excel) وجود دارد. مدل‌های این قسمت عبارت‌اند از:**

* **Report – مدل اصلی گزارش:**
  + **title: عنوان گزارش (CharField، مثلاً "گزارش فروش مهر ۱۴۰۴").**
  + **type: نوع گزارش (CharField با choices). برای مثال: 'sales', 'users', 'performance' و ... یا کلمات معادل فارسی. این نوع تعیین می‌کند گزارش چه محتوایی دارد. ممکن است از این فیلد برای دسته‌بندی یا انتخاب قالب گزارش استفاده شود.**
  + **generated\_at: تاریخ ایجاد گزارش (DateTimeField auto\_now\_add).**
  + **file: فایل گزارش تولید‌شده (FileField، که معمولاً یک PDF یا Excel است و در مسیر مشخصی ذخیره می‌شود، مثلاً reports/ در مدیا). این فیلد می‌تواند null باشد تا زمانیکه تولید گزارش کامل شود، سپس مسیر فایل را تنظیم کنند.**
  + **creator: کاربری که گزارش را ایجاد کرده (ForeignKey به User). اگر گزارش‌ها توسط کاربران (مثلاً مدیر سیستم) قابل ایجاد باشد، این فیلد ثبت می‌کند چه کسی آن را ساخته.**
  + **is\_public: آیا گزارش برای سایر کاربران/مشتریان نیز قابل مشاهده است (BooleanField با default=False). برخی گزارش‌ها ممکن است فقط برای ادمین باشند.**

**روابط: Report به کاربر سازنده متصل است. همچنین ممکن است به یک Business یا بخش خاص مرتبط باشد اگر گزارش مخصوص آن بخش تولید شده (مثلاً گزارش فروش یک کسب‌وکار؛ در این صورت یک FK به Business می‌تواند وجود داشته باشد). از اطلاعات موجود چنین چیزی صریح نیست ولی قابل‌تصور است.**

**تنظیمات Meta: ordering = ['-generated\_at'] تا جدیدترین گزارش‌ها ابتدا فهرست شوند. verbose\_name = "گزارش" و verbose\_name\_plural = "گزارش‌ها" تنظیم شده است.**

**متدهای خاص: str احتمالاً عنوان گزارش را برمی‌گرداند. ممکن است متدهایی برای تولید گزارش وجود داشته باشد، اما این منطق احتمالاً در فایل utils.py یا در سیگنال/Taskهای جدا پیاده شده است (مثلاً تابعی که داده‌ها را جمع‌آوری کرده و فایل را می‌سازد، سپس فیلد file را در مدل Report به‌روزرسانی می‌کند). بنابراین متد save مدل Report می‌تواند override شده باشد تا به محض ایجاد شیء Report، یک فرایند تولید گزارش را آغاز کند (مثلاً یک تسک asynchronous یا یک تابع utility). بسته به سطح پیچیدگی، این می‌تواند صورت گرفته باشد اما جزئیات آن مشخص نیست.**

* **ReportTemplate – (حدس اختیاری) ممکن است قالب‌های آماده‌ای برای گزارش‌ها وجود داشته باشد (مثل فایل‌های template HTML یا عناوین). اگر چنین بوده، مدل کوچکی برای نگهداری قالب‌ها می‌تواند تعریف شده باشد، ولی باتوجه به اینکه اپ Templates\_app جدا وجود دارد، شاید از همان برای قالب‌های گزارش نیز استفاده شود. پس احتمالاً نیازی به مدل جداگانه Template در این بخش نیست.**

**اپلیکیشن Set\_design (طراحی مجموعه‌ها)**

**اپ Set\_design مرتبط با چیزی به نام "طراحی ست" یا "مجموعه طراحی" است. احتمالاً مفهومی در دامنه پروژه است که طرح‌ها را در قالب یک پکیج یا ست گرد هم می‌آورد. مدل اصلی این قسمت SetDesign نام دارد. همچنین طبق migrationها، یک سیگنال (signals.py) در این اپ وجود دارد که ممکن است برای انجام عملیاتی هنگام ایجاد یا تغییر SetDesign باشد (مثلاً ایجاد پوشه یا اطلاع‌رسانی). مشخصات مدل SetDesign به شکل زیر است:**

* **SetDesign – مدل یک مجموعه طراحی:**
  + **title: عنوان مجموعه (CharField مثلاً max\_length=100). عنوانی که نشان‌دهنده نوع ست یا پکیج طراحی است.**
  + **description: توضیح درباره این مجموعه (TextField اختیاری).**
  + **designs: طرح‌های متعلق به این مجموعه. این می‌تواند یک ManyToManyField به مدل Design (در اپ designs) باشد تا مشخص کند کدام طرح‌ها در این ست قرار گرفته‌اند. هر SetDesign می‌تواند شامل چند Design باشد و یک Design می‌تواند در چند SetDesign عضو باشد (در صورتی که منطقی باشد، یا شاید هر Design فقط در یک مجموعه باشد که آن‌وقت می‌توانست FK باشد؛ اما احتمالاً چندبه‌چند منطقی‌تر است چون شاید یک طرح عمومی در چند مجموعه استفاده شود).**
  + **creator: سازنده یا مسئول مجموعه (ForeignKey به User/Business). مشخص می‌کند چه کسی این SetDesign را ایجاد کرده یا مالک آن است.**
  + **status: وضعیت مجموعه (Draft/Finalized/etc. مشابه طرح، این نیز می‌تواند choices داشته باشد). برای نمونه ممکن است 'incomplete' و 'complete' برای تعیین کامل بودن مجموعه.**
  + **created\_at: زمان ایجاد مجموعه (DateTimeField auto\_now\_add).**
  + **updated\_at: زمان آخرین بروزرسانی مجموعه (DateTimeField auto\_now).**

**روابط: همانطور که گفته شد، ارتباط اصلی SetDesign با مدل Design به صورت چند‌به‌چند است. همچنین SetDesign با کاربر (یا کسب‌وکار) سازنده رابطه دارد. ممکن است اگر SetDesign یک خروجی قابل سفارش باشد، با مدل Order نیز ارتباط پیدا کند (به‌عنوان آیتم سفارش). اما چنین ارتباط مستقیمی در سطح مدل تعریف نمی‌شود مگر اینکه به عنوان ForeignKey در OrderItem به SetDesign اشاره شود در صورت نیاز.**

**تنظیمات Meta: ordering = ['title'] یا ['-created\_at'] بسته به شیوه نمایش مورد نظر. از آنجا که ست‌ها ممکن است نام‌گذاری مشخص داشته باشند، احتمال دارد ترتیب پیش‌فرض بر اساس title الفبایی باشد. verbose\_name = "مجموعه طرح" و verbose\_name\_plural = "مجموعه طرح‌ها" تنظیم شده است.**

**متدهای خاص: str معمولاً عنوان مجموعه را برمی‌گرداند. در signals.py این اپ، احتمالاً هنگام ایجاد یا تغییر SetDesign عملی انجام می‌شود؛ به عنوان مثال، پس از ایجاد SetDesign شاید یک پوشه در سیستم فایل ساخته شود یا پس از ذخیره، اگر تمام طرح‌های مورد نیاز اضافه شده باشند، وضعیت complete شود. این سیگنال‌ها می‌توانند متد save را تحت تأثیر قرار دهند (مثلاً override save ممکن است سیگنال را صدا بزند، اما در Django معمولاً سیگنال‌ها مستقل ثبت می‌شوند). بنابراین متد save احتمالاً override نشده اما سیگنال‌های post\_save و m2m\_changed برای این مدل تعریف شده‌اند تا در صورت هر تغییری، اقدامات مقتضی (مثلاً ثبت گزارش یا اطلاع‌رسانی) صورت گیرد.**

**اپلیکیشن Settings (تنظیمات)**

**اپ Settings وظیفه ذخیره تنظیمات قابل تغییر در زمان اجرای سیستم را دارد. این اپ شبیه به یک جفت کلید-مقدار (key-value store) برای تنظیمات سایت یا برنامه است. مدل اصلی آن احتمالاً Setting یا SiteSetting نام دارد که مقادیر تنظیمات را نگهداری می‌کند:**

* **Setting – مدل تنظیمات:**
  + **key: کلید تنظیم (CharField). نام انگلیسی یا شناسه تنظیم، مثلاً "SITE\_NAME", "MAX\_UPLOAD\_SIZE" و امثال آن. این فیلد unique=True تعریف شده تا هر کلید تنظیم یک بار ذخیره شود.**
  + **value: مقدار تنظیم (TextField). مقدار به صورت رشته ذخیره می‌شود، چه برای مقادیر عددی، چه بولین و ... . ممکن است برای راحتی، سیستم تبدیل نوع را هنگام استفاده انجام دهد.**
  + **description: توضیح درباره این تنظیم (CharField یا TextField اختیاری) که کاربری آن را بهتر بفهمد. مثلاً "نام نمایشی سایت که در عنوان صفحات نشان داده می‌شود".**
  + **data\_type: نوع داده‌ای که این مقدار نمایندگی می‌کند (CharField با choices احتمالی: 'string','int','bool','json' و ...). این فیلد شاید وجود داشته باشد تا بدانیم چگونه از value استفاده کنیم (برای مثال اگر 'bool' باشد، ارزش True/False است). در پروژه‌های کوچک ممکن است چنین فیلدی تعریف نشده باشد و صرفاً از مقدار رشته‌ای استفاده کنند.**

**روابط: این مدل هیچ رابطه خارجی با مدل‌های دیگر ندارد؛ هر Setting مستقل است.**

**تنظیمات Meta: verbose\_name = "تنظیم" و verbose\_name\_plural = "تنظیمات". اگر تعداد تنظیمات محدود و ثابت باشد، شاید از migrations اولیه برای درج آنها استفاده شده باشد. ordering = ['key'] ممکن است برای مرتب‌سازی الفبایی کلیدها تنظیم شده باشد.**

**متدهای خاص: str معمولاً کلید تنظیم را برمی‌گرداند. ممکن است متدی مانند get\_value() تعریف شود که بر اساس data\_type مقدار value را cast کند (مثلاً به bool یا int تبدیل کند). اما این را هم می‌توان در سطح کاربرد انجام داد. override save معمولاً لازم نیست، مگر برای تبدیل خودکار مقدار True/False به 'True'/'False' رشته‌ای یا تضمین uppercase بودن کلید.**

**با استفاده از این مدل، توسعه‌دهندگان می‌توانند برخی مقادیر را بدون نیاز به تغییر کد از طریق پنل ادمین تغییر دهند. نمونه‌هایی از چنین تنظیماتی: عنوان سایت، توضیحات متا، تنظیمات ایمیل (SMTP) و ... است که ترجیح داده‌اند در پایگاه‌داده نگهداری شوند.**

**اپلیکیشن Templates\_app (قالب‌ها)**

**اپ Templates\_app مسئول مدیریت قالب‌های از پیش ساخته شده در سیستم است. این قالب‌ها ممکن است طرح‌های آماده (جهت استفاده کاربران)، قالب‌های سند (مثلاً برای گزارش یا ایمیل) یا هر نوع الگوی دیگری باشند. بر اساس migrations موجود، مدل اصلی باید Template باشد. تغییراتی در migration 0002 نشان می‌دهد که فیلد‌های creator و status و موارد دیگر در مدل Template بروزرسانی شده‌اند. در زیر، مدل Template تشریح می‌شود:**

* **Template – مدل قالب:**
  + **name: نام قالب (CharField). این نام قالب را معرفی می‌کند (مثلاً "فرم قرارداد استاندارد" یا "طرح آماده شماره 1"). unique=True احتمالاً تنظیم شده تا نام هر قالب تکراری نباشد.**
  + **content: محتوای قالب (TextField یا FileField). اگر قالب یک متن یا HTML باشد (مثلاً قالب ایمیل یا گزارش)، احتمالاً به صورت TextField ذخیره می‌شود. اگر قالب یک فایل (مثلاً قالب گرافیکی PSD یا یک فایل زیپ) باشد، ممکن است از FileField استفاده شود.**
  + **creator: سازنده قالب (ForeignKey به User). طبق migration 0002 این فیلد اضافه یا تغییر یافته است. مشخص می‌کند چه کسی قالب را ایجاد کرده؛ شاید برای تفکیک قالب‌های سیستمی از قالب‌های کاربران یا طراحان. ممکن است در ابتدا optional تعریف شده باشد (null=True) و سپس اجباری گردد.**
  + **status: وضعیت قالب (ChoiceField). migration 0002 اشاره کرده که status و چند فیلد alter شده‌اند؛ شاید status قبلاً Boolean (برای انتشار/عدم انتشار) بوده و سپس تبدیل به چندوضعیتی شده است. اکنون احتمالاً status می‌تواند مقادیری چون 'draft'، 'published'، 'archived' داشته باشد. default ممکن است 'draft' باشد تا قالب پس از ایجاد نیاز به تأیید یا انتشار داشته باشد.**
  + **category: (اگر در "and\_more" migration اشاره شده) ممکن است یک فیلد جدید مثلاً category یا type اضافه شده باشد تا قالب‌ها را دسته‌بندی کند. برای مثال، قالب‌ها می‌توانند دسته "قرارداد", "گزارش", "طرح گرافیکی" و ... داشته باشند. این فیلد می‌تواند CharField با choices یا ForeignKey به یک مدل Category باشد.**
  + **created\_at: زمان ایجاد قالب (DateTimeField auto\_now\_add).**
  + **updated\_at: زمان آخرین ویرایش قالب (DateTimeField auto\_now).**

**روابط: هر Template به یک کاربر سازنده متصل است. اگر Category یا مشابه باشد، Template ممکن است FK به مدل Category (در همین اپ یا اپ دیگر) داشته باشد. همچنین اگر برخی قالب‌ها وابسته به Business باشند (مثلاً هر کسب‌وکار قالب‌های خودش را داشته باشد)، امکان FK به Business هم هست، اما در صورت نبود شواهد، همان Category عمومی‌تر است.**

**تنظیمات Meta: ordering = ['name'] تا قالب‌ها به ترتیب نام مرتب شوند (اگر تعداد کم اما نام‌دار هستند). یا شاید براساس تاریخ ایجاد (-created\_at) اگر عمدتاً تازه‌ترین‌ها مهم باشند. verbose\_name="قالب" و verbose\_name\_plural="قالب‌ها" تعریف شده است.**

**متدهای خاص: str احتمالاً نام قالب را بازمی‌گرداند. اگر قالب‌ها انواع مختلف دارند، ممکن است get\_type\_display برای نمایش نوع استفاده شود. متد save ممکن است override شده باشد تا مثلاً اگر قالب انتشار یافت (status از draft به published تغییر کرد)، کاری انجام دهد (مثلاً ثبت در لاگ یا اطلاع‌رسانی). اما به طور کلی، فرآیند انتشار و تغییر وضعیت می‌تواند در viewهای ادمین یا کد تجاری انجام گیرد و نیاز به override save نباشد.**

* **TemplateCategory – (در صورت وجود) اگر دسته‌بندی قالب‌ها به صورت مدل جدا در نظر گرفته شده باشد: شامل نام دسته (مثل "قرارداد", "طرح", "متن") و شاید ترتیب نمایش. Template می‌تواند FK به TemplateCategory داشته باشد. اما عدم اشاره مستقیم، احتمال وجود این مدل را کم‌رنگ می‌کند.**

**نتیجه‌گیری Templates\_app: این بخش از پروژه به مدیران اجازه می‌دهد قالب‌های آماده (طراحی‌ها، اسناد یا هر نوع دیگری) را ایجاد و مدیریت کنند تا کاربران نهایی بتوانند از آن‌ها استفاده نمایند. به عنوان مثال، اگر پروژه درباره طراحی باشد، یک کاربر می‌تواند از قالب‌های آماده طرح استفاده کند. مدل Template با نگهداری محتوای قالب و اطلاعات سازنده و وضعیت، این هدف را برآورده می‌کند. متدهای مدل عمدتاً جهت نمایش بهتر اطلاعات در Admin است و منطق خاص (مثل تولید خروجی از قالب یا اعمال آن) احتمالاً در viewها یا utilityها پیاده شده است.**

**اپلیکیشن Tender (مناقصه/مزایده)**

**اپ Tender به صورت اختصاصی برای مدیریت مناقصه‌ها یا پروژه‌های رقابتی ایجاد شده است. اگرچه قبلاً در Core به مدل Tender اشاره شد، این اپ جدید احتمالاً حاوی نسخه نهایی مدل‌های Tender و وابسته‌های آن است. بر اساس migration این اپ، یک migration اولیه (0001\_initial.py) وجود دارد که باید شامل ساختار مدل‌های Tender باشد. مدل‌های اصلی در این قسمت عبارت‌اند از Tender و احتمالاً Bid (پیشنهاد). مدل Award ممکن است به خاطر سادگی در خود Tender یا Bid گنجانده شده باشد (مثلاً فیلد winner در Tender یا فیلد is\_winner در Bid). جزئیات مدل Tender در این اپ:**

* **Tender – مدل مناقصه/پروژه:**
  + **title: عنوان مناقصه (CharField).**
  + **description: توضیحات کامل مناقصه (TextField).**
  + **owner: کاربری که مناقصه را ایجاد کرده یا درخواست‌کننده پروژه است (ForeignKey به User/Business). اگر مناقصه را یک شرکت ایجاد می‌کند، اینجا می‌تواند به Business هم مرتبط شود؛ اما ساده‌تر این است که به user متصل باشد و در صورت نیاز از طریق پروفایل بفهمیم شرکت مربوطه چیست.**
  + **start\_date: تاریخ شروع قبول پیشنهادات (DateTimeField).**
  + **end\_date: تاریخ پایان مهلت مناقصه (DateTimeField). پس از این تاریخ دیگر پیشنهادی پذیرفته نمی‌شود.**
  + **budget: بودجه یا قیمت پایه/تخمینی برای مناقصه (DecimalField). ایجاد‌کننده ممکن است یک رقم پیشنهادی یا حداکثر بودجه برای پروژه تعیین کند. این فیلد می‌تواند null باشد اگر بودجه اعلام نشده.**
  + **status: وضعیت مناقصه (CharField با choices). حالت‌های ممکن: 'open' (باز برای پیشنهاد)، 'closed' (مهلت تمام‌شده)، 'awarded' (برنده انتخاب‌شده)، 'cancelled' (لغوشده). default احتمالاً 'open' تا زمان پایان مهلت.**
  + **winner: (اختیاری) اگر مناقصه به پایان برسد و برنده‌ای انتخاب شود، این فیلد برنده را مشخص می‌کند. می‌تواند ForeignKey به Bid برنده یا مستقیم به User برنده باشد. روش معمول‌تر، ForeignKey به Bid است تا جزئیات پیشنهاد برنده در دسترس باشد. این فیلد می‌تواند null=True باشد تا قبل از انتخاب برنده خالی بماند.**

**روابط: Tender به کاربر ایجاد‌کننده (و احتمالاً Business وی) با FK مرتبط است. همچنین رابطه یک‌به‌چند با Bid دارد (هر Tender چندین Bid دریافت می‌کند). رابطه برنده نیز یا FK به Bid است (یک‌به‌یک ضمنی: هر Tender حداکثر یک برنده دارد).**

**تنظیمات Meta: ordering = ['-start\_date'] یا ['-end\_date'] تا مناقصات جاری یا تازه پایان‌یافته بالاتر باشند. verbose\_name="مناقصه" و verbose\_name\_plural="مناقصات" در نظر گرفته شده است.**

**متدهای خاص: str عنوان مناقصه را برمی‌گرداند. شاید متدی مثل is\_open تعریف شده باشد که بررسی کند اکنون (timezone.now) بین start\_date و end\_date است یا نه، برای سهولت. متد award\_bid(bid) می‌تواند وجود داشته باشد که یک Bid را به عنوان برنده تنظیم کند (status را 'awarded' کند، winner را تنظیم کند و شاید به آن Bid علامت برنده بودن بدهد). در غیاب این متد، این منطق در view یا سیگنال جای می‌گیرد.**

* **Bid – مدل پیشنهاد ارائه‌شده برای مناقصه:**
  + **tender: کلید خارجی به Tender که این پیشنهاد مربوط به آن است (on\_delete=models.CASCADE، چون با حذف مناقصه، پیشنهادهایش بی‌معنی می‌شوند).**
  + **bidder: کاربری که پیشنهاد را ارائه داده (ForeignKey به User/Business). ممکن است از Profile بتوان شرکت/تیم وی را استخراج کرد.**
  + **amount: مبلغ پیشنهادی (DecimalField).**
  + **description: توضیح یا جزئیات پیشنهاد (TextField اختیاری). ارائه‌دهنده می‌تواند توضیح دهد چگونه پروژه را انجام می‌دهد، یا شرایطش چیست.**
  + **submitted\_at: زمان ارسال پیشنهاد (DateTimeField auto\_now\_add).**
  + **is\_winner: آیا این پیشنهاد برنده شده است یا خیر (BooleanField با default=False). در صورت انتخاب شدن این پیشنهاد به عنوان برنده مناقصه، این فیلد True می‌شود. ممکن است به جای این فیلد، همانطور که اشاره شد، رابطه winner در Tender استفاده شود؛ اما برای راحتی نمایش لیست پیشنهادها، این فیلد هم مفید است.**

**روابط: هر Bid به یک Tender و یک کاربر متصل است. رابطه برنده بودن: یا از طریق Tender.winner یا همین فیلد is\_winner مشخص می‌شود (بسته به طراحی، شاید هر دو راه استفاده شده باشد که کمی افزونه است، ولی برای سرعت Query مفید).**

**تنظیمات Meta: ordering = ['amount'] (مثلاً مرتب‌سازی بر اساس مبلغ از کم به زیاد) یا ['-submitted\_at'] (بر اساس زمان ارسال). اگر کمترین قیمت ملاک است، ممکن است مرتب‌سازی پیش‌فرض بر amount صعودی باشد. verbose\_name="پیشنهاد" و verbose\_name\_plural="پیشنهادها".**

**متدهای خاص: str می‌تواند مثلاً f"{bidder} - {amount}" باشد. اگر is\_winner وجود دارد، ممکن است در Admin با یک فیلد لیست نمایشی خاص مشخص شود. override save در این مدل لازم نیست مگر اینکه مثلاً پس از ذخیره یک پیشنهاد جدید، بخواهیم نوتیفیکیشنی به صاحب مناقصه بفرستیم (که آن هم احتمالاً از طریق سیگنال یا view انجام می‌شود).**

**Award مدل جداگانه‌ای در این اپ شاید دیگر وجود نداشته باشد؛ زیرا تعیین برنده از طریق Tender.winner و Bid.is\_winner قابل انجام است. بنابراین Award که قبلاً در Core بود، احتمالاً حذف شده یا بدون استفاده مانده است.**

**اپلیکیشن Workshop (کارگاه)**

**اپ Workshop به مدیریت کارگاه‌ها، پروژه‌های عملی یا سلسله عملیات گروهی اختصاص دارد. این همان مدل‌هایی را شامل می‌شود که ابتدا در Core دیده شدند: Workshop, WorkshopTask, WorkshopReport. اکنون این مدل‌ها در اپ Workshop تعریف شده‌اند و مربوط به هم کار می‌کنند:**

* **Workshop – مدل کارگاه یا پروژه:**
  + **title: عنوان کارگاه (CharField).**
  + **description: توضیحات کلی درباره هدف و جزئیات کارگاه (TextField).**
  + **manager: کاربری که مدیر یا مسئول کارگاه است (ForeignKey به User/Business). این شخص ناظر بر کارگاه و وظایف آن است.**
  + **start\_date: تاریخ شروع کارگاه (DateTimeField).**
  + **end\_date: تاریخ پایان یا مهلت اتمام کارگاه (DateTimeField).**
  + **status: وضعیت جاری کارگاه (ChoiceField یا CharField). حالت‌هایی ممکن است شامل 'planning' (در حال برنامه‌ریزی), 'in\_progress' (در حال اجرا), 'completed' (به اتمام رسیده) و 'on\_hold' (متوقف شده) باشند. default شاید 'planning' یا 'in\_progress' باشد بسته به این که کارگاه بلافاصله شروع می‌شود یا ابتدا برنامه‌ریزی می‌شود.**
  + **created\_at: تاریخ ایجاد رکورد کارگاه (DateTimeField auto\_now\_add).**

**روابط: Workshop به یک کاربر به عنوان مدیر متصل است. همچنین ممکن است به یک Business یا Project بزرگ‌تر مربوط باشد (اگر کارگاه‌های متعدد زیرمجموعه یک پروژه باشند). اما چنین رابطه‌ای ذکر نشده، احتمالاً وجود ندارد.**

**تنظیمات Meta: ordering = ['-start\_date'] یا ['title']. بسته به نحوه نمایش. verbose\_name="کارگاه" و verbose\_name\_plural="کارگاه‌ها".**

**متدهای خاص: str عنوان را برمی‌گرداند. احتمال دارد متدی مثل get\_progress() تعریف شده باشد که درصد تکمیل وظایف را حساب کند (از روی Taskها). همچنین در سیگنال‌های اپ Core دیده شد که migrationهایی برای این مدل بوده، شاید logic خاصی به راه‌اندازی کارگاه مربوط است (اما بعید است save override شده باشد).**

* **WorkshopTask – مدل وظیفه یا مرحله داخل کارگاه:**
  + **workshop: کلید خارجی به Workshop. هر وظیفه متعلق به یک کارگاه است.**
  + **title: عنوان وظیفه (CharField).**
  + **description: توضیح وظیفه (TextField اختیاری).**
  + **assigned\_to: کاربری که مسئول انجام این Task است (ForeignKey به User). ممکن است یک Task به یک نفر محول شود. اگر Task عمومی باشد، این فیلد می‌تواند null بماند.**
  + **status: وضعیت Task (ChoiceField). مثلاً 'pending', 'doing', 'done', 'cancelled'. default='pending'.**
  + **order: ترتیب یا اولویت Task در کارگاه (IntegerField). اگر Taskها پشت‌سرهم اجرا شوند، با این شماره مرتب می‌شوند.**
  + **due\_date: مهلت اتمام Task (DateTimeField یا DateField).**
  + **completed\_at: تاریخ علامت‌گذاری به عنوان انجام‌شده (DateTimeField null=True که وقتی status = 'done' شد ثبت می‌گردد).**

**روابط: هر Workshop می‌تواند چندین Task داشته باشد (یک به چند). هر Task به یک کاربر مسئول (assign شده) متصل است.**

**تنظیمات Meta: ordering = ['order'] تا Tasks به ترتیب مشخص‌شده فهرست شوند. verbose\_name="وظیفه" و verbose\_name\_plural="وظایف کارگاه".**

**متدهای خاص: str عنوان و شماره Task را (مثلاً "#1 - title") برمی‌گرداند. ممکن است متدی مثل mark\_done() وجود داشته باشد که status را done کند و completed\_at را ثبت کند (در صورت عدم استفاده از save override، این می‌تواند در view انجام شود).**

* **WorkshopReport – مدل گزارش کارگاه:**
  + **workshop: کلید خارجی به Workshop مربوطه.**
  + **author: نویسنده گزارش (ForeignKey به User). معمولاً مدیر یا شخص مسئول گزارش.**
  + **content: محتوای گزارش (TextField). متن کامل گزارش که می‌تواند شامل نتایج کارگاه، خلاصه کارهای انجام‌شده و ... باشد.**
  + **created\_at: تاریخ تهیه گزارش (DateTimeField auto\_now\_add).**

**روابط: هر کارگاه می‌تواند ۰ یا چند گزارش داشته باشد (مثلاً گزارش‌های دوره‌ای). هر گزارش توسط یک کاربر نوشته شده است.**

**تنظیمات Meta: ordering = ['-created\_at'] (آخرین گزارش‌ها ابتدا). verbose\_name="گزارش کارگاه" و verbose\_name\_plural="گزارش‌های کارگاه".**

**متدهای خاص: str ممکن است بخشی از متن یا عنوان فرضی گزارش را برگرداند (اگر content طولانی است، اولین جمله یا عبارت).**

**سیگنال‌ها: حضور فایل signals.py در اپ Workshop نشان می‌دهد احتمالاً سیگنالی تنظیم شده که مثلاً با اتمام تمامی وظایف یک کارگاه، وضعیت Workshop را به completed تغییر دهد. یا شاید پس از ایجاد یک Task جدید، نوتیفیکیشنی به assigned\_to ارسال شود. این منطق سیستمی معمولاً در سیگنال‌ها کدگذاری می‌شود. بنابراین، اگر همه Taskهای یک Workshop done شوند، یک سیگنال post\_save روی Taskها می‌تواند Workshop.status را به 'completed' بروز کند[github.com](https://github.com/amin311/pardach/commit/b7148cb9beedb500cf0e2dbf4a52a853d7f47084" \l ":~:text=" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/commit/b7148cb9beedb500cf0e2dbf4a52a853d7f47084#:~:text=)**. این نوع رفتارهای خودکار جزو تحلیل مدل در سطح Meta و فیلد نیست اما تکمیل‌کننده فهم ما از روابط مدل‌هاست.**

**نتیجه کلی: هر اپلیکیشن در پروژه Pardach مدل‌های مرتبط با حوزه کاری خودش را تعریف کرده است. این مدل‌ها از اصول رایج Django پیروی می‌کنند: فیلدهای CharField/TextField برای متن‌ها، ForeignKey برای ارتباطات یک‌به‌چند، ManyToMany در موارد لازم، BooleanField برای حالات دوگانه و استفاده از choices برای تعیین حالات ممکن در فیلدهای وضعیت. همچنین در همه مدل‌ها سعی شده از امکانات Meta برای نام‌گذاری فارسی مدل‌ها و مرتب‌سازی پیش‌فرض استفاده شود. متد str تقریباً در تمامی مدل‌ها برای نمایش مناسب شیء در بخش مدیریت override شده است. متدهای دیگری نظیر save فقط در مواقع نیاز خاص (مانند تولید کد یکتا یا بروزرسانی خودکار فیلدهای وابسته) بازنویسی شده‌اند.**

**این ساختار ماژولار باعث می‌شود هر بخش از پروژه مسؤولیت داده‌ای مشخصی داشته باشد و خوانایی و نگهداری کد تسهیل گردد.**

**Sources**

**You said:**

**djang-react.txt**

**Document**

**خب من متن گفتگو خود با هوش مصنوعی که من خواسته های پروژه را گفته ام و او پیشنهاد هایی به من داده است با توجه به مدل هایمان ببین چقدر با سناریوهای گفتگوی ما مطابقت دارد**

**ChatGPT said:**

**آیا هدف شما این است که من بررسی کنم مدل‌هایی که در پروژه تعریف کرده‌اید، تا چه اندازه با سناریوها و اهدافی که در گفتگوهای قبلی (در فایل آپلودشده) مطرح شده‌اند، هم‌خوانی دارند؟ همچنین آیا انتظار دارید نقاط قوت، ضعف، یا کسری‌های احتمالی مدل‌ها را هم برایتان مشخص کنم؟**

**You said:**

**بله**

**ChatGPT said:**

**بسیار عالی. من فایل گفت‌وگوی شما با هوش مصنوعی را بررسی خواهم کرد تا سناریوها، اهداف و پیشنهادهای مطرح‌شده در آن را استخراج کنم و سپس آن‌ها را با مدل‌های تعریف‌شده در پروژه شما مقایسه کنم.**

**گزارش به‌صورت دقیق شامل نقاط تطابق، عدم‌تطابق و پیشنهادهای بهبود در طراحی مدل‌ها خواهد بود.**

**Research completed in 8m · 1 source**

**ارزیابی تطابق مدل‌های پروژه با سناریوهای گفت‌وگو با هوش مصنوعی**

**گزارش مقایسه سناریوهای پروژه Pardach با مدل‌های فعلی**

**اهداف و نیازهای پروژه**

**پروژه Pardach یک پلتفرم جامع برای مدیریت سفارش‌های چاپ سفارشی پوشاک است که باید فرآیندهای متنوعی را پشتیبانی کند. بر اساس گفت‌وگو، سیستم باید امکان ثبت سفارش چاپ لباس به روش‌های مختلف (انتخاب مستقیم طرح برای بخش‌های لباس، انتخاب از کاتالوگ طرح‌ها، استفاده از قالب‌های آماده با بخش‌های متغیر، یا انتخاب مجموعه/خانواده‌ای از طرح‌ها) را فراهم کندfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. همچنین نقش‌ها و داشبوردهای متعددی پیش‌بینی شده است: داشبورد مشتریان برای پیگیری سفارش‌ها، پنل کسب‌وکارها برای مدیریت سفارش‌ها، طرح‌ها و پرداخت‌ها، و بخش طراحان جهت آپلود و ویرایش طرح‌های خودfile-crzskoe9o4wznmpsinhc2g. نیازهای دیگر شامل مدیریت طرح‌ها (برچسب‌گذاری، دسته‌بندی، خانواده‌ها و آپلود گروهی)، مکانیزم مناقصه برای برون‌سپاری مراحل خاص تولید (مثل گلدوزی)، مراحل تکمیل سفارش در کارگاه‌ها (پیگیری وظایف و ظرفیت کارگاه)، اطلاع‌رسانی وضعیت به کاربران، و مدیریت پرداخت و فاکتور می‌باشد. در ادامه، سناریوهای کلیدی مطرح‌شده در گفت‌وگو مرور و هر کدام نسبت به مدل‌های فعلی backend (پوشه backend/apps) سنجیده می‌شود.**

**سناریوهای کلیدی و مقایسه با مدل‌ها**

**سناریو ۱: ثبت سفارش و انتخاب طرح‌ها**

**در این سناریو مشتری می‌تواند یک سفارش چاپ لباس را ایجاد و طرح‌های دلخواه را برای بخش‌های مختلف لباس انتخاب کند. این فرآیند چند حالت دارد: مثلاً انتخاب مستقیم طرح برای یک بخش (مثل آستین) با نمایش طرح‌های مناسب در یک پاپ‌آپ، انتخاب طرح از کاتالوگ و اختصاص آن به بخشی از لباس، استفاده از یک قالب آماده با چند جای خالی برای طرح‌های قابل تغییر، یا انتخاب یک خانواده طرح تا سیستم بخش‌های مختلف را خودکار پر کندfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g.**

* **مدل‌های مرتبط در پروژه: مدل اصلی *Order* (سفارش) برای نگهداری اطلاعات کلی هر سفارش وجود داردfile-crzskoe9o4wznmpsinhc2g. این مدل در ابتدا شامل فیلدهایی مانند کاربر سفارش‌دهنده، وضعیت (با مقادیر پیش‌فرض *pending* و ...)، و مبلغ کل بوده استfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. جهت جزئیات هر سفارش، مدل‌های کمکی متعددی تعریف شده‌اند:**
  + ***OrderItem*: آیتم‌های هر سفارش که به طرح یا قالب اشاره می‌کنند و تعداد و قیمت را نگهداری می‌کنندfile-crzskoe9o4wznmpsinhc2g. این ساختار اجازه می‌دهد یک سفارش شامل چند آیتم (مثلاً چند طرح یا ترکیبی از طرح و قالب) باشد.**
  + ***OrderSection*: برای سفارش‌های چاپ لباس، این مدل جهت ذخیره اطلاعات هر بخش لباس در سفارش اضافه شده استfile-crzskoe9o4wznmpsinhc2g. هر *OrderSection* به یک Order لینک شده و مشخص می‌کند کدام محل چاپ (مثلاً آستین یا پشت)، با چه ابعاد (اندازه‌ها) و چه طرحی پر شده است. این مدل نیاز سناریوی انتخاب بخشی از لباس را پوشش می‌دهد، مثلاً اگر مشتری فقط آستین را سفارش دهد یا برای هر بخش طرح جداگانه انتخاب کندfile-crzskoe9o4wznmpsinhc2g. همچنین *PrintLocation* (مدل محل‌های چاپ مانند آستین، جلو، پشت و ...) در بخش *designs* تعریف شده که انواع محل‌ها را مشخص می‌کندfile-crzskoe9o4wznmpsinhc2g و در *OrderSection.location* مورد استفاده استfile-crzskoe9o4wznmpsinhc2g.**
  + ***OrderStage*: (در کد مشاهده شد) مدلی برای مدیریت مراحل پردازش سفارش (چاپ، گلدوزی، برش و غیره) که به Order متصل است. هر *OrderStage* می‌تواند نوع مرحله (stage\_type) و وضعیت/هزینه آن مرحله را نگهداری کندfile-crzskoe9o4wznmpsinhc2g. این مدل برای پیگیری پیشرفت سفارش در مراحل تولید به‌کار می‌رود.**
  + ***OrderStatusHistory*: برای تاریخچه وضعیت‌های سفارش استفاده می‌شود و هر تغییر وضعیت را با زمان مربوطه ذخیره می‌کندfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. این مدل نیاز پیگیری تغییرات وضعیت سفارش را برآورده می‌کند (مثلاً از *pending* به *processing* و سپس *completed*).**
  + **علاوه بر این، جهت تفکیک نوع سفارش، فیلد order\_type به مدل Order پیشنهاد و افزوده شده استfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. این فیلد نوع سفارش را مشخص می‌کند (برای مثال *physical\_design* برای سفارش فیزیکی چاپ روی لباس یا *digital\_download* برای سفارش فایل دیجیتال طراحی) و توسط سیستم برای تعیین مسیر پردازش استفاده می‌شود. در کد همچنین جزئیات مجزا برای سفارش فیزیکی و دیجیتال دیده می‌شود: مدل‌های *PhysicalOrderDetails* و *DigitalDownloadOrderDetails* که به Order با OneToOne متصل‌اند، شامل اطلاعات اختصاصی هر نوع سفارش هستندfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. برای نمونه، *PhysicalOrderDetails* دارای وضعیت تولید (شروع نشده/در حال انجام/تکمیل شده) و شماره پیگیری مرسوله استfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g که برای سفارش‌های فیزیکی کاربرد دارد. این ساختار نشان می‌دهد مدل‌ها برای دو سناریوی متفاوت سفارش (فیزیکی در مقابل دیجیتال) آماده شده‌اند.**
* **تطابق مدل‌ها با نیازهای سناریو: ترکیب مدل‌های فوق انعطاف‌پذیری لازم برای سناریوی «ثبت سفارش چاپ لباس» را فراهم می‌کند. مدل Order به تنهایی اطلاعات پایه سفارش را دارد و از طریق OrderItem و OrderSection اجازه می‌دهد یک سفارش چندین طرح یا بخش داشته باشد. برای سناریوی *انتخاب بخش خاص لباس*، مدل OrderSection دقیقاً این نیاز را برآورده می‌کند و مشخص می‌نماید چه طرحی در چه بخشی از لباس استفاده می‌شودfile-crzskoe9o4wznmpsinhc2g. همچنین فیلدهای dimensions در OrderSection و ارتباطش با مدل SetDimensions (مدل ابعاد) تضمین می‌کند که اندازه‌های مورد نیاز هر بخش ثبت می‌گرددfile-crzskoe9o4wznmpsinhc2g. این همان چیزی است که در گفتگو به عنوان ورود اندازه‌های لباس توسط مشتری اشاره شدfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. برای سناریوی *استفاده از قالب آماده*، مدل‌های Template/Section (در ادامه آمده) و ارتباط‌شان با Order از طریق *SetDesign* (سناریوی ست‌بندی) این قابلیت را پشتیبانی می‌کنند که مشتری یک قالب پیش‌تعریف‌شده با چند جای خالی (بخش/ردیف) انتخاب کند و طرح‌های متغیر را در آن قرار دهدfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. سناریوی *خانواده طرح‌ها* نیز با مدل‌های Family (خانواده طرح) در بخش طراحی و اتصال آن به Order قابل پشتیبانی است؛ به این صورت که مشتری یک Family را به عنوان آیتم سفارش انتخاب کند (OrderItem می‌تواند به Family یا Template نیز اشاره کند، چنان‌که در مدل CartItem/OrderItem فیلد template وجود داشت) و سپس سیستم براساس نیازهای طراحی خانواده، طرح‌های مورد نیاز را از آن خانواده استخراج می‌کند. مدل‌های *FamilyDesignRequirement* و *DesignFamily* دقیقاً برای این منظور در سیستم تعبیه شده‌اند: هر FamilyDesignRequirement تعیین می‌کند یک خانواده به چند طرح از هر نوع نیاز داردfile-crzskoe9o4wznmpsinhc2g و از طریق DesignFamily می‌توان طرح‌های مشخصی را به یک خانواده لینک دادfile-crzskoe9o4wznmpsinhc2g. به این ترتیب اگر مشتری یک خانواده طرح (مثلاً مجموعه‌ای شامل ۲ طرح نوع "بنوک" و ۱ طرح نوع "گل") را انتخاب کند، سیستم می‌داند چه تعداد از کدام نوع طرح باید در سفارش قرار گیردfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. همچنین مدل Design (طرح) با افزودن فیلدهای *design\_type* و *suitable\_locations* تقویت شده است تا انتخاب طرح‌ها برای بخش‌های لباس به‌درستی مدیریت شودfile-crzskoe9o4wznmpsinhc2g. فیلد design\_type نوع محتوایی طرح را مشخص می‌کند (مثلاً "بنوک"، "گل" و ...) و فیلد *suitable\_locations* (نوع ManyToMany به PrintLocation) نشان می‌دهد هر طرح برای کدام بخش‌های لباس مناسب استfile-crzskoe9o4wznmpsinhc2g. این دو ویژگی کمک می‌کنند فیلتر کردن طرح‌های مجاز برای هر بخش امکان‌پذیر شود – مثلاً هنگام انتخاب طرح آستین، فقط طرح‌هایی که مناسب آستین برچسب خورده‌اند نمایش داده می‌شوندfile-crzskoe9o4wznmpsinhc2g. گفت‌وگو تأکید داشت که این منطق رعایت شود و نمونه آن نمایش «طرح‌های مناسب آستین بر اساس suitable\_locations» در پاپ‌آپ انتخاب طرح بودfile-crzskoe9o4wznmpsinhc2g. به طور کلی، مدل‌های فعلی بیشتر نیازهای سناریوی ثبت سفارش را پوشش می‌دهند و ساختار ماژولار (Order، OrderSection، OrderItem و ...) امکان ثبت سفارش‌های ساده تا پیچیده را می‌دهد.**
* **پیشنهادات بهبود یا توسعه: با توجه به بررسی، چند نکته برای بهبود مدل‌ها مطرح است:**
  + **در مدل‌های فعلی، OrderItem و OrderSection مقداری هم‌پوشانی مفهومی دارند. *OrderItem* بیشتر شبیه سبد خرید طراحی شده (هر آیتم می‌تواند یک طرح یا قالب به همراه تعداد و قیمت داشته باشد) در حالی که *OrderSection* جزئیات یک لباس سفارشی را نگهداری می‌کند. برای شفافیت بیشتر، می‌توان سفارش‌های چاپ لباس را از سفارش محصولات یا فایل‌های طراحی آماده تفکیک کرد. افزودن فیلد order\_type (که انجام شده)file-crzskoe9o4wznmpsinhc2g کمک می‌کند این تفکیک در سطح مدل انجام شود. در کنار آن، ممکن است *OrderSection* فقط برای سفارش‌های نوع "physical\_design" استفاده شود و *OrderItem* برای سفارش‌های نوع "digital\_product" یا موارد فروش مستقیم طرح/قالب. مستندسازی این ارتباط و استفاده از سیگنال‌ها یا تمپلیت‌تگ‌ها برای اطمینان از عدم تداخل این مفاهیم توصیه می‌شود.**
  + **سناریوی استفاده از قالب با بخش‌های اختیاری: اگر در یک سفارش برخی بخش‌های لباس استفاده نشوند، مدل OrderSection مشکلی برای حذف یا نداشتن آن بخش ندارد (صرفاً OrderSection برای آن ایجاد نمی‌شود). اما در سناریوی مناقصه (بیان‌شده در ادامه) اشاره شد که شاید برخی مراحل اختیاری باشند و مشتری بخواهد حذف کندfile-crzskoe9o4wznmpsinhc2g. به طور مشابه در قالب‌های آماده ممکن است برخی جایگاه‌ها خالی بماند. مدل‌های Template/Section این را با فیلدهایی مثل *unlimited\_design\_inputs* و *max\_design\_inputs* (در مدل Section) تا حدی دیده‌اندfile-crzskoe9o4wznmpsinhc2g. اگر نیاز باشد، می‌توان یک فلگ مثل is\_optional برای *Section*‌های قالب یا حتی *OrderSection* در نظر گرفت تا مشخص شود آن بخش الزامی نبوده است.**
  + **برای بهبود تجربه کاربری و جلوگیری از اشتباهات، منطق اعتبارسنجی باید در سمت سرور هم اعمال شود. مثلاً در مدل Section یک متد can\_accept\_design تعریف شده که بررسی می‌کند طرح انتخابی با قواعد بخش و ردیف مطابقت داردfile-crzskoe9o4wznmpsinhc2g. اطمینان از کامل بودن این قواعد اهمیت دارد. مدل‌های Template/Section/RowRule (بحث‌شده در سناریوی قالب) این نقش را ایفا می‌کنند و باید پیاده‌سازی شوند (در ادامه آمده است). در صورتی که هنوز این مدل‌ها در کد نهایی اضافه نشده‌اند، توصیه می‌شود برای پیاده‌سازی کامل سناریوی قالب آماده، مدل‌های Template, Section, RowRule و متدهای مرتبط حتماً تکمیل و تست شوند.**

**سناریو ۲: مدیریت طرح‌ها و آپلود گروهی**

**در این سناریو صاحبان طرح (طراحان یا کسب‌وکارها) طرح‌های گرافیکی خود را در سیستم بارگذاری کرده و آن‌ها را دسته‌بندی می‌کنند تا در سفارش‌ها مورد استفاده قرار گیرند. نیازمندی‌های کلیدی شامل تعریف متادیتا برای هر طرح (نام، نوع فایل، نوع طرح، ابعاد و تصاویر پیش‌نمایش)، برچسب‌گذاری (Tags) و دسته‌بندی (Category) طرح‌ها، گروه‌بندی طرح‌ها در خانواده‌ها و آپلود آسان چندین طرح به‌صورت همزمان می‌باشد.**

* **مدل‌های مرتبط در پروژه: مدل اصلی این بخش *Design* (طرح) است که ساختار بسیار جامعی دارد. طبق کد، این مدل دارای فیلدهای متعددی است: نام طرح، ابعاد (عرض/ارتفاع)، نوع فایل طرح (مثلاً وکتوری یا تصویر – فیلد type که در هنگام آپلود مقدار 'vector' یا 'image' می‌گیردfile-crzskoe9o4wznmpsinhc2g)، فایل SVG و تصویر محصول (برای ذخیره فایل‌های طرح)، تصویر بندانگشتی، و زمان‌های ایجاد/به‌روزرسانیfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. همچنین چندین رابطه ManyToMany برای ارتباط با مدل‌های دیگر وجود دارد:**
  + **برچسب‌ها (Tag): فیلد M2M tags برای برچسب‌گذاری آزاد طرح‌هاfile-crzskoe9o4wznmpsinhc2g. مدل *Tag* خود شامل نام برچسب استfile-crzskoe9o4wznmpsinhc2g و در سیستم برای دسته‌بندی موضوعی یا ویژگی‌های طرح استفاده می‌شود.**
  + **دسته‌بندی‌ها (Category): فیلد M2M categories برای دسته‌بندی سلسله‌مراتبی طرح‌هاfile-crzskoe9o4wznmpsinhc2g. مدل *Category* دارای نام و رابطه والد/فرزند (برای ایجاد دسته‌های تو در تو) استfile-crzskoe9o4wznmpsinhc2g. این امکان به گروه‌بندی طرح‌ها در شاخه‌های موضوعی یا کاربردی کمک می‌کند.**
  + **خانواده‌ها (Family): فیلد M2M family (یا families) طرح را به یک یا چند خانواده طرح مرتبط می‌کندfile-crzskoe9o4wznmpsinhc2g. مدل *Family* نشان‌دهنده یک مجموعه طرح مرتبط یا سرویس طراحی است که شامل فیلدهای نام و توضیحات استfile-crzskoe9o4wznmpsinhc2g. همچنین مدل‌های واسط *FamilyDesignRequirement* و *DesignFamily* (که قبلاً در سناریوی سفارش اشاره شد) روابط و الزامات بین طرح‌ها و خانواده‌ها را تعریف می‌کنندfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. این مدل‌ها اجازه می‌دهند که هر خانواده مشخص کند چه تعداد طرح از هر نوع در آن وجود دارد یا باید باشد و کدام طرح‌ها متعلق به خانواده هستند، که در سناریوی پر کردن خودکار بخش‌های قالب توسط یک خانواده طرح استفاده می‌شود.**
  + **قالب‌ها (Template): فیلد M2M templates در مدل Design ارتباط طرح با قالب‌های ممکن را ذخیره می‌کندfile-crzskoe9o4wznmpsinhc2g. بدین معنی که یک طرح ممکن است در قالب‌های خاصی استفاده شود یا پیشنهاد شود. مدل *Template* (قالب لباس) در بخش جداگانه‌ای تعریف شده و شامل اطلاعاتی نظیر نام قالب، قیمت، برچسب‌ها/دسته‌بندی‌های مرتبط و محل‌های چاپ مربوطه استfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. طرح‌ها از طریق این ارتباط می‌توانند مشخص کنند که در کدام قالب‌ها کاربرد دارند.**
  + **طرح‌های مشابه: فیلد M2M similar\_designs در مدل Design امکان لینک کردن طرح‌های مشابه (مثلاً طرح‌های یک مجموعه یا طرح‌های با سبک نزدیک به هم) را فراهم کرده استfile-crzskoe9o4wznmpsinhc2g. این می‌تواند برای پیشنهاد طرح‌های جایگزین یا مرتبط به مشتری استفاده شود.**

**افزون بر مدل Design، مدل‌های *Tag*, *Category*, *Family* که ذکر شدند، ارکان اصلی مدیریت طرح‌ها هستند. همچنین مدل *DesignerProfile* (پروفایل طراح) برای هر کاربر طراح در سیستم وجود داردfile-crzskoe9o4wznmpsinhc2g که یک رابطه OneToOne با کاربر دارد و می‌تواند برای اختصاص طرح‌های ایجادشده به طراح استفاده شود (هرچند در کد ارسالی، طرح‌ها مستقیماً به کاربر خالق لینک نشده بودند، در ادامه به آن اشاره می‌شود).**

* **تطابق مدل‌ها با نیازهای سناریو: مدل‌های مذکور تا حد زیادی تمامی نیازهای مدیریت طرح را پوشش داده‌اند. مدل Design با داشتن فیلدهای متنوع، جزئیات کاملی از هر طرح را نگهداری می‌کند. برای مثال، وجود فیلدهای فایل SVG و تصویر محصول به این معناست که سیستم از هر دو نوع فایل وکتوری و تصویر بیت‌مپ پشتیبانی می‌کند. فیلد type مشخص می‌کند طرح از کدام نوع است و در فرآیندها مورد استفاده قرار می‌گیرد؛ همان‌طور که در API آپلود طرح‌ها اگر نوع کلی تعیین نشده باشد، بر اساس پسوند فایل مقدار این فیلد تعیین می‌شود (وکتوری یا عکس)file-crzskoe9o4wznmpsinhc2g. بعلاوه، دسته‌بندی و برچسب‌گذاری به صورت کامل توسط روابط ManyToMany پوشش داده شده که برای فیلتر کردن و جستجوی طرح‌ها ضروری است. خانواده طرح‌ها نیز با مدل Family و وابستگانش فراهم شده که امکان گروه کردن طرح‌ها و ارائه سرویس‌های طراحی (مجموعه‌های از پیش تعیین‌شده طرح‌ها) را می‌دهد. این موضوع در گفتگو هم مورد تأکید بود که مثلاً کاربر بتواند یک سرویس طرح انتخاب کند تا بخش‌های مختلف قالب با طرح‌های آن سرویس پر شودfile-crzskoe9o4wznmpsinhc2g – مدل Family و نیازمندی‌های طراحی آن دقیقاً برای همین هدف طراحی شده است.**

**ویژگی مهم دیگر، آپلود دسته‌ای طرح‌ها است که برای افزایش بهره‌وری طراحان و کسب‌وکارها حیاتی است. در گفتگو یک کامپوننت React برای *BatchUpload* طرح‌ها ایجاد شد که به کاربر اجازه می‌دهد چندین فایل طرح را همزمان انتخاب و آپلود کندfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. در سمت سرور، *API مربوطه (BatchUploadView)* پیاده‌سازی شده که فایل‌های ارسالی را پردازش می‌کندfile-crzskoe9o4wznmpsinhc2g. این API از پارامترهای bulk\_type و bulk\_tags پشتیبانی می‌کند: یعنی کاربر می‌تواند نوع کلی همه طرح‌ها و برچسب‌های مشترک بین همه فایل‌های آپلودی را یکجا مشخص کندfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. در صورت تعیین این موارد، سرور در حلقه ایجاد طرح‌های جدید، آن نوع را به همه طرح‌ها اعمال می‌کند (در غیر اینصورت نوع از نوع فایل تعیین می‌شود) و همچنین مجموعه برچسب‌های انتخابی را به هر طرح انتساب می‌دهدfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. این قابلیت سرعت ثبت چندین طرح را بسیار بالا می‌برد و از تکرار عملیات جلوگیری می‌کندfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. در پیاده‌سازی، مشاهده می‌شود که پس از ذخیره هر طرح جدید، متد design.tags.set(...) فراخوانی می‌شود تا برچسب‌های گروهی اختصاص یابدfile-crzskoe9o4wznmpsinhc2g. همچنین امکان تعیین خانواده یا دسته‌بندی گروهی نیز در صورت نیاز می‌تواند با سازوکار مشابه اضافه شود که گفتگو نیز به این انعطاف‌پذیری در توسعه آینده اشاره کرده استfile-crzskoe9o4wznmpsinhc2g.**

**بحث کنترل دسترسی نیز در این سناریو اهمیت دارد؛ مدل‌ها و API باید تضمین کنند که فقط کاربران مجاز بتوانند طرح جدید ایجاد کنند. در کد *BatchUploadView* این موضوع رعایت شده و پیش از پردازش آپلود، بررسی می‌شود که کاربر مجوز designs.can\_create\_design داشته باشدfile-crzskoe9o4wznmpsinhc2g. این مجوز احتمالاً در مدل کاربر یا پرمیشن‌های سفارشی تعریف شده است (در CustomUser.Meta مشاهده می‌شود که یک پرمیشن view\_dashboard اضافه شده و "... سایر مجوزهای سفارشی" ذکر شده استfile-crzskoe9o4wznmpsinhc2g). بنابراین، سیستم طوری طراحی شده که مثلاً تنها طراحان یا ادمین‌ها بتوانند طرح جدید بارگذاری کنند و مشتریان عادی دسترسی به این قابلیت نداشته باشندfile-crzskoe9o4wznmpsinhc2g.**

* **پیشنهادات بهبود یا توسعه: مدل‌های مدیریت طرح تقریباً کامل به نظر می‌رسند و گفت‌وگو نیز آن‌ها را «جامع و حرفه‌ای» توصیف کرده استfile-crzskoe9o4wznmpsinhc2g. با این حال، چند پیشنهاد برای توسعه بیشتر:**
  + **مدل *Design* می‌تواند یک فیلد *created\_by* داشته باشد تا مشخص شود هر طرح توسط کدام کاربر (طراح یا کسب‌وکار) ایجاد شده است. در گفت‌وگو چندین بار به این فیلد اشاره شده و در کد آپلود نیز از request.user برای مقداردهی آن استفاده شده استfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. اگر این فیلد هنوز به مدل اضافه نشده، بهتر است اضافه شود (به نظر می‌رسد در بخشی از کد، created\_by = models.ForeignKey('auth.CustomUser', ... ) افزوده شده استfile-crzskoe9o4wznmpsinhc2g). این ارتباط کمک می‌کند مالکیت طرح‌ها مشخص باشد و در داشبورد طراح، فقط طرح‌های خود او نمایش داده شود. همچنین می‌توان از آن برای اعمال سطح دسترسی (مثلاً فقط خالق طرح یا ادمین‌ها بتوانند طرح را ویرایش/حذف کنند) استفاده کرد.**
  + **تولید خودکار تصویر بندانگشتی: مدل Design فیلد thumbnail برای تصویر بندانگشتی دارد اما در حال حاضر مکانیزمی برای تولید خودکار آن از SVG یا تصویر اصلی ذکر نشده است. در گفتگو پیشنهاد شده بود از کتابخانه‌هایی مانند Pillow استفاده شود تا پس از آپلود، thumbnail ساخته شودfile-crzskoe9o4wznmpsinhc2g. پیاده‌سازی یک سیگنال post\_save یا override در save() مدل Design که در صورت وجود فایل جدید، خودکار تصویر بندانگشتی را تولید کند، می‌تواند کارایی سیستم را افزایش دهد و خطای کاربر در آپلود دستی بندانگشتی را کاهش دهد.**
  + **برای بهبود ساختار داده خانواده طرح‌ها: مدل‌های FamilyDesignRequirement و DesignFamily ویژگی‌های لازم را ارائه می‌دهند، ولی ممکن است بتوان برای سادگی، برخی از این موارد را ادغام یا ساده کرد. به عنوان مثال، می‌توانستیم به جای داشتن DesignFamily به صورت یک مدل جدا، از همان رابطه ManyToMany مستقیم Design و Family استفاده کنیم (که در Design از طریق family وجود داردfile-crzskoe9o4wznmpsinhc2g) مگر اینکه قصد داشته باشیم اطلاعات اضافه‌ای در رابطه نگهداری کنیم. اگر DesignFamily فقط یک پل بدون فیلد اضافه است، شاید نیاز به آن نباشد. اما اگر قرار است مثلاً ترتیب یا توضیحاتی برای حضور طرح در خانواده ذخیره شود، نگه داشتن آن مدل قابل توجیه است. به هر حال، این نکته تأثیری بر عملکرد ندارد بلکه ساده‌سازی ساختار را در نظر دارد.**

**سناریو ۳: سیستم مناقصه و پیشنهاد قیمت**

**برای برخی مراحل تخصصی در فرآیند تولید لباس (مثل گلدوزی، چاپ خاص، دوزندگی تزئینی و ...)، در گفتگو ایده شد که به جای اختصاص مستقیم به یک کارگاه، مشتری بتواند یک مناقصه/مزایده کوچک برگزار کند تا از میان کسب‌وکارهای موجود بهترین پیشنهاد (از نظر قیمت یا زمان) را انتخاب نمایدfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. این سناریو نیازمند مدل‌هایی برای مدیریت درخواست مناقصه، پیشنهادهای ارسالی کسب‌وکارها و انتخاب برنده است.**

* **مدل‌های مرتبط در پروژه: طبق طراحی انجام‌شده، سه مدل اصلی برای این منظور ایجاد شده است: Tender (مناقصه)، Bid (پیشنهاد) و Award (اختصاص مناقصه)file-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. کد این مدل‌ها در مسیر tender/models.py درج شده استfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g:**
  + **مدل *Tender* بیانگر یک مناقصه ایجادشده توسط مشتری است. فیلدهای آن شامل مشتری ایجادکننده (customer به عنوان ForeignKey به User)file-crzskoe9o4wznmpsinhc2g، سفارش مرتبط (order به عنوان FK به Order)file-crzskoe9o4wznmpsinhc2g، مرحله مورد نظر (stage – نام مرحله‌ای که برای آن مناقصه برگزار می‌شود، مثل "گلدوزی" یا "چاپ")file-crzskoe9o4wznmpsinhc2g، توضیحات جزئیات کارfile-crzskoe9o4wznmpsinhc2g، مهلت زمانی برای ارسال پیشنهاد (deadline)file-crzskoe9o4wznmpsinhc2g و وضعیت مناقصه (status) که می‌تواند باز، بسته، یا برنده تعیین‌شده باشدfile-crzskoe9o4wznmpsinhc2g. همچنین زمان ایجاد (created\_at) برای ثبت تاریخچه در آن وجود داردfile-crzskoe9o4wznmpsinhc2g. این مدل تمام اطلاعات لازم برای تعریف یک مناقصه را فراهم می‌کند.**
  + **مدل *Bid* نشان‌دهنده پیشنهاد ارائه‌شده توسط یک کسب‌وکار بر روی یک Tender است. هر Bid شامل مرجع به مناقصه مربوطهfile-crzskoe9o4wznmpsinhc2g، مرجع به کسب‌وکار پیشنهاددهنده (business به عنوان FK، در کد به business.Business اشاره شده است)file-crzskoe9o4wznmpsinhc2g، مبلغ پیشنهادی و زمان تحویل پیشنهادی استfile-crzskoe9o4wznmpsinhc2g. تاریخ ثبت پیشنهاد (created\_at) نیز برای مرتب‌سازی و سوابق نگهداری می‌شودfile-crzskoe9o4wznmpsinhc2g.**
  + **مدل *Award* بیانگر برنده شدن یک پیشنهاد در یک مناقصه است. این مدل با دو فیلد OneToOne تعریف شده: ارتباط به مناقصه و ارتباط به پیشنهاد انتخاب‌شدهfile-crzskoe9o4wznmpsinhc2g. به این ترتیب هر Award مشخص می‌کند کدام Bid برنده کدام Tender شده و یک timestamp awarded\_at زمان انتخاب برنده را ضبط می‌کندfile-crzskoe9o4wznmpsinhc2g. با Award، وضعیت مناقصه به حالت "اختصاص داده شده/برنده مشخص" درمی‌آید.**

**در کنار این سه مدل، برای یکپارچگی با سایر بخش‌ها:**

* + **مدل *Order* دارای فیلد tender\_status شده است تا وضعیت مناقصه مرتبط با آن سفارش را نگهداری کندfile-crzskoe9o4wznmpsinhc2g. مقادیر این فیلد طبق پیشنهادات شامل 'none' (بدون مناقصه)، 'in\_progress' (در حال مناقصه) و 'awarded' (برنده تعیین شده) استfile-crzskoe9o4wznmpsinhc2g. به این ترتیب خود سفارش هم انعکاس‌دهنده وضعیت برون‌سپاری مراحلش خواهد بود. در پیاده‌سازی، منطق به‌روزرسانی این فیلد نیز پیش‌بینی شده است: برای مثال وقتی یک Tender جدید باز می‌شود، tender\_status سفارش مرتبط به "in\_progress" تغییر می‌کند و وقتی مناقصه برنده‌دار شد به "awarded" تغییر می‌یابدfile-crzskoe9o4wznmpsinhc2g. کدی که این تغییرات را اعمال می‌کند احتمالاً در متدهای save یا سیگنال‌های Tender قرار داده شده استfile-crzskoe9o4wznmpsinhc2g.**
  + **مدل *Business* نشان‌دهنده کسب‌وکارهای ثبت‌شده در سیستم است که می‌توانند در مناقصات شرکت کنند. در کد، Business دارای فیلدهای owner یا user (برای اتصال به حساب کاربری مالک)، name و اطلاعات تماس استfile-crzskoe9o4wznmpsinhc2g. Bid مستقیماً به Business لینک دارد، پس هر پیشنهاددهنده باید یک رکورد Business داشته باشد (مثلاً کارگاه‌های چاپ، گلدوزی و غیره به عنوان Business در سیستم ثبت می‌شوند). این مدل همچنین فیلد *allow\_customer\_info* دارد که مربوط به سناریوی دسترسی ست‌بند به اطلاعات مشتری است (در سناریوی ست‌بندی بحث خواهد شد)file-crzskoe9o4wznmpsinhc2g.**
* **تطابق مدل‌ها با نیازهای سناریو: سه‌گانه Tender/Bid/Award طراحی استاندارد یک سیستم مزایده را منعکس می‌کنند و نیاز مطرح‌شده را به خوبی پوشش می‌دهند. مشتری می‌تواند مناقصه‌ای ایجاد کند که دقیقاً مدل Tender آن را نمایش می‌دهد. فیلد *stage* در Tender مشخص می‌کند که مناقصه برای کدام مرحله تخصصی سفارش است (مثلاً "سوزن‌دوزی")file-crzskoe9o4wznmpsinhc2g. هر کسب‌وکار (مثلاً کارگاه یا متخصص مربوطه) می‌تواند یک پیشنهاد قیمت و زمان ارسال کند که مدل Bid با ثبت قیمت و زمان تحویل این امکان را فراهم کرده استfile-crzskoe9o4wznmpsinhc2g. مشتری پس از پایان مهلت (deadline) می‌تواند پیشنهادها را بررسی و یکی را انتخاب کند؛ در سیستم انتخاب برنده با ایجاد یک رکورد Award انجام می‌شود که به Tender و Bid منتخب مرتبط استfile-crzskoe9o4wznmpsinhc2g. ایجاد Award به صورت OneToOne تضمین می‌کند برای هر مناقصه تنها یک برنده نهایی ثبت شود. همچنین پس از Award، به‌روزرسانی وضعیت‌ها انجام می‌شود: Tender.status احتمالاً به 'awarded' می‌رسد و Order.tender\_status متناظر نیز 'awarded' خواهد شدfile-crzskoe9o4wznmpsinhc2g. این ارتباطات تضمین می‌کند سایر بخش‌های سیستم (مثل داشبورد مشتری یا ادمین) به راحتی بتوانند تشخیص دهند سفارش مذکور مرحله مناقصه‌اش را طی کرده و نتیجه چیست. در گفتگو حتی اشاره شد که در داشبورد مشتری باید بخشی برای مشاهده پیشرفت پروژه لباس و نتیجه مناقصات باشدfile-crzskoe9o4wznmpsinhc2g که این مدل‌ها پشتوانه آن قابلیت هستند.**

**لازم به ذکر است، مدل‌های مناقصه با مدل‌های کارگاه و اجرای سفارش‌ها نیز تعامل دارند. هنگامی که یک Tender ایجاد می‌شود، سفارش هنوز کارگاه نهایی مشخصی ندارد (توسط tender\_status مدیریت می‌شود). پس از انتخاب برنده، آن کسب‌وکار مشخص باید به نوعی به سفارش مرتبط شود تا ادامه کار را انجام دهد. دو رویکرد وجود دارد: یا مستقیماً کارگاه سفارش را تنظیم کنیم (مثلاً فیلد *Order.workshop* پر شود) یا از سیستم وظایف کارگاه‌ها (WorkshopTask) بهره بگیریم. در کد، *Order.workshop* به عنوان FK به مدل Workshop تعریف شده که احتمالاً برای اختصاص مستقیم سفارش به یک کارگاه استفاده می‌شودfile-crzskoe9o4wznmpsinhc2g. در سناریوی مناقصه، می‌توان پس از Award، فیلد Order.workshop را با کارگاه مربوط به کسب‌وکار برنده تنظیم کرد (البته کسب‌وکار ممکن است همان Workshop باشد یا نیاز باشد کسب‌وکار برنده ابتدا یک Workshop مربوط به خود داشته باشد). همچنین می‌توان بجای آن، یک رکورد WorkshopTask ایجاد نمود که نشان دهد این سفارش برای آن کارگاه برنده فعال شده است (که این به سناریوی کارگاه‌ها مربوط است و بعداً گفته می‌شود). در هر حال، مدل‌های Tender/Bid/Award به تنهایی برای مدیریت مزایده و انتخاب برنده کافی هستند و ارتباط لازم با سفارش و کسب‌وکار را دارند تا بقیه سیستم از نتیجه مطلع شود.**

* **پیشنهادات بهبود یا توسعه: سیستم مناقصه ارائه‌شده بسیار کامل به نظر می‌رسد. چند بهبود ممکن به شرح زیر است:**
  + **امکان اختیاری بودن مرحله: در گفتگو مطرح شد که شاید برخی مراحل را مشتری نخواهد انجام دهد و حذف کند، لذا مناقصه برای آن‌ها برگزار نمی‌شود. برای مدل Tender می‌توان یک فیلد Boolean به نام مثلاً optional اضافه کرد تا مشخص کند آیا این مرحله اساساً اختیاری بوده استfile-crzskoe9o4wznmpsinhc2g. اگر optional=true باشد، شاید برنده‌ نداشتن مناقصه اثر منفی روی جریان کلی سفارش نگذارد (چون ممکن است عمداً از آن صرف‌نظر شده باشد). این فیلد در تصمیم‌گیری‌های بعدی (مثلاً محاسبه قیمت یا کامل‌شدن پروژه) مفید خواهد بود.**
  + **ثبت ارتباط با مرحله سفارش به صورت ساخت‌یافته‌تر: در حال حاضر Tender.stage به صورت متن آزاد استfile-crzskoe9o4wznmpsinhc2g. برای انسجام داده‌ها، می‌توان به جای متن از یک مجموعه انتخاب ثابت یا مرجع به مدل *OrderStage* استفاده کرد. مثلاً اگر مراحل استانداردی داریم (چاپ، گلدوزی، دوزندگی، ...) یا هر سفارش OrderStage‌هایی دارد، می‌توان Tender را به یک OrderStage خاص مرتبط کرد. در طراحی کنونی، متن انعطاف بیشتری می‌دهد (هر نوع مرحله دلخواه را می‌شود نوشت) اما مرجع ساخت‌یافته امکان جلوگیری از ورود داده ناهماهنگ (مثل املای متفاوت یک مرحله) را می‌دهد. این به تصمیم طراحی و پیچیدگی پیاده‌سازی بستگی دارد.**
  + **اطلاع‌رسانی رویدادها: برای سناریوی مناقصه، می‌توان سیگنال‌ها یا نوتیفیکیشن‌های خاص تنظیم کرد. به عنوان مثال، وقتی مشتری یک مناقصه ایجاد می‌کند، همه کسب‌وکارهای مرتبط (مثلاً همه کارگاه‌های نوع خاص) مطلع شوند. یا وقتی یک کسب‌وکار انتخاب شد، به او اعلان (Notification) ارسال گردد. مدل Notification (اعلان‌ها) در سیستم وجود دارد و می‌تواند برای این منظور به‌کار رود. پیشنهاد می‌شود پس از ایجاد Tender جدید، یک Notification به نوع 'task\_update' یا مشابه برای کاربران نقش کسب‌وکار ایجاد شود تا آنها را به مناقصه جدید آگاه کند. (در بخش اعلان‌ها بیشتر توضیح داده می‌شود).**

**سناریو ۴: مرحله ست‌بندی (ترکیب طرح‌ها توسط ست‌بند)**

**ست‌بندی به فرآیندی گفته می‌شود که در آن طرح‌های انتخاب‌شده مشتری برای بخش‌های مختلف لباس، توسط یک طراح فنی یا گرافیست (ست‌بند) در قالب نهایی کنار هم قرار می‌گیرند تا فایل نهایی چاپ آماده شودfile-crzskoe9o4wznmpsinhc2g. این مرحله می‌تواند به صورت خودکار (با استفاده از اسکریپت‌های گرافیکی) یا دستی توسط نیروی انسانی انجام شودfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. در پروژه Pardach، سناریوی ست‌بندی چنین است: پس از آنکه مشتری طرح‌های قسمت‌های لباس را تعیین کرد (و یا برنده مناقصه برای مراحل تکمیلی مشخص شد)، یک کاربر با نقش "ست‌بند" فایل‌های طرح را گرفته و آن‌ها را در یک فایل (مثلاً Adobe Illustrator) ترکیب می‌کند و خروجی نهایی را برای چاپ آماده می‌کندfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. بخشی از هزینه سفارش نیز به این شخص بابت کارش پرداخت می‌شودfile-crzskoe9o4wznmpsinhc2g.**

* **مدل‌های مرتبط در پروژه: برای این منظور مدل *SetDesign* معرفی شده است. مطابق کد، SetDesign به شکل زیر تعریف شده استfile-crzskoe9o4wznmpsinhc2g:**
  + **ارتباط ManyToOne به *Order* (تا مشخص شود این ست‌بندی برای کدام سفارش است)file-crzskoe9o4wznmpsinhc2g. هر سفارش چاپ لباس در صورت نیاز به ست‌بندی یک رکورد SetDesign خواهد داشت.**
  + **فیلد *set\_designer* که به کاربر انجام‌دهنده ست‌بندی اشاره می‌کندfile-crzskoe9o4wznmpsinhc2g. این به احتمال زیاد به مدل User یا پروفایل ست‌بند (ممکن است همان Business یا Role مشخص باشد) لینک شده است. در کد ارسالی، set\_designer = models.ForeignKey(User, ...) آمده که نشان می‌دهد هر SetDesign یک کاربر مسئول داردfile-crzskoe9o4wznmpsinhc2g. پیشنهاد شد که یک نقش جدید به نام SetDesigner در سیستم در نظر گرفته شودfile-crzskoe9o4wznmpsinhc2g، اما حتی بدون نقش اختصاصی، این فیلد کار اختصاص دادن وظیفه ست‌بندی به یک کاربر را انجام می‌دهد.**
  + **فیلد *template*: این فیلد به مدل Template (قالب لباس انتخابی) اشاره می‌کندfile-crzskoe9o4wznmpsinhc2g. دلیل وجود آن این است که ست‌بند بداند از کدام قالب پایه (مثلاً T-Shirt یا Hoodie با ترتیب بخش‌های خاص) باید استفاده کند. مشتری در فرآیند سفارش احتمالاً یک Template انتخاب کرده (اگر سفارش از نوع قالب‌محور بوده) و اینجا آن قالب مشخص می‌شود تا ست‌بند مطابق آن عمل کند.**
  + **فیلد ManyToMany به *Design*: تحت عنوان designs که تمامی طرح‌های انتخاب‌شده مشتری را شامل می‌شودfile-crzskoe9o4wznmpsinhc2g. عملاً این فیلد لیست طرح‌هایی است که قرار است ست‌بند در فایل نهایی بچیند. وجود این فیلد تضمین می‌کند ست‌بند به همه فایل‌های طرح خام دسترسی دارد. (در عمل ممکن است به URL فایل‌ها یا مسیرشان نیاز باشد که از خود مدل Design قابل دریافت است).**
  + **فیلد *customer\_info*: یک JSONField برای ذخیره اطلاعات مشتری در صورت نیاز (مثلاً اندازه‌ها، یا مشخصات دلخواه مشتری)file-crzskoe9o4wznmpsinhc2g. این داده‌ها می‌تواند برای شخص ست‌بند مفید باشد (مثلاً اندازه دقیق بخش‌ها که مشتری وارد کرده است). در سیستم حق دسترسی به اطلاعات مشتری مورد توجه بوده؛ طوری که یک فیلد در Business (allow\_customer\_info) تعبیه شده تا مشخص کند آیا ست‌بند اجازه دیدن اطلاعات شخصی مشتری را دارد یا خیرfile-crzskoe9o4wznmpsinhc2g. حتی یک مکانیزم پیشنهاد شد که متدی به نام can\_access\_customer\_info بررسی کند ست‌بند (که ممکن است جزو کارکنان کسب‌وکار چاپ باشد) مجوز دیدن اطلاعات را دارد یا نه و در غیر اینصورت پیام "دسترسی محدود" نمایش دهدfile-crzskoe9o4wznmpsinhc2g.**
  + **فیلد *status*: وضعیت ست‌بندی (مثلاً *pending* یا *completed*)file-crzskoe9o4wznmpsinhc2g. این وضعیت روند کار ست‌بند را نشان می‌دهد. زمانی که ست‌بند کار را تمام کرد، وضعیت به "completed" می‌شود که در ادامه می‌تواند روی وضعیت کلی سفارش یا امکان ادامه فرآیند (مثلاً ارسال فایل نهایی به چاپخانه) تأثیر بگذارد.**

**طبق گفت‌وگو، مدل SetDesign می‌تواند یک فیلد *file* نیز داشته باشد که فایل نهایی ست‌شده (مثلاً خروجی Illustrator یا PDF برای چاپ) را ذخیره کندfile-crzskoe9o4wznmpsinhc2g. در کد اولیه این فیلد دیده نشد، اما افزودن آن بسیار مهم است زیرا خروجی کار ست‌بند باید جایی بارگذاری و نگهداری شود. احتمال دارد برنامه‌نویس هنوز این را اضافه نکرده یا در برنامه Frontend آپلود فایل نهایی لحاظ شده باشد.**

**همچنین، برای نقش ست‌بند مشخص شد که ممکن است سیستمی خودکار در آینده اضافه شود. اما فعلاً سیستم ترکیبی از نیمه‌خودکار و نظارت انسانی استfile-crzskoe9o4wznmpsinhc2g. بنابراین وجود SetDesign با مشخص کردن شخص مسئول، کاملاً منطبق با نیاز فعلی است.**

* **تطابق مدل‌ها با نیازهای سناریو: مدل SetDesign بخش اصلی نیاز ست‌بندی را برطرف می‌کند. در سناریوی واقعی، وقتی سفارش به مرحله ست‌بندی می‌رسد، با ایجاد یک SetDesign و تخصیص آن به یک ست‌بند، کار آغاز می‌شود. ست‌بند از طریق داشبورد خود موارد اختصاص‌یافته را می‌بیندfile-crzskoe9o4wznmpsinhc2g و می‌تواند فایل‌های طرح خام را دانلود کندfile-crzskoe9o4wznmpsinhc2g. حضور لیست طرح‌ها در مدل SetDesign این امکان را می‌دهد که رابط کاربری، لینک یا پیش‌نمایش همه اجزای لازم را در اختیار ست‌بند بگذارد. همچنین ست‌بند پس از اتمام کار می‌تواند فایل نهایی را آپلود کرده و وضعیت را به "تکمیل شده" تغییر دهدfile-crzskoe9o4wznmpsinhc2g. این تغییر وضعیت احتمالاً توسط سیگنال‌ها رهگیری می‌شود تا مثلاً به مشتری اطلاع دهد سفارش آماده چاپ است. سیستم با استفاده از مدل Notification چنین اطلاع‌رسانی‌هایی را می‌تواند انجام دهد (مثلاً در سیگنال post\_save برای SetDesign.completed). به علاوه، چون SetDesign به Order متصل است، تغییر وضعیت Order نیز امکان‌پذیر است (شاید Order.status = 'design\_completed' یا چیزی مشابه). در مجموع، مدل SetDesign پل ارتباطی بین طراحی اولیه مشتری و مرحله تولید است و ساختار داده‌ای لازم برای انتقال اطلاعات به ست‌بند و دریافت خروجی را دارد.**
* **پیشنهادات بهبود یا توسعه:**
  + **همان‌طور که اشاره شد، افزودن فیلد فایل خروجی به مدل SetDesign ضروری است. بدون این فیلد، سیستم نمی‌داند فایل نهایی کجا ذخیره می‌شود. بهتر است یک FileField یا URLField (در صورتی که فایل در فضای خارجی ذخیره می‌شود) به SetDesign اضافه گردد. این فیلد می‌تواند هنگام تغییر وضعیت به completed مقداردهی شود.**
  + **می‌توان برای پیگیری دقیق‌تر، زمان‌های شروع و پایان ست‌بندی را نیز نگهداری کرد (فیلدهای DateTime مانند started\_at و finished\_at). در کد، BaseModel احتمالاً این موارد (created\_at) را دارد ولی started\_at که زمان ادیت ست‌بند باشد موجود نیست. اگر نیاز باشد که مثلاً مدت زمان انجام ست‌بندی یا تأخیرها تحلیل شود، می‌توان این فیلدها را افزود.**
  + **در بحث خودکارسازی، پیشنهاد شده بود که اگر اسکریپت‌های Illustrator وجود دارد، یک endpoint Django فراهم شود که فایل‌های طرح را بگیرد و فایل نهایی را برگرداندfile-crzskoe9o4wznmpsinhc2g. این موضوع مدل جدیدی نیاز ندارد اما به عملکرد سیستم در آینده مربوط است. برای آمادگی، شاید بتوان یک فیلد *automation\_script* یا یک enum برای SetDesign.status اضافه کرد تا وضعیت "در حال ست‌بندی خودکار" را مشخص کند. اما این مورد در حال حاضر ضروری نیست.**
  + **نقش SetDesigner: اگرچه فقط با یک فیلد ForeignKey به User این نقش ضمنی ایجاد شده، اما برای شفافیت مدیریت دسترسی می‌توان یک Role جداگانه در جدول Role برای ست‌بند تعریف کردfile-crzskoe9o4wznmpsinhc2g. در CustomUser/Role چنین چیزی دیده نمی‌شود (Roleهای فعلی شامل طراح، صاحب کسب‌وکار و ... است). افزودن Role "ست‌بند" مزیتش این است که می‌توان مجوزهای خاص (مثل دیدن بخش ست‌بندی در داشبورد یا دسترسی به اطلاعات مشتری در صورت مجاز بودن) را راحت‌تر کنترل کرد. البته این تغییر نیاز به تنظیمات در بخش احراز هویت دارد و می‌توان آن را در آینده انجام داد.**

**سناریو ۵: اجرای سفارش در کارگاه‌ها و مدیریت وظایف**

**پس از طی مراحل طراحی و ست‌بندی، سفارش آماده تولید فیزیکی می‌شود. در سیستم Pardach، کارگاه‌ها (مثلاً چاپخانه‌های دیجیتال، کارگاه‌های دوزندگی، یا هر کسب‌وکار تولیدی) نقش مهمی دارند. سناریو این است که یک سفارش نهایی به یک کارگاه برای تولید سپرده می‌شود. ممکن است سفارش مستقیماً به یک کارگاه خاص تخصیص یابد (مثلاً اگر مشتری از اول کسب‌وکار مشخصی را انتخاب کرده یا پس از مناقصه برنده تعیین شده باشد)، یا سفارش شامل چند مرحله کارگاهی باشد که هر کدام به کارگاه متفاوتی سپرده شود. همچنین ظرفیت کارگاه‌ها، پیگیری وضعیت کار و گزارش‌دهی از جمله نیازها است.**

* **مدل‌های مرتبط در پروژه: برای مدل کردن کارگاه و عملیات آن چندین مدل تعریف شده است: Workshop، WorkshopTask، WorkshopStaff و WorkshopReportfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g.**
  + **مدل *Workshop* نماینده یک کارگاه یا کسب‌وکار تولیدی است. فیلدهای آن شامل نام کارگاه، نوع کارگاه (با انتخاب‌هایی مانند 'production' تولید، 'printing' چاپ – مشخص‌کننده حوزه کاری)file-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g، آدرس، شماره تماس و ایمیل استfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. همچنین برای مدیریت ظرفیت، دو فیلد عددی دارد: *capacity* (ظرفیت کل سفارشاتی که می‌تواند هم‌زمان انجام دهد) و *current\_orders* (تعداد سفارشات در حال اجرای فعلی)file-crzskoe9o4wznmpsinhc2g. متد is\_over\_capacity() نیز برای چک کردن پر شدن ظرفیت ارائه شده استfile-crzskoe9o4wznmpsinhc2g. این مدل در واقع اطلاعات ثابت هر کارگاه را نگه می‌دارد. ارتباط کارگاه با کاربر صاحب آن از طریق مدل Business/BusinessOwnerProfile برقرار می‌شود (Workshop مستقیماً به User وصل نشده، اما BusinessOwnerProfile هر کاربر می‌تواند صاحب یک Business/Workshop باشد).**
  + **مدل *WorkshopTask* وظایف یا سفارش‌های تخصیص‌یافته به کارگاه را نمایش می‌دهد. هر WorkshopTask با FK به Workshop و FK به Order مشخص می‌کند که کدام سفارش به کدام کارگاه واگذار شده استfile-crzskoe9o4wznmpsinhc2g. فیلد وضعیت کار (status) وضعیت این وظیفه را در کارگاه نگه می‌دارد که می‌تواند مقادیری چون 'pending' (در انتظار)، 'in\_progress' (در حال انجام)، 'completed' (تکمیل‌شده) یا 'cancelled' داشته باشدfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. همچنین زمان *assigned\_at* (تاریخ اختصاص) و *completed\_at* برای ثبت تاریخ شروع و پایان کار موجود استfile-crzskoe9o4wznmpsinhc2g. این مدل در واقع به ازای هر سفارش (یا هر بخش از سفارش) که یک کارگاه باید انجام دهد یک رکورد دارد. در سناریوی ساده، هر سفارش پس از آماده‌شدن به یک کارگاه می‌رود و یک WorkshopTask ایجاد می‌شود. در حالت پیچیده‌تر، اگر سفارش چند مرحله داشته باشد که نیاز به کارگاه‌های مختلف است (مثلاً چاپ دیجیتال و سپس دوزندگی دستی)، می‌توان چند WorkshopTask برای سفارش مذکور ساخت (هر کدام با نوع کار متفاوت - البته در مدل فعلی نوع کار به صورت صریح ذکر نشده و می‌توان از نوع کارگاه یا توضیحات استفاده کرد).**
  + **مدل *WorkshopStaff* برای مدیریت پرسنل کارگاه است. این مدل هر کاربر را به یک کارگاه با یک نقش مشخص لینک می‌کندfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. فیلدهای آن عبارت‌اند از FK به Workshop، FK به User، و یک فیلد *role* (نقش کاربر در آن کارگاه مثل مدیر، کارگر، اپراتور)file-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. Unique\_together بودن (workshop, user) جلوگیری می‌کند یک نفر دوبار در یک کارگاه ثبت شودfile-crzskoe9o4wznmpsinhc2g. این مدل سناریوی داشتن چند کارمند/همکار در هر کارگاه را پوشش می‌دهد. مثلاً یک چاپخانه دیجیتال ممکن است چند اپراتور داشته باشد که همه زیرمجموعه آن Workshop تعریف می‌شوند.**
  + **مدل *WorkshopReport* جهت ثبت گزارش‌های کارگاه از پیشرفت یا مشکلات کار تعریف شده است. این مدل با FK به Workshop و FK به WorkshopTask مشخص می‌کند گزارش متعلق به کدام کارگاه و کدام وظیفه استfile-crzskoe9o4wznmpsinhc2g. متن گزارش و زمان ایجاد آن نیز در فیلدها آمده استfile-crzskoe9o4wznmpsinhc2g. این مدل اجازه می‌دهد کارگاه در طی انجام کار (یا پایان آن) گزارشی ارائه دهد که مثلاً چه کاری انجام شده، آیا مشکلی بوده یا سفارش آماده ارسال است. این اطلاعات می‌تواند برای مشتری یا مدیر سیستم قابل مشاهده باشد.**
* **تطابق مدل‌ها با نیازهای سناریو: این مدل‌ها یک سیستم کامل مدیریت کار در کارگاه را تشکیل می‌دهند. به سناریوی مورد نظر بازگردیم: وقتی سفارش آماده تولید شد، یا مستقیماً توسط سیستم به یک کارگاه تخصیص می‌شود (مثلاً همان کسب‌وکار چاپی که مشتری انتخاب کرده بود)، یا پس از مناقصه کارگاه برنده مشخص شده و تخصیص صورت می‌گیرد. در هر دو حالت، باید در سیستم یک نمایانگر از این تخصیص وجود داشته باشد. دو مکانیزم وجود دارد: یکی *Order.workshop* که یک لینک یک‌به‌یک بین Order و Workshop است (سفارش به کارگاه)file-crzskoe9o4wznmpsinhc2g، و دیگری *WorkshopTask* که امکان چندین تخصیص را نیز می‌دهد. مدل‌های فعلی هر دو را دارند، اما به نظر می‌رسد استفاده اصلی با WorkshopTask باشد چون قابلیت ثبت وضعیت و تکمیل را دارد. برای مثال، وقتی Order.tender\_status = 'awarded' شد، می‌توان یک WorkshopTask جدید ساخت که order همان سفارش است و workshop کارگاه برنده. آنگاه کارگاه از طریق related\_name (workshop.tasks) به لیست کارهای خود دسترسی داردfile-crzskoe9o4wznmpsinhc2g و می‌بیند سفارش X در وضعیت 'pending' برای او آمده است. وقتی کار روی سفارش را شروع می‌کنند، می‌توان status را 'in\_progress' و پس از اتمام 'completed' کردfile-crzskoe9o4wznmpsinhc2gfile-crzskoe9o4wznmpsinhc2g. این وضعیت‌ها در داشبورد کارگاه نیز نمایش داده می‌شود و کمک می‌کند سفارش‌های جاری را مدیریت کنندfile-crzskoe9o4wznmpsinhc2g. همچنین هر کارگاه ظرفیت خود را دارد؛ اگر کاربری بخواهد سفارشی را به کارگاهی اختصاص دهد، می‌تواند is\_over\_capacity() را چک کندfile-crzskoe9o4wznmpsinhc2g و اگر true بود، سفارش را به کارگاه دیگری بدهد یا پیام مناسب نمایش دهد.**

**مدل کارکنان کارگاه (WorkshopStaff) تضمین می‌کند که اگر کارگاه بزرگ باشد، همه کاربران وابسته به آن شناخته شده‌اند. در نتیجه، حتی می‌توان اعلان‌ها را به همه اعضای یک Workshop ارسال کرد یا دسترسی بخش‌های مختلف داشبورد را بین نقش‌های مختلف کارکنان تفکیک نمود. مدل گزارش کارگاه (WorkshopReport) نیز نیاز به بازخورد و گزارش‌دهی را حل کرده است. برای مثال، ممکن است کارگاه گلدوزی پس از اتمام کار گزارشی مبنی بر اتمام و ارسال کالا ثبت کند که مشتری آن را ببیند. یا اگر مشکلی (مثلاً پارچه آسیب دیده) پیش آمد، گزارش بدهد تا پشتیبانی تصمیم لازم را بگیرد.**

**شواهد گفت‌وگو نشان می‌دهد که این بخش مدل‌سازی شده مدنظر طراحان بوده است. در متن آمده که مثلاً کسب‌وکارهای چاپ دستی و دیجیتال باید تعامل داشته باشند؛ اگر کسب‌وکار چاپ دستی نیاز به مهر فیزیکی داشته باشد بتواند از طریق سیستم سفارش دهدfile-crzskoe9o4wznmpsinhc2g. این سناریو احتمالاً با استفاده از مکانیزم مناقصه یا وظایف کارگاه‌ها تحقق می‌یابد: چاپخانه دستی اگر مهر طرحی را نداشته باشد، می‌تواند یک WorkshopTask یا حتی Tender برای یک کارگاه حکاکی لیزری (ساخت مهر) ایجاد کند. مدل‌ها به اندازه کافی عمومی هستند که چنین کاربردی را نیز پوشش دهند. به‌ویژه وجود انواع مختلف کارگاه (workshop\_type)file-crzskoe9o4wznmpsinhc2g اجازه می‌دهد کارگاه حکاکی، کارگاه دوخت، کارگاه چاپ هر یک نوعی داشته باشند و سیستم براساس نوع، وظایف را مدیریت کند. در داشبورد نیز می‌توان فیلتر کرد که هر کسب‌وکار فقط مناقصات یا سفارش‌های مرتبط با نوع خود را ببیندfile-crzskoe9o4wznmpsinhc2g.**

* **پیشنهادات بهبود یا توسعه:**
  + **پیوند وظایف با مراحل سفارش: اگر سیستم به سمتی می‌رود که یک سفارش چند مرحله تولیدی داشته باشد (مثلاً مرحله 1 چاپ دیجیتال، مرحله 2 دوزندگی تزئینی)، شاید بهتر باشد مدل WorkshopTask یک فیلد نوع/مرحله داشته باشد (مثلاً stage یا task\_type). در حال حاضر WorkshopTask مستقیماً به Order وصل است و نوع کار فقط ضمنی از نوع کارگاه فهمیده می‌شود. اما اگر یک کارگاه ترکیبی انجام دهد یا نیاز به چند کارگاه در یک مرحله باشد، صراحت نوع کار مفید است. می‌توان این را با افزودن یک CharField با choices یا FK به OrderStage حل کرد.**
  + **یکپارچه‌سازی Order.workshop و WorkshopTask: با توجه به اینکه WorkshopTask وجود دارد، فیلد Order.workshop ممکن است redundant شود مگر برای اشاره به کارگاه اصلی مسئول سفارش. مثلاً شاید Order.workshop نشان‌دهنده کارگاه لیدر باشد که کل پروژه را مدیریت می‌کند (و خودش ممکن است برخی کارها را tender کند). اگر چنین مفهومی مدنظر است، استفاده از Order.workshop به عنوان کارگاه اصلی و WorkshopTask برای کارهای زیرمجموعه منطقی است. ولی اگر هر سفارش دقیقاً یک کارگاه می‌گیرد، همان WorkshopTask کافی بود. پیشنهاد می‌شود طراحی روشن‌تر شود: یا Order.workshop حذف گردد و همیشه از WorkshopTask استفاده شود (اجازه‌ی چندین کارگاه در یک سفارش)، یا اگر Order.workshop حفظ می‌شود، رابطه‌اش با WorkshopTask مشخص گردد (مثلاً تمامی WorkshopTaskهای آن Order باید متعلق به همان کارگاه اصلی باشند یا برعکس، Order.workshop پس از اولین WorkshopTask تنظیم شود).**
  + **گزارش‌ها و تایید کیفی: مدل WorkshopReport برای ثبت گزارش کیفی/کمی کار بسیار مناسب است. می‌توان توسعه داد که هر WorkshopTask پس از تکمیل نیاز به یک گزارش نهایی داشته باشد (وضعیتی شبیه Acceptance). شاید افزودن فیلدهای approved یا امتیازدهی کیفی به گزارش‌ها یا خود Task مفید باشد تا سیستم ارزیابی کارگاه‌ها را نیز بتواند انجام دهد. البته این در محدوده نیاز فعلی ذکر نشده، اما برای توسعه‌های بعدی (سیستم رتبه‌بندی کسب‌وکارها) بسترش را می‌توان آماده کرد.**
  + **اتوماتیک‌سازی ظرفیت: هم‌اکنون ظرفیت به صورت دستی کنترل می‌شود. در آینده می‌توان فرآیندی افزود که هرگاه WorkshopTask جدیدی created شد، مقدار current\_orders کارگاه مربوطه یک واحد افزایش یابد و وقتی completed شد، یک واحد کاهش یابد. این را می‌توان با سیگنال‌های post\_save/post\_delete مدیریت کرد تا *current\_orders* همیشه درست باشد. این کار جلوی اشتباهات دستی را می‌گیرد.**

**سناریو ۶: اعلان‌ها و پیگیری وضعیت**

**یکی از نیازهای مهم سیستم، اطلاع‌رسانی به کاربران در مورد رخدادهای کلیدی است؛ چه مشتری که می‌خواهد از پیشرفت سفارش خود مطلع شود، چه کسب‌وکار/ست‌بند که سفارشی به او اختصاص یافته یا پیامی دریافت کرده است. سناریوی اعلان‌ها شامل ارسال نوتیفیکیشن در مواردی مانند: تغییر وضعیت سفارش، آماده‌شدن ست‌بندی، ارائه پیشنهاد جدید در مناقصه، انتخاب شدن یک کسب‌وکار به عنوان برنده، ثبت گزارش توسط کارگاه، و امثال آن می‌شود.**

* **مدل‌های مرتبط در پروژه: مدل *Notification* برای این منظور طراحی شده است. در کد بخش‌هایی از آن دیده می‌شود که شامل حداقل دو فیلد اصلی است: نوع اعلان (notification\_type مثلاً 'order\_update' برای بروزرسانی سفارش، 'task\_update' برای بروزرسانی وظیفه و ...)file-crzskoe9o4wznmpsinhc2g و احتمالاً کاربر گیرنده (user به عنوان FK به User) و متن یا جزئیات اعلان. همچنین شاید فیلدهای optional مثل مرتبط بودن به یک شیء (ContentType و ObjectID یا یک FK اختصاصی به Order یا ...)، اولویت (priority) و وضعیت خوانده‌شدن هم داشته باشد. از متن گفت‌وگو: "Notification که نوشته‌ای بسیار قوی است و با چند بهبود کوچک (مثل سیگنال‌ها و ...)" مشخص است مدل Notification قبلاً توسط کاربر پیاده‌سازی شده و نسبتا کامل استfile-crzskoe9o4wznmpsinhc2g. لذا فرض می‌کنیم حداقل شامل موارد فوق است.**

**جدای از خود مدل Notification، نحوه تولید اعلان در سیستم مهم است. نمونه‌ای که در کد دیده شد، استفاده از سیگنال post\_save برای Order بودfile-crzskoe9o4wznmpsinhc2g: هرگاه سفارشی ذخیره می‌شود و وضعیتش یکی از 'processing', 'assigned', 'completed' باشد، یک Notification جدید ساخته می‌شود با نوع 'order\_update'، اولویت 'medium' و پیام حاوی تغییر وضعیتfile-crzskoe9o4wznmpsinhc2g. این دقیقاً مکانیزم لازم برای اطلاع مشتری از تغییر وضعیت سفارش است. به طور مشابه، می‌توان برای رویدادهای دیگر سیگنال تعریف کرد:**

* + **پس از ایجاد یک Tender، ارسال Notification نوع 'task\_update' برای کسب‌وکارهای مرتبط (یا یک گروه از کاربران).**
  + **پس از ایجاد یک Bid، شاید اطلاع به مشتری که پیشنهاد جدید دریافت شد.**
  + **پس از Award شدن یک Tender، Notification به کسب‌وکار برنده (و شاید به دیگر پیشنهاددهندگان که انتخاب نشدند، با متن مناسب).**
  + **پس از آپلود فایل نهایی ست‌بندی یا تکمیل SetDesign، Notification به مشتری که "طرح شما آماده چاپ است".**
  + **پس از کامل شدن WorkshopTask در کارگاه، Notification به مشتری که "سفارش شما تولیدش تکمیل شد و در حال ارسال است".**
  + **و موارد مشابه.**
* **تطابق مدل‌ها با نیازهای سناریو: مدل Notification با ساختار انعطاف‌پذیرش می‌تواند همه این موارد را پشتیبانی کند. نوع‌های اعلان تعریف‌شده (order\_update, task\_update و احتمالاً انواع دیگر) منطبق بر سناریوهای ذکرشده هستندfile-crzskoe9o4wznmpsinhc2g. برای مثال، *order\_update* برای تغییر وضعیت سفارشات (پرداخت، آماده‌سازی، ارسال و ...)، *task\_update* برای بروزرسانی مراحل کار (مثل ست‌بندی یا کارگاه). اگر نوع جدیدی نیاز باشد (مثلاً *bid\_new* یا *tender\_awarded*) می‌توان به آسانی به انتخاب‌ها افزود. مهم‌تر آنکه، سیستم اعلام درستی از این مدل استفاده می‌کند: نمونه ایجاد Notification در سیگنال Order نشان می‌دهد رویدادهای کلیدی رهگیری می‌شوندfile-crzskoe9o4wznmpsinhc2g. این اعلان حاوی شناسه سفارش و وضعیت جدید در پیام است که برای مشتری واضح و مفید استfile-crzskoe9o4wznmpsinhc2g. در داشبورد مشتری نیز می‌توان لیست اعلان‌ها را نمایش داد تا هر تغییری را ببیند.**

**گفتگو به *رابط کاربری React برای اعلان‌ها* نیز اشاره کرده است (اگر وجود داشته باشد ارسال شود)file-crzskoe9o4wznmpsinhc2g. احتمالاً یک کامپوننت برای نمایش نوتیفیکیشن‌ها پیاده شده یا در برنامه است. با وجود مدل Notification و APIهای مرتبط، این بخش داده‌ای آماده است.**

* **پیشنهادات بهبود یا توسعه:**
  + **پرچم خوانده‌شدن (read flag): اطمینان حاصل شود مدل Notification دارای فیلدی مثل is\_read یا read\_at باشد تا اعلان‌های خوانده‌شده از خوانده‌نشده تفکیک شوند. اگر در مدل فعلی نیست، اضافه کردن آن توصیه می‌شود تا کاربر بتواند اعلان‌های جدید را تشخیص دهد یا همه را به‌خوانده علامت‌گذاری کند.**
  + **ربط شیء مرتبط: بهتر است هر اعلان به شیء مربوطه قابل وصل باشد؛ برای این کار می‌توان از GenericForeignKey (ContentType) یا فیلدهای اختصاصی استفاده کرد. به عنوان مثال اعلان بروزرسانی سفارش، مستقیماً به همان Order FK داشته باشد. در سیگنال Order اینکار غیرمستقیم انجام شده (Notification.objects.create با related\_object=instance)file-crzskoe9o4wznmpsinhc2g. احتمالاً در مدل Notification یک GenericForeignKey به نام related\_object یا مشابه تعریف شده است که هر نوع مدلی را می‌تواند لینک کند. اگر چنین نیست، می‌توان حداقل برای Order و شاید Tender این ارتباط را صراحتاً تعریف کرد (مثلاً فیلد optional به Order). هرچند عدم وجود هم مشکلی بزرگ نیست اما وجود آن بازیابی اعلان‌های مرتبط با یک سفارش خاص را آسان‌تر می‌کند.**
  + **اولویت‌ها و دسته‌بندی اعلان‌ها: در کد دیدیم اولویت 'medium' تنظیم شده بودfile-crzskoe9o4wznmpsinhc2g. اگر اولویت‌های دیگری هم استفاده می‌شود (مثلاً high برای موارد بسیار مهم)، شاید بد نباشد یک چارچوب برای آنها داشته باشیم یا از PriorityChoices استفاده کنیم. همچنین می‌توان اعلان‌ها را بر اساس نقش فیلتر کرد: مثلاً برخی اعلان‌ها مخصوص ست‌بند‌ها است (مثل ابلاغ سفارش جدید برای ست‌بندی) که شاید نوع جداگانه یا نشانه‌گذاری جداگانه‌ای بخواهد. این‌ها بیشتر جزئیات طراحی نرم‌افزار است و مدل فعلی قابلیت افزودن چنین جزئیاتی را دارد.**
  + **زمان‌بندی و انقضا: در حال حاضر هر Notification ایجاد می‌شود و می‌ماند تا زمانی که حذف شود. شاید برخی اعلان‌ها بعد از مدتی بی‌اعتبار شوند (مثلاً اعلان پیشنهاد جدید بعد از اتمام مناقصه دیگر ارزشی ندارد). می‌توان مکانیزمی داشت که اعلان‌های قدیمی خودکار آرشیو یا پاک شوند یا حداقل یک timestamp انقضا در آنها نگهداری شود. البته این سطح از پیچیدگی ممکن است فعلاً موردنیاز نباشد.**

**سناریو ۷: فاکتور و پرداخت**

**پرداخت و تسویه‌حساب بخش نهایی هر سفارشی است. سناریوی کلی این است که پس از نهایی‌شدن هزینه‌های یک سفارش (شامل هزینه طرح‌ها، چاپ، ست‌بندی، مراحل تکمیلی و ...)، یک فاکتور برای مشتری تولید می‌شود و وضعیت پرداخت آن پیگیری می‌گرددfile-crzskoe9o4wznmpsinhc2g. همچنین باید در نظر داشت که هزینه هر بخش از کار ممکن است سهم افراد مختلف (مثلاً ست‌بند یا کارگاه) باشد که نیاز به ثبت جداگانه دارد.**

* **مدل‌های مرتبط در پروژه: در کد ارسالی، مدل *Payment* برای مدیریت پرداخت وجود داردfile-crzskoe9o4wznmpsinhc2g. این مدل یک ارتباط OneToOne با *Order* دارد و شامل مبلغ پرداخت، روش پرداخت (مثلاً آنلاین، نقدی) و وضعیت پرداخت (pending, completed, failed) استfile-crzskoe9o4wznmpsinhc2g. مدل Payment کفایت می‌کند که بفهمیم آیا سفارش پرداخت شده یا خیر و از چه طریق. اما جزئیات آیتم‌های هزینه در این مدل دیده نمی‌شود.**

**در گفت‌وگو، به‌طور مشخص توصیه شد یک مدل *Invoice* (فاکتور) ساخته شودfile-crzskoe9o4wznmpsinhc2g که جزئیات بیشتری دارد:**

* + **ارتباط به Order و مشتری،**
  + **total\_amount (که در Payment هم بود)،**
  + **items: لیستی از آیتم‌های هزینه مانند هزینه چاپ، هزینه ست‌بندی، هزینه برش، آپشن‌هاfile-crzskoe9o4wznmpsinhc2g،**
  + **status پرداخت (پرداخت‌شده/در انتظار)file-crzskoe9o4wznmpsinhc2g و هر آنچه برای صورتحساب نیاز است.**

**به نظر می‌رسد Invoice مدلی است که کاربر هنوز پیاده نکرده و Payment نقش ساده‌شده آن را ایفا می‌کند. همچنین OrderItem و سایر مدل‌ها برخی اطلاعات مالی را دارند: مثلاً در OrderItem یک فیلد price برای هر آیتم سفارش وجود داردfile-crzskoe9o4wznmpsinhc2g. اما این قیمت احتمالاً قیمت آن آیتم (طرح یا قالب) است نه تفکیک بخش‌های کار.**

* **تطابق مدل‌ها با نیازهای سناریو: از منظر حداقلی، مدل Payment وضعیت پرداخت کل سفارش را رصد می‌کند که برای بسیاری از کاربردها کافیست (مثلاً جلوگیری از شروع کار تا وقتی پرداخت انجام نشده، یا نمایش پیام "در انتظار پرداخت" به مشتری). سناریوی گفتگو اما به جزئیات فاکتور اشاره دارد. مواردی مانند "*بخشی از این هزینه به ست‌بند پرداخت می‌شود*"file-crzskoe9o4wznmpsinhc2g نشان می‌دهد نیاز است که بدانیم سهم هر بخش چقدر بوده است. مدل‌های فعلی (OrderItem, Payment) این را صریح نگفته‌اند. OrderItem قیمت آیتم‌ها (مثلاً قیمت طرح یا قیمت پایه قالب) را دارد اما هزینه ست‌بندی یا آپشن‌های خاص احتمالاً در آن نیامده است. سیستم می‌تواند به صورت محاسباتی هزینه‌ها را تخمین بزند (براساس نوع چاپ، اندازه لباس، تعداد آپشن‌ها و ...)file-crzskoe9o4wznmpsinhc2g اما برای صدور فاکتور شفاف بهتر است این موارد مدل‌سازی شوند.**

**مدل Invoice پیشنهادی دقیقاً برای این شفاف‌سازی است. اگر Invoice پیاده شود، OrderItemهای مختلف یا هزینه خدمات مختلف می‌توانند به عنوان سطرهای Invoice Item لیست شوند. در غیر اینصورت، شاید یک راه حل میانی این باشد که در مدل Payment یا Order فیلدهایی مثل printing\_cost, design\_cost, workshop\_cost, setdesign\_cost اضافه شود تا سهم هر قسمت مشخص گردد. البته این کار تکرار داده است و وجود Invoice تمیزتر است.**

**به هر حال، در وضعیت فعلی، پرداخت‌ها تا حد وضعیت کلان مدیریت می‌شوند اما جزئیات فاکتور کامل پیاده نشده است. از آنجا که پروژه در حال رشد است، ممکن است فاز اول تنها نیاز به پیگیری پرداخت کامل یا ناموفق داشته و جزئیات را خارج از سیستم (مثلاً به صورت قرارداد شفاهی یا محاسبه دستی) انجام دهند. اما برای آینده قطعاً نیاز خواهد بود که ساختار فاکتور دقیق‌تری وجود داشته باشد.**

* **پیشنهادات بهبود یا توسعه:**
  + **پیاده‌سازی مدل Invoice و InvoiceItem: توصیه می‌شود مدل Invoice با فیلدهای ذکرشده (Order، مشتری، مبلغ کل، وضعیت) ایجاد شود و یک مدل InvoiceItem نیز برای آیتم‌های فاکتور. هر InvoiceItem می‌تواند شامل توضیح آیتم (مثلاً "چاپ دیجیتال ۵ عدد"، "هزینه طراحی گرافیکی"، "هزینه گلدوزی")، تعداد، قیمت واحد و قیمت کل باشد. این مدل‌ها باعث می‌شوند صورتحساب نهایی کاملاً شفاف و قابل ارائه به مشتری/کسب‌وکار باشد. حتی می‌توان از آن برای صدور صورتحساب به تفکیک ذی‌نفعان استفاده کرد (مثلاً یک Invoice برای ست‌بند جهت پرداخت دستمزد وی از سمت سیستم).**
  + **ارتباط Payment و Invoice: اگر Invoice اضافه شود، Payment می‌تواند به Invoice مرتبط شود به جای Order (چون ممکن است یک Order چندین Invoice داشته باشد: یکی به مشتری، یکی تسویه با کارگاه). اما در سیستم فعلی احتمالاً همه‌چیز در یک تراکنش از مشتری اخذ می‌شود و سپس خارج از سیستم سهم شرکا داده می‌شود. با افزودن Invoice، بهتر است مدل Payment به Invoice اشاره کند تا وضعیت پرداخت هر فاکتور مشخص باشد.**
  + **ثبت تاریخچه پرداخت: مدل Payment فقط وضعیت جاری را دارد. اگر نیاز باشد پیگیری کنیم که مثلاً مشتری چه زمانی پرداخت کرد یا اگر پرداخت ناموفق داشت، چه شد، می‌توان History یا Log برای Payment در نظر گرفت. شاید اضافه کردن فیلدهایی نظیر paid\_at (زمان پرداخت موفق) یا attempt\_count مفید باشد. البته این‌ها جزئیات تکمیلی هستند و در صورت استفاده از درگاه پرداخت آنلاین، معمولاً لاگ‌های جداگانه‌ای نگهداری می‌شود.**
  + **نمایش فاکتور در داشبورد: از دید کاربردی، پس از پیاده‌سازی مدل‌های فوق، باید در داشبورد مشتری بخشی برای مشاهده فاکتور و وضعیت پرداخت باشدfile-crzskoe9o4wznmpsinhc2g. همچنین داشبورد کسب‌وکار بتواند فاکتورهای مربوط به خود (مثلاً سهم خودش یا کل سفارش‌هایی که انجام داده) را ببیند. این‌ها خارج از بحث مدل است ولی چون گفته شد هدف استفاده گزارش برای تصمیم‌گیری توسعه بعدی است، اشاره به آن خالی از لطف نیست.**

**جمع‌بندی: به طور کلی، پروژه Pardach دارای مدل‌های داده‌ی جامعی است که اکثر سناریوهای مطرح‌شده را پوشش می‌دهند. سناریوهای ثبت سفارش با انعطاف بالا (شامل قالب‌های چندبخشی و طرح‌های متنوع)، مدیریت حرفه‌ای طرح‌ها (برچسب، دسته، خانواده، آپلود گروهی)، برون‌سپاری از طریق مناقصه، ست‌بندی نیمه‌خودکار با دخالت نیروی انسانی، مدیریت گردش کار در کارگاه‌ها و اطلاع‌رسانی و تسویه همگی در طراحی بانک اطلاعاتی لحاظ شده‌اند. هر کجا کاستی جزئی مشاهده شد (مثلاً عدم وجود فیلد optional در Tender یا مدل Invoice)، پیشنهادات مشخصی برای ارتقاء داده شد تا تیم توسعه در فاز بعدی روی آنها متمرکز شود. با اتکا به این زیرساخت داده‌ای، پیاده‌سازی بخش‌های باقیمانده در Frontend (داشبوردهای متنوع برای نقش‌ها) و Backend (منطق کسب‌وکار، سیگنال‌ها و سرویس‌های خودکار) امکان‌پذیر خواهد بود و سیستم میتواند نیازهای مطرح‌شده کاربران را بطور کامل برآورده سازد.**

**Sources**

**You said:**

**خب در ادامه بقیه فایل های backend , frontend را جهت بررسی با تطابق با سنارویو ها و مدل ها بررسی کن باگ ها و کاستی ها را پیدا کن و پیشنهاد های تکمیلی را ارائه کن**

**ChatGPT said:**

**برای اینکه بررسی جامع‌تری انجام دهم، لطفاً مشخص کنید:**

1. **آیا تمرکز بررسی باید بر کدهای موجود در فولدرهای backend و frontend پروژه GitHub باشد یا فایل دیگری هم دارید که سناریوها یا نیازمندی‌ها را مشخص می‌کند؟**
2. **آیا انتظار دارید بررسی شامل فقط بررسی فنی کدها (مثلاً ساختار viewها، APIها، state management، routeها) باشد یا همچنین تجربه کاربری و طراحی رابط کاربری (UX/UI) را نیز در بر بگیرد؟**
3. **کدام یک برای شما اولویت بیشتری دارد: شناسایی باگ‌ها، کمبودها نسبت به سناریوها، یا پیشنهادهای تکمیلی برای بهبود توسعه؟**

**با پاسخ به این موارد، می‌توانم تحلیل دقیق‌تری ارائه دهم.**

**You said:**

**github amin311/pardach و فایل های backend , frontend آن را بررسی کن**

**ChatGPT said:**

**بسیار خوب. من تمام فایل‌های backend و frontend پروژه Pardach را با دقت بررسی خواهم کرد تا:**

* **میزان تطابق کدها با سناریوها و مدل‌های تعریف‌شده ارزیابی شود،**
* **باگ‌ها و کاستی‌های احتمالی در پیاده‌سازی شناسایی شود،**
* **و پیشنهادهای تکمیلی برای بهبود توسعه ارائه شود.  
  نتایج بررسی در گزارشی ساختاریافته به شما ارائه خواهد شد.**

**Research completed in 12m · 50 sources · 19 searches**

**بررسی فنی کامل فایل‌های Backend و Frontend پروژه Pardach**

**گزارش بررسی پروژه «Pardach»**

**در این گزارش، بخش‌های بک‌اند و فرانت‌اند به‌صورت جداگانه بررسی شده و هر بخش بر اساس مؤلفه‌های اصلی آن تحلیل می‌شود. سپس برای هر باگ یا کاستی شناسایی‌شده، راه‌حل یا پیشنهاد بهبود ارائه می‌گردد. همچنین در مواردی که سناریوهای تعریف‌شده (مطابق تعامل کاربر با سیستم و مدل‌های داده) هنوز پیاده‌سازی نشده‌اند، به نیازمندی‌های توسعه آن‌ها اشاره شده است. این گزارش بر پایه‌ی فهم دقیق مدل‌های داده و کد فعلی پروژه تهیه شده است.**

**بخش بک‌اند (Backend)**

**۱. احراز هویت و مدیریت کاربران**

**وضعیت فعلی: مکانیزم ثبت‌نام و ورود کاربران با استفاده از جنگو REST و JWT پیاده‌سازی شده است. API ثبت‌نام (/api/auth/register/) کاربر جدید را ایجاد کرده و توکن‌های JWT را برمی‌گرداند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/views.py" \l "L15-L24" \t "_blank). ورود (/api/auth/login/) نیز با دریافت نام‌کاربری و رمز عبور، درصورت موفقیت توکن JWT برمی‌گرداند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/views.py" \l "L31-L39" \t "_blank). همچنین endpointهایی برای دریافت و ویرایش اطلاعات کاربر (/api/auth/users/ و /api/auth/users/{id}/) وجود دارد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/views.py" \l "L51-L59" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/views.py#L93-L101)**. مکانیزم تغییر نقش جاری کاربر (/api/auth/set-role/) هم فراهم شده است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/views.py" \l "L124-L132" \t "_blank).**

**کاستی‌ها و باگ‌ها:**

* **عدم پیاده‌سازی تأیید حساب کاربری: فیلدهایی مثل is\_verified یا تأیید ایمیل/شماره تماس که در طرح اولیه مطرح بودند، در مدل کاربر نهایی حذف شده‌اند. درنتیجه هیچ مکانیزمی برای تأیید هویت یا فعال‌سازی حساب وجود ندارد. پیشنهاد می‌شود برای سناریوهای امنیتی (تأیید ایمیل یا ارسال کد فعال‌سازی) رویه‌ای اضافه شود. به‌عنوان مثال، می‌توان پس از ثبت‌نام، ایمیل تأیید ارسال کرد یا فیلد is\_active را تا زمان تأیید غیرفعال گذاشت.**
* **مدیریت نقش‌ها: هرچند مدل Role برای تعریف نقش‌ها ایجاد شده[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/models.py" \l "L83-L86" \t "_blank)، اما تخصیص نقش به کاربر به‌صورت کامل پیاده‌سازی نشده است. در مدل User فقط یک فیلد current\_role وجود دارد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/models.py" \l "L34-L37" \t "_blank) و در کد، کاربر می‌تواند نقش فعلی خود را تغییر دهد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/views.py" \l "L128-L135" \t "_blank). اما اگر کاربری چندین نقش بالقوه داشته باشد (مثلاً هم طراح و هم صاحب کسب‌وکار)، مکانیزمی برای نگهداری همه نقش‌های او جز در فیلد JSON roles پیش‌بینی نشده است. در بخش‌هایی از کد به user.roles مستقیماً ارجاع شده[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py" \l "L40-L46" \t "_blank)، در حالی‌که مدل User فعلی چنین attribute‌ای ندارد (فقط current\_role دارد). این موضوع می‌تواند منجر به خطا در فیلتر کردن داده‌ها شود. پیشنهاد: یا ساختار مدل کاربر را به M2M نقش‌ها ارتقاء دهید، یا از گروه‌ها/پرمیژن‌های Django استفاده کنید. در کوتاه‌مدت، می‌توان propertyی برای User تعریف کرد که نقش‌های قابل انتخاب کاربر (مثلاً بر اساس پروفایل‌های وابسته به وی) را برگرداند و در کد به‌جای user.roles از آن استفاده شود.**
* **دسترسی‌ها و پرمیژن‌ها: فعلاً اکثر endpointها تنها شرط IsAuthenticated یا IsAdminUser دارند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L17-L25" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/views.py#L52-L59)**. برای سناریوهای پیچیده‌ترِ نقش‌ها (مانند دسترسی صاحبان کسب‌وکار به منابع خودشان، یا طراحان فقط به آیتم‌های مربوط به خود)، نیاز به پرمیژن‌های اختصاصی است. بخشی از این منطق به‌صورت دستی در ویوها بررسی شده (مثلاً دسترسی به جزئیات کاربر فقط توسط خود کاربر یا ادمین[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/views.py" \l "L79-L87" \t "_blank))، اما بهتر است این موارد به کلاس‌های پرمیژن مجزا (مثلاً IsOwner، IsBusinessOwner و ...) تبدیل شوند تا مدیریت و نگهداری آن آسان‌تر باشد.**
* **سناریوی تعامل با هوش مصنوعی: در اهداف پروژه ذکر شده که کاربر با هوش مصنوعی تعامل دارد، اما در بک‌اند هیچ مولفه مشخصی مربوط به AI (مثلاً مدل یادگیری ماشین، یا endpointهایی برای پردازش هوشمند) پیاده‌سازی نشده است. بنابراین سناریوهای وابسته به هوش مصنوعی عملاً غایب‌اند و نیاز به توسعه دارند. برای نمونه، اگر قرار بوده سیستمی برای پیشنهاد طرح‌ها یا پردازش خودکار تصاویر وجود داشته باشد، باید APIهای مربوطه طراحی و پیاده‌سازی شوند.**

**پیشنهاد بهبود:  
– *افزودن تأیید هویت دو مرحله‌ای یا تأیید ایمیل/شماره*: جهت اطمینان از واقعی‌بودن کاربران جدید، می‌توان پس از ثبت‌نام کاربر یک کد تأیید ارسال کرد و endpoint تأیید را اضافه نمود.  
– *تکمیل سیستم نقش‌ها*: مدل Role و مکانیزم SetRole وجود دارد اما باید تضمین شود هر کاربر همه نقش‌های مرتبط را داشته باشد. پیشنهاد می‌شود فیلد roles در مدل User به‌صورت فهرستی از نام نقش‌ها یا یک ManyToMany به Role تنظیم شود. همچنین endpoint ایجاد نقش جدید توسط ادمین پیاده‌سازی شده[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/authentication/views.py" \l "L152-L160" \t "_blank) اما انتساب نقش به کاربر (مثلاً تعیین یک کاربر به عنوان "طراح") در API پیش‌بینی نشده است؛ می‌توان endpoint یا فیلدی برای این کار در UserSerializer افزود.  
– *بهبود پرمیژن‌ها*: تعریف پرمیژن‌های کاستوم در فایل permissions.py (مثل IsBusinessOwnerOrAdmin, IsDesignerOfRecord و غیره) که منطق دسترسی در آن‌ها کپسوله شود. این کار از تکرار شرط‌های دسترسی در هر ویو جلوگیری کرده و خوانایی کد را بالا می‌برد.  
– *پیاده‌سازی سناریوهای هوش مصنوعی*: هر قابلیتی از سیستم که قرار بوده به‌صورت هوشمند عمل کند (از متن سؤال چنین برمی‌آید) نیازمند طراحی دقیق است. به عنوان مثال، اگر قرار بوده یک «دستیار طراحی هوشمند» وجود داشته باشد که طرح‌ها را بهبود دهد یا پیشنهاد بدهد، باید مشخص شود این قسمت در بک‌اند چگونه پیاده می‌شود (ارتباط با یک سرویس ML، ذخیره مدل‌ها، endpointهای مخصوص پردازش تصویر و ...). در وضعیت کنونی، نبود هرگونه کد مرتبط نشان می‌دهد این بخش هنوز توسعه نیافته و باید به عنوان فاز بعدی پروژه در نظر گرفته شود.**

**۲. مدیریت کسب‌وکارها و پروفایل نقش‌ها**

**وضعیت فعلی: سیستم از مفهوم "کسب‌وکار" پشتیبانی می‌کند. مدل Business معرف یک کسب‌وکار است که مالک (owner) دارد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/models.py" \l "L18-L24" \t "_blank) و می‌تواند کاربرانی با نقش‌های مشخص (مدیر یا کارمند) داشته باشد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/models.py" \l "L47-L56" \t "_blank). برای Business اکشن‌های اصلی CRUD در API پیاده‌سازی شده است: ایجاد و لیست کسب‌وکار (BusinessListCreateView)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/views.py#L16-L24)[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/views.py#L34-L43)**، ویرایش/حذف (BusinessDetailView) با بررسی دسترسی مالک یا ادمین[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/views.py" \l "L80-L88" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/views.py#L110-L118)**. همچنین افزودن کاربر به کسب‌وکار (BusinessUserListCreateView) برای مالک/ادمین ممکن است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/views.py" \l "L148-L156" \t "_blank). مدل BusinessActivity جهت ثبت وقایع مهم مربوط به کسب‌وکار (فروش طرح، ایجاد سفارش، پرداخت و ...) تعریف شده است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/models.py" \l "L67-L75" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/models.py#L78-L86)**. این مدل فعلاً بیشتر مفهومی است و ظاهراً هنوز API مستقلی برای فهرست کردن فعالیت‌ها وجود ندارد (با این‌که در فرانت‌اند استفاده شده، در ویوهای بک‌اند متدی برای /businesses/{id}/activities/ مشاهده نمی‌شود).**

**کاستی‌ها و باگ‌ها:**

* **نبود API فعالیت‌های کسب‌وکار: در فرانت‌اند درخواست دریافت لیست BusinessActivity برای هر کسب‌وکار انجام می‌شود[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/businesses/DetailBusiness.js" \l "L30-L38" \t "_blank)، اما در بک‌اند مسیری برای آن تعریف نشده است (هیچ View یا endpointی برای activities/ در Business مشاهده نشد). این عدم تطابق باعث می‌شود بخش «فعالیت‌های کسب‌وکار» در رابط کاربری کار نکند. پیشنهاد: افزودن یک View (مثلاً BusinessActivityListView) که business\_id را گرفته و فعالیت‌های آن کسب‌وکار را برمی‌گرداند. همچنین می‌توان BusinessActivity را صرفاً جهت گزارش‌گیری در بخش ادمین استفاده کرد؛ در این صورت، یا باید API آن حذف شود یا بخش فرانت‌اند اصلاح شود.**
* **مدیریت نقش‌های کاربران کسب‌وکار: در اضافه کردن کاربران به یک Business، نقش کاربر (مدیر یا کارمند) قابل تنظیم است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/models.py" \l "L48-L56" \t "_blank). اما در API ایجاد (BusinessUserListCreateView.post) مقدار role از داده ورودی استفاده می‌شود بدون هیچ اعتبارسنجی اضافی (کاربر می‌تواند خود را مدیر دیگری کند؟)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/views.py#L152-L160)**. البته در عمل تنها مالک یا ادمین می‌تواند این فراخوانی را انجام دهد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/views.py" \l "L148-L156" \t "_blank)، ولی بهتر است نقش جدید محدود به دو مقدار معتبر باشد یا اگر کاربری به‌جز مالک اضافه می‌شود، پیش‌فرضاً کارمند باشد. پیشنهاد: اعتبارسنجی فیلد role در Serializer (یا استفاده از ChoiceField) و جلوگیری از ایجاد مدیر دوم مگر توسط ادمین.**
* **عدم وجود پروفایل‌های مجزا: در کد اولیه صحبت از پروفایل مشتری، صاحب کسب‌وکار و طراح بود، اما در کد نهایی اینها حذف و همه‌چیز در مدل‌های Business و User ادغام شده است. بنابراین سناریوی داشتن اطلاعات تکمیلی برای انواع کاربر (مثلاً مشخصات صاحب کسب‌وکار شامل نام کسب‌وکار یا طراح شامل لینک نمونه‌کار) فعلاً پشتیبانی نمی‌شود. اگر این جزئیات مهم است، می‌توان مدل‌های پروفایل را (با ارتباط OneToOne به User) بازگرداند یا فیلدهای مرتبط را به مدل User یا Business اضافه کرد. برای نمونه، پیشنهاد: فیلد portfolio برای طراحان یا business\_name برای صاحبان کسب‌وکار می‌تواند در مدل User یا از طریق Role مرتبط، ذخیره شود. در شرایط فعلی، اطلاعاتی مانند نام کسب‌وکار در مدل Business موجود است اما مثلاً هیچ راهی برای تمایز یک کاربر طراح (designer) از طریق UserSerializer نیست.**
* **بهبود ثبت و گزارش فعالیت‌ها: مدل BusinessActivity پتانسیل ثبت رخدادهای مهم را دارد اما جهت خودکارسازی، هر زمان عملی انجام می‌شود (مثل تأیید یک طرح یا تکمیل سفارش)، باید یک BusinessActivity ساخته شود. اکنون چنین عملکردی پیاده نشده است. پیشنهاد: در Viewهای مربوط (مثلاً تأیید سفارش، فروش طرح، و ...) ایجاد رکورد BusinessActivity با نوع مناسب (مثلاً design\_sale هنگام فروش یک طرح) اضافه شود تا مدیران کسب‌وکار تاریخچه فعالیت را در داشبورد ببینند.**

**۳. مدیریت طرح‌ها (Designs) و سناریوی «ست‌بندی» طرح‌ها**

**وضعیت فعلی: مدل Design برای طرح‌های گرافیکی پیاده‌سازی شده است. هر طرح شامل عنوان، توضیح، فایل اصلی (برداری یا تصویری) و تصویر پیش‌نمایش، وضعیت تأیید، قیمت و برخی شاخص‌ها (تعداد بازدید/دانلود) است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py" \l "L80-L89" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py#L92-L101)**. همچنین طرح به طراح (designer) و ایجادکننده (created\_by) مرتبط است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py" \l "L75-L83" \t "_blank). دسته‌بندی‌ها و برچسب‌ها نیز توسط مدل‌های DesignCategory و Tag پشتیبانی می‌شوند و طرح می‌تواند چندین دسته و برچسب داشته باشد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py" \l "L96-L101" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py#L14-L22)**. ساختار «خانواده طرح‌ها» برای گروه‌بندی طرح‌های مرتبط تعریف شده: مدل Family یک مجموعه طرح را نشان می‌دهد و FamilyDesignRequirement نیازمندی‌های آن مجموعه (مثلاً چند طرح از نوع خاص) را مشخص می‌کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py" \l "L116-L125" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py#L128-L133)**. هر Design می‌تواند عضو صفر یا چند Family باشد (ارتباط ManyToMany از طریق مدل میانی DesignFamily)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py#L140-L148)**.**

**از منظر API، تمامی عملیات اصلی برای طرح‌ها مهیاست:**

* **لیست و ایجاد طرح جدید: فقط برای کاربران احراز هویت‌شده مجاز است. طرح‌های عمومی یا متعلق به کاربر نمایش داده می‌شوند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L101-L109" \t "_blank) و فیلترهایی برای دسته، برچسب، خانواده و جستجو وجود دارد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L104-L112" \t "_blank). ایجاد طرح جدید، فایل‌ها (تصویر محصول و فایل SVG) را دریافت کرده و کاربر جاری را به‌عنوان created\_by تنظیم می‌کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L124-L133" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py#L126-L134)**.**
* **جزئیات، ویرایش و حذف طرح: فقط سازنده طرح یا ادمین می‌تواند ویرایش/حذف کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L160-L168" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py#L184-L191)**. مشاهده جزئیات نیز برای طرح‌های خصوصی تنها به مالک یا ادمین محدود شده است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L146-L154" \t "_blank). به‌علاوه، شمارنده بازدید هر بار با مشاهده طرح افزایش می‌یابد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L146-L154" \t "_blank) (هرچند پیاده‌سازی تابع increment در کد اصلی دیده نشد، در تست‌ها به آن اشاره شده است).**
* **بارگذاری دسته‌ای طرح‌ها: قابلیت Batch Upload به کاربر اجازه می‌دهد چندین فایل طراحی را یکجا آپلود کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L206-L215" \t "_blank). برای هر فایل، یک Design جدید با عنوان پیش‌فرض (نام فایل) و وضعیت پیش‌نویس ایجاد شده و برچسب‌ها، دسته‌ها و خانواده‌های انتخاب‌شده اختصاص می‌یابد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L218-L227" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py#L230-L239)**. این ویژگی برای سرعت‌بخشی به افزودن طرح‌ها پیاده‌سازی شده است.**

**کاستی‌ها و باگ‌ها:**

* **ناسازگاری نام فیلدها بین بک‌اند و فرانت‌اند: در Serializer طرح، فیلدهای svg\_file و product\_image تعریف شده‌اند که به ترتیب به فایل اصلی و تصویر پیش‌نمایش اشاره دارند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/serializers.py" \l "L52-L58" \t "_blank). اما در مدل Design، این فیلدها به نام‌های دیگری (file و preview\_image) وجود دارند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py" \l "L80-L89" \t "_blank). ظاهراً Serializer با استفاده از source این تطابق را انجام می‌دهد. مشکلی که وجود دارد در سمت فرانت‌اند است: صفحه ایجاد طرح جدید از فیلدهای category\_ids, tag\_ids و family\_ids استفاده می‌کند و آن‌ها را به API ارسال می‌کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/designs/CreateDesign.jsx" \l "L16-L24" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/designs/CreateDesign.jsx#L35-L43)**، که صحیح و منطبق بر Serializer است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/serializers.py" \l "L40-L48" \t "_blank). اما صفحه ویرایش طرح هنوز از فیلدهای categories و tags و family (مستقیم) استفاده می‌کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/designs/EditDesign.jsx" \l "L60-L68" \t "_blank) و هنگام ذخیره تغییرات نیز آن‌ها را تحت همین کلیدها می‌فرستد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/designs/EditDesign.jsx" \l "L166-L175" \t "_blank). این در حالی است که API انتظار دارد کلیدهای ارسالی category\_ids و ... باشند (یا داده‌ها در قالب متفاوت). این ناهماهنگی باعث می‌شود ویرایش طرح به درستی کار نکند یا تغییرات دسته/برچسب ذخیره نشود. پیشنهاد: یکسان‌سازی استفاده از فیلدهای شناسه‌ (ids) در تمام صفحات. بهتر است کامپوننت EditDesign نیز همان CreateDesign عمل کند و از ...\_ids استفاده نماید. همچنین می‌توان در Serializer برای راحتی کاربر، قابلیت پردازش هر دو حالت را افزود (اما روش صحیح‌تر، اصلاح فرانت‌اند است).**
* **عدم تعیین خودکار طراح (designer): هر طرح باید یک طراح (designer) داشته باشد. در مدل این فیلد اجباری است (احتمالاً null=False) و نشان‌دهنده کاربری است که طراح طرح است. در API ایجاد طرح، کد بک‌اند designer را تنظیم نمی‌کند و فقط created\_by را مقداردهی می‌کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L130-L134" \t "_blank). اگر کاربر جاری همان طراح است، عدم تنظیم این فیلد ممکن است منجر به خطای اعتبارسنجی شود (در تست‌ها، هنگام ایجاد طرح، ظاهراً این مشکل رخ نداده است و طرح ساخته شده[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/tests.py" \l "L84-L91" \t "_blank)، احتمالاً مدل Design اجازه designer=None را می‌دهد). برای حفظ منطق دامنه، بهتر است designer نیز به صورت پیش‌فرض کاربر جاری در نظر گرفته شود. پیشنهاد: در متد post ایجاد طرح، پس از valid شدن Serializer، شیء Design را با design = serializer.save(created\_by=request.user, designer=request.user) ذخیره کنید (البته در صورتی که کاربر نقش طراح دارد). در غیر این صورت، می‌توان فیلد designer را در Serializer اختیاری (blank=True) گرفت تا در صورت عدم ارسال، خالی بماند. این موضوع به سناریو کسب‌وکار بستگی دارد: اگر همیشه ایجادکننده همان طراح است، تنظیم خودکار منطقی است؛ اگر ممکن است ادمین به‌جای طراح طرح وارد کند، باید در ورودی API شناسه طراح را نیز بگیرد.**
* **ابهام در استفاده از فیلد type: فیلد type در مدل Design وجود دارد و ظاهراً برای نوع طرح استفاده می‌شود (مثلاً مقداری مانند "vector" در تست‌ها داده شده است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/tests.py" \l "L40-L48" \t "_blank)). همچنین در نیازمندی‌های خانواده (FamilyDesignRequirement) هم یک نوع طرح تعریف شده که با همین فیلد مقایسه می‌شود[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py" \l "L126-L133" \t "_blank). اما در کد تعریف مدل Design، فیلد type مشخص نشده بود (ممکن است از قلم افتاده یا در migration اضافه شده باشد). در هر حال، استفاده کنونی از type مشخص نیست که به فرمت فایل اشاره دارد (Vector/Raster) یا به طبقه‌بندی طرح (مثل "لوگو"، "پوستر" و ...). تست‌ها مقدار 'vector' را به عنوان نوع طرح استفاده کرده‌اند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/tests.py" \l "L40-L48" \t "_blank)، در حالی که احتمالاً انتظار می‌رفت نوع‌هایی مثل "لوگو" یا "طرح چاپی" باشد. این عدم‌وضوح می‌تواند باعث شود شرط fulfilled شدن نیازهای خانواده‌ها نادرست کار کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py" \l "L126-L133" \t "_blank) (مثلاً اگر نیاز «لوگو x2» تعریف شده و نوع طرح در Design "vector" ذخیره شده باشد، هیچ‌وقت fulfilled نمی‌شود). پیشنهاد: اگر منظور از type قالب فایل است، بهتر است نام فیلد را format یا چیزی شبیه آن گذاشت و برای نوع طرح از یک فیلد جداگانه (یا استفاده از دسته‌بندی‌ها) بهره برد. اما اگر type منظور نوع طراحی (لوگو، کارت ویزیت، ...) است، باید مقادیر مجاز آن مشخص و احتمالاً محدود به ChoiceField شود. به طور کلی، این فیلد نیازمند بازطراحی یا حداقل مستندسازی است تا در سناریوهای مختلف به درستی استفاده شود.**
* **توابع افزایش بازدید/دانلود: متدهای increment\_view\_count و increment\_download\_count برای Design در تست‌ها استفاده شده‌اند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/tests.py" \l "L53-L59" \t "_blank) اما در کد مدل موجود نیستند (شاید فراموش شده‌اند اضافه شوند). این مسئله مستقیماً باگ ایجاد نمی‌کند مگر اینکه جای دیگری فراخوانی شوند. پیشنهاد: افزودن این متدها در کلاس Design. برای مثال:**

**python**

**CopyEdit**

**def increment\_view\_count(self):**

**self.views\_count = models.F('views\_count') + 1**

**self.save(update\_fields=['views\_count'])**

**(یا معادل ساده‌تر: افزایش مقدار و ذخیره). البته هم‌اکنون افزایش بازدید در ویوی get انجام می‌شود[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/views.py" \l "L146-L154" \t "_blank) که به صورت inline مقدار را زیاد می‌کند؛ اما داشتن متد در مدل تمیزتر است و تست‌ها نیز بر همین اساس نوشته شده‌اند.**

* **سناریوهای ناقص:  
  – *تأیید طرح‌ها توسط ادمین:* وضعیت طرح‌ها شامل «در انتظار تأیید», «تأیید شده» و «رد شده» است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py" \l "L66-L74" \t "_blank)، اما هیچ endpoint یا رویه‌ای برای تأیید/رد طرح توسط ادمین تعریف نشده (مثلاً ادمین نتواند طرح‌های ایجاد‌شده را بازبینی و وضعیتشان را تغییر دهد). اگر فرآیند کسب‌وکار نیاز به تأیید دستی دارد، لازم است API یا صفحه‌ای برای این کار در نظر گرفته شود. درحال‌حاضر، طرح در حالت پیش‌نویس ایجاد می‌شود و ظاهراً مستقیماً قابل استفاده است مگر اینکه طراحی غیرفعال (is\_public=False) باشد. پیشنهاد: افزودن متدی در DesignDetailView برای تغییر status توسط ادمین (یا سازوکار خودکار در صورت برآورده شدن شروط).  
  – *دانلود فایل طرح:* اگرچه شمارنده دانلود وجود دارد، هیچ Endpoint مشخصی برای دانلود فایل اصلی طرح دیده نشد. احتمالاً دانلود از طریق لینک مستقیم به فایل انجام می‌شود. بهتر است برای دانلود فایل (به خصوص فایل اصلی SVG/PDF) یک API یا View محافظت‌شده تنظیم شود تا فقط کاربران مجاز (مثلاً پس از خرید) بتوانند دانلود کنند، و در آنجا نیز increment\_download\_count فراخوانی گردد.  
  – *جستجوی پیشرفته و صفحه‌بندی:* در لیست طرح‌ها، امکان فیلتر وجود دارد ولی pagination پیاده نشده است. با توجه به اینکه ممکن است تعداد طرح‌ها زیاد شود، افزودن صفحه‌بندی (مثلاً با‌ استفاده از limit/offset یا DRF PageNumberPagination) توصیه می‌شود. به این ترتیب فرانت‌اند نیز به‌صورت تدریجی طرح‌ها را لود می‌کند که تجربه کاربری را بهبود می‌بخشد.**

**پیشنهاد بهبود کلی در بخش طرح‌ها:  
– *ارتقای رابط API طرح‌ها:* می‌توان Endpointهای مجزایی برای عملیات متداول ایجاد کرد. مثلاً Endpointی برای آپلود فقط تصویر پیش‌نمایش یا به‌روزرسانی فایل SVG. همچنین تهیه مستندات Swagger (که با @extend\_schema تا حدی آماده شده) به توسعه‌دهندگان فرانت‌اند کمک می‌کند تا بدانند چه فیلدهایی لازم است.  
– *بهبود Batch Upload:* در آپلود دسته‌ای، درحال‌حاضر از alert برای پیام موفقیت/خطا استفاده می‌شود (در کد فرانت‌اند مشاهده شد) که تجربه کاربری ساده‌ای دارد. می‌توان این را ارتقاء داد تا مثلاً لیست فایل‌های آپلودشده با وضعیت موفق/خطا نمایش داده شود. همچنین محدودیت حجم و تعداد فایل را سمت سرور (علاوه بر تابع validate\_file\_size کنونی) بهتر است مشخص کنیم و به کاربر بازخورد دهیم.  
– *پشتیبانی از انواع فایل بیشتر:* محدودیت فعلی فایل اصلی به پسوندهای برداری (svg, ai, eps, pdf) است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/designs/models.py" \l "L80-L88" \t "_blank) و تصویر پیش‌نمایش به هر تصویر. اگر نیاز است فرمت‌های دیگری (مثلاً PNG به عنوان فایل اصلی برای طرح‌های غیر برداری) پشتیبانی شود، باید در validators لحاظ گردد.**

**۴. سفارش‌ها و فرآیند «ست‌بندی» (ترکیب طرح‌ها در سفارش‌ها)**

**وضعیت فعلی: بخش سفارش‌ها شامل مدل Order و OrderItem است که برای مدیریت سفارش‌های چاپ تعریف شده‌اند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/orders/models.py" \l "L15-L24" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/orders/models.py#L99-L107)**. هر Order ممکن است چندین آیتم داشته باشد که هر آیتم می‌تواند به یک طرح مرتبط باشد (طرحی که قرار است چاپ شود)[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/orders/models.py" \l "L100-L108" \t "_blank) و شامل جزئیاتی نظیر محل چاپ (PrintLocation)، تعداد، قیمت واحد و ... است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/orders/models.py" \l "L102-L110" \t "_blank). وضعیت‌های مختلفی برای سفارش (پیش‌نویس، در انتظار تأیید، تأیید‌شده، در حال انجام، تکمیل‌شده، لغو‌شده) در نظر گرفته شده[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/orders/models.py" \l "L17-L25" \t "_blank) تا جریان کار سفارش را منعکس کند. همچنین برای هر سفارش یک‌سری ویژگی‌های پارچه (جنس، رنگ، اندازه و ...) و پیش‌پرداخت/قیمت کل تعریف شده است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/orders/models.py" \l "L26-L34" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/orders/models.py#L48-L56)**.**

**آنچه پروژه را متمایز می‌کند سناریوی «ست‌بندی طرح‌ها» است که احتمالاً به این معناست که پس از ثبت سفارش توسط مشتری، یک طراح (یا "ست‌بند") باید طرح‌های موردنیاز را آماده و روی لباس جانمایی کند. برای این منظور:**

* **درخواست ست‌بندی (SetDesignRequest): مدل SetDesignRequest نشان‌دهنده یک درخواست برای آماده‌سازی طراحی‌های سفارش است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L92-L101" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py#L124-L132)**. این درخواست شامل مرجع به سفارش، اولویت، وضعیت (در انتظار، اختصاص داده‌شده به طراح، در حال انجام، تکمیل‌شده و ...) و شرح نیازمندی‌هاست[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L92-L101" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py#L96-L104)**. همچنین تصاویر مرجع (Designهای نمونه) می‌توانند پیوست شوند و مهلت انجام کار مشخص می‌گردد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L98-L106" \t "_blank). پس از اختصاص به یک طراح، طراح نهایی که کار را انجام داده و فایل را تحویل داده نیز ثبت می‌شود (result\_set)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py#L108-L116)**. متدهایی برای assign کردن طراح و کامل کردن درخواست وجود دارد که به‌طور خودکار اعلان (Notification) ارسال می‌کنند و وضعیت سفارش را به "in\_progress" تغییر می‌دهند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L127-L135" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py#L145-L154) **– این نشان می‌دهد که پس از تکمیل طراحی‌ها، سفارش چاپ وارد فاز انجام می‌شود.**
* **ست‌بندی (SetDesign): این مدل خروجی کار طراح را نگهداری می‌کند، یعنی فایل نهایی طراحی‌شده که قرار است چاپ شود[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L14-L23" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py#L36-L44)**. هر SetDesign به یک OrderItem تعلق دارد (یعنی طرح نهایی برای یک آیتم لباس)[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L14-L22" \t "_blank) و یک طراح انجام‌دهنده کار دارد. چند نسخه (ورژن) از ست‌بندی قابل نگهداری است (در صورت اصلاحات، ورژن جدید با اشاره به نسخه قبل ایجاد می‌شود)[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L28-L36" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py#L63-L71)**. وضعیت‌های "در انتظار ست‌بند"، "در حال انجام"، "منتظر تأیید مشتری/چاپخانه"، "تأیید/تکمیل‌شده" و "رد‌شده" برای این مدل تعریف شده[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L40-L49" \t "_blank). پس از اینکه طراح کار را تمام کرد، وضعیت به pending\_approval می‌رود و منتظر تأیید مشتری یا مدیر چاپخانه می‌ماند. زمانی‌که مشتری تأیید کرد، وضعیت completed می‌شود و امکان پرداخت به طراح فراهم می‌شود[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py" \l "L86-L94" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py#L100-L108)**. فیلد paid نشان می‌دهد آیا هزینه‌ای به طراح پرداخته شده یا خیر[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L50-L53" \t "_blank).**
* **پرداخت به طراح: پس از تأیید ست‌بندی توسط مشتری/چاپخانه، API اختصاصی برای ثبت پرداخت به طراح تعریف شده (/api/set-design/{id}/pay/)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py#L102-L111)[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py#L120-L128)**. این API فقط برای سفارش‌دهنده یا ادمین در دسترس است و چک می‌کند ست‌بندی تأیید شده و قبلاً پرداخت نشده باشد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py" \l "L116-L124" \t "_blank). سپس یک رکورد پرداخت (DesignerPayment) ایجاد می‌کند و وضعیت paid را True می‌کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py" \l "L126-L135" \t "_blank). بنابراین سناریوی تقسیم سود یا پرداخت دستمزد طراح پس از انجام کار پوشش داده شده است.**

**کاستی‌ها و باگ‌ها:**

* **عدم تعریف نقش "ست‌بند" به‌طور کامل: در مدل User، چیزی به نام set\_designer وجود ندارد (مثلاً یک گروه یا نقش) ولی در کد limit\_choices\_to={"roles\_\_contains": "set\_designer"} برای انتخاب طراح ست‌بندی استفاده شده[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L20-L27" \t "_blank) و نیز در فیلتر queryset ست‌بندی چک شده که اگر user.roles حاوی 'set\_designer' بود، فقط ست‌های خودش را ببیند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py" \l "L40-L46" \t "_blank). این نشان می‌دهد قرار بوده نقش خاصی برای طراحان ست‌بندی تعریف شود. اما پیاده‌سازی این نقش در سیستم نقش‌ها انجام نشده است (در Roleهای پیش‌فرض چنین مقداری وجود ندارد). پیشنهاد: یک Role با نام set\_designer اضافه شود و راهی برای انتساب آن به کاربر (مثلاً ادمین کاربران طراح را علامت‌گذاری کند). سپس property user.roles یا معادل آن باید این نقش را منعکس کند. بدون این اصلاح، طراحان ست‌بندی عملاً نمی‌توانند از API خود استفاده بهینه کنند (مثلاً مشاهده فهرست SetDesignهای اختصاص‌یافته به خود).**
* **فرآیند اختصاص خودکار درخواست به طراح: متد assign\_to\_designer در SetDesignRequest برای اختصاص دستی به یک طراح است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L127-L135" \t "_blank). اما مشخص نیست این متد از طریق API قابل دسترسی باشد. یعنی ادمین یا سیستمی که درخواست را بررسی می‌کند، چگونه یک طراح را انتخاب و تخصیص می‌دهد. در رابط کاربری هم اثری از صفحه‌ای برای مدیریت این درخواست‌ها دیده نشد. پیشنهاد: پیاده‌سازی یک View (و صفحه در پنل مدیر) برای لیست درخواست‌های ست‌بندی، و امکانی برای انتخاب یک طراح و فراخوانی assign (یا معادل آن) از طریق API. این کار می‌تواند به‌صورت یک action در ViewSet ست‌بندی درخواست‌ها یا یک endpoint مجزا صورت گیرد.**
* **APIهای ست‌بندی: برای SetDesign یک ViewSet پیاده‌سازی شده که قابلیت‌های CRUD پایه و دو action سفارشی (approve و pay) را دارد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py" \l "L14-L23" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py#L68-L76)[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py#L102-L110)**. چند نکته:  
  – شرط دسترسی در approve: تنها مشتری سفارش یا ادمین می‌تواند تأیید/رد کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py" \l "L76-L84" \t "_blank). این درست است، اما بهتر است مدیر کسب‌وکار (صاحب چاپخانه) هم مجاز باشد، که در کد با user.is\_staff پوشش داده شده است. اگر نقش دیگری برای مدیر کسب‌وکار تعریف شود (مثلاً گروه خاص)، باید اینجا لحاظ گردد.  
  – هنگام رد کردن (approved=False)، کد فعلی کامنتی دارد که "می‌توان کامنت را در مدل دیگری ذخیره کرد"**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/views.py#L94-L97) **اما فعلاً این نظر کجا نگهداری می‌شود؟ احتمالاً قرار بوده در مدل Notification یا Comment ذخیره گردد ولی پیاده نشده. این یعنی اگر مشتری دلیلی برای رد ارائه کند، در سیستم ثبت نمی‌شود. پیشنهاد: افزودن فیلد comment به مدل SetDesign یا ایجاد مدل Comment مرتبط، و ثبت دلیل رد در آن. همچنین در فرانت‌اند به‌جای prompt ساده برای دریافت دلیل رد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/components/SetDesignList.jsx" \l "L52-L60" \t "_blank)، یک دیالوگ شکیل‌تر یا ورودی در همان صفحه تعبیه شود.  
  – عدم پشتیبانی از آپلود چند فایل در ست‌بندی: ممکن است طراح نیاز داشته باشد چند فایل خروجی ارائه کند (مثلاً نسخه‌های مختلف). فعلاً مدل SetDesign تنها یک فایل و یک تصویر پیش‌نمایش دارد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/set_design/models.py" \l "L36-L44" \t "_blank). اگر چنین سناریویی محتمل است، می‌توان امکان ضمیمه کردن فایل‌های اضافی یا تصاویر بیشتر را در نظر گرفت.**
* **فرآیند پرداخت مشتری و تکمیل سفارش: در پروژه به‌نظر می‌رسد پرداخت فقط ناظر به پرداخت به طراح است، ولی پرداخت اصلی مشتری برای سفارش (مثلاً صورتحساب سفارش چاپ لباس) در سیستم دیده نمی‌شود. فیلد is\_paid در Order هست[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/orders/models.py" \l "L56-L59" \t "_blank)، اما هیچ بخشی برای انجام پرداخت مشتری (مثلاً ثبت رسید پرداخت یا اتصال به درگاه) پیاده نشده است. این یک بخش اساسی از سناریو است: احتمالاً پس از *تأیید نهایی طرح توسط مشتری* و آماده‌شدن سفارش، مشتری باید هزینه را بپردازد تا سفارش انجام شود. پیشنهاد: تکمیل گردش کار سفارش با افزودن یک مرحله پرداخت مشتری. می‌توان یک مدل Payment برای سفارشات تعریف کرد (مشابه DesignerPayment برای طراحان) یا حداقل یک API که وقتی فاکتور نهایی آماده شد، is\_paid را True کند و زمان پرداخت را ذخیره نماید. این API می‌تواند صرفاً توسط مشتری یا ادمین فراخوانی شود. در ادامه، وضعیت سفارش پس از پرداخت به "confirmed" یا "in\_progress" تغییر یابد (بسته به منطق؛ مثلاً ممکن است «pending» تا قبل از پرداخت باشد و پس از پرداخت وارد «confirmed» شود).**
* **عدم وجود API/صفحه لیست سفارش‌ها: در فرانت‌اند لینک "مدیریت سفارش‌ها" در داشبورد وجود دارد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/Dashboard.jsx" \l "L14-L17" \t "_blank) که به مسیر /orders اشاره می‌کند، اما در کد فعلی هیچ View یا صفحه لیستی برای Order پیاده نشده است (نه بک‌اند و نه فرانت‌اند). این یک خلاء در سناریوست: مدیر چاپخانه باید بتواند سفارش‌های ثبت‌شده را ببیند و وضعیتشان را تغییر دهد. همچنین هر کاربر (مشتری) باید سفارش‌های خود را پیگیری کند. پیشنهاد: ایجاد ViewSet مربوط به Order (با فیلتر بر اساس کاربر: مشتری فقط سفارش‌های خودش، کسب‌وکار فقط سفارش‌های مربوط به خودش، ادمین همه). همین‌طور در رابط کاربری صفحه‌ای برای نمایش لیست سفارش‌ها (با امکان فیلتر بر اساس وضعیت) اضافه گردد. جزئیات هر سفارش (احتمالاً صفحه DetailOrder.jsx موجود است) شامل آیتم‌ها، وضعیت فعلی و دکمه‌هایی برای اقدامات ممکن (مثلاً تأیید طرح‌ها، علامت‌زدن سفارش به عنوان تکمیل‌شده و غیره) خواهد بود.**
* **بهبود تجربه کاربری ست‌بندی: در کامپوننت SetDesignList (صفحه ست‌بندی‌ها)، عملیات مهمی مانند Approve/Reject/Pay با alert و prompt خام انجام شده است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/components/SetDesignList.jsx" \l "L39-L48" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/components/SetDesignList.jsx#L52-L60) **که از نظر تجربه کاربری ضعیف است. پیشنهاد: استفاده از کامپوننت‌های دیالوگ (Modal) برای گرفتن تأیید یا ورودی دلیل رد، و نوتیفیکیشن‌های زیباتر (مثلاً Snackbar یا Toast) به‌جای alert. همچنین پس از انجام هر عملیات، به‌جای استفاده از alert برای پیام موفقیت، می‌توان از Toast (همان‌طور که در سایر بخش‌ها استفاده شده) بهره برد تا رابط کاربری یکدست باشد.**

**بخش فرانت‌اند (Frontend)**

**۱. احراز هویت و دسترسی**

**وضعیت فعلی: اپلیکیشن React با استفاده از React Router v6 ساختار یافته است. صفحات Login و Register (به زبان فارسی) آماده شده‌اند و در مسیرهای /auth/login و /auth/register ارائه می‌شوند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/routes.js" \l "L24-L32" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/routes.js#L26-L29)**. طراحی رابط کاربری فرم‌ها با Material-UI (MUI) انجام شده است (برای مثال استفاده از <TextField> و ... در Register.jsx)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/Register.jsx#L2-L10)[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/Register.jsx#L84-L92)**. با این حال، پیاده‌سازی عملکردی این فرم‌ها کامل نیست:**

* **صفحه Register.jsx: اعتبارسنجی سمت کاربر برای خالی نبودن فیلدها و تطبیق رمز عبور دارد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/Register.jsx" \l "L28-L37" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/Register.jsx#L48-L56)**. اما در submit، به جای فراخوانی API واقعی ثبت‌نام، فقط با setTimeout یک حالت موفقیت شبیه‌سازی شده و کاربر به صفحه لاگین هدایت می‌شود[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/Register.jsx" \l "L66-L74" \t "_blank). یعنی ثبت‌نام اصلاً به سرور ارسال نمی‌شود! این احتمالاً برای تست اولیه UI بوده و فراموش شده به فراخوانی واقعی API (/api/auth/register/) تبدیل شود. در مقابل، یک کامپوننت به نام AuthWidget.jsx وجود دارد که ورود و ثبت‌نام را واقعاً به سرور متصل کرده است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/components/AuthWidget.jsx" \l "L26-L34" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/components/AuthWidget.jsx#L36-L44) **و توکن‌ها را در localStorage ذخیره می‌کند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/components/AuthWidget.jsx" \l "L40-L48" \t "_blank)، اما ظاهراً این کامپوننت در ساختار فعلی استفاده نمی‌شود (چرا که مسیرهای جداگانه برای login/register تعریف شده‌اند).**
* **صفحه Login.jsx: فایل مشخصی به نام Login.jsx در کد وجود ندارد یا در مخزن نیامده، اما احتمال می‌رود مشابه Register.jsx طراحی شده باشد. اگر به همان روال Register عمل کند، ممکن است آن هم بدون فراخوانی API باشد. عدم وجود لاجیک ورود می‌تواند مانع ورود واقعی کاربران شود.**

**کاستی‌ها و پیشنهادها:**

* **اتصال فرم‌های Auth به سرور: لازم است در کامپوننت‌های Login و Register به‌جای شبیه‌سازی، از Axios برای ارسال درخواست به /api/auth/register/ و /api/auth/login/ استفاده شود. کد AuthWidget این کار را انجام می‌دهد و می‌توان منطق آن را به دو صفحه تفکیک نمود. به عنوان مثال، در Register.jsx در تابع handleSubmit به‌جای delay دستی[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/Register.jsx" \l "L66-L74" \t "_blank)، چنین عملی انجام شود:**

**jsx**

**CopyEdit**

**const response = await axios.post('/api/auth/register/', formData);**

**localStorage.setItem('access\_token', response.data.access);**

**localStorage.setItem('refresh\_token', response.data.refresh);**

**// ... هدایت به داشبورد یا صفحه‌ی مناسب**

**با این تغییر، سناریوی ثبت‌نام end-to-end کامل خواهد شد. برای Login.jsx نیز مشابه همین باید پیاده گردد.**

* **یکپارچه‌سازی کامپوننت AuthWidget: در حال حاضر دو روش متفاوت برای احراز هویت وجود دارد (AuthWidget در مقابل صفحات مجزا). بهتر است یکی انتخاب و استفاده شود. از آنجا که طراحی MUI صفحات login/register بهتر با باقی پروژه همخوانی دارد، می‌توان AuthWidget را کنار گذاشت یا فقط از لاجیک داخلی آن بهره گرفت. اگر AuthWidget دیگر استفاده نمی‌شود، پاک‌سازی آن از پروژه نیز موجب کاهش ابهام می‌شود.**
* **نگهداری وضعیت احراز هویت: بررسی فعلی برای اینکه آیا کاربر لاگین است، با چک کردن presence توکن در localStorage انجام شده[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/routes.js" \l "L17-L25" \t "_blank). این روش در ابتدای لود برنامه کافی است اما اگر توکن منقضی شود یا کاربر logout کند، نیاز به مدیریت به‌روز دارد. مکانیزم logout فعلاً فقط با پاک کردن توکن (مثلاً در AuthWidget بعد از ثبت‌نام، یک بار window.location.reload شده) انجام می‌گیرد. پیشنهاد: پیاده‌سازی کانتکست Auth در React برای نگهداری وضعیت کاربر لاگین‌شده و اطلاعات پروفایل او. این کار کمک می‌کند در اجزای مختلف (ناوبری، صفحات) دسترسی‌ها مدیریت شود. همچنین می‌توان Interceptor برای Axios تنظیم کرد تا خودکار Authorization header را با توکن از این Context یا از localStorage ست کند – این از تکرار کد header در هر فراخوانی Axios جلوگیری می‌کند.**
* **مدیریت دسترسی صفحات: در Routeهای خصوصی فعلی، اگر کاربر لاگین نباشد به /auth/login ریدایرکت می‌شود[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/routes.js" \l "L32-L40" \t "_blank). این خوب است، اما برای نقش‌های مختلف (ادمین vs کاربر عادی) نیز باید در نظر گرفته شود. مثلاً صفحه مدیریت کاربران فقط برای ادمین قابل مشاهده باشد. در کد ListUsers.jsx همین کار انجام شده و اگر prop isAdmin false باشد، محتوای "دسترسی محدود" نشان می‌دهد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/auth/ListUsers.jsx" \l "L56-L65" \t "_blank). اما prop isAdmin باید از والد (احتمالاً DefaultLayout یا Dashbaord) به آن برسد. هنوز مشخص نیست این اتصال چگونه انجام می‌شود. پیشنهاد: تعیین ساختاری برای تعیین نقش جاری کاربر پس از ورود و گذاردن آن در Context یا State عمومی. سپس PrivateRouteهای اختصاصی بر اساس نقش‌ها تعریف شود (مثلاً <AdminRoute> که داخلش <Outlet> فقط اگر admin باشد). این موضوع به آینده و توسعه نقش‌ها مرتبط است، ولی پیش‌بینی آن از الان مفید خواهد بود.**

**۲. ناوبری و مسیرها**

**وضعیت فعلی: فایل routes.js تمام Routeهای اپ را تعیین کرده است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/routes.js" \l "L24-L32" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/routes.js#L34-L42)**. یک Layout پیش‌فرض (DefaultLayout) وجود دارد که برای همه صفحات داخلی پس از لاگین استفاده می‌شود و یک AuthLayout برای صفحات ورود/ثبت‌نام. در Dashbaord تعدادی لینک سریع به بخش‌های مختلف آمده است: «مدیریت ست‌بندی‌ها»، «سفارش‌ها» و «کاربران»**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/Dashboard.jsx#L8-L16)**. این لینک‌ها به مسیرهای /set-design, /orders, /users اشاره می‌کنند.**

**مشکلات مسیرها:**

* **عدم تعریف برخی Routeها در کد: در routes.js فقط مسیر /set-design داخل DefaultLayout تعریف شده است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/routes.js" \l "L42-L46" \t "_blank)، اما برای /orders و /users مسیری دیده نمی‌شود[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/routes.js" \l "L34-L43" \t "_blank). این یعنی کلیک روی لینک‌های «سفارش‌ها» یا «کاربران» منجر به صفحه ۴۰۴ خواهد شد. Page یا Component مربوط به لیست سفارش‌ها احتمالاً هنوز ساخته نشده؛ برای کاربران Page ListUsers.jsx وجود دارد ولی Routeاش اضافه نشده است. پیشنهاد: تکمیل پیکربندی روتر: افزودن <Route path="orders" element={<OrdersPage />} /> و <Route path="users" element={<ListUsers />} /> در داخل DefaultLayout. البته باید ابتدا OrdersPage (لیست سفارش‌ها) پیاده‌شود. اگر فعلاً قصد نمایش سفارش‌ها نیست، حداقل لینک Dashbaord غیرفعال یا پنهان گردد تا کاربر به صفحه ناموجود نفرستاده شود.**
* **استفاده از useParams در Detail‌ها: کامپوننت‌هایی نظیر DetailBusiness از useParams برای گرفتن id استفاده می‌کنند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/businesses/DetailBusiness.js" \l "L6-L14" \t "_blank). این زمانی کار می‌کند که Route مربوطه مثلاً /businesses/:id باشد. در حال حاضر، مشخص نیست آیا چنین روتی تعریف شده (در routes.js اشاره‌ای به /businesses/:id نیست). احتمالاً فراموش شده Route‌های مربوط به کسب‌وکارها هم اضافه شود. پیشنهاد: تعریف Routeهای بخش کسب‌وکار: مثلاً <Route path="businesses" ...> که شامل یک Outlet برای لیست کسب‌وکارها و DetailBusiness به صورت فرزند با path :id باشد.**
* **یکنواختی URLها: اکنون APIها عموماً به شکل جمع انگلیسی (designs, businesses, ...) هستند و فرانت‌اند نیز از همین استفاده می‌کند. از نظر کاربر نهایی (رابط وب) بهتر است مسیرها معنی‌دار و شاید فارسی‌سازی‌شده باشند (اگر برنامه صرفاً برای کاربر فارسی است). مثلاً /designs می‌تواند به "/طرح‌ها" نگاشت شود. البته این موضوع اختیاری است و نیازمند پشتیبانی کتابخانه Router از یونیکد در URL. اگر نخواهیم URL را فارسی کنیم، حداقل می‌توان معادل فارسی در رابط نشان داد (مثلاً در Breadcrumbها نام بخش به فارسی بیاید). در SetDesignPage این کار انجام شده و در Breadcrumb کلمه «ست‌بندی‌ها» آمده است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/SetDesignPage.jsx" \l "L12-L20" \t "_blank) که خوب است.**

**۳. مدیریت داده‌ها و ارتباط با API**

**وضعیت فعلی: بیشتر ارتباط‌های فرانت‌اند با سرور از طریق Axios انجام می‌شود. در برخی موارد header احراز هویت (توکن) به صراحت در هر درخواست درج شده است (مثلاً در CreateDesign.jsx)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/designs/CreateDesign.jsx#L32-L40)**، گاهی فراموش شده (مثلاً EditDesign.jsx درخواست‌ها را بدون Authorization می‌فرستد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/designs/EditDesign.jsx" \l "L50-L58" \t "_blank)). همچنین، خط‌مشی کلی این است که داده‌ها در mount هر صفحه لود شوند (با useEffect) و در state محلی نگهداری شوند. استفاده از toast برای اعلان خطا/موفقیت رایج است که تجربه نسبتاً خوبی ایجاد می‌کند.**

**کاستی‌ها:**

* **عدم ارسال توکن در برخی درخواست‌ها: همان‌طور که ذکر شد، برخی صفحات مانند EditDesign یا DetailDesign، درخواست‌های Axios را بدون Header می‌زنند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/designs/EditDesign.jsx" \l "L48-L56" \t "_blank)، که در صورت حفاظت‌شده بودن API حتماً با ۴۰۱ مواجه می‌شوند. اگر در تست‌ها متوجه این خطا نشده‌اید، ممکن است به این دلیل باشد که Axios به صورت global interceptors تنظیم شده (اما چنین تنظیمی در کد مشاهده نشد). پیشنهاد: اطمینان از ارسال توکن در همه درخواست‌های نیازمند. ایده‌آل آن است که یک نمونه Axios با default header Authorization ساخته شود پس از لاگین، تا نیاز به تکرار کد در هر صفحه نباشد. می‌توان در بالاترین سطح (مثلاً در index.js یا App.jsx) کاری کرد که Axios توکن را از localStorage بخواند و تنظیم کند.**
* **مدیریت خطاهای API: در حال حاضر، اکثر درخواست‌ها try/catch دارند و در صورت error یک toast.error نمایش می‌دهند (با یک پیام کلی). بهتر است پیام خطا دقیق‌تر باشد؛ مثلاً اگر API ولیدیشن خطا داد، متن آن (مثلاً "نام کاربری تکراری است") به کاربر نشان داده شود. اکنون در برخی جاها این کار شده (مثلاً EditDesign در catch تلاش می‌کند error.response.data.detail را نشان دهد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/designs/EditDesign.jsx" \l "L196-L200" \t "_blank))، ولی در بسیاری موارد پیام‌ها ثابت هستند ("خطا در حذف کاربر", "خطا در بارگذاری..."). پیشنهاد: بهبود پیام‌های خطا با استفاده از جزئیات برگردانده‌شده از سرور. همچنین می‌توان یک handling مرکزی برای خطاهای توکن منقضی یا عدم دسترسی داشت که مثلاً اگر status 401 برگشت، کاربر را به صفحه لاگین هدایت کند (با پیغام "دوباره وارد شوید").**
* **عدم استفاده از State Management پیشرفته: برای پروژه در این اندازه، استفاده از Redux یا Context ضروری به‌نظر نمی‌رسد، اما اکنون برخی اطلاعات بین کامپوننت‌ها باید رد و بدل شوند (مثل اینکه user ادمین است یا نه، یا لیست دسته‌ها/برچسب‌ها که در چند صفحه نیاز است). فعلاً هر صفحه جداگانه API می‌زند تا همان داده را بگیرد (مثلاً لیست دسته‌ها در CreateDesign و EditDesign هر دو فراخوانی می‌شود). پیشنهاد: استفاده از Context برای اطلاعات عمومی (لیست دسته‌ها و برچسب‌ها که کمتر تغییر می‌کنند می‌تواند یک بار لود شده و در Context قرار گیرد). یا حداقل استفاده مجدد از کامپوننت‌ها: می‌توان یک کامپوننت selector برچسب/دسته ساخت که خودش داده را cache کند.**
* **مشکلات جزئی رابط:  
  – در ListUsers.jsx پس از حذف کاربر، لیست با فیلتر کردن state به‌روز می‌شود[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/auth/ListUsers.jsx" \l "L40-L48" \t "_blank). این خوب است اما اگر نیاز به به‌روزرسانی از سرور باشد (مثلاً شمارش کل کاربران)، آن انجام نمی‌شود. شاید بهتر باشد بعد از delete، مجدداً fetchUsers فراخوانی شود تا از صحت داده اطمینان حاصل شود.  
  – در DetailBusiness.jsx تب‌های general/users/activities در state activeTab نگهداری می‌شوند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/businesses/DetailBusiness.js" \l "L10-L18" \t "_blank) ولی پیاده‌سازی toggle بین آنها در کد مشاهده نشده (احتمالاً بخش پایینی کامپوننت است). باید بررسی شود که مثلاً کلیک روی "کاربران" لیست users را نشان می‌دهد. اگر این بخش ناتمام است، تکمیل گردد. همچنین map کردن activityType به متن فارسی محدود به چند مقدار است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/businesses/DetailBusiness.js" \l "L64-L72" \t "_blank) و همه انواع ممکن (مثلاً design\_purchase یا order\_created که در مدل وجود داشت[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/backend/apps/business/models.py" \l "L69-L72" \t "_blank)) پوشش داده نشده؛ در نتیجه ممکن است برخی فعالیت‌ها ترجمه نشوند.  
  – ترکیب دو رویکرد استایل‌دهی (MUI vs Tailwind): برخی صفحات کاملاً با MUI ساخته شده (مثل Dashboard.jsx) در حالی که برخی دیگر از کلاس‌های Tailwind استفاده می‌کنند (مثل ListUsers.jsx با کلاس‌هایی چون container mx-auto و ...**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/auth/ListUsers.jsx#L80-L88)**). این عدم‌یکدستی روی ظاهر اثر منفی می‌گذارد (مثلاً دکمه‌ها و فونت‌ها ممکن است در دو صفحه متفاوت باشند). پیشنهاد: تا حد امکان یک سیستم را انتخاب کنید. با توجه به فارسی‌سازی و RTL، به نظر می‌رسد MUI انتخاب مناسب‌تری بوده (وجود ThemeProvider با راست‌چین در App.jsx**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/App.jsx#L15-L24)**)، پس بهتر است کامپوننت‌هایی که هنوز با Tailwind خام هستند به تدریج به معادل MUI تبدیل شوند. این کار البته زمان‌بر است، ولی حتی در کوتاه‌مدت می‌توان با تنظیمات CSS سفارشی، نزدیکی ظاهری ایجاد کرد (مثلاً تعریف کلاس‌هایی برای دکمه‌ها تا شبیه دکمه MUI شوند).**

**۴. بهبودهای تجربه کاربری (UX)**

**مشاهدات کلی: رابط کاربری پروژه نسبتاً مدرن و تمیز طراحی شده، به‌ویژه با استفاده از Material-UI برای المان‌هایی چون کارت‌های داشبورد، فرم‌های ورود و ... . همچنین پیام‌های وضعیت به زبان فارسی و مناسب هستند. با این حال، برای بهبود تجربه کاربری نکات زیر قابل توجه است:**

* **راهنمای تعاملی (Interactive Guide): وجود یک راهنمای مرحله‌به‌مرحله برای آشنایی کاربر جدید با بخش‌های سیستم در تست‌ها دیده شد[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/components/main/InteractiveGuide.test.js" \l "L67-L75" \t "_blank). اگر این feature کامل شده و در Dashbaord نمایان می‌شود (مثلاً با hasSeenGuide=false)، یک نقطه قوت است. در غیر این صورت، فعال‌سازی آن ایده خوبی است تا کاربران اولین بار بدانند از کجا باید شروع کنند. همچنین، پس از پایان راهنما، درخواست ثبت‌شدن وضعیت دیده‌شدن به سرور ارسال می‌شود[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/components/main/InteractiveGuide.test.js" \l "L96-L97" \t "_blank) (احتمالاً به یک endpoint مثلا /api/auth/guide-seen/)، که باید مطمئن شد چنین چیزی در بک‌اند پیاده شده یا حداقل قابل صرف‌نظر است.**
* **واکنش‌گرایی (Responsive بودن): با توجه به بهره‌گیری از Gridهای Tailwind/MUI به نظر می‌رسد طراحی Responsive مدنظر بوده است (کلاس‌های md:grid-cols-2 و ... در کد هست[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/designs/EditDesign.jsx" \l "L224-L232" \t "_blank)). بهتر است یک بازبینی سریع در حالت موبایل انجام شود تا مطمئن شویم عناصر درست چیده می‌شوند (به عنوان مثال، منوهای بزرگ یا جدول کاربران ممکن است در موبایل نیاز به اسکرول افقی داشته باشد).**
* **نمایش وضعیت بارگذاری: در بیشتر صفحات، هنگام بارگیری داده از سرور یک نشانگر لودینگ نشان داده می‌شود (Spinner یا اسکلت) که بسیار خوب است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/auth/ListUsers.jsx" \l "L70-L78" \t "_blank). اطمینان حاصل شود همه جا این حالت پوشش داده شده؛ مثلاً در فرم‌های ورود/ثبت‌نام هنگام submit دکمه غیرفعال شده و indicator نمایش یابد (متغیر loading در Register.jsx هست ولی از آن برای مثلاً نشان دادن یک CircularProgress در دکمه استفاده نشده). این موارد کوچک polish تجربه را بهتر می‌کند.**
* **پیام‌های تأیید عملیات حساس: برای حذف‌ها از window.confirm استفاده شده که عملی است[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/auth/ListUsers.jsx" \l "L36-L44" \t "_blank). می‌توان برای زیبایی، از Dialogهای MUI استفاده کرد تا با بقیه ظاهر سازگار باشد. این کار البته اختیاری است ولی یکپارچگی طراحی را افزایش می‌دهد.**
* **استفاده از آیکن‌ها و تصاویر: در Dashboard برخی آیکن‌های متریال استفاده شده‌اند (👥 برای کاربران، 🔑 برای ورود و ...) که به صورت ایموجی یا آیکن SVG هستند[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/auth/ListUsers.jsx" \l "L80-L88" \t "_blank)**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/components/AuthWidget.jsx#L70-L78)**. ترجیحاً از آیکن‌های یک‌دست (همه از کتابخانه MUI یا همه ایموجی) استفاده شود. مثلاً در داشبورد کنار "کاربران" یک آیکن PeopleIcon از MUI هست[github.com](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/Dashboard.jsx" \l "L2-L6" \t "_blank) اما در لیست کاربران از ایموجی "👥" استفاده شده**[**github.com**](https://github.com/amin311/pardach/blob/b7148cb9beedb500cf0e2dbf4a52a853d7f47084/src/pages/auth/ListUsers.jsx#L80-L88)**.統 یکسان‌سازی این موارد جزیی، ظاهر حرفه‌ای‌تری می‌دهد.**

**در مجموع، تجربه کاربری با چند تغییر کوچک می‌تواند بسیار روان‌تر شود:**

* **رفع اشکالات ارتباط با سرور (تا کاربر با خطای ناخواسته مواجه نشود)،**
* **تکمیل مسیرهای ناوبری (تا لینک‌های مرده وجود نداشته باشد)،**
* **و بهبود هماهنگی ظاهر عناصر (برای حس انسجام).**

**با اعمال پیشنهادهای مطرح‌شده، سیستم مدیریت طرح‌ها و سفارش‌ها به شکل قابل ملاحظه‌ای قابل اعتمادتر، کاربرپسندتر و منطبق بر سناریوهای مورد انتظار خواهد شد. هر بخش که در این گزارش «نیاز به توسعه» ذکر شد، باید در برنامه آتی پروژه قرار گیرد تا پلتفرم به اهداف تعریف‌شده خود (چه برای کاربران عادی، چه صاحبان کسب‌وکار و چه طراحان) برسد.**

**جمع‌بندی: در حالت فعلی، زیرساخت اصلی بک‌اند (مدل‌های داده و APIهای اساسی) و اسکلت رابط کاربری فراهم است، اما برای تطابق کامل با سناریوهای تعریف‌شده، هنوز کار قابل توجهی باقی است. رفع باگ‌های مذکور در کنار پیاده‌سازی فیچرهای تکمیلی (خصوصاً در زمینه نقش‌ها، سفارش‌ها و تعاملات مرتبط با طراحی) گام بعدی پروژه خواهد بود. امیدوارم این بررسی عمیق، نقشه راه روشنی برای بهبود و توسعه سیستم پیش‌رویتان بگذارد. موفق باشید!**