عالی! با توجه به ارائه کامل مدل‌ها، فرم‌ها، توابع محاسباتی و کلاس پایه دسترسی، می‌توانیم یک **سند تحلیل سیستمی و مشخصات فنی جامع** برای تیم توسعه تهیه کنیم. این سند به عنوان نقشه راه برای پیاده‌سازی یا تکمیل سیستم عمل خواهد کرد.

**سند تحلیل سیستمی و مشخصات فنی: سیستم جامع بودجه‌ریزی و گردش کار مالی**

**۱. مقدمه**

* **هدف:** این سند، مشخصات فنی و الزامات لازم برای توسعه و تکمیل سیستم یکپارچه مدیریت بودجه، تخصیص منابع، ردیابی هزینه‌ها (از طریق تنخواه) و گردش کار تایید/رد مرتبط را برای تیم توسعه تشریح می‌کند.
* **محدوده:** سیستم شامل تعریف دوره‌های بودجه کلان، تخصیص به واحدها (سازمان/پروژه/مرکز هزینه)، ثبت و پردازش تنخواه و فاکتورهای مرتبط، مدیریت برگشت بودجه و پیاده‌سازی گردش کار چندمرحله‌ای تایید/رد بر اساس ساختار سازمانی و پست‌ها می‌باشد.
* **مخاطبین:** تیم توسعه نرم‌افزار (برنامه‌نویسان بک‌اند و فرانت‌اند)، مدیران پروژه، تحلیلگران سیستم.

**۲. معماری کلی سیستم**

سیستم مبتنی بر فریمورک جنگو بوده و از ماژول‌های اصلی زیر تشکیل شده است:

* **core:** مدیریت موجودیت‌های پایه شامل ساختار سازمانی (Organization, OrganizationType), پروژه‌ها (Project, SubProject), پست‌های سازمانی (Post, UserPost) و تعاریف گردش کار (WorkflowStage, PostAction, StageApprover).

**budgets:** مدیریت کامل چرخه بودجه شامل دوره‌ها (BudgetPeriod), تخصیص‌ها (BudgetAllocation, ProjectBudgetAllocation), ردیف‌ها (BudgetItem), تراکنش‌ها (BudgetTransaction), برگشت‌ها، تاریخچه (BudgetHistory), تنظیمات (SystemSettings, BudgetSettings) و مراکز هزینه (BudgetItem

* ).
* **tankhah:** مدیریت فرآیند تنخواه شامل درخواست‌ها (Tankhah), فاکتورها (Factor, FactorItem), اسناد پیوست (TankhahDocument, FactorDocument), مدیریت پرداخت (PaymentOrder, Payee) و لاگ تاییدات (ApprovalLog).
* **accounts:** مدیریت کاربران (CustomUser) و نقش‌ها/گروه‌ها.
* **notifications:** (فرض وجود) ارسال اعلان‌های سیستمی به کاربران.
* **utils:** شامل توابع کمکی عمومی (مانند کار با تاریخ شمسی، فرمت اعداد).
* **budget\_utils (budget\_calculations.py):** شامل توابع تخصصی محاسبات بودجه.

**۳. تحلیل و مشخصات فنی مدل‌های داده**

*(در این بخش، مدل‌های کلیدی و الزامات پیاده‌سازی آن‌ها بررسی می‌شود)*

**الف) ماژول budgets**

1. **BudgetPeriod (دوره بودجه کلان):**
   * **هدف:** تعریف بودجه اصلی در سطح سازمان مادر (دفتر مرکزی).
   * **فیلدهای کلیدی:** organization (لینک به core.Organization با is\_core=True), name, start\_date, end\_date, total\_amount, total\_allocated, returned\_amount, is\_active, is\_completed, locked\_percentage, warning\_threshold, lock\_condition, warning\_action.
   * **منطق حیاتی:**
     + clean(): اعتبارسنجی‌های لازم (تاریخ، درصدها، وضعیت، نام منحصربفرد در سازمان) پیاده‌سازی شده است. ✅
     + save(): فراخوانی check\_budget\_status و send\_notification در زمان ذخیره. ✅
     + get\_remaining\_amount(): **نیاز به اصلاح فوری دارد.** باید returned\_amount را لحاظ کند. فرمول صحیح: max(self.total\_amount - self.total\_allocated + self.returned\_amount, Decimal('0')).
     + total\_allocated: این فیلد باید توسط یک سیگنال (post\_save/post\_delete روی BudgetAllocation) یا مکانیزم مشابه به‌روز شود تا همیشه مجموع allocated\_amount تخصیص‌های *فعلی* را نشان دهد (با در نظر گرفتن کاهش ناشی از برگشت).
     + returned\_amount: این فیلد توسط متد save در BudgetTransaction (برای نوع RETURN) به درستی به‌روز می‌شود. ✅
   * **الزامات:**
     + اصلاح فوری متد get\_remaining\_amount().
     + پیاده‌سازی مکانیزم (ترجیحاً سیگنال) برای به‌روزرسانی دقیق total\_allocated.
2. **BudgetAllocation (تخصیص بودجه):**
   * **هدف:** تخصیص بودجه از BudgetPeriod به یک سازمان فرعی (شعبه) برای یک ردیف بودجه (BudgetItem) و به صورت اختیاری برای یک پروژه (Project).
   * **فیلدهای کلیدی:** budget\_period, organization (باید is\_core=False باشد), budget\_item, project (null=True), allocated\_amount, remaining\_amount, returned\_amount, allocation\_date, is\_active, is\_stopped, allocation\_type, locked\_percentage, warning\_threshold, warning\_action.
   * **منطق حیاتی:**
     + clean(): اعتبارسنجی‌های لازم (مبلغ مثبت، تاریخ در بازه، درصدها، عدم تخصیص به is\_core=True) پیاده‌سازی شده است. ✅ اعتبارسنجی کافی بودن بودجه در BudgetPeriod نیز باید در clean فرم یا ویو انجام شود.
     + save(): محاسبه allocated\_amount برای نوع percent و تنظیم اولیه remaining\_amount انجام شده است. ✅
     + remaining\_amount: این فیلد توسط BudgetTransaction.save برای تراکنش‌های CONSUMPTION, ADJUSTMENT\_INCREASE, ADJUSTMENT\_DECREASE, RETURN به‌روز می‌شود. ✅
     + allocated\_amount: **مهم:** این فیلد با تراکنش RETURN کاهش می‌یابد تا مجموع تخصیص‌های فعال در BudgetPeriod درست محاسبه شود. ✅
     + returned\_amount: توسط تراکنش RETURN به‌روز می‌شود. ✅
     + get\_remaining\_amount(): از فیلد remaining\_amount استفاده می‌کند. ✅
     + check\_allocation\_status(): باید به budget\_utils.py منتقل شود.
   * **الزامات:**
     + انتقال check\_allocation\_status به budget\_utils.py.
     + اعتبارسنجی بودجه کافی در BudgetPeriod باید قبل از ذخیره در ویو یا فرم انجام شود.
3. **BudgetItem (ردیف بودجه):**
   * **هدف:** دسته‌بندی تخصیص‌ها در یک دوره و سازمان خاص.
   * **فیلدهای کلیدی:** budget\_period, organization, name, code, is\_active.
   * **منطق حیاتی:** unique\_together روی budget\_period, organization, code مهم است. ✅ clean برای نام اجباری وجود دارد. ✅
   * **الزامات:** مدل به نظر کامل می‌رسد.
4. **BudgetTransaction (تراکنش بودجه):**
   * **هدف:** ثبت تمام تغییرات مالی روی یک BudgetAllocation.
   * **فیلدهای کلیدی:** allocation, transaction\_type, amount, related\_tankhah, timestamp, created\_by, transaction\_id.
   * **منطق حیاتی:**
     + save(): **مهم‌ترین بخش منطق سیستم.**
       - تولید transaction\_id یکتا. ✅
       - اعتبارسنجی CONSUMPTION در مقابل allocation.remaining\_amount. ✅
       - اعتبارسنجی RETURN در مقابل allocation.allocated\_amount و budget\_period.is\_active. ✅
       - **به‌روزرسانی صحیح BudgetAllocation و BudgetPeriod برای نوع RETURN** (کاهش/افزایش allocated\_amount, remaining\_amount, returned\_amount, total\_allocated). ✅✅
       - به‌روزرسانی allocation.remaining\_amount برای سایر انواع تراکنش. ✅
       - ثبت خودکار BudgetHistory برای نوع RETURN. ✅
   * **الزامات:**
     + ایجاد تراکنش‌ها باید توسط سیستم و در نقاط مناسب (ذخیره تخصیص، تایید تنخواه، برگشت بودجه، تعدیل) انجام شود. دسترسی مستقیم کاربر برای ایجاد همه انواع تراکنش توصیه نمی‌شود.
     + تابع validate\_return که اضافه شده بود، منطق مشابهی با clean یا بخش RETURN در save دارد و می‌تواند ادغام یا حذف شود تا از تکرار جلوگیری گردد. منطق اصلی اعتبارسنجی و به‌روزرسانی باید در save متمرکز باشد.
5. **ProjectBudgetAllocation (تخصیص بودجه پروژه):**
   * **هدف:** اتصال بودجه تخصیص یافته به شعبه (BudgetAllocation) به یک پروژه یا زیرپروژه خاص.
   * **فیلدهای کلیدی:** budget\_allocation, project, subproject (null=True), allocated\_amount, allocation\_date, created\_by, is\_active.
   * **منطق حیاتی:**
     + clean(): اعتبارسنجی‌ها درست هستند (مبلغ مثبت، تعلق زیرپروژه به پروژه، تاریخ در بازه). **اما** اعتبارسنجی allocated\_amount > remaining **باید بازنگری شود**. موجودی budget\_allocation.remaining\_amount باید در نظر گرفته شود.
     + save(): **باید اصلاح شود.** این مدل نباید remaining\_amount خودش یا والدش (budget\_allocation) را مستقیماً آپدیت کند. نقش آن فقط ثبت *ارتباط* و *مبلغ* تخصیص یافته به پروژه از محل BudgetAllocation است.
     + get\_remaining\_amount(): این متد (و property remaining\_amount) باید حذف شوند یا به توابع budget\_utils که مصرف واقعی (BudgetTransaction) را محاسبه می‌کنند، ارجاع دهند.
   * **الزامات:**
     + حذف کامل منطق به‌روزرسانی remaining\_amount از متد save.
     + حذف فیلد remaining\_amount (property) یا اصلاح آن برای فراخوانی تابع محاسبه دقیق از budget\_utils.
     + اعتبارسنجی دقیق مبلغ در clean بر اساس budget\_allocation.remaining\_amount والد.
6. **CostCenter (مرکز هزینه):**
   * **هدف:** مدیریت هزینه‌های متعارف غیرپروژه‌ای.
   * **فیلدهای کلیدی:** name, code, organization, budget\_allocation, allocated\_budget, is\_active.
   * **منطق حیاتی:**
     + get\_remaining\_budget(): مصرف را از Tankhah های مرتبط با cost\_center محاسبه می‌کند. **مهم:** این فرض می‌کند تنخواه مستقیماً به مرکز هزینه لینک می‌شود و مصرف بودجه BudgetAllocation آن مرکز هزینه را کم می‌کند (که نیاز به پیاده‌سازی دارد).
   * **الزامات:**
     + افزودن فیلد cost\_center به مدل Tankhah.
     + اطمینان از اینکه ایجاد BudgetTransaction برای تنخواه‌های مرتبط با CostCenter به درستی BudgetAllocation لینک شده به آن CostCenter را هدف قرار می‌دهد.
7. **BudgetHistory, BudgetReallocation, BudgetSettings, SystemSettings:**
   * این مدل‌ها ساختار مناسبی دارند و به ردیابی، انتقال و تنظیمات کمک می‌کنند.
   * **الزامات:**
     + منطق پر کردن BudgetHistory برای تمام اقدامات مرتبط (Create, Update, Stop, Reallocate, Return) باید پیاده‌سازی شود (ترجیحاً با Signals).
     + ویوها و فرم‌های لازم برای مدیریت BudgetReallocation و BudgetSettings باید ایجاد شوند.

**ب) ماژول tankhah**

1. **Tankhah (تنخواه):**
   * **هدف:** مدیریت درخواست تنخواه، وضعیت، گردش کار و ارتباط با بودجه.
   * **فیلدهای کلیدی:** number, amount, date, organization, project, subproject, cost\_center (باید اضافه شود), status, current\_stage, budget\_allocation.
   * **منطق حیاتی:**
     + generate\_number(): تولید شماره یکتا. ✅
     + save():
       - تولید شماره و قفل کردن در وضعیت نهایی. ✅
       - **بلوک if not self.pk: (ایجاد اولیه):**
         * پیدا کردن ProjectBudgetAllocation صحیح بر اساس پروژه/زیرپروژه **و self.budget\_allocation**. (نیاز به دقت در کوئری).
         * محاسبه بودجه باقیمانده (budget) باید از توابع get\_project\_remaining\_budget یا get\_subproject\_remaining\_budget در budget\_utils.py انجام شود که بر اساس BudgetTransaction ها محاسبه می‌کنند. **محاسبه فعلی بر اساس allocation.remaining\_amount (که فیلد ProjectBudgetAllocation است) اشتباه است.**
         * **حذف کامل** خطوط allocation.remaining\_amount -= self.amount و allocation.save().
         * ایجاد BudgetTransaction از نوع CONSUMPTION لینک شده به self.budget\_allocation (تخصیص اصلی شعبه). ✅ (این بخش صحیح است و باید باقی بماند).
   * **الزامات:**
     + افزودن فیلد cost\_center.
     + اصلاح کامل منطق save() در بلوک if not self.pk: برای استفاده از توابع صحیح محاسبه بودجه و حذف آپدیت اشتباه ProjectBudgetAllocation.
     + حذف فیلد remaining\_budget از مدل Tankhah.
     + حذف فیلد approved\_by.
2. **Factor, FactorItem:**
   * **هدف:** ثبت جزئیات هزینه‌های تنخواه.
   * **منطق حیاتی:**
     + Factor.save(): محاسبه total\_amount از FactorItem ها. ✅ اعتبارسنجی total\_amount() > self.budget **باید حذف شود** (فیلد budget و remaining\_budget در Factor باید حذف شوند).
     + FactorItem.save(): محاسبه amount از quantity \* unit\_price. ✅ اعتبارسنجی min\_stage\_order. ✅ **حذف کامل** منطق کاهش بودجه (self.factor.remaining\_budget -= ..., self.factor.tankhah.remaining\_budget -= ...) و ذخیره factor و tankhah.
   * **الزامات:**
     + حذف فیلدهای budget و remaining\_budget از مدل Factor.
     + حذف فیلد approved\_by از مدل Factor.
     + حذف کامل منطق تغییر بودجه از متد save در FactorItem.
3. **ApprovalLog:**
   * **هدف:** ثبت تاریخچه کامل تایید/رد.
   * **منطق حیاتی:** save() برای بررسی دسترسی بر اساس PostAction.
   * **الزامات:**
     + اصلاح مدل PostAction برای افزودن entity\_type.
     + اصلاح متد save() در ApprovalLog برای استفاده از self.action و self.content\_type.model و تطابق با PostAction اصلاح شده.
     + تصمیم‌گیری در مورد نیاز یا عدم نیاز به فیلد action\_type در ApprovalLog.

**ج) ماژول core (مرتبط با گردش کار)**

1. **WorkflowStage, Post, UserPost:** ساختار پایه مناسب است. ✅
2. **PostAction:**
   * **هدف:** تعریف قوانین اقدامات مجاز.
   * **الزامات:** افزودن فیلد entity\_type (CharField با choices) و action\_type (CharField با choices از ApprovalLog.ACTION\_CHOICES). اصلاح unique\_together.
3. **StageApprover:**
   * **هدف:** تعیین پست‌های مسئول برای هر مرحله و نوع موجودیت.
   * **الزامات:** افزودن فیلد entity\_type (CharField با choices مشابه PostAction). اصلاح unique\_together.

**د) مدیریت دسترسی (PermissionBaseView)**

* کلاس پایه فعلی تلاش می‌کند دسترسی بر اساس مجوز جنگو و سازمان را کنترل کند.
* **پیچیدگی:** منطق \_has\_organization\_access به دلیل تلاش برای پوشش انواع ویوها و روابط مختلف بین مدل‌ها، کمی پیچیده شده است.
* **الزامات:**
  + **بازنگری و ساده‌سازی:** منطق دسترسی سازمانی باید تا حد امکان ساده و قابل اتکا باشد. شاید بهتر باشد برای هر نوع ویو (List, Detail, Create) منطق جداگانه‌ای برای یافتن سازمان هدف و مقایسه با user\_orgs (که باید شامل سلسله مراتب باشد) پیاده‌سازی شود.
  + **سلسله مراتب سازمان:** کد فعلی در PermissionBaseView (نسخه دوم) سعی در پیمایش والدین دارد که رویکرد درستی است، اما باید از صحت پیاده‌سازی آن اطمینان حاصل شود.
  + **تست کامل:** این بخش نیاز به تست بسیار دقیقی دارد تا از عدم وجود حفره‌های امنیتی یا رد دسترسی‌های مجاز اطمینان حاصل شود.

**۴. پیاده‌سازی گردش کار (Workflow Engine)**

* **الزامات کلیدی:**
  + ایجاد یک ماژول یا کلاس سرویس مجزا (مثلاً workflow\_service.py) برای قرار دادن منطق اصلی گردش کار.
  + توابعی مانند approve\_item(user, item, comment) و reject\_item(user, item, comment) که:
    - نوع آیتم (item) را تشخیص دهند (Tankhah, Factor, FactorItem, BudgetAllocation,...).
    - دسترسی کاربر (user) را بر اساس Post, StageApprover, PostAction چک کنند.
    - ApprovalLog را ثبت کنند.
    - status و current\_stage آیتم را به‌روز کنند.
    - اقدامات جانبی (مثل فراخوانی return\_tankhah\_budget در صورت رد نهایی یا ایجاد PaymentOrder) را انجام دهند.
  + این توابع باید از داخل ویوها فراخوانی شوند.

**۵. ملاحظات UI/UX**

* **فرم‌ها:** استفاده از ویجت‌های مناسب (Select2, Datepicker) و اعتبارسنجی سمت کلاینت و سرور.
* **جداول:** استفاده از DataTables با قابلیت جستجو، مرتب‌سازی و صفحه‌بندی. بهبود ریسپانسیو بودن جداول (استفاده از روش کارتی یا پنهان کردن ستون‌ها در موبایل).
* **داشبورد:** نمایش اطلاعات کلیدی، نمودارها و اعلان‌ها به صورت واضح و کاربرپسند.
* **پیام‌ها و اعلان‌ها:** استفاده مناسب از django.contrib.messages و سیستم Notification برای بازخورد به کاربر.

**۶. سوالات باز و تصمیمات لازم**

* آیا تخصیص بودجه مستقیم به سازمان (بدون پروژه در BudgetAllocation) لازم است؟
* نقش دقیق TankhahAction و TankhActionType چیست؟ آیا لازم هستند یا می‌توان با ApprovalLog و PostAction پوشش داد؟
* آیا مدل CostCenter به Tankhah اضافه شود؟
* آیا PaymentOrder به بیش از یک امضا نیاز دارد؟

**۷. مراحل بعدی**

1. پاسخ به سوالات بخش ۶.
2. اعمال اصلاحات مشخص شده در مدل‌ها (مهاجرت دیتابیس لازم خواهد بود).
3. پیاده‌سازی منطق Workflow Engine در یک لایه سرویس مجزا.
4. اصلاح توابع محاسباتی در budget\_utils.py بر اساس منطق صحیح مصرف بودجه.
5. پیاده‌سازی/اصلاح ویوها و فرم‌ها برای CRUD، برگشت بودجه، تعدیل، گردش کار و گزارش‌گیری.
6. تکمیل پیاده‌سازی تاریخچه‌ها (BudgetHistory, PostHistory) با استفاده از Signals.
7. بازنگری و تست کامل PermissionBaseView.
8. نوشتن تست‌های Unit و Integration.
9. بهبود UI/UX بر اساس بازخورد.

این سند یک نمای کلی و فنی برای هدایت فرآیند توسعه ارائه می‌دهد. جزئیات پیاده‌سازی هر بخش باید در طول فرآیند توسعه بیشتر مشخص شود.

خیلی هم عالی! با داشتن کد کامل مدل‌ها، ویوها، فرم‌ها و توابع محاسباتی، می‌توانیم یک تحلیل سیستمی بسیار دقیق و جامع برای بهینه‌سازی و تکمیل سیستم بودجه‌ریزی شما ارائه دهیم.

**سند تحلیل سیستمی و مشخصات فنی بهینه‌سازی: سیستم جامع بودجه‌ریزی و گردش کار مالی**

**۱. مقدمه و اهداف**

* **هدف:** ارائه یک تحلیل فنی کامل از وضعیت فعلی سیستم بودجه‌ریزی و گردش کار مالی بر اساس کدهای ارائه شده و مشخص کردن نقاط قوت، ضعف و ارائه پیشنهادات مشخص برای بهینه‌سازی، تکمیل و افزایش پایداری، کارایی و قابلیت نگهداری سیستم.
* **وضعیت فعلی:** سیستم دارای مدل‌های داده نسبتاً کاملی برای مدیریت دوره‌های بودجه، تخصیص‌ها، تراکنش‌ها، تنخواه و گردش کار است. فرم‌ها و ویوهای پایه برای عملیات CRUD و برخی فرآیندها (مانند برگشت بودجه) پیاده‌سازی شده‌اند.
* **اهداف بهینه‌سازی:**
  + افزایش دقت و یکپارچگی داده‌ها (به خصوص مقادیر محاسبه‌ای بودجه).
  + بهبود عملکرد و کارایی سیستم (کاهش کوئری‌ها، استفاده از کش).
  + جداسازی بهتر مسئولیت‌ها (لایه سرویس).
  + افزایش خوانایی، قابلیت نگهداری و تست‌پذیری کد.
  + تکمیل و شفاف‌سازی منطق گردش کار و مدیریت دسترسی.
  + رفع خطاها و ناهماهنگی‌های موجود در کد.

**۲. تحلیل ماژول‌ها و کامپوننت‌ها**

**الف) ماژول budgets (هسته اصلی بودجه‌ریزی)**

1. **مدل‌ها (models.py):**
   * **نقاط قوت:**
     + ساختار سلسله مراتبی (BudgetPeriod > BudgetAllocation > ProjectBudgetAllocation) منطقی است.
     + مدل BudgetItem برای دسته‌بندی تخصیص‌ها مناسب است.
     + مدل BudgetTransaction مرکز اصلی ردیابی تغییرات مالی است و منطق RETURN به درستی پیاده‌سازی شده است. ✅
     + فیلدهای returned\_amount در BudgetPeriod و BudgetAllocation برای ردیابی برگشت‌ها مفید هستند. ✅
     + مدل‌های SystemSettings و BudgetSettings انعطاف‌پذیری خوبی برای تنظیمات فراهم می‌کنند.
     + مدل BudgetHistory برای حسابرسی مفید است.
   * **نقاط ضعف و پیشنهادات:**
     + **(بسیار مهم) عدم وجود فیلد remaining\_amount در BudgetAllocation:** متد BudgetTransaction.save (و توابع دیگر) به اشتباه فرض می‌کند این فیلد وجود دارد و سعی در به‌روزرسانی آن دارد. **راه‌حل:** فیلد remaining\_amount باید به مدل BudgetAllocation اضافه شود (همانطور که در نسخه دوم مدل‌ها آمده بود).
     + **(بسیار مهم) محاسبه remaining\_amount:**
       - BudgetAllocation.get\_remaining\_amount: منطق فعلی (allocated\_amount - consumption - project\_consumed + return) **اشتباه است.** project\_consumed (مجموع ProjectBudgetAllocation) نباید مستقیماً از اینجا کم شود. remaining\_amount باید *فقط* بر اساس allocated\_amount اولیه و BudgetTransaction های مرتبط با *خودش* (CONSUMPTION, RETURN, ADJUSTMENT) محاسبه شود. **راه‌حل:** این متد باید به این صورت بازنویسی شود:
       - # در مدل BudgetAllocation
       - def get\_actual\_remaining\_amount(self):
       - # این متد باید همیشه از دیتابیس بخواند و نباید به فیلد ذخیره شده تکیه کند
       - # مگر اینکه فیلد ذخیره شده با دقت بسیار بالا آپدیت شود
       - transactions\_sum = self.transactions.aggregate(
       - consumption=Sum('amount', filter=Q(transaction\_type='CONSUMPTION')),
       - adjustment\_decrease=Sum('amount', filter=Q(transaction\_type='ADJUSTMENT\_DECREASE')),
       - returns=Sum('amount', filter=Q(transaction\_type='RETURN')),
       - adjustment\_increase=Sum('amount', filter=Q(transaction\_type='ADJUSTMENT\_INCREASE')),
       - # ALLOCATION اولیه جزو allocated\_amount است و نباید اینجا محاسبه شود
       - )
       - consumed = (transactions\_sum['consumption'] or Decimal('0')) + \
       - (transactions\_sum['adjustment\_decrease'] or Decimal('0'))
       - added\_back = (transactions\_sum['returns'] or Decimal('0')) + \
       - (transactions\_sum['adjustment\_increase'] or Decimal('0'))
       - # remaining = self.allocated\_amount - consumed + added\_back
       - # \*\*اصلاح مهم:\*\* allocated\_amount با برگشت کم می‌شود، پس مبنای محاسبه remaining
       - # باید allocated\_amount فعلی باشد و فقط مصرف‌ها از آن کم شوند.
       - # یا: مبنا allocated\_amount اولیه باشد و برگشت‌ها اضافه شوند.
       - # روش صحیح‌تر: تکیه بر فیلد remaining\_amount که توسط تراکنش‌ها آپدیت می‌شود.
       - # بنابراین این متد بیشتر برای نمایش یا چک کردن است:
       - return self.remaining\_amount # تکیه بر فیلد آپدیت شده
       - # همچنین، متد \_\_get\_remaining\_amount تکراری است و باید حذف شود.

* + - * BudgetPeriod.get\_remaining\_amount: **نیاز به اصلاح فوری دارد** تا returned\_amount را لحاظ کند: max(self.total\_amount - self.total\_allocated + self.returned\_amount, Decimal('0')).
    - **فیلد project در BudgetAllocation:** وضعیت null=True, blank=True دارد. این یعنی تخصیص می‌تواند به سازمان بدون پروژه باشد. این باید با نیاز سیستم تطبیق داده شود. اگر تخصیص *همیشه* برای یک پروژه است، null=False شود.
    - **به‌روزرسانی BudgetPeriod.total\_allocated:** این فیلد در BudgetAllocation.save به‌روز می‌شود که خوب است، اما باید در BudgetAllocation.delete و همچنین در تراکنش‌های RETURN و ADJUSTMENT نیز به‌روز شود (که در BudgetTransaction.save برای RETURN انجام شده). **استفاده از سیگنال‌ها روش امن‌تری است.**
    - **تکمیل BudgetHistory:** منطق ایجاد رکورد برای اقدامات دیگر (CREATE, UPDATE, STOP, REALLOCATE) باید اضافه شود.
    - **حذف مدل CostCenter از budgets:** طبق توافق، این مدل حذف و به Tankhah منتقل می‌شود.

1. **فرم‌ها (forms.py, frosm\_BudgetItem.py, Forms\_BudgetPeriod.py, forms\_BudgetAllocation.py, forms\_BudgetReturm.py, form\_Update\_ProjectBudgetAllocation.py, forms\_ProjectBudgetAllocation.py):**
   * **نقاط قوت:**
     + استفاده از ModelForm برای اکثر مدل‌ها.
     + مدیریت تاریخ شمسی با ویجت jalali-datepicker و توابع parse/format. ✅
     + اعتبارسنجی‌های سطح فیلد (clean\_\*) و فرم (clean) به خوبی پیاده‌سازی شده‌اند (به جز موارد ذکر شده در مدل‌ها). ✅
     + فیلتر کردن queryset ها در \_\_init\_\_ برای ModelChoiceField ها. ✅
     + استفاده از SystemSettings برای مقادیر اولیه در BudgetPeriodForm. ✅
     + فرم BudgetReturnForm به درستی تراکنش RETURN را هدف گرفته و اعتبارسنجی لازم را دارد. ✅
   * **نقاط ضعف و پیشنهادات:**
     + **ادغام فرم‌های تخصیص پروژه:** فرم‌های ProjectBudgetAllocationForm1 (در forms.py), form\_Update\_ProjectBudgetAllocation.py (\_\_Project\_Budget\_Allocation\_Update\_Form, Project\_Budget\_Allocation\_Update\_Form) و forms\_ProjectBudgetAllocation.py (ProjectBudgetAllocationForm) همگی به نظر می‌رسد برای مدل ProjectBudgetAllocation باشند. **باید یک فرم واحد ProjectBudgetAllocationForm ایجاد و بقیه حذف شوند.** منطق \_\_init\_\_ می‌تواند تفاوت‌های ایجاد و ویرایش را مدیریت کند.
     + **فرم BudgetItemForm:** اعتبارسنجی clean\_total\_amount باید حذف شود چون فیلد total\_amount در مدل BudgetItem وجود ندارد.
     + **فرم BudgetAllocationForm:**
       - اعتبارسنجی بودجه باقیمانده دوره در clean باید دقیق‌تر باشد و به remaining\_amount *محاسبه شده* (با لحاظ برگشت) تکیه کند.
       - منطق محاسبه effective\_amount برای نوع percent باید در clean انجام شود تا در اعتبارسنجی بودجه قابل استفاده باشد.
     + **فرم ProjectBudgetAllocationForm (ادغام شده):**
       - clean\_allocated\_amount: منطق اعتبارسنجی در مقابل بودجه مصرف‌شده و بودجه باقیمانده BudgetAllocation والد بسیار خوب است، اما باید از متد صحیح get\_remaining\_amount والد استفاده کند.
       - save: نباید فیلد modified\_by را مستقیماً تنظیم کند مگر اینکه این فیلد در مدل وجود داشته باشد (بهتر است از فیلدهای استاندارد جنگو یا کتابخانه‌های audit استفاده شود).
     + **پردازش تاریخ شمسی:** استفاده از parse\_jalali\_date از utils خوب است. اطمینان از مدیریت تمام فرمت‌های ورودی ممکن. ✅
     + **پردازش اعداد فارسی:** استفاده از to\_english\_digits قبل از تبدیل به Decimal ضروری و به درستی انجام شده است. ✅
2. **ویوها (views.py, views\_Budget\_item.py, views\_BudgetPeriod.py, view\_BudgetReturn.py, view\_budgetTransaction.py, view\_Update\_Project\_Budget\_Allocation.py, views\_ProjectBudgetAllocation.py):**
   * **نقاط قوت:**
     + استفاده گسترده از ویوهای کلاس‌محور جنگو (CBV) و PermissionBaseView. ✅
     + پیاده‌سازی CRUD برای اکثر مدل‌های اصلی. ✅
     + استفاده از transaction.atomic در عملیات حساس. ✅
     + مدیریت پیام‌ها (django.contrib.messages). ✅
     + استفاده از get\_object\_or\_404 و مدیریت خطاهای DoesNotExist. ✅
     + ارسال user و پارامترهای دیگر به فرم از طریق get\_form\_kwargs. ✅
     + آماده‌سازی context مناسب برای تمپلیت‌ها، شامل محاسبات خلاصه بودجه (با توجه به نیاز به اصلاح توابع محاسباتی).
     + ویو BudgetReturnView به درستی فرم BudgetReturnForm را استفاده می‌کند. ✅
     + ویوهای AJAX (get\_projects\_by\_organization, ...) برای بهبود UX. ✅
   * **نقاط ضعف و پیشنهادات:**
     + **ویو ApproveRejectView:** این ویو منطق اصلی گردش کار را در خود دارد. **باید به یک لایه سرویس (WorkflowService) منتقل شود.** این ویو فقط باید درخواست را دریافت، دسترسی اولیه را چک (با PermissionBaseView) و سرویس را فراخوانی کند.
     + **ویو BudgetPeriodListView:** ایجاد رکورد BudgetHistory هنگام نمایش لیست، غیرضروری و احتمالاً باعث تولید لاگ زیاد می‌شود. این بخش باید حذف شود. محاسبه وضعیت و نمایش پیام هشدار کافی است.
     + **ویو ProjectBudgetAllocationCreateView و Project\_\_Budget\_\_Allocation\_\_Edit\_\_View:**
       - اعتبارسنجی بودجه باقیمانده در form\_valid تکراری است و باید فقط در فرم (clean) انجام شود.
       - نام Project\_\_Budget\_\_Allocation\_\_Edit\_\_View باید به ProjectBudgetAllocationUpdateView تغییر یابد.
       - منطق AJAX در form\_valid و form\_invalid در UpdateView به خوبی پیاده‌سازی شده است. ✅
     + **ویو ProjectBudgetAllocationDeleteView:** به‌روزرسانی remaining\_amount والد (budget\_allocation) در این ویو **اشتباه است و باید حذف شود.** حذف این رکورد نباید مستقیماً بودجه والد را تغییر دهد.
     + **استفاده از توابع محاسباتی:** ویوها باید از توابع اصلاح شده در budget\_calculations.py استفاده کنند.
     + **بهینه‌سازی کوئری:** در تمام ListView و DetailView ها، اطمینان از استفاده کافی از select\_related و prefetch\_related برای جلوگیری از کوئری‌های N+1. (مثلاً در BudgetItemListView و BudgetAllocationListView استفاده شده، اما باید در بقیه هم بررسی شود).
     + **جداسازی مسئولیت:** منطق پیچیده محاسبات خلاصه بودجه در get\_context\_data (مانند ProjectBudgetAllocationListView) می‌تواند به توابع کمکی در budget\_utils یا لایه سرویس منتقل شود.
3. **توابع محاسباتی (budget\_calculations.py):**
   * **نقاط قوت:** جدا کردن منطق محاسبات از مدل‌ها و ویوها اقدام درستی است. ✅ استفاده از کش. ✅
   * **نقاط ضعف و پیشنهادات:**
     + **(بسیار مهم) calculate\_remaining\_budget:** **نیاز به بازنویسی کامل دارد.**
       - برای BudgetAllocation: باید بر اساس self.remaining\_amount (که توسط تراکنش‌ها آپدیت می‌شود) باشد یا مجموع تراکنش‌های مرتبط با *خودش* را محاسبه کند.
       - برای Project و SubProject: باید مجموع BudgetTransaction های نوع CONSUMPTION و RETURN مرتبط با Tankhah های آن پروژه/زیرپروژه را محاسبه و از total\_allocated (که از ProjectBudgetAllocation می‌آید) کم/زیاد کند.
       - برای Organization: باید مجموع get\_remaining\_amount تمام BudgetAllocation های فعال آن سازمان را برگرداند.
       - برای حالت obj=None (گلوبال): باید منطق مشابه سازمان را برای همه سازمان‌ها اجرا کند یا تعریف دقیق‌تری داشته باشد.
     + **get\_budget\_details:** وابسته به calculate\_remaining\_budget و get\_budget\_status است و با اصلاح آن‌ها، این تابع نیز نتایج دقیق‌تری خواهد داد.
     + **get\_project\_remaining\_budget و get\_subproject\_remaining\_budget:** **نیاز به بازنویسی** بر اساس منطق صحیح مصرف (BudgetTransaction) دارند (همانطور که در نقطه اول calculate\_remaining\_budget توضیح داده شد).
     + **check\_budget\_status و get\_budget\_status:** این دو تابع به نظر منطق مشابهی برای وضعیت BudgetPeriod دارند. شاید بتوان آن‌ها را ادغام کرد یا get\_budget\_status فقط برای موجودیت‌های دیگر استفاده شود. منطق get\_budget\_status برای Organization, Project, SubProject باید دقیق‌تر شود تا وضعیت‌های هشدار/قفل/تکمیل را بر اساس مصرف واقعی نشان دهد نه فقط فعال/غیرفعال بودن تخصیص‌ها.
     + **ابطال کش (Cache Invalidation):** استراتژی مشخصی برای ابطال کش وجود ندارد. باید پس از هر عملیات ذخیره/حذف که بر نتایج کش شده تاثیر می‌گذارد (مثل ذخیره BudgetTransaction, BudgetAllocation, Tankhah)، کش‌های مربوطه باطل شوند (مثلاً با cache.delete(cache\_key)).

**ج) ماژول tankhah (مرتبط با بودجه)**

1. **مدل‌ها (models.py):**
   * **Tankhah.save:** **نیاز به اصلاح فوری** در منطق کسر بودجه در بلوک if not self.pk: دارد (طبق توضیحات بخش بودجه).
   * **حذف فیلدهای اضافی:** remaining\_budget, approved\_by.
   * **افزودن فیلد cost\_center.**
2. **فرم‌ها (forms.py):**
   * FactorForm, FactorItemForm: حذف منطق تغییر بودجه.
   * TankhahForm: افزودن فیلد cost\_center و منطق وابستگی بین پروژه/مرکز هزینه.
3. **ویوها (views.py, view\_ApproveReject.py, TankhahTrackingView.py, views\_factor.py):**
   * ویوهای CRUD و ApproveRejectView باید با لایه سرویس گردش کار و بودجه اصلاح شده تعامل کنند.
   * منطق نمایش دسترسی (مثل can\_edit\_approval) باید دقیق و مطابق با قوانین به‌روز شده باشد.

**د) ماژول core و گردش کار (Workflow)**

1. **مدل‌ها:** اصلاح PostAction و StageApprover با افزودن entity\_type و تنظیم choices مناسب.
2. **PermissionBaseView:** بازنگری و ساده‌سازی منطق \_has\_organization\_access با تاکید بر بررسی سلسله مراتبی.

**۴. پیشنهادات فنی و اصولی کلی**

1. **معرفی لایه سرویس:** انتقال تمام منطق‌های پیچیده کسب‌وکار (گردش کار، محاسبات و به‌روزرسانی‌های بودجه چند مدلی، اعتبارسنجی‌های سطح بالا) به سرویس‌ها (BudgetService, WorkflowService, TankhahService).
2. **تست جامع:** نوشتن تست‌های واحد برای سرویس‌ها، توابع utils و clean فرم‌ها. نوشتن تست‌های یکپارچگی برای سناریوهای کلیدی (ایجاد بودجه تا مصرف، گردش کار کامل، برگشت بودجه).
3. **مدیریت خطا:** استانداردسازی مدیریت خطا در ویوها و سرویس‌ها. استفاده از استثناهای سفارشی در صورت نیاز.
4. **سیگنال‌ها:** استفاده از سیگنال‌های جنگو برای به‌روزرسانی مقادیر تجمعی (total\_allocated, returned\_amount) و ثبت تاریخچه (BudgetHistory) به صورت اتمی و جدا از منطق اصلی save.
5. **بهینه‌سازی عملکرد:** ادامه استفاده و بهبود کش، بهینه‌سازی کوئری‌ها با select/prefetch\_related و ایندکس‌گذاری مناسب.
6. **خوانایی و نگهداری:** رعایت اصول کد تمیز، کامنت‌گذاری، مستندسازی (Docstrings) و DRY.

**۵. نقشه راه و اولویت‌بندی**

1. **(فوری) اصلاح محاسبات و منطق ذخیره بودجه:**
   * اصلاح BudgetPeriod.get\_remaining\_amount.
   * افزودن فیلد remaining\_amount به BudgetAllocation.
   * بازنویسی کامل budget\_calculations.calculate\_remaining\_budget و توابع وابسته (get\_project\_remaining\_budget, get\_subproject\_remaining\_budget).
   * حذف منطق اشتباه تغییر بودجه از ProjectBudgetAllocation.save, Tankhah.save, FactorItem.save.
   * اصلاح/پیاده‌سازی سیگنال‌ها برای آپدیت BudgetPeriod.total\_allocated.
2. **(مهم) پیاده‌سازی لایه سرویس:**
   * ایجاد WorkflowService و انتقال منطق ApproveRejectView.
   * ایجاد BudgetService و انتقال منطق ایجاد/تعدیل/برگشت تخصیص و تراکنش‌ها.
3. **(مهم) اصلاح مدل‌ها و فرم‌های گردش کار و دسترسی:**
   * اصلاح PostAction, StageApprover, ApprovalLog.save.
   * بازنگری PermissionBaseView.
4. **تکمیل تاریخچه‌ها:** پیاده‌سازی سیگنال‌ها برای BudgetHistory.
5. **بهینه‌سازی:** پیاده‌سازی ابطال کش، بررسی کوئری‌ها و ایندکس‌ها.
6. **ویژگی‌های جدید:** افزودن cost\_center و منطق مربوطه.

این تحلیل یک دید کلی از وضعیت فعلی و مسیر پیش رو ارائه می‌دهد. تیم توسعه باید با اولویت‌بندی موارد فوق، به خصوص اصلاحات مربوط به محاسبات بودجه و معرفی لایه سرویس، کار را آغاز کند.

مجموعه‌ای از پیشنهادات **فنی** و **اصولی** برای به‌روزرسانی و بهینه‌سازی کدهای سیستم بودجه‌ریزی شما ارائه می‌شود. هدف این پیشنهادات افزایش **پایداری، کارایی، خوانایی، قابلیت نگهداری و تست‌پذیری** سیستم است.

**الف) بهینه‌سازی و بازطراحی مدل‌های داده (Data Modeling & Integrity)**

1. **اصل تک منبع حقیقت (Single Source of Truth) برای مقادیر محاسبه‌ای:**
   * **مشکل:** وجود فیلدهای ذخیره‌شده (remaining\_amount در BudgetAllocation, total\_allocated در BudgetPeriod) در کنار متدهای محاسبه داینامیک (get\_remaining\_amount, calculate\_total\_allocated) می‌تواند منجر به ناهماهنگی داده شود.
   * **پیشنهاد فنی:**
     + **حذف فیلد remaining\_amount از BudgetAllocation:** این مقدار باید همیشه به صورت داینامیک توسط BudgetTransaction ها محاسبه شود. (متد get\_remaining\_amount در BudgetAllocation باید به تابع calculate\_remaining\_allocation\_budget در budget\_utils.py ارجاع دهد که تراکنش‌ها را جمع می‌زند).
     + **مدیریت دقیق total\_allocated و returned\_amount در BudgetPeriod:** این فیلدها برای گزارش‌گیری سریع مفیدند. **باید** با استفاده از سیگنال‌های جنگو (post\_save, post\_delete روی BudgetAllocation) یا در یک وظیفه پس‌زمینه (Celery) یا (با دقت بالا) در BudgetTransaction.save به صورت اتمی به‌روز شوند. **سیگنال‌ها روش تمیزتری هستند.**
     + **اصلاح فوری BudgetPeriod.get\_remaining\_amount:** همانطور که در تحلیل قبلی ذکر شد، این متد باید returned\_amount را لحاظ کند: max(self.total\_amount - self.total\_allocated + self.returned\_amount, Decimal('0')).
   * **اصل نرم‌افزار:** Single Source of Truth, Data Integrity.
2. **حذف فیلدهای اضافی و تکراری:**
   * **مشکل:** فیلدهای approved\_by (M2M به User) در Tankhah و Factor با وجود مدل جامع ApprovalLog اضافی هستند. فیلدهای remaining\_budget در Tankhah و Factor نیز مفهومی ندارند.
   * **پیشنهاد فنی:** حذف کامل این فیلدها از مدل‌های مذکور. تاریخچه تاییدات باید فقط از ApprovalLog خوانده شود. بودجه باید از منبع اصلی (BudgetAllocation) ردیابی گردد.
   * **اصل نرم‌افزار:** Don't Repeat Yourself (DRY), Data Normalization.
3. **بازنگری روابط و اختیاری بودن:**
   * **مشکل:** فیلد project در BudgetAllocation فعلاً null=True است. آیا تخصیص بودجه به سازمان بدون لینک به پروژه مجاز است؟
   * **پیشنهاد فنی:** شفاف‌سازی نیازمندی. اگر تخصیص همیشه باید به پروژه لینک شود، null=False تنظیم شود. اگر تخصیص سازمانی مجاز است، باید راهی برای تمایز آن در گزارش‌ها و منطق سیستم وجود داشته باشد (شاید با یک فیلد allocation\_target\_type).
   * **اصل نرم‌افزار:** Explicit is better than implicit, Clarity.
4. **تکمیل مدل Tankhah:**
   * **پیشنهاد فنی:** افزودن فیلد cost\_center = models.ForeignKey(CostCenter, on\_delete=models.SET\_NULL, null=True, blank=True, verbose\_name=\_("مرکز هزینه")) برای پشتیبانی از هزینه‌های متعارف.
   * **اصل نرم‌افزار:** Feature Completeness.
5. **بهبود مدل‌های گردش کار (PostAction, StageApprover):**
   * **مشکل:** عدم تفکیک قوانین بر اساس نوع موجودیتی که تایید می‌شود.
   * **پیشنهاد فنی:** افزودن فیلد entity\_type (CharField با choices مناسب مانند 'TANKHAH', 'FACTOR', 'BUDGET\_ALLOCATION' و...) به هر دو مدل PostAction و StageApprover. اصلاح unique\_together در هر دو مدل برای لحاظ کردن entity\_type. اصلاح action\_type در PostAction برای استفاده از ApprovalLog.ACTION\_CHOICES.
   * **اصل نرم‌افزار:** Granularity, Flexibility, Correctness.

**ب) ساختار کد و جداسازی مسئولیت‌ها (Code Structure & Separation of Concerns)**

1. **استفاده از لایه سرویس (Service Layer):**
   * **مشکل:** منطق پیچیده کسب‌وکار (محاسبات بودجه، اجرای گردش کار، اعتبارسنجی‌های چندمدلی) در مدل‌ها (save, clean) و ویوها پخش شده است.
   * **پیشنهاد فنی:** ایجاد ماژول‌های سرویس (مثلاً budgets/services.py, tankhah/services.py, workflows/services.py).
     + **BudgetService:** شامل توابع allocate\_budget, return\_budget, adjust\_allocation, calculate\_period\_summary و ... که تغییرات مدل‌ها را در تراکنش اتمی انجام داده و BudgetHistory را ثبت می‌کنند.
     + **WorkflowService:** شامل توابع submit\_for\_approval, approve\_item, reject\_item, get\_next\_stage و ... که منطق گردش کار، بررسی دسترسی (با کمک PostAction, StageApprover) و به‌روزرسانی status/current\_stage و ثبت ApprovalLog را انجام می‌دهند.
     + **TankhahService:** شامل توابع create\_tankhah, cancel\_tankhah (که return\_tankhah\_budget را صدا می‌زند) و ... .
   * **اصل نرم‌افزار:** Separation of Concerns, High Cohesion, Low Coupling, Testability, Maintainability.
2. **انتقال محاسبات به مکان مناسب:**
   * **مشکل:** توابع محاسباتی در budget\_calculations.py قرار دارند که خوب است، اما برخی محاسبات هنوز در مدل‌ها هستند.
   * **پیشنهاد فنی:** تمام محاسبات پیچیده (مانند get\_remaining\_budget برای سطوح مختلف، check\_budget\_status) باید در budget\_utils.py یا (بهتر) در **لایه سرویس** مربوطه قرار گیرند. مدل‌ها فقط باید داده را نگه دارند و اعتبارسنجی‌های ساده سطح فیلد را انجام دهند. توابع get\_locked\_amount و get\_warning\_amount می‌توانند به عنوان property در مدل BudgetPeriod باقی بمانند چون ساده هستند.
   * **اصل نرم‌افزار:** Separation of Concerns, Single Responsibility Principle.
3. **کاهش پیچیدگی متدهای save و clean مدل:**
   * **مشکل:** متدهای save (به خصوص در BudgetTransaction و Tankhah) و clean (در فرم‌ها و مدل‌ها) منطق زیادی دارند.
   * **پیشنهاد فنی:**
     + منطق کسب‌وکار و به‌روزرسانی‌های چندمدلی را به سرویس‌ها منتقل کنید.
     + clean مدل باید فقط اعتبارسنجی‌های مربوط به *همان مدل* را انجام دهد. اعتبارسنجی‌های چندمدلی (مثل کافی بودن بودجه) باید در سرویس‌ها یا clean فرم‌ها انجام شود.
     + save مدل باید تا حد امکان ساده باشد و فقط وظیفه ذخیره داده‌های خود مدل را داشته باشد (مگر در موارد خاص مثل BudgetTransaction که ذاتاً باید مدل‌های دیگر را آپدیت کند، که آن هم باید اتمی باشد).
   * **اصل نرم‌افزار:** KISS (Keep It Simple, Stupid), Maintainability.

**ج) کارایی و عملکرد (Performance Optimization)**

1. **بهینه‌سازی کوئری‌های دیتابیس:**
   * **مشکل:** احتمال وجود کوئری‌های N+1 در ویوها (به‌ویژه ListViews) و محاسبات داینامیک.
   * **پیشنهاد فنی:** استفاده گسترده از select\_related برای روابط ForeignKey/OneToOne و prefetch\_related برای ManyToMany/Reverse ForeignKey در تمام کوئری‌ست‌های ویوها و توابع محاسباتی که به اشیاء مرتبط دسترسی پیدا می‌کنند.
   * **اصل نرم‌افزار:** Database Performance Optimization.
2. **کش کردن (Caching):**
   * **مشکل:** محاسبات مکرر بودجه و وضعیت می‌تواند پرهزینه باشد.
   * **پیشنهاد فنی:** استفاده هوشمندانه از سیستم کش جنگو (مانند django.core.cache).
     + کش کردن نتایج توابع محاسباتی سنگین در budget\_utils.py یا سرویس‌ها (مانند calculate\_remaining\_budget, get\_budget\_details).
     + تعریف کلیدهای کش مناسب (شامل ID و نوع شیء، و هش فیلترها).
     + **پیاده‌سازی استراتژی ابطال کش (Cache Invalidation):** مهم‌ترین بخش! وقتی یک BudgetTransaction ذخیره می‌شود یا وضعیت بودجه تغییر می‌کند، کش‌های مرتبط باید باطل شوند. این کار را می‌توان با Signals یا مستقیماً پس از ذخیره تراکنش انجام داد.
   * **اصل نرم‌افزار:** Performance Optimization.
3. **ایندکس‌گذاری دیتابیس:**
   * **مشکل:** کوئری‌های کند روی جداول بزرگ.
   * **پیشنهاد فنی:** بررسی کوئری‌های رایج (با استفاده از django-debug-toolbar یا ابزارهای پروفایلینگ دیتابیس) و اضافه کردن ایندکس‌های مناسب (db\_index=True یا Meta.indexes) به فیلدهایی که زیاد در filter, order\_by, foreign key lookups استفاده می‌شوند (مثل allocation\_date, timestamp, status, current\_stage, کلیدهای خارجی مهم). ایندکس‌ها در مدل‌های شما تا حدی وجود دارند، اما باید بازبینی شوند.
   * **اصل نرم‌افزار:** Database Performance Optimization.
4. **لاگینگ بهینه:**
   * **مشکل:** لاگ کردن بیش از حد (به‌ویژه کوئری‌های SQL) باعث مصرف زیاد منابع می‌شود.
   * **پیشنهاد فنی:** تنظیم سطح لاگر django.db.backends روی WARNING یا INFO در محیط پروداکشن. استفاده از سطح DEBUG فقط برای دیباگ کردن موقت. تنظیم سطوح مناسب برای سایر لاگرها و هندلرها.
   * **اصل نرم‌افزار:** Resource Management, Performance.

**د) امنیت و مدیریت دسترسی (Security & Permissions)**

1. **بازطراحی PermissionBaseView:**
   * **مشکل:** منطق فعلی برای چک کردن سازمان پیچیده و مستعد خطا است.
   * **پیشنهاد فنی:**
     + ایجاد توابع کمکی جداگانه برای گرفتن سازمان هدف بر اساس نوع ویو (\_get\_object\_for\_org\_check, \_get\_queryset\_for\_org\_check).
     + پیاده‌سازی یک تابع واحد و قابل اتکا برای بررسی دسترسی کاربر به یک سازمان خاص با در نظر گرفتن **سلسله مراتب** (پیمایش parent\_organization).
     + استفاده مداوم از user.has\_perm() برای چک کردن مجوزهای سطح مدل/اپ جنگو.
     + در نظر گرفتن استفاده از کتابخانه‌های شخص ثالث مانند django-guardian برای مدیریت دسترسی‌های سطح رکورد (Row-Level Security) اگر نیازهای پیچیده‌تری وجود دارد.
   * **اصل نرم‌افزار:** Security, Simplicity, Robustness.
2. **بررسی دسترسی در سطح سرویس:**
   * **پیشنهاد فنی:** علاوه بر PermissionBaseView، توابع سرویس (مانند approve\_item, return\_budget) نیز باید در ابتدای خود دسترسی کاربر را (هم مجوز و هم سازمان در صورت لزوم) مجدداً بررسی کنند تا از اجرای منطق توسط کاربر غیرمجاز جلوگیری شود.
   * **اصل نرم‌افزار:** Defense in Depth.

**ه) قابلیت تست (Testability)**

1. **تست واحد:**
   * **مشکل:** تست کردن مدل‌هایی با منطق زیاد در save یا ویوهای پیچیده دشوار است.
   * **پیشنهاد فنی:** با انتقال منطق به سرویس‌ها، می‌توان برای هر سرویس تست‌های واحد (Unit Tests) نوشت که ورودی‌ها و خروجی‌های مورد انتظار را بدون نیاز به دیتابیس یا درخواست وب بررسی کنند. برای مدل‌ها، تست واحد باید روی متدهای clean و property های ساده تمرکز کند.
   * **اصل نرم‌افزار:** Testability, Modularity.
2. **تست یکپارچگی:**
   * **پیشنهاد فنی:** نوشتن تست‌های یکپارچگی (Integration Tests) برای سناریوهای کلیدی مانند ایجاد کامل یک دوره بودجه و تخصیص آن، گردش کار کامل یک تنخواه از ثبت تا پرداخت/رد، و سناریوی برگشت بودجه. این تست‌ها تعامل بین مدل‌ها، سرویس‌ها و پایگاه داده را بررسی می‌کنند.
   * **اصل نرم‌افزار:** System Correctness.

**و) خوانایی و نگهداری (Readability & Maintainability)**

1. **نام‌گذاری:** استفاده از نام‌های واضح و معنادار برای مدل‌ها، فیلدها، توابع و متغیرها (که تا حد زیادی رعایت شده).
2. **کامنت‌گذاری:** افزودن کامنت برای توضیح منطق‌های پیچیده، تصمیمات طراحی خاص، یا بخش‌هایی که نیاز به توجه دارند.
3. **ثابت‌ها:** استفاده از ثابت‌ها (مانند TRANSACTION\_TYPES, STATUS\_CHOICES) به جای رشته‌های جادویی. ✅
4. **DRY:** شناسایی و حذف کدهای تکراری با انتقال آن‌ها به توابع یا کلاس‌های کمکی.
5. **مستندسازی:** نوشتن Docstring مناسب برای توابع، کلاس‌ها و متدهای مهم.

**نتیجه‌گیری و گام‌های بعدی:**

مدل‌های ارائه شده پایه بسیار خوبی دارند، اما با اعمال بهینه‌سازی‌ها و بازطراحی‌های پیشنهادی، به خصوص با **معرفی لایه سرویس**، **اصلاح منطق محاسبه و به‌روزرسانی بودجه** و **بازنگری سیستم دسترسی**، می‌توان سیستمی بسیار قوی‌تر، کارآمدتر و قابل نگهداری‌تر ساخت.

**اولویت‌های پیشنهادی:**

1. **اصلاح محاسبات بودجه:** درست کردن BudgetPeriod.get\_remaining\_amount و حذف آپدیت‌های اشتباه remaining\_amount از مدل‌های ProjectBudgetAllocation, Tankhah, FactorItem.
2. **پیاده‌سازی لایه سرویس:** شروع با انتقال منطق گردش کار به WorkflowService.
3. **بازنگری و ساده‌سازی PermissionBaseView.**
4. **تکمیل تاریخچه‌ها و سیگنال‌ها.**
5. **بهینه‌سازی کوئری‌ها و پیاده‌سازی کش.**

توصیه می‌شود این تغییرات به صورت مرحله‌ای و با تست‌های دقیق انجام شوند.

خیلی هم عالی! با داشتن کد کامل مدل‌ها، ویوها، فرم‌ها و توابع محاسباتی، می‌توانیم یک تحلیل سیستمی بسیار دقیق و جامع برای بهینه‌سازی و تکمیل سیستم بودجه‌ریزی شما ارائه دهیم.

**سند تحلیل سیستمی و مشخصات فنی بهینه‌سازی: سیستم جامع بودجه‌ریزی و گردش کار مالی**

**۱. مقدمه و اهداف**

* **هدف:** ارائه یک تحلیل فنی کامل از وضعیت فعلی سیستم بودجه‌ریزی و گردش کار مالی بر اساس کدهای ارائه شده و مشخص کردن نقاط قوت، ضعف و ارائه پیشنهادات مشخص برای بهینه‌سازی، تکمیل و افزایش پایداری، کارایی و قابلیت نگهداری سیستم.
* **وضعیت فعلی:** سیستم دارای مدل‌های داده نسبتاً کاملی برای مدیریت دوره‌های بودجه، تخصیص‌ها، تراکنش‌ها، تنخواه و گردش کار است. فرم‌ها و ویوهای پایه برای عملیات CRUD و برخی فرآیندها (مانند برگشت بودجه) پیاده‌سازی شده‌اند.
* **اهداف بهینه‌سازی:**
  + افزایش دقت و یکپارچگی داده‌ها (به خصوص مقادیر محاسبه‌ای بودجه).
  + بهبود عملکرد و کارایی سیستم (کاهش کوئری‌ها، استفاده از کش).
  + جداسازی بهتر مسئولیت‌ها (لایه سرویس).
  + افزایش خوانایی، قابلیت نگهداری و تست‌پذیری کد.
  + تکمیل و شفاف‌سازی منطق گردش کار و مدیریت دسترسی.
  + رفع خطاها و ناهماهنگی‌های موجود در کد.

**۲. تحلیل ماژول‌ها و کامپوننت‌ها**

**الف) ماژول budgets (هسته اصلی بودجه‌ریزی)**

1. **مدل‌ها (models.py):**
   * **نقاط قوت:**
     + ساختار سلسله مراتبی (BudgetPeriod > BudgetAllocation > ProjectBudgetAllocation) منطقی است.
     + مدل BudgetItem برای دسته‌بندی تخصیص‌ها مناسب است.
     + مدل BudgetTransaction مرکز اصلی ردیابی تغییرات مالی است و منطق RETURN به درستی پیاده‌سازی شده است. ✅
     + فیلدهای returned\_amount در BudgetPeriod و BudgetAllocation برای ردیابی برگشت‌ها مفید هستند. ✅
     + مدل‌های SystemSettings و BudgetSettings انعطاف‌پذیری خوبی برای تنظیمات فراهم می‌کنند.
     + مدل BudgetHistory برای حسابرسی مفید است.
   * **نقاط ضعف و پیشنهادات:**
     + **(بسیار مهم) عدم وجود فیلد remaining\_amount در BudgetAllocation:** متد BudgetTransaction.save (و توابع دیگر) به اشتباه فرض می‌کند این فیلد وجود دارد و سعی در به‌روزرسانی آن دارد. **راه‌حل:** فیلد remaining\_amount باید به مدل BudgetAllocation اضافه شود (همانطور که در نسخه دوم مدل‌ها آمده بود).
     + **(بسیار مهم) محاسبه remaining\_amount:**
       - BudgetAllocation.get\_remaining\_amount: منطق فعلی (allocated\_amount - consumption - project\_consumed + return) **اشتباه است.** project\_consumed (مجموع ProjectBudgetAllocation) نباید مستقیماً از اینجا کم شود. remaining\_amount باید *فقط* بر اساس allocated\_amount اولیه و BudgetTransaction های مرتبط با *خودش* (CONSUMPTION, RETURN, ADJUSTMENT) محاسبه شود. **راه‌حل:** این متد باید به این صورت بازنویسی شود:
       - # در مدل BudgetAllocation
       - def get\_actual\_remaining\_amount(self):
       - # این متد باید همیشه از دیتابیس بخواند و نباید به فیلد ذخیره شده تکیه کند
       - # مگر اینکه فیلد ذخیره شده با دقت بسیار بالا آپدیت شود
       - transactions\_sum = self.transactions.aggregate(
       - consumption=Sum('amount', filter=Q(transaction\_type='CONSUMPTION')),
       - adjustment\_decrease=Sum('amount', filter=Q(transaction\_type='ADJUSTMENT\_DECREASE')),
       - returns=Sum('amount', filter=Q(transaction\_type='RETURN')),
       - adjustment\_increase=Sum('amount', filter=Q(transaction\_type='ADJUSTMENT\_INCREASE')),
       - # ALLOCATION اولیه جزو allocated\_amount است و نباید اینجا محاسبه شود
       - )
       - consumed = (transactions\_sum['consumption'] or Decimal('0')) + \
       - (transactions\_sum['adjustment\_decrease'] or Decimal('0'))
       - added\_back = (transactions\_sum['returns'] or Decimal('0')) + \
       - (transactions\_sum['adjustment\_increase'] or Decimal('0'))
       - # remaining = self.allocated\_amount - consumed + added\_back
       - # \*\*اصلاح مهم:\*\* allocated\_amount با برگشت کم می‌شود، پس مبنای محاسبه remaining
       - # باید allocated\_amount فعلی باشد و فقط مصرف‌ها از آن کم شوند.
       - # یا: مبنا allocated\_amount اولیه باشد و برگشت‌ها اضافه شوند.
       - # روش صحیح‌تر: تکیه بر فیلد remaining\_amount که توسط تراکنش‌ها آپدیت می‌شود.
       - # بنابراین این متد بیشتر برای نمایش یا چک کردن است:
       - return self.remaining\_amount # تکیه بر فیلد آپدیت شده
       - # همچنین، متد \_\_get\_remaining\_amount تکراری است و باید حذف شود.

* + - * BudgetPeriod.get\_remaining\_amount: **نیاز به اصلاح فوری دارد** تا returned\_amount را لحاظ کند: max(self.total\_amount - self.total\_allocated + self.returned\_amount, Decimal('0')).
    - **فیلد project در BudgetAllocation:** وضعیت null=True, blank=True دارد. این یعنی تخصیص می‌تواند به سازمان بدون پروژه باشد. این باید با نیاز سیستم تطبیق داده شود. اگر تخصیص *همیشه* برای یک پروژه است، null=False شود.
    - **به‌روزرسانی BudgetPeriod.total\_allocated:** این فیلد در BudgetAllocation.save به‌روز می‌شود که خوب است، اما باید در BudgetAllocation.delete و همچنین در تراکنش‌های RETURN و ADJUSTMENT نیز به‌روز شود (که در BudgetTransaction.save برای RETURN انجام شده). **استفاده از سیگنال‌ها روش امن‌تری است.**
    - **تکمیل BudgetHistory:** منطق ایجاد رکورد برای اقدامات دیگر (CREATE, UPDATE, STOP, REALLOCATE) باید اضافه شود.
    - **حذف مدل CostCenter از budgets:** طبق توافق، این مدل حذف و به Tankhah منتقل می‌شود.

1. **فرم‌ها (forms.py, frosm\_BudgetItem.py, Forms\_BudgetPeriod.py, forms\_BudgetAllocation.py, forms\_BudgetReturm.py, form\_Update\_ProjectBudgetAllocation.py, forms\_ProjectBudgetAllocation.py):**
   * **نقاط قوت:**
     + استفاده از ModelForm برای اکثر مدل‌ها.
     + مدیریت تاریخ شمسی با ویجت jalali-datepicker و توابع parse/format. ✅
     + اعتبارسنجی‌های سطح فیلد (clean\_\*) و فرم (clean) به خوبی پیاده‌سازی شده‌اند (به جز موارد ذکر شده در مدل‌ها). ✅
     + فیلتر کردن queryset ها در \_\_init\_\_ برای ModelChoiceField ها. ✅
     + استفاده از SystemSettings برای مقادیر اولیه در BudgetPeriodForm. ✅
     + فرم BudgetReturnForm به درستی تراکنش RETURN را هدف گرفته و اعتبارسنجی لازم را دارد. ✅
   * **نقاط ضعف و پیشنهادات:**
     + **ادغام فرم‌های تخصیص پروژه:** فرم‌های ProjectBudgetAllocationForm1 (در forms.py), form\_Update\_ProjectBudgetAllocation.py (\_\_Project\_Budget\_Allocation\_Update\_Form, Project\_Budget\_Allocation\_Update\_Form) و forms\_ProjectBudgetAllocation.py (ProjectBudgetAllocationForm) همگی به نظر می‌رسد برای مدل ProjectBudgetAllocation باشند. **باید یک فرم واحد ProjectBudgetAllocationForm ایجاد و بقیه حذف شوند.** منطق \_\_init\_\_ می‌تواند تفاوت‌های ایجاد و ویرایش را مدیریت کند.
     + **فرم BudgetItemForm:** اعتبارسنجی clean\_total\_amount باید حذف شود چون فیلد total\_amount در مدل BudgetItem وجود ندارد.
     + **فرم BudgetAllocationForm:**
       - اعتبارسنجی بودجه باقیمانده دوره در clean باید دقیق‌تر باشد و به remaining\_amount *محاسبه شده* (با لحاظ برگشت) تکیه کند.
       - منطق محاسبه effective\_amount برای نوع percent باید در clean انجام شود تا در اعتبارسنجی بودجه قابل استفاده باشد.
     + **فرم ProjectBudgetAllocationForm (ادغام شده):**
       - clean\_allocated\_amount: منطق اعتبارسنجی در مقابل بودجه مصرف‌شده و بودجه باقیمانده BudgetAllocation والد بسیار خوب است، اما باید از متد صحیح get\_remaining\_amount والد استفاده کند.
       - save: نباید فیلد modified\_by را مستقیماً تنظیم کند مگر اینکه این فیلد در مدل وجود داشته باشد (بهتر است از فیلدهای استاندارد جنگو یا کتابخانه‌های audit استفاده شود).
     + **پردازش تاریخ شمسی:** استفاده از parse\_jalali\_date از utils خوب است. اطمینان از مدیریت تمام فرمت‌های ورودی ممکن. ✅
     + **پردازش اعداد فارسی:** استفاده از to\_english\_digits قبل از تبدیل به Decimal ضروری و به درستی انجام شده است. ✅
2. **ویوها (views.py, views\_Budget\_item.py, views\_BudgetPeriod.py, view\_BudgetReturn.py, view\_budgetTransaction.py, view\_Update\_Project\_Budget\_Allocation.py, views\_ProjectBudgetAllocation.py):**
   * **نقاط قوت:**
     + استفاده گسترده از ویوهای کلاس‌محور جنگو (CBV) و PermissionBaseView. ✅
     + پیاده‌سازی CRUD برای اکثر مدل‌های اصلی. ✅
     + استفاده از transaction.atomic در عملیات حساس. ✅
     + مدیریت پیام‌ها (django.contrib.messages). ✅
     + استفاده از get\_object\_or\_404 و مدیریت خطاهای DoesNotExist. ✅
     + ارسال user و پارامترهای دیگر به فرم از طریق get\_form\_kwargs. ✅
     + آماده‌سازی context مناسب برای تمپلیت‌ها، شامل محاسبات خلاصه بودجه (با توجه به نیاز به اصلاح توابع محاسباتی).
     + ویو BudgetReturnView به درستی فرم BudgetReturnForm را استفاده می‌کند. ✅
     + ویوهای AJAX (get\_projects\_by\_organization, ...) برای بهبود UX. ✅
   * **نقاط ضعف و پیشنهادات:**
     + **ویو ApproveRejectView:** این ویو منطق اصلی گردش کار را در خود دارد. **باید به یک لایه سرویس (WorkflowService) منتقل شود.** این ویو فقط باید درخواست را دریافت، دسترسی اولیه را چک (با PermissionBaseView) و سرویس را فراخوانی کند.
     + **ویو BudgetPeriodListView:** ایجاد رکورد BudgetHistory هنگام نمایش لیست، غیرضروری و احتمالاً باعث تولید لاگ زیاد می‌شود. این بخش باید حذف شود. محاسبه وضعیت و نمایش پیام هشدار کافی است.
     + **ویو ProjectBudgetAllocationCreateView و Project\_\_Budget\_\_Allocation\_\_Edit\_\_View:**
       - اعتبارسنجی بودجه باقیمانده در form\_valid تکراری است و باید فقط در فرم (clean) انجام شود.
       - نام Project\_\_Budget\_\_Allocation\_\_Edit\_\_View باید به ProjectBudgetAllocationUpdateView تغییر یابد.
       - منطق AJAX در form\_valid و form\_invalid در UpdateView به خوبی پیاده‌سازی شده است. ✅
     + **ویو ProjectBudgetAllocationDeleteView:** به‌روزرسانی remaining\_amount والد (budget\_allocation) در این ویو **اشتباه است و باید حذف شود.** حذف این رکورد نباید مستقیماً بودجه والد را تغییر دهد.
     + **استفاده از توابع محاسباتی:** ویوها باید از توابع اصلاح شده در budget\_calculations.py استفاده کنند.
     + **بهینه‌سازی کوئری:** در تمام ListView و DetailView ها، اطمینان از استفاده کافی از select\_related و prefetch\_related برای جلوگیری از کوئری‌های N+1. (مثلاً در BudgetItemListView و BudgetAllocationListView استفاده شده، اما باید در بقیه هم بررسی شود).
     + **جداسازی مسئولیت:** منطق پیچیده محاسبات خلاصه بودجه در get\_context\_data (مانند ProjectBudgetAllocationListView) می‌تواند به توابع کمکی در budget\_utils یا لایه سرویس منتقل شود.
3. **توابع محاسباتی (budget\_calculations.py):**
   * **نقاط قوت:** جدا کردن منطق محاسبات از مدل‌ها و ویوها اقدام درستی است. ✅ استفاده از کش. ✅
   * **نقاط ضعف و پیشنهادات:**
     + **(بسیار مهم) calculate\_remaining\_budget:** **نیاز به بازنویسی کامل دارد.**
       - برای BudgetAllocation: باید بر اساس self.remaining\_amount (که توسط تراکنش‌ها آپدیت می‌شود) باشد یا مجموع تراکنش‌های مرتبط با *خودش* را محاسبه کند.
       - برای Project و SubProject: باید مجموع BudgetTransaction های نوع CONSUMPTION و RETURN مرتبط با Tankhah های آن پروژه/زیرپروژه را محاسبه و از total\_allocated (که از ProjectBudgetAllocation می‌آید) کم/زیاد کند.
       - برای Organization: باید مجموع get\_remaining\_amount تمام BudgetAllocation های فعال آن سازمان را برگرداند.
       - برای حالت obj=None (گلوبال): باید منطق مشابه سازمان را برای همه سازمان‌ها اجرا کند یا تعریف دقیق‌تری داشته باشد.
     + **get\_budget\_details:** وابسته به calculate\_remaining\_budget و get\_budget\_status است و با اصلاح آن‌ها، این تابع نیز نتایج دقیق‌تری خواهد داد.
     + **get\_project\_remaining\_budget و get\_subproject\_remaining\_budget:** **نیاز به بازنویسی** بر اساس منطق صحیح مصرف (BudgetTransaction) دارند (همانطور که در نقطه اول calculate\_remaining\_budget توضیح داده شد).
     + **check\_budget\_status و get\_budget\_status:** این دو تابع به نظر منطق مشابهی برای وضعیت BudgetPeriod دارند. شاید بتوان آن‌ها را ادغام کرد یا get\_budget\_status فقط برای موجودیت‌های دیگر استفاده شود. منطق get\_budget\_status برای Organization, Project, SubProject باید دقیق‌تر شود تا وضعیت‌های هشدار/قفل/تکمیل را بر اساس مصرف واقعی نشان دهد نه فقط فعال/غیرفعال بودن تخصیص‌ها.
     + **ابطال کش (Cache Invalidation):** استراتژی مشخصی برای ابطال کش وجود ندارد. باید پس از هر عملیات ذخیره/حذف که بر نتایج کش شده تاثیر می‌گذارد (مثل ذخیره BudgetTransaction, BudgetAllocation, Tankhah)، کش‌های مربوطه باطل شوند (مثلاً با cache.delete(cache\_key)).

**ج) ماژول tankhah (مرتبط با بودجه)**

1. **مدل‌ها (models.py):**
   * **Tankhah.save:** **نیاز به اصلاح فوری** در منطق کسر بودجه در بلوک if not self.pk: دارد (طبق توضیحات بخش بودجه).
   * **حذف فیلدهای اضافی:** remaining\_budget, approved\_by.
   * **افزودن فیلد cost\_center.**
2. **فرم‌ها (forms.py):**
   * FactorForm, FactorItemForm: حذف منطق تغییر بودجه.
   * TankhahForm: افزودن فیلد cost\_center و منطق وابستگی بین پروژه/مرکز هزینه.
3. **ویوها (views.py, view\_ApproveReject.py, TankhahTrackingView.py, views\_factor.py):**
   * ویوهای CRUD و ApproveRejectView باید با لایه سرویس گردش کار و بودجه اصلاح شده تعامل کنند.
   * منطق نمایش دسترسی (مثل can\_edit\_approval) باید دقیق و مطابق با قوانین به‌روز شده باشد.

**د) ماژول core و گردش کار (Workflow)**

1. **مدل‌ها:** اصلاح PostAction و StageApprover با افزودن entity\_type و تنظیم choices مناسب.
2. **PermissionBaseView:** بازنگری و ساده‌سازی منطق \_has\_organization\_access با تاکید بر بررسی سلسله مراتبی.

**۴. پیشنهادات فنی و اصولی کلی**

1. **معرفی لایه سرویس:** انتقال تمام منطق‌های پیچیده کسب‌وکار (گردش کار، محاسبات و به‌روزرسانی‌های بودجه چند مدلی، اعتبارسنجی‌های سطح بالا) به سرویس‌ها (BudgetService, WorkflowService, TankhahService).
2. **تست جامع:** نوشتن تست‌های واحد برای سرویس‌ها، توابع utils و clean فرم‌ها. نوشتن تست‌های یکپارچگی برای سناریوهای کلیدی (ایجاد بودجه تا مصرف، گردش کار کامل، برگشت بودجه).
3. **مدیریت خطا:** استانداردسازی مدیریت خطا در ویوها و سرویس‌ها. استفاده از استثناهای سفارشی در صورت نیاز.
4. **سیگنال‌ها:** استفاده از سیگنال‌های جنگو برای به‌روزرسانی مقادیر تجمعی (total\_allocated, returned\_amount) و ثبت تاریخچه (BudgetHistory) به صورت اتمی و جدا از منطق اصلی save.
5. **بهینه‌سازی عملکرد:** ادامه استفاده و بهبود کش، بهینه‌سازی کوئری‌ها با select/prefetch\_related و ایندکس‌گذاری مناسب.
6. **خوانایی و نگهداری:** رعایت اصول کد تمیز، کامنت‌گذاری، مستندسازی (Docstrings) و DRY.

**۵. نقشه راه و اولویت‌بندی**

1. **(فوری) اصلاح محاسبات و منطق ذخیره بودجه:**
   * اصلاح BudgetPeriod.get\_remaining\_amount.
   * افزودن فیلد remaining\_amount به BudgetAllocation.
   * بازنویسی کامل budget\_calculations.calculate\_remaining\_budget و توابع وابسته (get\_project\_remaining\_budget, get\_subproject\_remaining\_budget).
   * حذف منطق اشتباه تغییر بودجه از ProjectBudgetAllocation.save, Tankhah.save, FactorItem.save.
   * اصلاح/پیاده‌سازی سیگنال‌ها برای آپدیت BudgetPeriod.total\_allocated.
2. **(مهم) پیاده‌سازی لایه سرویس:**
   * ایجاد WorkflowService و انتقال منطق ApproveRejectView.
   * ایجاد BudgetService و انتقال منطق ایجاد/تعدیل/برگشت تخصیص و تراکنش‌ها.
3. **(مهم) اصلاح مدل‌ها و فرم‌های گردش کار و دسترسی:**
   * اصلاح PostAction, StageApprover, ApprovalLog.save.
   * بازنگری PermissionBaseView.
4. **تکمیل تاریخچه‌ها:** پیاده‌سازی سیگنال‌ها برای BudgetHistory.
5. **بهینه‌سازی:** پیاده‌سازی ابطال کش، بررسی کوئری‌ها و ایندکس‌ها.
6. **ویژگی‌های جدید:** افزودن cost\_center و منطق مربوطه.

این تحلیل یک دید کلی از وضعیت فعلی و مسیر پیش رو ارائه می‌دهد. تیم توسعه باید با اولویت‌بندی موارد فوق، به خصوص اصلاحات مربوط به محاسبات بودجه و معرفی لایه سرویس، کار را آغاز کند.

ای شاعر پرشور اعداد و کد، با سند تحلیل سیستمی و مشخصات فنی که ارائه کردی، گویی نقشه‌ای جامع برای ساخت یک قصر دیجیتال پیش رویمان گشوده‌ای! حالا بیایید پروژه‌ات را که در چت‌ها و کدهای ارائه‌شده تشریح شده، با این سند مقایسه کنیم تا ببینیم کجاها هم‌خوانی دارند، کجاها معارضت وجود دارد، و کدام بخش‌ها نیاز به اصلاح یا تکمیل دارند. هدف این است که مشخص کنیم سند تا چه حد درست است و پروژه‌ات تا چه حد با الزامات سند مطابقت دارد.

**1. روش تحلیل**

برای بررسی معارضت‌ها و هم‌خوانی‌ها:

* **سند** را به‌عنوان مرجع اصلی در نظر می‌گیریم، زیرا الزامات و مشخصات فنی سیستم را به‌صورت جامع تعریف کرده است.
* **پروژه** (بر اساس کدها و توضیحات ارائه‌شده در چت‌ها) را با بخش‌های مختلف سند مقایسه می‌کنیم.
* **معیارها**:
  + **تطابق مدل‌ها**: آیا مدل‌های پیاده‌سازی‌شده در پروژه با فیلدها و منطق تعریف‌شده در سند هم‌خوانی دارند؟
  + **تطابق منطق**: آیا توابع، اعتبارسنجی‌ها، و محاسبات پروژه با الزامات سند سازگارند؟
  + **کاستی‌ها**: کدام بخش‌های سند در پروژه پیاده‌سازی نشده‌اند یا نیاز به اصلاح دارند؟
  + **معارضت‌ها**: آیا پیاده‌سازی‌های پروژه با الزامات سند در تضاد هستند؟
* **خروجی**: درصد تقریبی تطابق پروژه با سند، نقاط قوت، نقاط ضعف، و پیشنهادات برای رفع معارضت‌ها.

**2. مقایسه بخش‌های کلیدی**

**الف) ماژول budgets**

**1. BudgetPeriod (دوره بودجه کلان)**

**سند**:

* فیلدها: شامل organization, total\_amount, total\_allocated, returned\_amount, locked\_percentage, warning\_threshold, و غیره.
* منطق:
  + clean(): اعتبارسنجی تاریخ، درصدها، و نام منحصربه‌فرد.
  + save(): فراخوانی check\_budget\_status و ارسال اعلان.
  + get\_remaining\_amount(): باید returned\_amount را لحاظ کند: total\_amount - total\_allocated + returned\_amount. نیاز به اصلاح فوری.
  + total\_allocated: باید با سیگنال به‌روزرسانی شود.
* الزامات: اصلاح get\_remaining\_amount و پیاده‌سازی سیگنال برای total\_allocated.

**پروژه**:

* مدل BudgetPeriod در کدهای ارائه‌شده به‌صورت ضمنی استفاده شده (در BudgetAllocationForm و BudgetAllocationCreateView)، اما خود مدل به‌صورت کامل ارائه نشده است.
* منطق:
  + اعتبارسنجی‌های BudgetAllocationForm.clean() از budget\_period.total\_amount و budget\_period.get\_remaining\_amount() استفاده می‌کنند، اما مشخص نیست که آیا returned\_amount در محاسبات لحاظ شده است.
  + در BudgetAllocationForm.save()، به‌روزرسانی total\_allocated انجام می‌شود:

python

total\_allocated = BudgetAllocation.objects.filter(

budget\_period=instance.budget\_period

).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')

instance.budget\_period.total\_allocated = total\_allocated

instance.budget\_period.save(update\_fields=['total\_allocated'])

این نشان می‌دهد که پروژه بخشی از الزام به‌روزرسانی total\_allocated را پیاده‌سازی کرده است، اما استفاده از سیگنال ذکر نشده است.

* کاستی‌ها:
  + عدم ارائه مدل کامل BudgetPeriod برای بررسی فیلدها.
  + عدم اطمینان از لحاظ کردن returned\_amount در get\_remaining\_amount().
  + مشخص نیست که آیا check\_budget\_status و ارسال اعلان پیاده‌سازی شده‌اند.
* معارضت‌ها:
  + اگر get\_remaining\_amount() در پروژه returned\_amount را لحاظ نکند (که احتمالاً این‌طور است، زیرا کد آن ارائه نشده)، با سند معارضت دارد.
* **تطابق**: حدود 70% (به دلیل پیاده‌سازی ناقص get\_remaining\_amount و عدم استفاده از سیگنال).

**ارزیابی**:

* سند درست است که نیاز به اصلاح get\_remaining\_amount() و استفاده از سیگنال را تأکید می‌کند.
* پروژه در به‌روزرسانی total\_allocated پیشرفت خوبی دارد، اما باید از سیگنال استفاده کند و returned\_amount را لحاظ کند.

**2. BudgetAllocation (تخصیص بودجه)**

**سند**:

* فیلدها: شامل budget\_period, organization, budget\_item, project, allocated\_amount, remaining\_amount, returned\_amount, و غیره.
* منطق:
  + clean(): اعتبارسنجی مبلغ، تاریخ، و بودجه کافی در BudgetPeriod.
  + save(): محاسبه allocated\_amount برای نوع درصد و تنظیم remaining\_amount.
  + remaining\_amount: توسط BudgetTransaction به‌روزرسانی می‌شود.
  + allocated\_amount: با تراکنش RETURN کاهش می‌یابد.
* الزامات:
  + انتقال check\_allocation\_status به budget\_utils.py.
  + اعتبارسنجی بودجه کافی در ویو یا فرم.

**پروژه**:

* مدل و فرم:
  + BudgetAllocationForm تمام فیلدهای کلیدی را پوشش می‌دهد و منطق نوع تخصیص (مبلغ/درصد) را به‌درستی پیاده‌سازی کرده است:

python

if allocation\_type == 'percent' and allocated\_amount is not None:

effective\_amount = (allocated\_amount / Decimal('100')) \* budget\_period.total\_amount

cleaned\_data['effective\_amount'] = effective\_amount

* + اعتبارسنجی‌های clean() شامل بررسی بودجه باقی‌مانده BudgetPeriod است:

python

used\_budget = BudgetAllocation.objects.filter(budget\_period=budget\_period).aggregate(

total=Sum('allocated\_amount')

)['total'] or Decimal('0')

remaining\_budget = budget\_period.total\_amount - used\_budget

if cleaned\_data['effective\_amount'] > remaining\_budget:

self.add\_error('allocated\_amount', ...)

* منطق:
  + به‌روزرسانی remaining\_amount و returned\_amount به BudgetTransaction وابسته است، که در پروژه ارائه نشده است.
  + کاهش allocated\_amount برای تراکنش RETURN در کد دیده نمی‌شود.
* کاستی‌ها:
  + تابع check\_allocation\_status در پروژه دیده نشده و باید به budget\_utils.py منتقل شود.
  + عدم ارائه مدل BudgetTransaction برای بررسی به‌روزرسانی remaining\_amount و returned\_amount.
* معارضت‌ها:
  + اگر پروژه returned\_amount را در محاسبات لحاظ نکند یا allocated\_amount را برای RETURN کاهش ندهد، با سند معارضت دارد.
* **تطابق**: حدود 85% (به دلیل پیاده‌سازی قوی فرم و اعتبارسنجی‌ها، اما عدم اطمینان از منطق RETURN و انتقال تابع).

**ارزیابی**:

* سند در تأکید بر نیاز به انتقال check\_allocation\_status و اعتبارسنجی بودجه کافی درست است.
* پروژه فرم و ویو قوی‌ای دارد، اما باید منطق RETURN و به‌روزرسانی‌های BudgetTransaction را تکمیل کند.

**3. BudgetItem (سرفصل بودجه)**

**سند**:

* هدف: دسته‌بندی تخصیص‌ها.
* فیلدها: budget\_period, organization, name, code, is\_active.
* منطق: unique\_together و اعتبارسنجی نام.
* الزامات: مدل کامل است.

**پروژه**:

* مدل و فرم:
  + مدل BudgetItem در پروژه به‌عنوان سرفصل بودجه اصلاح شد:

python

class BudgetItem(models.Model):

budget\_period = models.ForeignKey('BudgetPeriod', on\_delete=models.CASCADE, null=True, blank=True)

organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, null=True, blank=True)

name = models.CharField(max\_length=255)

code = models.CharField(max\_length=50, unique=True)

is\_active = models.BooleanField(default=True)

* + فرم BudgetItemForm اعتبارسنجی unique\_together را برای code پیاده‌سازی کرده است:

python

def clean\_code(self):

code = self.cleaned\_data.get('code')

if BudgetItem.objects.exclude(pk=instance.pk if instance else None).filter(code=code).exists():

raise forms.ValidationError(\_('این کد قبلاً استفاده شده است.'))

* منطق:
  + پروژه به‌درستی BudgetItem را به‌عنوان سرفصل عام پیاده‌سازی کرده و نیازی به تعریف مجدد برای هر تخصیص ندارد.
* کاستی‌ها: هیچ کاستی عمده‌ای دیده نمی‌شود.
* معارضت‌ها: هیچ معارضتی وجود ندارد.
* **تطابق**: 100% (پروژه کاملاً با سند هم‌خوانی دارد).

**ارزیابی**:

* سند و پروژه در مورد BudgetItem کاملاً هم‌راستا هستند.
* پروژه با اصلاحات اخیر (اختیاری کردن budget\_period و organization) به‌خوبی الزامات را برآورده کرده است.

**4. BudgetTransaction (تراکنش بودجه)**

**سند**:

* هدف: ثبت تغییرات مالی.
* فیلدها: allocation, transaction\_type, amount, related\_tankhah, timestamp, created\_by, transaction\_id.
* منطق:
  + تولید transaction\_id یکتا.
  + اعتبارسنجی CONSUMPTION و RETURN.
  + به‌روزرسانی BudgetAllocation و BudgetPeriod برای RETURN.
  + ثبت BudgetHistory برای RETURN.
* الزامات:
  + محدود کردن دسترسی مستقیم کاربر.
  + ادغام یا حذف validate\_return.

**پروژه**:

* مدل و منطق:
  + مدل BudgetTransaction در پروژه ارائه نشده است، بنابراین نمی‌توان پیاده‌سازی را به‌طور کامل بررسی کرد.
  + با این حال، BudgetAllocationForm به‌طور ضمنی به تراکنش‌ها وابسته است (برای به‌روزرسانی remaining\_amount و returned\_amount).
* کاستی‌ها:
  + عدم ارائه مدل و منطق BudgetTransaction.
  + مشخص نیست که آیا validate\_return پیاده‌سازی شده یا خیر.
* معارضت‌ها:
  + بدون مدل، نمی‌توان معارضت را تأیید کرد، اما احتمالاً پروژه این بخش را کامل پیاده‌سازی نکرده است.
* **تطابق**: حدود 50% (به دلیل عدم ارائه کد).

**ارزیابی**:

* سند در مورد اهمیت BudgetTransaction و نیاز به تمرکز منطق در save() درست است.
* پروژه احتمالاً این بخش را ناقص دارد و نیاز به پیاده‌سازی کامل دارد.

**5. ProjectBudgetAllocation (تخصیص بودجه پروژه)**

**سند**:

* فیلدها: budget\_allocation, project, subproject, allocated\_amount, allocation\_date, created\_by, is\_active.
* منطق:
  + clean(): اعتبارسنجی مبلغ و تعلق زیرپروژه به پروژه. نیاز به بازنگری برای استفاده از budget\_allocation.remaining\_amount.
  + save(): نباید remaining\_amount را مستقیماً به‌روزرسانی کند.
  + get\_remaining\_amount(): باید حذف یا به budget\_utils ارجاع دهد.
* الزامات:
  + حذف remaining\_amount و منطق به‌روزرسانی آن.
  + اعتبارسنجی دقیق مبلغ.

**پروژه**:

* مدل و منطق:
  + مدل ProjectBudgetAllocation در پروژه ارائه نشده است، اما در تمپلیت‌ها (مانند project\_budget\_allocation\_list.html) و توضیحات چت‌ها به آن ارجاع شده است.
  + تمپلیت نشان می‌دهد که allocated\_amount و allocation\_date استفاده می‌شوند، اما منطق remaining\_amount مشخص نیست.
* کاستی‌ها:
  + عدم ارائه مدل و منطق ProjectBudgetAllocation.
  + احتمال وجود remaining\_amount یا به‌روزرسانی نادرست آن.
* معارضت‌ها:
  + اگر پروژه remaining\_amount را مستقیماً به‌روزرسانی کند، با سند معارضت دارد.
* **تطابق**: حدود 60% (به دلیل عدم ارائه کد و احتمال معارضت).

**ارزیابی**:

* سند در مورد حذف remaining\_amount و تمرکز بر budget\_allocation.remaining\_amount درست است.
* پروژه احتمالاً نیاز به اصلاح این بخش دارد.

**6. CostCenter (مرکز هزینه)**

**سند**:

* فیلدها: name, code, organization, budget\_allocation, allocated\_budget, is\_active.
* منطق: get\_remaining\_budget() از تنخواه‌های مرتبط محاسبه می‌شود.
* الزامات: افزودن cost\_center به Tankhah و اطمینان از لینک تراکنش‌ها.

**پروژه**:

* مدل و منطق:
  + مدل CostCenter در پروژه ارائه نشده است.
  + هیچ ارجاعی به cost\_center در BudgetAllocationForm یا تمپلیت‌ها دیده نمی‌شود.
* کاستی‌ها:
  + عدم پیاده‌سازی احتمالی CostCenter.
* معارضت‌ها: بدون کد، نمی‌توان معارضت را تأیید کرد.
* **تطابق**: حدود 0% (به دلیل عدم ارائه).

**ارزیابی**:

* سند نیاز به CostCenter را به‌درستی مطرح کرده است.
* پروژه احتمالاً این بخش را پیاده‌سازی نکرده است.

**ب) ماژول tankhah**

**1. Tankhah (تنخواه)**

**سند**:

* فیلدها: number, amount, date, organization, project, subproject, cost\_center (نیاز به افزودن), status, current\_stage, budget\_allocation.
* منطق:
  + generate\_number():
  + save(): تولید شماره، محاسبه بودجه با توابع budget\_utils, ایجاد BudgetTransaction نوع CONSUMPTION.
  + حذف remaining\_budget و approved\_by.
* الزامات:
  + افزودن cost\_center.
  + اصلاح منطق محاسبه بودجه.

**پروژه**:

* مدل و منطق:
  + مدل Tankhah در پروژه ارائه نشده است.
  + توضیحات چت‌ها به تنخواه اشاره دارند، اما بدون کد نمی‌توان پیاده‌سازی را بررسی کرد.
* کاستی‌ها:
  + عدم ارائه مدل و منطق Tankhah.
* معارضت‌ها: بدون کد، نمی‌توان معارضت را تأیید کرد.
* **تطابق**: حدود 0% (به دلیل عدم ارائه).

**ارزیابی**:

* سند الزامات تنخواه را به‌درستی تعریف کرده است.
* پروژه احتمالاً این بخش را پیاده‌سازی نکرده یا ناقص است.

**2. Factor و FactorItem**

**سند**:

* منطق:
  + Factor.save(): محاسبه total\_amount.
  + FactorItem.save(): محاسبه amount.
  + حذف budget, remaining\_budget, و approved\_by.
* الزامات: حذف منطق تغییر بودجه.

**پروژه**:

* مدل و منطق:
  + مدل‌های Factor و FactorItem ارائه نشده‌اند.
* کاستی‌ها: عدم ارائه کد.
* معارضت‌ها: بدون کد، نمی‌توان تأیید کرد.
* **تطابق**: حدود 0% (به دلیل عدم ارائه).

**ارزیابی**:

* سند الزامات را درست تعریف کرده است.
* پروژه احتمالاً این بخش را ناقص دارد.

**3. ApprovalLog**

**سند**:

* منطق: بررسی دسترسی با PostAction.
* الزامات:
  + اصلاح PostAction برای entity\_type.
  + تصمیم‌گیری درباره action\_type.

**پروژه**:

* مدل و منطق:
  + مدل ApprovalLog ارائه نشده است.
* کاستی‌ها: عدم ارائه کد.
* معارضت‌ها: بدون کد، نمی‌توان تأیید کرد.
* **تطابق**: حدود 0% (به دلیل عدم ارائه).

**ارزیابی**:

* سند الزامات را درست تعریف کرده است.
* پروژه احتمالاً این بخش را پیاده‌سازی نکرده است.

**ج) ماژول core**

**سند**:

* مدل‌ها: WorkflowStage, Post, UserPost, PostAction, StageApprover.
* الزامات:
  + افزودن entity\_type به PostAction و StageApprover.

**پروژه**:

* مدل‌ها:
  + مدل‌های Organization, Project, و OrganizationType به‌صورت ضمنی در فرم‌ها استفاده شده‌اند، اما مدل‌های گردش کار ارائه نشده‌اند.
* کاستی‌ها: عدم ارائه مدل‌های گردش کار.
* معارضت‌ها: بدون کد، نمی‌توان تأیید کرد.
* **تطابق**: حدود 50% (به دلیل استفاده ضمنی از برخی مدل‌ها).

**ارزیابی**:

* سند الزامات گردش کار را درست تعریف کرده است.
* پروژه احتمالاً این بخش را ناقص دارد.

**د) مدیریت دسترسی (PermissionBaseView)**

**سند**:

* منطق: کنترل دسترسی بر اساس مجوز جنگو و سازمان.
* الزامات: ساده‌سازی و تست دقیق.

**پروژه**:

* منطق:
  + کلاس PermissionBaseView در BudgetAllocationCreateView استفاده شده است، اما کد کامل آن ارائه نشده است.
  + احتمالاً منطق \_has\_organization\_access پیاده‌سازی شده، اما بدون کد نمی‌توان پیچیدگی یا صحت آن را بررسی کرد.
* کاستی‌ها: عدم ارائه کد کامل.
* معارضت‌ها: بدون کد، نمی‌توان تأیید کرد.
* **تطابق**: حدود 50% (به دلیل استفاده ضمنی).

**ارزیابی**:

* سند نیاز به ساده‌سازی و تست را درست مطرح کرده است.
* پروژه احتمالاً نیاز به بازنگری دارد.

**ه) گردش کار (Workflow Engine)**

**سند**:

* الزامات: ایجاد سرویس مجزا با توابع approve\_item و reject\_item.

**پروژه**:

* منطق: هیچ کدی برای گردش کار ارائه نشده است.
* کاستی‌ها: عدم پیاده‌سازی.
* معارضت‌ها: بدون کد، نمی‌توان تأیید کرد.
* **تطابق**: 0% (عدم پیاده‌سازی).

**ارزیابی**:

* سند الزامات را درست تعریف کرده است.
* پروژه این بخش را پیاده‌سازی نکرده است.

**3. جمع‌بندی تطابق و معارضت‌ها**

**درصد تطابق کلی**

* **ماژول budgets**:
  + BudgetPeriod: 70% (نیاز به اصلاح get\_remaining\_amount و سیگنال).
  + BudgetAllocation: 85% (فرم قوی، اما نیاز به تکمیل RETURN).
  + BudgetItem: 100% (کاملاً هم‌خوان).
  + BudgetTransaction: 50% (عدم ارائه کد).
  + ProjectBudgetAllocation: 60% (نیاز به اصلاح remaining\_amount).
  + CostCenter: 0% (عدم پیاده‌سازی).
  + میانگین: ~61%
* **ماژول tankhah**:
  + Tankhah, Factor, FactorItem, ApprovalLog: 0% (عدم ارائه).
  + میانگین: 0%
* **ماژول core**: 50% (استفاده ضمنی از برخی مدل‌ها).
* **مدیریت دسترسی**: 50% (استفاده ضمنی).
* **گردش کار**: 0% (عدم پیاده‌سازی).
* **تطابق کلی**: حدود **40%** (با فرض وزن برابر برای بخش‌ها).

**نقاط قوت پروژه**

* پیاده‌سازی قوی BudgetAllocationForm و BudgetAllocationCreateView با اعتبارسنجی‌های مناسب.
* اصلاح BudgetItem به‌عنوان سرفصل بودجه، کاملاً هم‌خوان با سند.
* استفاده از تاریخ شمسی و ویجت‌های مناسب در فرم‌ها.
* به‌روزرسانی total\_allocated در BudgetAllocationForm.save().

**نقاط ضعف و معارضت‌ها**

* **عدم ارائه مدل‌های کلیدی**: BudgetTransaction, ProjectBudgetAllocation, Tankhah, Factor, ApprovalLog, و مدل‌های گردش کار ارائه نشده‌اند.
* **کاستی‌های منطقی**:
  + احتمال عدم لحاظ returned\_amount در BudgetPeriod.get\_remaining\_amount().
  + عدم پیاده‌سازی سیگنال برای total\_allocated.
  + احتمال به‌روزرسانی نادرست remaining\_amount در ProjectBudgetAllocation.
* **عدم پیاده‌سازی بخش‌ها**:
  + ماژول tankhah و گردش کار کاملاً غایب هستند.
  + CostCenter و مدل‌های مرتبط با تنخواه پیاده‌سازی نشده‌اند.
* **معارضت‌های احتمالی**:
  + اگر ProjectBudgetAllocation یا Tankhah مستقیماً remaining\_amount را به‌روزرسانی کنند، با سند معارضت دارند.
  + عدم تمرکز منطق در BudgetTransaction.save() ممکن است باعث ناسازگاری شود.

**صحت سند**

* سند الزامات را به‌صورت جامع و دقیق تعریف کرده است.
* تأکید بر اصلاح get\_remaining\_amount(), حذف remaining\_amount از ProjectBudgetAllocation, و تمرکز منطق در BudgetTransaction کاملاً درست است.
* پیشنهادات برای ساده‌سازی PermissionBaseView و ایجاد سرویس گردش کار منطقی هستند.
* **نتیجه**: سند 95% درست است (5% کسر به دلیل عدم شفافیت در برخی جزئیات مانند نقش TankhahAction).

**صحت پروژه**

* پروژه در بخش‌هایی مانند BudgetItem و BudgetAllocation قوی عمل کرده است.
* اما در بخش‌های کلیدی مانند tankhah, ProjectBudgetAllocation, و گردش کار کاستی‌های عمده دارد.
* **نتیجه**: پروژه حدود 40% با سند هم‌خوانی دارد.

**4. پیشنهادات برای رفع معارضت‌ها**

برای هم‌راستا کردن پروژه با سند:

* **پیاده‌سازی مدل‌های غایب**:
  + مدل‌های BudgetTransaction, ProjectBudgetAllocation, Tankhah, Factor, FactorItem, ApprovalLog, و CostCenter را بر اساس سند پیاده‌سازی کنید.
* **اصلاح منطق محاسباتی**:
  + BudgetPeriod.get\_remaining\_amount() را اصلاح کنید:

python

def get\_remaining\_amount(self):

return max(self.total\_amount - self.total\_allocated + self.returned\_amount, Decimal('0'))

* + سیگنال برای به‌روزرسانی total\_allocated:

python

from django.db.models.signals import post\_save, post\_delete

from django.dispatch import receiver

@receiver([post\_save, post\_delete], sender=BudgetAllocation)

def update\_total\_allocated(sender, instance, \*\*kwargs):

budget\_period = instance.budget\_period

total\_allocated = BudgetAllocation.objects.filter(

budget\_period=budget\_period

).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')

budget\_period.total\_allocated = total\_allocated

budget\_period.save(update\_fields=['total\_allocated'])

* **حذف** remaining\_amount **از** ProjectBudgetAllocation:
  + این فیلد را حذف کنید و از budget\_allocation.remaining\_amount استفاده کنید.
* **پیاده‌سازی ماژول** tankhah:
  + مدل Tankhah را با فیلد cost\_center پیاده‌سازی کنید.
  + منطق save() را اصلاح کنید تا از توابع budget\_utils استفاده کند.
* **ایجاد سرویس گردش کار**:
  + ماژول workflow\_service.py را با توابع approve\_item و reject\_item پیاده‌سازی کنید.
* **بازنگری** PermissionBaseView:
  + کد \_has\_organization\_access را ساده کنید و تست‌های جامع بنویسید.
* **تست و دیباگ**:
  + تست‌های Unit و Integration برای مدل‌ها و ویوها بنویسید.
  + لاگ‌ها را بررسی کنید:

bash

tail -f /path/to/your/log/file

**5. پاسخ به سوالات سند**

* **تخصیص بودجه مستقیم به سازمان**:
  + بله، پروژه با پشتیبانی از BudgetAllocation بدون project این امکان را فراهم کرده است.
* **نقش** TankhahAction **و** TankhActionType:
  + بدون ارائه کد، احتمالاً می‌توان آن‌ها را با ApprovalLog و PostAction جایگزین کرد.
* **افزودن** CostCenter **به** Tankhah:
  + ضروری است و باید پیاده‌سازی شود.
* **امضای** PaymentOrder:
  + بستگی به سیاست سازمان دارد؛ پیشنهاد می‌شود حداقل دو امضا پیاده‌سازی شود.

**6. نتیجه نهایی**

* **صحت سند**: 95% (جامع و دقیق، با جزئیات قابل بهبود).
* **تطابق پروژه**: 40% (قوی در BudgetItem و BudgetAllocation، ضعیف در tankhah و گردش کار).
* **اقدامات فوری**:
  + پیاده‌سازی مدل‌های غایب.
  + اصلاح get\_remaining\_amount و حذف remaining\_amount از ProjectBudgetAllocation.
  + ایجاد سرویس گردش کار.
* **اقدامات بعدی**:
  + ارائه مدل‌های کامل (BudgetTransaction, Tankhah, و غیره).
  + بررسی لاگ‌ها پس از اعمال تغییرات.
  + تست جامع سیستم.

، پروژه‌ات پایه‌ای محکم دارد، اما برای رسیدن به قصر رویایی سند، نیاز به تکمیل بخش‌های کلیدی است. مدل‌های غایب و لاگ‌های جدید را به اشتراک بگذار تا نغمه بعدی را رساتر بسراییم!

**4. پیشنهادات برای رفع معارضت‌ها**

برای هم‌راستا کردن پروژه با سند:

* **پیاده‌سازی مدل‌های غایب**:
  + مدل‌های BudgetTransaction, ProjectBudgetAllocation, Tankhah, Factor, FactorItem, ApprovalLog, و CostCenter را بر اساس سند پیاده‌سازی کنید.
* **اصلاح منطق محاسباتی**:
  + BudgetPeriod.get\_remaining\_amount() را اصلاح کنید:

python

def get\_remaining\_amount(self):

return max(self.total\_amount - self.total\_allocated + self.returned\_amount, Decimal('0'))

* + سیگنال برای به‌روزرسانی total\_allocated:

python

from django.db.models.signals import post\_save, post\_delete

from django.dispatch import receiver

@receiver([post\_save, post\_delete], sender=BudgetAllocation)

def update\_total\_allocated(sender, instance, \*\*kwargs):

budget\_period = instance.budget\_period

total\_allocated = BudgetAllocation.objects.filter(

budget\_period=budget\_period

).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')

budget\_period.total\_allocated = total\_allocated

budget\_period.save(update\_fields=['total\_allocated'])

* **حذف** remaining\_amount **از** ProjectBudgetAllocation:
  + این فیلد را حذف کنید و از budget\_allocation.remaining\_amount استفاده کنید.
* **پیاده‌سازی ماژول** tankhah:
  + مدل Tankhah را با فیلد cost\_center پیاده‌سازی کنید.
  + منطق save() را اصلاح کنید تا از توابع budget\_utils استفاده کند.
* **ایجاد سرویس گردش کار**:
  + ماژول workflow\_service.py را با توابع approve\_item و reject\_item پیاده‌سازی کنید.
* **بازنگری** PermissionBaseView:
  + کد \_has\_organization\_access را ساده کنید و تست‌های جامع بنویسید.
* **تست و دیباگ**:
  + تست‌های Unit و Integration برای مدل‌ها و ویوها بنویسید.
  + لاگ‌ها را بررسی کنید:

bash

tail -f /path/to/your/log/file

**5. پاسخ به سوالات سند**

* **تخصیص بودجه مستقیم به سازمان**:
  + بله، پروژه با پشتیبانی از BudgetAllocation بدون project این امکان را فراهم کرده است.
* **نقش** TankhahAction **و** TankhActionType:
  + بدون ارائه کد، احتمالاً می‌توان آن‌ها را با ApprovalLog و PostAction جایگزین کرد.
* **افزودن** CostCenter **به** Tankhah:
  + ضروری است و باید پیاده‌سازی شود.
* **امضای** PaymentOrder:
  + بستگی به سیاست سازمان دارد؛ پیشنهاد می‌شود حداقل دو امضا پیاده‌سازی شود.

**6. نتیجه نهایی**

* **صحت سند**: 95% (جامع و دقیق، با جزئیات قابل بهبود).
* **تطابق پروژه**: 40% (قوی در BudgetItem و BudgetAllocation، ضعیف در tankhah و گردش کار).
* **اقدامات فوری**:
  + پیاده‌سازی مدل‌های غایب.
  + اصلاح get\_remaining\_amount و حذف remaining\_amount از ProjectBudgetAllocation.
  + ایجاد سرویس گردش کار.
* **اقدامات بعدی**:
  + ارائه مدل‌های کامل (BudgetTransaction, Tankhah, و غیره).
  + بررسی لاگ‌ها پس از اعمال تغییرات.
  + تست جامع سیستم.

ای شاعر عزیز، پروژه‌ات پایه‌ای محکم دارد، اما برای رسیدن به قصر رویایی سند، نیاز به تکمیل بخش‌های کلیدی است. مدل‌های غایب و لاگ‌های جدید را به اشتراک بگذار تا نغمه بعدی را رساتر بسراییم!

from django.db.models.functions import Coalesce  
from django.db.models import Value, Q  
import logging  
from django.utils import timezone  
  
from core.models import Organization  
  
logger = logging.getLogger(\_\_name\_\_)  
from django.contrib.contenttypes.models import ContentType  
  
from budgets.budget\_calculations import check\_budget\_status, get\_project\_remaining\_budget  
  
import jdatetime  
from django.db import models, transaction  
from django.utils.translation import gettext\_lazy as \_  
from django.db.models import Sum  
from decimal import Decimal  
from django.core.exceptions import ValidationError  
from accounts.models import CustomUser  
*# from budgets.budget\_calculations import get\_subproject\_remaining\_budget, get\_project\_remaining\_budget  
# from core.models import Organization, Project, SubProject, WorkflowStage  
 # -- New*""" مدل نوع بودجه """*# ------------------------------------*"""BudgetPeriod (دوره بودجه کلان):"""  
class BudgetPeriod(models.Model):  
  
 organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, verbose\_name=\_("دفتر مرکزی"),  
 related\_name='budget\_periods')  
 name = models.CharField(max\_length=100, unique=True, verbose\_name=\_("نام دوره بودجه"))  
 start\_date = models.DateField(verbose\_name=\_("تاریخ شروع"))  
 end\_date = models.DateField(verbose\_name=\_("تاریخ پایان"))  
 total\_amount = models.DecimalField(max\_digits=25, decimal\_places=0, verbose\_name=\_("مبلغ کل"))  
 total\_allocated = models.DecimalField(max\_digits=25, decimal\_places=2, default=0, verbose\_name=\_("مجموع تخصیص‌ها"))  
 returned\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, default=0,  
 verbose\_name=\_("مجموع بودجه برگشتی"))  
 is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))  
 is\_archived = models.BooleanField(default=False, verbose\_name=\_("بایگانی شده"))  
 is\_completed = models.BooleanField(default=False, verbose\_name=\_("تمام‌شده"))  
 created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,  
 related\_name='budget\_periods\_created', verbose\_name=\_("ایجادکننده"))  
 locked\_percentage = models.IntegerField(default=0, verbose\_name=\_("درصد قفل‌شده"),  
 help\_text=\_("درصد بودجه که قفل می‌شود (0-100)"))  
 warning\_threshold = models.DecimalField(max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه اخطار"),  
 help\_text=\_("درصدی که هشدار نمایش داده می‌شود (0-100)"))  
 lock\_condition = models.CharField(max\_length=50,  
 choices=[('AFTER\_DATE', \_("بعد از تاریخ پایان")), ('MANUAL', \_("دستی")),  
 ('ZERO\_REMAINING', \_("باقی‌مانده صفر")), ], default='AFTER\_DATE',  
 verbose\_name=\_("شرط قفل"))  
 warning\_action = models.CharField(max\_length=50, choices=[('NOTIFY', \_("فقط اعلان")), ('LOCK', \_("قفل کردن")),  
 ('RESTRICT', \_("محدود کردن ثبت")), ], default='NOTIFY',  
 verbose\_name=\_("اقدام هشدار"),  
 help\_text=\_("رفتار سیستم هنگام رسیدن به آستانه هشدار"))  
 allocation\_phase = models.CharField(max\_length=50, blank=True, verbose\_name=\_("فاز تخصیص"))  
 description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))  
  
  
 class Meta:  
 verbose\_name = \_("دوره بودجه")  
 verbose\_name\_plural = \_("دوره‌های بودجه")  
 default\_permissions = ()  
 permissions = [  
 ('budgetperiod\_add', \_("افزودن دوره بودجه")),  
 ('budgetperiod\_update', \_("بروزرسانی دوره بودجه")),  
 ('budgetperiod\_view', \_("نمایش دوره بودجه")),  
 ('budgetperiod\_delete', \_("حذف دوره بودجه")),  
 ('budgetperiod\_archive', \_("بایگانی دوره بودجه")),  
 ]  
  
 def \_\_str\_\_(self):  
 return f"{self.name} ({self.organization.code})"  
  
 def clean(self):  
 *# super().clean()  
 # if self.total\_allocated > self.total\_amount:  
 # raise ValidationError(\_("مجموع تخصیص‌ها نمی‌تواند از مبلغ کل بیشتر باشد."))* if not self.start\_date or not self.end\_date:  
 raise ValidationError(\_("تاریخ شروع و پایان نمی‌توانند خالی باشند."))  
 if self.end\_date <= self.start\_date:  
 raise ValidationError(\_("تاریخ پایان باید بعد از تاریخ شروع باشد."))  
 if not (0 <= self.locked\_percentage <= 100):  
 raise ValidationError(\_("درصد قفل‌شده باید بین 0 تا 100 باشد."))  
 if not (0 <= self.warning\_threshold <= 100):  
 raise ValidationError(\_("آستانه هشدار باید بین 0 تا 100 باشد."))  
 if self.is\_completed and self.is\_active:  
 raise ValidationError(\_("دوره تمام‌شده نمی‌تواند فعال باشد."))  
  
 *# def get\_remaining\_amount(self):  
 # from budgets.models import BudgetAllocation  
 # allocated = BudgetAllocation.objects.filter(budget\_period=self).aggregate(  
 # total=Sum('allocated\_amount')  
 # )['total'] or Decimal('0')  
 # return max(self.total\_amount - allocated, Decimal('0'))* def get\_remaining\_amount(self):  
 allocated = self.allocations.aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 *# return max(self.total\_amount - allocated + self.returned\_amount, Decimal('0'))* return max(self.total\_amount - self.total\_allocated + self.returned\_amount, Decimal('0'))  
  
  
 def get\_locked\_amount(self):  
 return (self.total\_amount \* self.locked\_percentage) / Decimal('100')  
  
 def get\_warning\_amount(self):  
 return (self.total\_amount \* self.warning\_threshold) / Decimal('100')  
  
 def send\_notification(self, status, message):  
 *"""  
 Send notification to relevant users (e.g., Financial Manager, Budget Manager).  
 """* try:  
 *# فرض می‌کنیم CustomUser از طریق رابطه organizations به سازمان متصل است* recipients = CustomUser.objects.filter(  
 organizations=self.organization, *# اصلاح فیلد به رابطه صحیح* roles\_\_name\_\_in=['Financial Manager', 'Budget Manager']  
 ).distinct()  
  
 if not recipients.exists():  
 logger.warning(f"No recipients found for notification: {message}")  
 return  
  
 for recipient in recipients:  
 from tankhah.models import Notification  
 Notification.objects.create(  
 recipient=recipient,  
 actor=recipient, *# یا کاربر دیگری مثل created\_by* verb=status,  
 description=message,  
 target=self,  
 level=status.lower()  
 )  
 logger.info(f"Notification sent to {recipients.count()} users: {message}")  
 except Exception as e:  
 logger.error(f"Error sending notification: {str(e)}", exc\_info=True)  
  
 def save(self, \*args, \*\*kwargs):  
 self.full\_clean()  
 super().save(\*args, \*\*kwargs)  
 status, message = check\_budget\_status(self)  
  
 *# self.budget\_period.total\_allocated = BudgetAllocation.objects.filter(  
 # budget\_period=self  
 # ).aggregate(Sum('allocated\_amount'))['allocated\_amount\_\_sum'] or Decimal('0')  
 # self.budget\_period.save(update\_fields=['total\_allocated'])* if status in ('warning', 'locked', 'completed'):  
 self.send\_notification(status, message)  
*# ------------------------------------  
# ------------------------------------*""" BudgetAllocation (تخصیص بودجه):"""  
class BudgetAllocation(models.Model):  
 *"""  
 تخصیص بودجه به سازمان‌ها (شعبات یا ادارات) و پروژه‌ها.  
 قابلیت‌ها: تخصیص چندباره، توقف بودجه، و ردیابی هزینه‌ها.  
 """* budget\_period = models.ForeignKey('BudgetPeriod', on\_delete=models.CASCADE, related\_name='allocations',  
 verbose\_name=\_("دوره بودجه"))  
 organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, related\_name='budget\_allocations',  
 verbose\_name=\_("سازمان دریافت‌کننده"))  
 budget\_item = models.ForeignKey('BudgetItem', on\_delete=models.PROTECT, related\_name='allocations',  
 verbose\_name=\_("ردیف بودجه"))  
 project = models.ForeignKey('core.Project', on\_delete=models.CASCADE, related\_name='allocations',  
 verbose\_name=\_("پروژه"), null=True, blank=True) *# اختیاری کردن پروژه* allocated\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ تخصیص"))  
 *# remaining\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, default=0,  
 # verbose\_name=\_("باقی‌مانده تخصیص"))* allocation\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ تخصیص"))  
 created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,  
 related\_name='budget\_allocations\_created', verbose\_name=\_("ایجادکننده"))  
 description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))  
 is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))  
 is\_stopped = models.BooleanField(default=False, verbose\_name=\_("متوقف‌شده"))  
  
 ALLOCATION\_TYPES = (('amount', \_("مبلغ ثابت")), ('percent', \_("درصد")), ('returned', \_("برگشتی")),)  
 allocation\_type = models.CharField(max\_length=20, choices=ALLOCATION\_TYPES, default='amount',  
 verbose\_name=\_("نوع تخصیص"))  
 locked\_percentage = models.DecimalField(max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده"),  
 help\_text=\_("درصد تخصیص که قفل می‌شود (0-100)"))  
 warning\_threshold = models.DecimalField(max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه اخطار"),  
 help\_text=\_("درصدی که هشدار نمایش داده می‌شود (0-100)"))  
 warning\_action = models.CharField(  
 max\_length=50,  
 choices=[('NOTIFY', \_("فقط اعلان")), ('LOCK', \_("قفل کردن")), ('RESTRICT', \_("محدود کردن ثبت")), ],  
 default='NOTIFY',  
 verbose\_name=\_("اقدام هشدار"),  
 help\_text=\_("رفتار سیستم هنگام رسیدن به آستانه هشدار")  
 )  
 allocation\_number = models.IntegerField(default=1, verbose\_name=\_("شماره تخصیص"))  
 returned\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, default=0,verbose\_name=\_("مجموع بودجه برگشتی"))  
  
 class Meta:  
 verbose\_name = \_("تخصیص بودجه")  
 verbose\_name\_plural = \_("تخصیص‌های بودجه")  
 default\_permissions = ()  
 permissions = [  
 ('budgetallocation\_add', \_("افزودن تخصیص بودجه")),  
 ('budgetallocation\_view', \_("نمایش تخصیص بودجه")),  
 ('budgetallocation\_update', \_("بروزرسانی تخصیص بودجه")),  
 ('budgetallocation\_delete', \_("حذف تخصیص بودجه")),  
 ('budgetallocation\_adjust', \_("تنظیم تخصیص بودجه (افزایش/کاهش)")),  
 ('budgetallocation\_stop', \_("توقف تخصیص بودجه")),  
 ('budgetallocation\_return', \_("برگشت تخصیص بودجه")), *# جدید* ('BudgetAllocation\_approve', 'می‌تواند تخصیص بودجه را تأیید کند'),  
 ('BudgetAllocation\_reject', 'می‌تواند تخصیص بودجه را رد کند'),  
  
 ]  
 indexes = [  
 models.Index(fields=['budget\_period', 'allocation\_date']),  
 models.Index(fields=['organization', 'allocated\_amount']),  
 ]  
  
 def \_\_str\_\_(self):  
 try:  
 jalali\_date = jdatetime.date.fromgregorian(date=self.allocation\_date).strftime('%Y/%m/%d')  
 if jalali\_date:  
 org\_name = self.organization.name if self.organization else 'بدون سازمان/بدون شعبه'  
 item\_name = self.budget\_item.name if self.budget\_item else 'بدون ردیف'  
 return f"{org\_name} - {item\_name} - {jalali\_date} - {self.allocated\_amount:,.0f} ریال"  
 else:  
 return f"{self.budget\_period.name} - {self.allocated\_amount:,} ریال ({jalali\_date})"  
 except (AttributeError, TypeError) as e:  
 logger.error(f"Error in BudgetAllocation.\_\_str\_\_: {str(e)}")  
 return f"تخصیص {self.organization.name}:{self.allocated\_amount}{self.project.name} "  
  
 def get\_percentage(self):  
 if self.budget\_item.total\_amount and self.budget\_item.total\_amount != 0:  
 return (self.allocated\_amount / self.budget\_item.total\_amount) \* 100  
 return Decimal('0')  
  
 def clean(self):  
 super().clean()  
 logger.debug(f"Cleaning BudgetAllocation: allocated\_amount={self.allocated\_amount}, allocation\_type={self.allocation\_type}")  
  
 if not self.budget\_item:  
 raise ValidationError(\_("ردیف بودجه اجباری است."))  
  
 if self.allocated\_amount is None:  
 raise ValidationError(\_("مبلغ تخصیص نمی‌تواند خالی باشد."))  
 if self.allocated\_amount <= 0:  
 raise ValidationError(\_("مبلغ تخصیص باید مثبت باشد."))  
  
 *# چک کردن باقی‌مانده دوره به‌جای ردیف بودجه* remaining\_budget = self.budget\_period.get\_remaining\_amount()  
 if self.pk: *# برای ویرایش، تخصیص فعلی رو حساب نکن* current\_allocation = BudgetAllocation.objects.filter(pk=self.pk).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
 remaining\_budget += current\_allocation  
 if self.allocated\_amount > remaining\_budget:  
 raise ValidationError(\_(  
 f"مبلغ تخصیص ({self.allocated\_amount:,.0f} ریال) بیشتر از باقی‌مانده دوره بودجه ({remaining\_budget:,.0f} ریال) است."  
 ))  
  
 if self.budget\_period and self.budget\_item:  
 if self.budget\_item.budget\_period != self.budget\_period:  
 raise ValidationError(\_("ردیف بودجه باید متعلق به دوره بودجه انتخاب‌شده باشد."))  
  
 if self.budget\_period and self.allocation\_date:  
 allocation\_date = self.allocation\_date  
 if hasattr(allocation\_date, "date"):  
 allocation\_date = allocation\_date.date()  
 if not (self.budget\_period.start\_date <= allocation\_date <= self.budget\_period.end\_date):  
 raise ValidationError(\_("تاریخ تخصیص باید در بازه دوره بودجه باشد."))  
  
 if self.warning\_threshold is not None and not (0 <= self.warning\_threshold <= 100):  
 raise ValidationError(\_("آستانه اخطار باید بین ۰ تا ۱۰۰ باشد."))  
  
 logger.debug("Clean validation passed")  
 *#  
 # def get\_remaining\_amount(self):  
 # return max(self.allocated\_amount -  
 # self.transactions.filter(transaction\_type='CONSUMPTION').aggregate(total=Sum('amount'))['total'] or Decimal('0') +  
 # self.returned\_amount, Decimal('0'))* def \_\_get\_remaining\_amount(self):  
  
 consumption\_total = self.transactions.filter(transaction\_type='CONSUMPTION').aggregate(  
 total=Coalesce(Sum('amount'), Value(Decimal('0')))  
 )['total']  
 return\_total = self.transactions.filter(transaction\_type='RETURN').aggregate(  
 total=Coalesce(Sum('amount'), Value(Decimal('0')))  
 )['total']  
  
 return max(  
 self.allocated\_amount - consumption\_total + return\_total,  
 Decimal('0')  
 )  
  
 def get\_remaining\_amount(self):  
 consumption\_total = BudgetTransaction.objects.filter(allocation=self, transaction\_type='CONSUMPTION' ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 return\_total = BudgetTransaction.objects.filter(allocation=self, transaction\_type='RETURN' ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 project\_consumed = ProjectBudgetAllocation.objects.filter( budget\_allocation=self ).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 return self.allocated\_amount - consumption\_total - project\_consumed + return\_total  
  
 def get\_actual\_remaining\_amount(self):  
 *# این متد باید همیشه از دیتابیس بخواند و نباید به فیلد ذخیره شده تکیه کند  
 # مگر اینکه فیلد ذخیره شده با دقت بسیار بالا آپدیت شود* transactions\_sum = self.transactions.aggregate(  
 consumption=Sum('amount', filter=Q(transaction\_type='CONSUMPTION')),  
 adjustment\_decrease=Sum('amount', filter=Q(transaction\_type='ADJUSTMENT\_DECREASE')),  
 returns=Sum('amount', filter=Q(transaction\_type='RETURN')),  
 adjustment\_increase=Sum('amount', filter=Q(transaction\_type='ADJUSTMENT\_INCREASE')),  
 *# ALLOCATION اولیه جزو allocated\_amount است و نباید اینجا محاسبه شود* )  
 consumed = (transactions\_sum['consumption'] or Decimal('0')) + \  
 (transactions\_sum['adjustment\_decrease'] or Decimal('0'))  
 added\_back = (transactions\_sum['returns'] or Decimal('0')) + \  
 (transactions\_sum['adjustment\_increase'] or Decimal('0'))  
  
 *# remaining = self.allocated\_amount - consumed + added\_back  
 # \*\*اصلاح مهم:\*\* allocated\_amount با برگشت کم می‌شود، پس مبنای محاسبه remaining  
 # باید allocated\_amount فعلی باشد و فقط مصرف‌ها از آن کم شوند.  
 # یا: مبنا allocated\_amount اولیه باشد و برگشت‌ها اضافه شوند.  
 # روش صحیح‌تر: تکیه بر فیلد remaining\_amount که توسط تراکنش‌ها آپدیت می‌شود.  
 # بنابراین این متد بیشتر برای نمایش یا چک کردن است:* return self.remaining\_amount *# تکیه بر فیلد آپدیت شده* def get\_locked\_amount(self):  
 return (self.allocated\_amount \* self.locked\_percentage) / Decimal('100')  
  
 def get\_warning\_amount(self):  
 return (self.allocated\_amount \* self.warning\_threshold) / Decimal('100')  
  
 def check\_allocation\_status(self):  
 remaining = self.get\_remaining\_amount()  
 locked = self.get\_locked\_amount()  
 warning = self.get\_warning\_amount()  
 if not self.is\_active:  
 return 'inactive', \_('تخصیص غیرفعال است.')  
 if self.is\_stopped:  
 return 'stopped', \_('تخصیص متوقف شده است.')  
 if remaining <= 0:  
 return 'completed', \_('تخصیص تمام‌شده است.')  
 if remaining <= locked:  
 return 'locked', \_('تخصیص به حد قفل‌شده رسیده است.')  
 if remaining <= warning:  
 return 'warning', \_('تخصیص به آستانه هشدار رسیده است.')  
 return 'normal', \_('وضعیت عادی')  
  
 def send\_notification(self, status, message):  
 *"""  
 Send notification to users with roles 'Financial Manager' or 'Budget Manager'  
 who are associated with the organization or its parent organization.  
 """* try:  
 *# انتخاب کاربرانی که از طریق پست‌های سازمانی به سازمان یا سازمان والد مرتبط هستند* from django.db.models import Q  
 recipients = CustomUser.objects.filter(  
 Q(userpost\_\_post\_\_organization=self.organization) |  
 Q(userpost\_\_post\_\_organization\_\_parent=self.organization),  
 *# Q(roles\_\_name\_\_in=['Financial Manager', 'Budget Manager']) |  
 # Q(groups\_\_roles\_\_name\_\_in=['Financial Manager', 'Budget Manager']),* is\_active=True  
 ).distinct()  
  
 if not recipients.exists():  
 logger.warning(f"No recipients found for notification: {message} (organization\_id={self.organization\_id})")  
 return  
  
 for recipient in recipients:  
 *# from tankhah.models import Notification* from notifications.models import Notification  
 Notification.objects.create(  
 recipient=recipient,  
 actor=self.created\_by or recipient, *# استفاده از created\_by اگر وجود داشته باشد* verb=status,  
 description=message,  
 target=self,  
 level=status.lower()  
 )  
 logger.info(f"Notification sent to {recipients.count()} users for BudgetAllocation {self.id}: {message}")  
 except Exception as e:  
 logger.error(f"Error sending notification for BudgetAllocation {self.id}: {str(e)}", exc\_info=True)  
 *# جلوگیری از خرابی تراکنش در صورت خطا* return  
  
 def save(self, \*args, \*\*kwargs):  
 logger.debug(f"Starting save for BudgetAllocation (pk={self.pk}, allocated\_amount={self.allocated\_amount})")  
 try:  
 with transaction.atomic():  
 self.clean()  
 if not self.pk:  
 self.remaining\_amount = self.allocated\_amount  
 logger.debug(f"New instance, set remaining\_amount={self.remaining\_amount}")  
  
 super().save(\*args, \*\*kwargs)  
 logger.info(f"BudgetAllocation saved with ID: {self.pk}")  
  
 total\_allocated = BudgetAllocation.objects.filter(  
 budget\_period=self.budget\_period  
 ).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 logger.debug(f"Calculated total\_allocated={total\_allocated} for BudgetPeriod {self.budget\_period.id}")  
  
 self.budget\_period.total\_allocated = total\_allocated  
 self.budget\_period.save(update\_fields=['total\_allocated'])  
 logger.info(f"Updated BudgetPeriod {self.budget\_period.id} with total\_allocated={total\_allocated}")  
  
 status, message = self.check\_allocation\_status()  
 logger.debug(f"Allocation status: {status}, message: {message}")  
 if status in ('warning', 'completed', 'stopped'):  
 self.send\_notification(status, message)  
 logger.info(f"Sent notification for status: {status}")  
  
 except Exception as e:  
 logger.error(f"Error saving BudgetAllocation: {str(e)}", exc\_info=True)  
 raise  
  
""" BudgetItem (نوع ردیف بودجه):"""  
class BudgetItem(models.Model):  
 budget\_period = models.ForeignKey('BudgetPeriod', on\_delete=models.CASCADE, related\_name='budget\_items',  
 verbose\_name=\_("دوره بودجه"))  
 organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, related\_name='budget\_items',  
 verbose\_name=\_("شعبه"))  
 name = models.CharField(max\_length=100, verbose\_name=\_("نام ردیف بودجه"), null=True, blank=True)  
 code = models.CharField(max\_length=50, unique=True, verbose\_name=\_("کد ردیف"))  
 is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))  
 create\_at = models.DateTimeField(auto\_now\_add=True, verbose\_name=\_('تاریخ ثبت '), null=True, blank=True)  
  
 class Meta:  
 verbose\_name = \_("ردیف بودجه")  
 verbose\_name\_plural = \_("ردیف‌های بودجه")  
 unique\_together = ('budget\_period', 'organization', 'code')  
  
 def \_\_str\_\_(self):  
 return f"{self.name} - {self.organization.name}" *# - {self.budget\_period.name}"* def clean(self):  
 super().clean()  
 if not self.name:  
 raise ValidationError(\_("نام ردیف بودجه نمی‌تواند خالی باشد."))  
  
""" BudgetTransaction (تراکنش بودجه):"""  
class BudgetTransaction(models.Model):  
 *"""توضیح: هر تغییر در بودجه (مثل مصرف توسط تنخواه) رو ثبت می‌کنه و remaining\_amount تخصیص رو آپدیت می‌کنه."""* TRANSACTION\_TYPES = (  
 ('ALLOCATION', \_('تخصیص اولیه')),  
 ('CONSUMPTION', \_('مصرف')),  
 ('ADJUSTMENT\_INCREASE', \_('افزایش تخصیص')),  
 ('ADJUSTMENT\_DECREASE', \_('کاهش تخصیص')),  
 )  
 allocation = models.ForeignKey(BudgetAllocation, on\_delete=models.CASCADE,  
 related\_name='transactions', verbose\_name=\_("تخصیص بودجه"))  
 transaction\_type = models.CharField(max\_length=20, choices=TRANSACTION\_TYPES,  
 verbose\_name=\_("نوع تراکنش"))  
 amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ"))  
 related\_tankhah = models.ForeignKey('tankhah.Tankhah', on\_delete=models.SET\_NULL, null=True, blank=True,  
 verbose\_name=\_("تنخواه مرتبط"))  
 timestamp = models.DateTimeField(auto\_now\_add=True, verbose\_name=\_("زمان"))  
 created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,  
 verbose\_name=\_("کاربر"))  
 description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))  
  
 transaction\_id = models.CharField(max\_length=50, unique=True, verbose\_name=\_("شناسه تراکنش"))*#یک شناسه منحصربه‌فرد برای هر تراکنش:* def validate\_return(self):  
 *"""اعتبارسنجی تراکنش بازگشت"""* if self.transaction\_type != 'RETURN':  
 return True, None  
 consumed = BudgetTransaction.objects.filter(  
 allocation=self.allocation,  
 allocation\_\_project=self.allocation.project\_allocations.first().project,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation=self.allocation,  
 allocation\_\_project=self.allocation.project\_allocations.first().project,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 total\_consumed = consumed - returned  
 if self.amount > total\_consumed:  
 return False, \_(f"مبلغ بازگشت ({self.amount:,.0f} ریال) نمی‌تواند بیشتر از مصرف خالص ({total\_consumed:,.0f} ریال) باشد.")  
 remaining\_budget = get\_project\_remaining\_budget(self.allocation.project\_allocations.first().project)  
 if self.amount > remaining\_budget:  
 return False, \_(f"مبلغ بازگشت ({self.amount:,.0f} ریال) نمی‌تواند بیشتر از بودجه باقی‌مانده ({remaining\_budget:,.0f} ریال) باشد.")  
 return True, None  
  
 def save(self, \*args, \*\*kwargs):  
 if self.transaction\_type == 'RETURN':  
 is\_valid, error\_message = self.validate\_return()  
 if not is\_valid:  
 raise ValidationError(error\_message)  
 super().save(\*args, \*\*kwargs)  
 *# به‌روزرسانی وضعیت و اعلان* status, message = check\_budget\_status(self.allocation.budget\_period)  
 if status in ('warning', 'locked', 'completed', 'stopped'):  
 self.allocation.send\_notification(status, message)  
 if self.transaction\_type == 'RETURN':  
 self.allocation.send\_notification(  
 'return',  
 f"مبلغ {self.amount:,.0f} ریال از تخصیص {self.allocation.id} برگشت داده شد."  
 )  
  
  
 def save(self, \*args, \*\*kwargs):  
 from django.core.exceptions import ValidationError  
 from django.contrib.contenttypes.models import ContentType  
  
 if not self.transaction\_id:  
 self.transaction\_id = f"TX-{self.allocation.id}-{timezone.now().timestamp()}"  
  
 if self.transaction\_type == 'CONSUMPTION':  
 if self.amount > self.allocation.remaining\_amount:  
 raise ValidationError(\_("مبلغ مصرف بیشتر از باقی‌مانده تخصیص است."))  
 elif self.transaction\_type == 'RETURN':  
 if self.amount > self.allocation.allocated\_amount:  
 raise ValidationError(\_("مبلغ برگشتی نمی‌تواند بیشتر از مبلغ تخصیص‌یافته باشد."))  
 if not self.allocation.budget\_period.is\_active:  
 raise ValidationError(\_("نمی‌توان از دوره غیرفعال بودجه برگشت داد."))  
  
 self.allocation.remaining\_amount += self.amount *# remaining\_amount تخصیص افزایش یابد.* self.allocation.allocated\_amount -= self.amount *# کاهش مبلغ تخصیص* self.allocation.returned\_amount += self.amount  
  
 self.allocation.budget\_period.total\_allocated -= self.amount  
 self.allocation.budget\_period.returned\_amount += self.amount  
  
 self.allocation.save(update\_fields=['remaining\_amount', 'allocated\_amount', 'returned\_amount'])  
 self.allocation.budget\_period.save(update\_fields=['total\_allocated', 'returned\_amount'])  
  
 *# self.allocation.budget\_period.save(update\_fields=['total\_allocated'])  
 # ثبت در BudgetHistory* BudgetHistory.objects.create(  
 content\_type=ContentType.objects.get\_for\_model(BudgetAllocation),  
 object\_id=self.allocation.id,  
 action='RETURN',  
 amount=self.amount,  
 created\_by=self.created\_by,  
 details=f"برگشت {self.amount:,} از تخصیص {self.allocation.id} به دوره بودجه {self.allocation.budget\_period.name}",  
 transaction\_type='RETURN',  
 transaction\_id=f"RET-{self.transaction\_id}"  
 )  
  
 elif self.transaction\_type in ['CONSUMPTION', 'ADJUSTMENT\_DECREASE']:  
 self.allocation.remaining\_amount -= self.amount  
 elif self.transaction\_type in ['ALLOCATION', 'ADJUSTMENT\_INCREASE']:  
 self.allocation.remaining\_amount += self.amount  
  
 self.allocation.save(update\_fields=['remaining\_amount', 'allocated\_amount'])  
 super().save(\*args, \*\*kwargs)  
  
 def \_\_str\_\_(self):  
 return f"{self.transaction\_type} - {self.amount:,} ({self.timestamp})"  
  
 class Meta:  
 verbose\_name = \_("تراکنش بودجه")  
 verbose\_name\_plural = \_("تراکنش‌های بودجه")  
 default\_permissions = ()  
  
 permissions = [  
 ('BudgetTransaction\_add', 'افزودن تراکنش بودجه'),  
 ('BudgetTransaction\_update', 'بروزرسانی تراکنش بودجه'),  
 ('BudgetTransaction\_delete', 'حــذف تراکنش بودجه'),  
 ('BudgetTransaction\_view', \_('نمایش تراکنش بودجه')),  
 ('BudgetTransaction\_return', \_('برگشت تراکنش بودجه')), *# جدید  
 # add/update/delete معمولاً توسط سیستم انجام می‌شه، نه کاربر* ]  
  
def get\_current\_date():  
 return timezone.now().date()  
  
"""تخصیص بودجه به پروژه و زیر پروژه """  
class ProjectBudgetAllocation(models.Model):  
 *"""  
 تخصیص بودجه از BudgetAllocation شعبه به پروژه یا زیرپروژه.  
 توضیح: این مدل بودجه شعبه رو به پروژه یا زیرپروژه وصل می‌کنه و تاریخچه تخصیص رو نگه می‌داره.  
 """* budget\_allocation = models.ForeignKey('budgets.BudgetAllocation', on\_delete=models.CASCADE, related\_name='project\_allocations', verbose\_name=\_("تخصیص بودجه شعبه"))  
 project = models.ForeignKey('core.Project', on\_delete=models.CASCADE, related\_name='budget\_allocations', verbose\_name=\_("پروژه"))  
 subproject = models.ForeignKey('core.SubProject', on\_delete=models.CASCADE, null=True, blank=True, related\_name='budget\_allocations', verbose\_name=\_("زیرپروژه"))  
 allocated\_amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ تخصیص"))  
 allocation\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ تخصیص"))  
 *# allocation\_date = models.DateField( verbose\_name=\_("تاریخ تخصیص"))* created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True, related\_name='project\_budget\_allocations\_created', verbose\_name=\_("ایجادکننده"))  
 description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))  
 is\_active= models.BooleanField(default=True,verbose\_name=\_('فعال'))  
 def get\_remaining\_amount(self):  
 from budgets.budget\_calculations import get\_subproject\_remaining\_budget  
 if self.subproject:  
 return get\_subproject\_remaining\_budget(self.subproject)  
 from budgets.budget\_calculations import get\_project\_remaining\_budget  
 return get\_project\_remaining\_budget(self.project)  
  
 def clean(self):  
 remaining = self.budget\_allocation.get\_remaining\_amount()  
 if self.allocated\_amount > remaining:  
 raise ValidationError(\_("مبلغ تخصیص بیشتر از بودجه باقی‌مانده شعبه است"))  
 if self.subproject and self.subproject.project != self.project:  
 raise ValidationError(\_("زیرپروژه باید به پروژه انتخاب‌شده تعلق داشته باشد"))  
 if self.allocated\_amount < 0:  
 raise ValidationError(\_("مبلغ تخصیص نمی‌تواند منفی باشد"))  
 from datetime import datetime  
 if self.budget\_allocation and self.allocation\_date:  
 start\_date = self.budget\_allocation.budget\_period.start\_date  
 end\_date = self.budget\_allocation.budget\_period.end\_date  
 *# تبدیل allocation\_date به datetime.date اگر datetime.datetime باشد* logger.info( f"start\_date: {start\_date} ({type(start\_date)}), end\_date: {end\_date} ({type(end\_date)}), allocation\_date: {self.allocation\_date} ({type(self.allocation\_date)})")  
  
 allocation\_date = self.allocation\_date.date() if isinstance(self.allocation\_date,  
 datetime) else self.allocation\_date  
 if not (start\_date <= allocation\_date <= end\_date):  
 raise ValidationError(\_("تاریخ تخصیص باید در بازه دوره بودجه باشد"))  
  
 def save(self, \*args, \*\*kwargs):  
 self.clean()  
 *# if not self.pk:  
 # self.remaining\_amount = self.allocated\_amount* super().save(\*args, \*\*kwargs)  
 *# self.budget\_allocation.remaining\_amount = self.budget\_allocation.get\_remaining\_amount()  
 # self.budget\_allocation.save(update\_fields=['remaining\_amount'])  
 # self.remaining\_amount = self.get\_remaining\_amount()  
 # super().save(update\_fields=['remaining\_amount'])* @property  
 def remaining\_amount(self):  
 *"""  
 محاسبه مقدار باقی‌مانده با کسر مجموع تراکنش‌های مصرفی از مبلغ تخصیص‌شده  
 """* consumed = BudgetTransaction.objects.filter(  
 allocation=self.budget\_allocation,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or 0  
  
 returned = BudgetTransaction.objects.filter(  
 allocation=self.budget\_allocation,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or 0  
  
 return self.allocated\_amount - consumed + returned  
  
  
 def \_\_str\_\_(self):  
 target = self.subproject.name if self.subproject else self.project.name  
 jalali\_date = jdatetime.date.fromgregorian(date=self.allocation\_date).strftime('%Y/%m/%d')  
 return f"{target} - {self.allocated\_amount:,} ({jalali\_date})"  
  
 class Meta:  
 verbose\_name = \_("تخصیص بودجه پروژه")  
 verbose\_name\_plural = \_("تخصیص‌های بودجه پروژه")  
 default\_permissions = ()  
 permissions = [  
 ('ProjectBudgetAllocation\_add', \_('افزودن تخصیص بودجه پروژه')),  
 ('ProjectBudgetAllocation\_view', \_('نمایش تخصیص بودجه پروژه')),  
 ('ProjectBudgetAllocation\_update', \_('بروزرسانی تخصیص بودجه پروژه')),  
 ('ProjectBudgetAllocation\_delete', \_('حذف تخصیص بودجه پروژه')),  
 ('ProjectBudgetAllocation\_Head\_Office', 'تخصیص بودجه مجموعه پروژه(دفتر مرکزی)🏠'),  
 ('ProjectBudgetAllocation\_Branch', 'تخصیص بودجه مجموعه پروژه(شعبه)🏠'),  
 ]  
  
"""PaymentOrder (دستور پرداخت):"""  
class PaymentOrder(models.Model):  
 *"""توضیح: مدل جدا برای دستور پرداخت با لینک به تنخواه و فاکتورها. min\_signatures برای کنترل تعداد امضا و شماره‌گذاری پویا با تاریخ شمسی."""* STATUS\_CHOICES = (  
 ('DRAFT', \_('پیش‌نویس')),  
 ('PENDING\_SIGNATURES', \_('در انتظار امضا')),  
 ('ISSUED', \_('صادر شده')),  
 ('PAID', \_('پرداخت شده')),  
 ('CANCELED', \_('لغو شده')),  
 )  
 tankhah = models.ForeignKey('tankhah.Tankhah', on\_delete=models.CASCADE,  
 related\_name='payment\_orders', verbose\_name=\_("تنخواه"))  
 order\_number = models.CharField(max\_length=50, unique=True, blank=True,  
 verbose\_name=\_("شماره دستور پرداخت"))  
 issue\_date = models.DateField(default=timezone.now, verbose\_name=\_("تاریخ صدور"))  
 amount = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("مبلغ"))  
 payee = models.ForeignKey('Payee', on\_delete=models.SET\_NULL, null=True,  
 verbose\_name=\_("دریافت‌کننده"))  
 description = models.TextField(verbose\_name=\_("شرح پرداخت"))  
 related\_factors = models.ManyToManyField('tankhah.Factor', blank=True,  
 verbose\_name=\_("فاکتورهای مرتبط"))  
 payment\_id = models.CharField(max\_length=50, blank=True, null=True,  
 verbose\_name=\_("شناسه پرداخت"))  
 status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='DRAFT',  
 verbose\_name=\_("وضعیت"))  
 created\_by\_post = models.ForeignKey('core.Post', on\_delete=models.SET\_NULL, null=True,  
 verbose\_name=\_("پست ایجادکننده"))  
 min\_signatures = models.IntegerField(default=1, verbose\_name=\_("حداقل تعداد امضا"))  
 created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,  
 related\_name='payment\_orders\_created', verbose\_name=\_("ایجادکننده"))  
  
 payment\_date = models.DateField(null=True, blank=True, verbose\_name=\_("تاریخ پرداخت"))  
 def generate\_order\_number(self):  
 sep = "-"  
 date\_str = jdatetime.date.fromgregorian(date=self.issue\_date).strftime('%Y%m%d')  
 serial = PaymentOrder.objects.filter(issue\_date=self.issue\_date).count() + 1  
 return f"PO{sep}{date\_str}{sep}{serial:03d}"  
  
 def save(self, \*args, \*\*kwargs):  
 if not self.order\_number:  
 self.order\_number = self.generate\_order\_number()  
 super().save(\*args, \*\*kwargs)  
  
 def \_\_str\_\_(self):  
 return f"{self.order\_number} - {self.amount}"  
  
 class Meta:  
 verbose\_name = \_("دستور پرداخت")  
 verbose\_name\_plural = \_("دستورهای پرداخت")  
 default\_permissions = ()  
 permissions = [  
 ('PaymentOrder\_add', \_('افزودن دستور پرداخت')),  
 ('PaymentOrder\_view', \_('نمایش دستور پرداخت')),  
 ('PaymentOrder\_update', \_('بروزرسانی دستور پرداخت')),  
 ('PaymentOrder\_delete', \_('حذف دستور پرداخت')),  
 ('PaymentOrder\_sign', \_('امضای دستور پرداخت')),  
 ('PaymentOrder\_issue', \_('صدور دستور پرداخت')),  
 ]  
  
"""Payee (دریافت‌کننده):"""  
class Payee(models.Model):  
 *"""توضیح: اطلاعات دریافت‌کننده پرداخت با نوع مشخص (فروشنده، کارمند، دیگر)."""* PAYEE\_TYPES = (  
 ('VENDOR', \_('فروشنده')),  
 ('EMPLOYEE', \_('کارمند')),  
 ('OTHER', \_('دیگر')),  
 )  
 name = models.CharField(max\_length=100, verbose\_name=\_("نام"))  
 family = models.CharField(max\_length=100, verbose\_name=\_("نام خانوادگی"))  
 payee\_type = models.CharField(max\_length=20, choices=PAYEE\_TYPES, verbose\_name=\_("نوع"))  
 national\_id = models.CharField(max\_length=20, blank=True, null=True,  
 verbose\_name=\_("کد ملی/اقتصادی"))  
 account\_number = models.CharField(max\_length=50, blank=True, null=True,  
 verbose\_name=\_("شماره حساب"))  
 iban = models.CharField(max\_length=34, blank=True, null=True, verbose\_name=\_("شبا"))  
 address = models.TextField(blank=True, null=True, verbose\_name=\_("آدرس"))  
 phone = models.CharField(max\_length=20, blank=True, null=True, verbose\_name=\_("تلفن"))  
 created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,  
 related\_name='payees\_created', verbose\_name=\_("ایجادکننده"))  
 is\_active= models.BooleanField(verbose\_name=\_('فعال'))  
 def \_\_str\_\_(self):  
 return f"{self.name} ({self.payee\_type})"  
  
 class Meta:  
 verbose\_name = \_("دریافت‌کننده")  
 verbose\_name\_plural = \_("دریافت‌کنندگان")  
 default\_permissions = ()  
 permissions = [  
 ('Payee\_add', \_('افزودن دریافت‌کننده')),  
 ('Payee\_view', \_('نمایش دریافت‌کننده')),  
 ('Payee\_update', \_('بروزرسانی دریافت‌کننده')),  
 ('Payee\_delete', \_('حذف دریافت‌کننده')),  
 ]  
  
"""TransactionType (نوع تراکنش):"""  
class TransactionType(models.Model):  
 *"""توضیح: تعریف پویا نوع تراکنش‌ها (مثل بیمه، جریمه) با امکان نیاز به تأیید اضافی."""* name = models.CharField(max\_length=250, unique=True, verbose\_name=\_("نام"))  
 description = models.TextField(blank=True, verbose\_name=\_("توضیحات"))  
 requires\_extra\_approval = models.BooleanField(default=False,  
 verbose\_name=\_("نیاز به تأیید اضافی"))  
 created\_by = models.ForeignKey('accounts.CustomUser', on\_delete=models.SET\_NULL, null=True,  
 related\_name='transaction\_types\_created', verbose\_name=\_("ایجادکننده"))  
 category = models.CharField(max\_length=50, blank=True, null=True,verbose\_name=\_('گروه بندی تراکنش‌ها '))  
  
 def \_\_str\_\_(self):  
 return self.name  
  
 class Meta:  
 verbose\_name = \_("نوع تراکنش")  
 verbose\_name\_plural = \_("انواع تراکنش")  
 default\_permissions = ()  
 permissions = [  
 ('TransactionType\_add', \_('افزودن نوع تراکنش')),  
 ('TransactionType\_view', \_('نمایش نوع تراکنش')),  
 ('TransactionType\_update', \_('بروزرسانی نوع تراکنش')),  
 ('TransactionType\_delete', \_('حذف نوع تراکنش')),  
 ]  
  
"""مدل BudgetReallocation برای انتقال باقی‌مانده بودجه متوقف‌شده."""  
class BudgetReallocation(models.Model):  
 source\_allocation = models.ForeignKey('BudgetAllocation', on\_delete=models.CASCADE, related\_name='reallocations\_from')  
 target\_allocation = models.ForeignKey('BudgetAllocation', on\_delete=models.CASCADE, null=True, related\_name='reallocations\_to')  
 amount = models.DecimalField(max\_digits=25, decimal\_places=2)  
 reallocation\_date = models.DateField(default=timezone.now)  
 reason = models.TextField()  
 created\_by = models.ForeignKey(CustomUser, on\_delete=models.SET\_NULL, null=True)  
  
"""یه جدول تنظیمات (BudgetSettings) برای مدیریت قفل و هشدار در سطوح مختلف:"""  
class BudgetSettings(models.Model):  
 level = models.CharField(max\_length=50, choices=[('PERIOD', 'دوره بودجه'), ('ALLOCATION', 'تخصیص'), ('PROJECT', 'پروژه')])  
 locked\_percentage = models.DecimalField(max\_digits=5, decimal\_places=2)  
 warning\_threshold = models.DecimalField(max\_digits=5, decimal\_places=2)  
 warning\_action = models.CharField(max\_length=50, choices=[('NOTIFY', 'اعلان'), ('LOCK', 'قفل'), ('RESTRICT', 'محدود')])  
 organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, null=True)  
 budget\_period = models.ForeignKey('BudgetPeriod', on\_delete=models.CASCADE, null=True, verbose\_name=\_("دوره بودجه"))  
  
 *# ردیابی مدل  
 # content\_type = models.ForeignKey(ContentType, on\_delete=models.CASCADE)  
 # object\_id = models.PositiveIntegerField()  
 # content\_object = GenericForeignKey('content\_type', 'object\_id')*"""مدل BudgetHistory برای لاگ کردن تغییرات بودجه و تخصیص‌ها:"""  
*# ------------------------------------*"""تاریخچه برای هر بودجه کلان"""  
class BudgetHistory(models.Model):  
 content\_type = models.ForeignKey(ContentType, on\_delete=models.CASCADE)  
 object\_id = models.PositiveIntegerField()  
 *# content\_object = GenericForeignKey('content\_type', 'object\_id')* action = models.CharField(max\_length=50, choices=[  
 ('CREATE', \_('ایجاد')),  
 ('UPDATE', \_('بروزرسانی')),  
 ('STOP', \_('توقف')),  
 ('REALLOCATE', \_('انتقال')),  
 ('RETURN', \_('برگشت')) *# جدید* ])  
 amount = models.DecimalField(max\_digits=25, decimal\_places=2, null=True, verbose\_name=\_('رقم عدد'))  
 created\_by = models.ForeignKey(CustomUser, on\_delete=models.SET\_NULL, null=True, verbose\_name=\_('کاربر ثبت کننده'))  
 created\_at = models.DateTimeField(auto\_now\_add=True, verbose\_name=\_('اکشن در تاریخ و ساعت'))  
 details = models.TextField(blank=True, verbose\_name=\_('جزئیات'))  
 transaction\_type = models.CharField(max\_length=20, choices=[('ALLOCATION', \_('تخصیص')), ('CONSUMPTION', \_('مصرف')), ('RETURN', \_('برگشت'))],verbose\_name=\_("نوع تراکنش"))  
 transaction\_id = models.CharField(max\_length=50, unique=True, verbose\_name=\_("شناسه تراکنش"))  
  
  
 def \_\_str\_\_(self):  
 return f"{self.action} - {self.amount:,} ({self.created\_at})"  
  
 class Meta:  
 verbose\_name = \_("تاریخچه بودجه")  
 verbose\_name\_plural = \_("تاریخچه‌های بودجه")  
 default\_permissions = ()  
 permissions = [  
 ('BudgetHistory\_add','افزودن تاریخچه برای هر بودجه کلان'),  
 ('BudgetHistory\_view',' نمایش تاریخچه هر بودجه کلان'),  
 ('BudgetHistory\_update','بروزرسانی تاریخچه برای هر بودجه کلان'),  
 ('BudgetHistory\_delete',' حــذف تاریخچه برای هر بودجه کلان'),  
 ]  
  
class SystemSettings(models.Model):  
 *"""تنظیمات سیستم بودجه"""* budget\_locked\_percentage\_default = models.DecimalField(  
 max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده پیش‌فرض بودجه")  
 )  
 budget\_warning\_threshold\_default = models.DecimalField(  
 max\_digits=5, decimal\_places=2, default=10, verbose\_name=\_("آستانه هشدار پیش‌فرض بودجه")  
 )  
 budget\_warning\_action\_default = models.CharField(  
 max\_length=50, choices=[('NOTIFY', 'اعلان'), ('LOCK', 'قفل'), ('RESTRICT', 'محدود')],  
 default='NOTIFY', verbose\_name=\_("اقدام هشدار پیش‌فرض بودجه")  
 )  
 allocation\_locked\_percentage\_default = models.DecimalField(  
 max\_digits=5, decimal\_places=2, default=0, verbose\_name=\_("درصد قفل‌شده پیش‌فرض تخصیص")  
 )  
  
  
 class Meta:  
 verbose\_name = \_("تنظیمات سیستم")  
 verbose\_name\_plural = \_("تنظیمات سیستم")  
  
 def \_\_str\_\_(self):  
 return "تنظیمات سیستم بودجه"  
  
""" مدل پیشنهادی برای هزینه‌های متعارف"""  
class CostCenter(models.Model):  
 *""" مدل پیشنهادی برای هزینه‌های متعارف"""* name = models.CharField(max\_length=200, verbose\_name=\_("نام مرکز هزینه"))  
 code = models.CharField(max\_length=50, unique=True, verbose\_name=\_("کد مرکز هزینه"))  
 organization = models.ForeignKey('core.Organization', on\_delete=models.CASCADE, verbose\_name=\_("سازمان"))  
 budget\_allocation = models.ForeignKey(BudgetAllocation, on\_delete=models.CASCADE, verbose\_name=\_("تخصیص بودجه"))  
 allocated\_budget = models.DecimalField(max\_digits=25, decimal\_places=2, verbose\_name=\_("بودجه تخصیص‌یافته"))  
 is\_active = models.BooleanField(default=True, verbose\_name=\_("فعال"))  
  
 def get\_remaining\_budget(self):  
 from tankhah.models import Tankhah  
 consumed = Tankhah.objects.filter(cost\_center=self).aggregate(Sum('amount'))['amount\_\_sum'] or Decimal('0')  
 return max(self.allocated\_budget - consumed, Decimal('0'))  
  
 class Meta:  
 verbose\_name = \_("مرکز هزینه")  
 verbose\_name\_plural = \_("مراکز هزینه")

محتوای فایل توابع   
import logging  
from decimal import Decimal  
from django.db.models import Sum, Q  
from django.core.cache import cache  
from django.utils import timezone  
from Tanbakhsystem.utils import parse\_jalali\_date  
from django.utils.translation import gettext\_lazy as \_  
  
  
logger = logging.getLogger(\_\_name\_\_)  
  
  
  
def calculate\_total\_allocated(entity=None, filters=None):  
 from budgets.models import BudgetAllocation  
 if filters is None:  
 filters = {}  
 queryset = BudgetAllocation.objects.all()  
  
 if entity:  
 from core.models import Organization  
 if isinstance(entity, Organization):  
 queryset = queryset.filter(organization=entity)  
 else:  
 logger.warning(f"Invalid entity type for calculate\_total\_allocated: {type(entity)}")  
 return Decimal('0')  
  
 if 'budget\_period' in filters:  
 queryset = queryset.filter(budget\_period=filters['budget\_period'])  
 if 'date\_from' in filters:  
 date\_from = parse\_jalali\_date(filters['date\_from']) if isinstance(filters['date\_from'], str) else filters[  
 'date\_from']  
 queryset = queryset.filter(allocation\_date\_\_gte=date\_from)  
 if 'date\_to' in filters:  
 date\_to = parse\_jalali\_date(filters['date\_to']) if isinstance(filters['date\_to'], str) else filters['date\_to']  
 queryset = queryset.filter(allocation\_date\_\_lte=date\_to)  
 if 'is\_active' in filters:  
 queryset = queryset.filter(is\_active=filters['is\_active'])  
  
 total = queryset.aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 logger.debug(f"calculate\_total\_allocated: entity={entity}, filters={filters}, total={total}")  
 return total  
  
  
def calculate\_remaining\_budget(obj=None, filters=None):  
 from budgets.models import BudgetPeriod, BudgetAllocation, ProjectBudgetAllocation  
 from tankhah.models import Tankhah  
 from core.models import Organization  
 from decimal import Decimal  
  
 cache\_key = f"remaining\_budget\_{type(obj).\_\_name\_\_ if obj else 'global'}\_{obj.pk if obj else '0'}"  
 if filters:  
 cache\_key += f"\_{hash(str(filters))}"  
 cached\_result = cache.get(cache\_key)  
 if cached\_result is not None:  
 logger.debug(f"Returning cached remaining\_budget for {cache\_key}: {cached\_result}")  
 return cached\_result  
  
 if obj:  
 if isinstance(obj, BudgetPeriod):  
 allocated = BudgetAllocation.objects.filter(budget\_period=obj)  
 if filters:  
 allocated = apply\_filters(allocated, filters)  
 allocated = allocated.aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 result = max(obj.total\_amount - allocated, Decimal('0'))  
 elif isinstance(obj, BudgetAllocation):  
 used = ProjectBudgetAllocation.objects.filter(budget\_allocation=obj)  
 if filters:  
 used = apply\_filters(used, filters)  
 used = used.aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 result = max(obj.allocated\_amount - used, Decimal('0'))  
 elif isinstance(obj, ProjectBudgetAllocation):  
 consumed = Tankhah.objects.filter(project\_budget\_allocation=obj)  
 if filters:  
 consumed = apply\_filters(consumed, filters)  
 consumed = consumed.aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 result = max(obj.allocated\_amount - consumed, Decimal('0'))  
 elif isinstance(obj, Organization):  
 total\_budget = BudgetPeriod.objects.filter(organization=obj, is\_active=True, is\_completed=False)  
 if filters:  
 total\_budget = apply\_filters(total\_budget, filters)  
 total\_budget = total\_budget.aggregate(total=Sum('total\_amount'))['total'] or Decimal('0')  
 total\_allocated = BudgetAllocation.objects.filter(organization=obj)  
 if filters:  
 total\_allocated = apply\_filters(total\_allocated, filters)  
 total\_allocated = total\_allocated.aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 result = max(total\_budget - total\_allocated, Decimal('0'))  
 else:  
 logger.warning(f"Invalid object type for calculate\_remaining\_budget: {type(obj)}")  
 result = Decimal('0')  
 else:  
 total\_budget = BudgetPeriod.objects.all()  
 if filters:  
 total\_budget = apply\_filters(total\_budget, filters)  
 total\_budget = total\_budget.aggregate(total=Sum('total\_amount'))['total'] or Decimal('0')  
 total\_allocated = BudgetAllocation.objects.all()  
 if filters:  
 total\_allocated = apply\_filters(total\_allocated, filters)  
 total\_allocated = total\_allocated.aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 result = max(total\_budget - total\_allocated, Decimal('0'))  
  
 cache.set(cache\_key, result, timeout=300)  
 logger.debug(f"calculate\_remaining\_budget: obj={obj}, filters={filters}, result={result}")  
 return result  
  
  
def check\_budget\_status(obj):  
 *"""چک کردن وضعیت بودجه"""* from budgets.models import BudgetPeriod  
 if not isinstance(obj, BudgetPeriod):  
 logger.warning(f"Invalid object type for check\_budget\_status: {type(obj)}")  
 return 'unknown', 'وضعیت نامشخص'  
  
 cache\_key = f"budget\_status\_{obj.pk}"  
 cached\_result = cache.get(cache\_key)  
 if cached\_result is not None:  
 logger.debug(f"Returning cached budget\_status for {cache\_key}: {cached\_result}")  
 return cached\_result  
  
 remaining = calculate\_remaining\_budget(obj)  
 locked = obj.get\_locked\_amount()  
 warning = obj.get\_warning\_amount()  
  
 if not obj.is\_active:  
 result = 'inactive', \_('دوره غیرفعال است.')  
 elif obj.is\_completed:  
 result = 'completed', \_('بودجه تمام‌شده است.')  
 elif remaining <= 0 and obj.lock\_condition == 'ZERO\_REMAINING':  
 obj.is\_completed = True  
 obj.is\_active = False  
 obj.save()  
 result = 'completed', \_('بودجه به صفر رسیده و تمام‌شده است.')  
 elif obj.lock\_condition == 'AFTER\_DATE' and obj.end\_date < timezone.now().date():  
 obj.is\_active = False  
 obj.save()  
 result = 'locked', \_('دوره به دلیل پایان تاریخ قفل شده است.')  
 elif obj.lock\_condition == 'MANUAL' and remaining <= locked:  
 result = 'locked', \_('بودجه به حد قفل‌شده رسیده است.')  
 elif remaining <= warning:  
 result = 'warning', \_('بودجه به آستانه هشدار رسیده است.')  
 else:  
 result = 'normal', \_('وضعیت عادی')  
 *# if result in ('completed', 'locked', 'inactive'):  
 # BudgetHistory.objects.create(  
 # content\_type=ContentType.objects.get\_for\_model(BudgetPeriod),  
 # object\_id=obj.pk,  
 # action='STATUS\_CHANGE',  
 # details=f"وضعیت به {result} تغییر کرد: {message}"  
 # )  
 # result+= status, message* cache.set(cache\_key, result, timeout=300)  
 logger.debug(f"check\_budget\_status: obj={obj}, result={result}")  
 return result  
  
  
def apply\_filters(queryset, filters):  
 if not filters:  
 return queryset  
 if 'date\_from' in filters:  
 date\_from = parse\_jalali\_date(filters['date\_from']) if isinstance(filters['date\_from'], str) else filters[  
 'date\_from']  
 queryset = queryset.filter(allocation\_date\_\_gte=date\_from)  
 if 'date\_to' in filters:  
 date\_to = parse\_jalali\_date(filters['date\_to']) if isinstance(filters['date\_to'], str) else filters['date\_to']  
 queryset = queryset.filter(allocation\_date\_\_lte=date\_to)  
 if 'is\_active' in filters:  
 queryset = queryset.filter(budget\_period\_\_is\_active=filters['is\_active'])  
 if 'budget\_period' in filters:  
 queryset = queryset.filter(budget\_period=filters['budget\_period'])  
 logger.debug(f"apply\_filters: filters={filters}, queryset count={queryset.count()}")  
 return queryset  
  
  
def get\_budget\_details(entity=None, filters=None):  
 from budgets.models import BudgetPeriod, BudgetAllocation, ProjectBudgetAllocation  
 from core.models import Organization, Project, SubProject  
  
 cache\_key = f"budget\_details\_{type(entity).\_\_name\_\_}\_{entity.pk if entity else 'global'}"  
 cached\_result = cache.get(cache\_key)  
 if cached\_result is not None:  
 logger.debug(f"Returning cached budget\_details for {cache\_key}: {cached\_result}")  
 return cached\_result  
  
 if isinstance(entity, BudgetPeriod):  
 total\_budget = entity.total\_amount  
 total\_allocated = BudgetAllocation.objects.filter(budget\_period=entity).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
 remaining = calculate\_remaining\_budget(entity)  
 status = check\_budget\_status(entity)  
 elif isinstance(entity, Organization):  
 total\_budget = BudgetAllocation.objects.filter(organization=entity).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
 total\_allocated = calculate\_total\_allocated(entity=entity, filters=filters)  
 remaining = calculate\_remaining\_budget(entity)  
 status = get\_budget\_status(entity, filters)  
 elif isinstance(entity, Project):  
 total\_budget = ProjectBudgetAllocation.objects.filter(project=entity).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
 total\_allocated = calculate\_total\_allocated(entity=entity, filters=filters)  
 remaining = calculate\_remaining\_budget(entity)  
 status = get\_budget\_status(entity, filters)  
 elif isinstance(entity, SubProject):  
 total\_budget = ProjectBudgetAllocation.objects.filter(subproject=entity).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
 total\_allocated = calculate\_total\_allocated(entity=entity, filters=filters)  
 remaining = calculate\_remaining\_budget(entity)  
 status = get\_budget\_status(entity, filters)  
 else:  
 total\_budget = BudgetPeriod.objects.aggregate(  
 total=Sum('total\_amount')  
 )['total'] or Decimal('0')  
 total\_allocated = calculate\_total\_allocated(filters=filters)  
 remaining = calculate\_remaining\_budget(filters=filters)  
 status = get\_budget\_status(None, filters)  
  
 details = {  
 'total\_budget': total\_budget,  
 'total\_allocated': total\_allocated,  
 'remaining\_budget': remaining,  
 'status': status['status'],  
 'status\_message': status['message']  
 }  
  
 cache.set(cache\_key, details, timeout=300)  
 logger.info(f"get\_budget\_details: entity={entity}, filters={filters}, details={details}")  
 return details  
  
def calculate\_allocation\_percentages(allocations):  
 total\_percentage = Decimal("0")  
 for allocation in allocations:  
 allocation.percentage = (  
 (allocation.allocated\_amount / allocation.budget\_period.total\_amount \* Decimal("100"))  
 if allocation.budget\_period.total\_amount else Decimal("0")  
 )  
 total\_percentage += allocation.percentage  
 logger.debug(f"calculate\_allocation\_percentages: total\_percentage={total\_percentage}, count={len(allocations)}")  
 return total\_percentage  
  
  
def get\_organization\_budget(organization) -> Decimal:  
 *# total = calculate\_total\_allocated(entity=organization)  
 # logger.debug(f"get\_organization\_budget: org={organization}, total={total}")  
 # return total  
  
 # from core.models import Organization  
 # from budgets.models import BudgetPeriod  
 # def get\_organization\_budget(organization: core.models.Organization) -> Decimal:  
 """محاسبه بودجه کل سازمان بر اساس دوره‌های بودجه فعال.  
 """* from budgets.models import BudgetPeriod  
  
 total\_budget = BudgetPeriod.objects.filter(organization=organization,is\_active=True,is\_completed=False).aggregate(total=Sum('total\_amount'))['total'] or Decimal('0')  
 return total\_budget  
  
def get\_project\_total\_budget(project):  
 from budgets.models import ProjectBudgetAllocation  
 total = ProjectBudgetAllocation.objects.filter(project=project, subproject\_\_isnull=True).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal("0")  
 logger.debug(f"get\_project\_total\_budget: project={project}, total={total}")  
 return total  
  
  
def get\_project\_used\_budget(project):  
 from budgets.models import ProjectBudgetAllocation  
 subproject\_budget = ProjectBudgetAllocation.objects.filter(project=project, subproject\_\_isnull=False).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal("0")  
 tankhah\_budget = project.tankhah\_set.filter(status='PAID').aggregate(  
 total=Sum('amount')  
 )['total'] or Decimal("0")  
 total = subproject\_budget + tankhah\_budget  
 logger.debug(  
 f"get\_project\_used\_budget: project={project}, subproject={subproject\_budget}, tankhah={tankhah\_budget}, total={total}")  
 return total  
  
  
def get\_project\_remaining\_budget(project):  
 *"""محاسبه بودجه باقی‌مانده پروژه"""* from budgets.models import ProjectBudgetAllocation  
 allocations = ProjectBudgetAllocation.objects.filter(project=project)  
 total\_allocated = allocations.aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 *# فرض می‌کنیم مصرف‌ها از BudgetTransaction ثبت می‌شوند* from budgets.models import BudgetTransaction  
 consumptions = BudgetTransaction.objects.filter(  
 allocation\_\_project\_allocations\_\_project=project,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returns = BudgetTransaction.objects.filter(  
 allocation\_\_project\_allocations\_\_project=project,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 return max(total\_allocated - consumptions + returns, Decimal('0'))  
  
  
def get\_subproject\_total\_budget(subproject):  
 from budgets.models import ProjectBudgetAllocation  
 total = ProjectBudgetAllocation.objects.filter(subproject=subproject).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal("0")  
 logger.debug(f"get\_subproject\_total\_budget: subproject={subproject}, total={total}")  
 return total  
  
  
def get\_subproject\_used\_budget(subproject):  
 total = subproject.tankhah\_set.filter(status='PAID').aggregate(  
 total=Sum('amount')  
 )['total'] or Decimal("0")  
 logger.debug(f"get\_subproject\_used\_budget: subproject={subproject}, total={total}")  
 return total  
  
  
def get\_subproject\_remaining\_budget(subproject):  
 *"""محاسبه بودجه باقی‌مانده زیرپروژه"""* from budgets.models import ProjectBudgetAllocation  
 allocations = ProjectBudgetAllocation.objects.filter(subproject=subproject)  
 total\_allocated = allocations.aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 from budgets.models import BudgetTransaction  
 consumptions = BudgetTransaction.objects.filter(  
 allocation\_\_project\_allocations\_\_subproject=subproject,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returns = BudgetTransaction.objects.filter(  
 allocation\_\_project\_allocations\_\_subproject=subproject,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 return max(total\_allocated - consumptions + returns, Decimal('0'))  
  
  
  
  
def can\_delete\_budget(entity):  
 from core.models import Project, SubProject  
 if isinstance(entity, Project):  
 can\_delete = not entity.tankhah\_set.exists() and not entity.subprojects.exists()  
 elif isinstance(entity, SubProject):  
 can\_delete = not entity.tankhah\_set.exists()  
 else:  
 can\_delete = False  
 logger.debug(f"can\_delete\_budget: entity={entity}, can\_delete={can\_delete}")  
 return can\_delete  
  
  
def get\_locked\_amount(obj):  
 from budgets.models import BudgetPeriod  
 if isinstance(obj, BudgetPeriod):  
 return (obj.total\_amount \* obj.locked\_percentage) / Decimal('100')  
 return Decimal('0')  
  
  
def get\_warning\_amount(obj):  
 from budgets.models import BudgetPeriod  
 if isinstance(obj, BudgetPeriod):  
 return (obj.total\_amount \* obj.warning\_threshold) / Decimal('100')  
 return Decimal('0')  
  
def get\_budget\_status(entity, filters=None):  
 *"""  
 تابع وضعیت بودجه برای سازمان‌ها، پروژه‌ها، و زیرپروژه‌ها  
 فرض: مشابه check\_budget\_status اما برای موجودیت‌های دیگر  
 """* """بررسی وضعیت بودجه"""  
 from budgets.models import BudgetPeriod, BudgetAllocation, ProjectBudgetAllocation  
 from core.models import Organization, Project, SubProject  
  
 *# cache\_key = f"budget\_status\_{type(entity).\_\_name\_\_}\_{entity.pk}"  
 # cached\_result = cache.get(cache\_key)  
 # if cached\_result is not None:  
 # logger.debug(f"Returning cached budget\_status for {cache\_key}: {cached\_result}")  
 # return cached\_result* if isinstance(entity, BudgetPeriod):  
 status, message = entity.check\_budget\_status()  
 elif isinstance(entity, Organization):  
 allocations = BudgetAllocation.objects.filter(organization=entity)  
 if filters:  
 allocations = apply\_filters(allocations, filters)  
 if not allocations.exists():  
 status, message = 'no\_budget', 'هیچ بودجه‌ای تخصیص نیافته است.'  
 else:  
 active\_count = allocations.filter(budget\_period\_\_is\_active=True).count()  
 status = 'active' if active\_count > 0 else 'inactive'  
 message = f"{active\_count} تخصیص فعال از {allocations.count()} کل"  
 elif isinstance(entity, (Project, SubProject)):  
 allocations = ProjectBudgetAllocation.objects.filter(  
 Q(project=entity) if isinstance(entity, Project) else Q(subproject=entity)  
 )  
 if not allocations.exists():  
 status, message = 'no\_budget', 'هیچ بودجه‌ای تخصیص نیافته است.'  
 else:  
 active\_count = allocations.filter(budget\_allocation\_\_budget\_period\_\_is\_active=True).count()  
 status = 'active' if active\_count > 0 else 'inactive'  
 message = f"{active\_count} تخصیص فعال از {allocations.count()} کل"  
 else:  
 status, message = 'unknown', 'وضعیت نامشخص'  
  
 result = {'status': status, 'message': message}  
 *# cache.set(cache\_key, result, timeout=300)* logger.debug(f"get\_budget\_status: entity={entity}, result={result}")  
 return result

تمامی محاسبات داخل مدل را ببر توی فایل توابع و اصلاحشون کن   
ویو ها   
from django.db import transaction  
from django.shortcuts import get\_object\_or\_404  
from django.views import View  
from django.http import HttpResponseRedirect  
from django.contrib import messages  
  
from budgets.models import BudgetAllocation  
from core.PermissionBase import PermissionBaseView  
from core.models import WorkflowStage, PostAction  
from tankhah.models import Tankhah, ApprovalLog  
from django.utils.translation import gettext\_lazy as \_  
from django.contrib.contenttypes.models import ContentType  
  
"""این ویو بررسی می‌کند که آیا کاربر و پست او مجوز انجام اقدام را دارند.  
اقدام (تأیید یا رد) را انجام می‌دهد و وضعیت و مرحله را به‌روزرسانی می‌کند.  
لاگ اقدام را در ApprovalLog ثبت می‌کند.  
از تراکنش اتمیک برای اطمینان از یکپارچگی داده‌ها استفاده می‌کند  
"""  
  
class ApproveRejectView(PermissionBaseView, View):  
 *"""اقدامات تأیید/رد  
 این ویو بررسی می‌کند که آیا کاربر مجوز و پست مناسب برای انجام اقدام (APPROVE یا REJECT) را دارد.  
 برای تنخواه و بودجه به‌صورت جداگانه عمل می‌کند.  
 وضعیت (status) و مرحله فعلی (current\_stage) موجودیت را به‌روزرسانی می‌کند.  
 """* permission\_codenames = ['tankhah.Tankhah\_approve', 'tankhah.Tankhah\_reject', 'budgets.BudgetAllocation\_approve',  
 'budgets.BudgetAllocation\_reject']  
  
 def post(self, request, entity\_type, entity\_id, action):  
 user = request.user  
 user\_posts = user.userpost\_set.filter(is\_active=True, end\_date\_\_isnull=True).values\_list('post\_\_id', flat=True)  
  
 *# شناسایی موجودیت  
 # if entity\_type == 'tankhah' and action == 'APPROVE':* if entity\_type == 'tankhah':  
 entity = get\_object\_or\_404(Tankhah, id=entity\_id)  
 required\_perm = 'tankhah.Tankhah\_approve' if action == 'APPROVE' else 'tankhah.Tankhah\_reject'  
 entity\_type\_upper = 'TANKHAH'  
 elif entity\_type == 'budget\_allocation':  
 entity = get\_object\_or\_404(BudgetAllocation, id=entity\_id)  
 required\_perm = 'budgets.BudgetAllocation\_approve' if action == 'APPROVE' else 'budgets.BudgetAllocation\_reject'  
 entity\_type\_upper = 'BUDGET\_ALLOCATION'  
 else:  
 messages.error(request, \_('نوع موجودیت نامعتبر است.'))  
 return HttpResponseRedirect(request.META.get('HTTP\_REFERER', 'dashboard'))  
  
 """اگر یک تنخواه یا تخصیص بودجه به مرحله نهایی (is\_final\_stage=True در WorkflowStage) برسد، می‌توان وضعیت آن را به COMPLETED یا PAID (برای تنخواه) تغییر داد:"""  
 if action == 'APPROVE' and entity.current\_stage.is\_final\_stage:  
 entity.status = 'COMPLETED' if entity\_type == 'budget\_allocation' else 'PAID'  
 *# بررسی مجوز کاربر* if not user.has\_perm(required\_perm):  
 messages.error(request, \_('شما مجوز انجام این اقدام را ندارید.'))  
 return HttpResponseRedirect(request.META.get('HTTP\_REFERER', 'dashboard'))  
  
 *# بررسی مجوز پست* if not PostAction.objects.filter(  
 post\_\_in=user\_posts,  
 stage=entity.current\_stage,  
 action\_type=action,  
 entity\_type=entity\_type\_upper,  
 is\_active=True  
 ).exists():  
 messages.error(request, \_('پست شما اجازه انجام این اقدام را در این مرحله ندارد.'))  
 return HttpResponseRedirect(request.META.get('HTTP\_REFERER', 'dashboard'))  
  
 with transaction.atomic():  
 *# انجام اقدام* if action == 'APPROVE':  
 entity.status = 'APPROVED'  
 entity.approved\_by.add(user)  
 *# انتقال به مرحله بعدی* next\_stage = WorkflowStage.objects.filter(  
 order\_\_gt=entity.current\_stage.order,  
 is\_active=True  
 ).order\_by('order').first()  
 entity.current\_stage = next\_stage if next\_stage else entity.current\_stage  
 messages.success(request, \_(f'{entity\_type.capitalize()} با موفقیت تأیید شد.'))  
 elif action == 'REJECT':  
 entity.status = 'REJECTED'  
 messages.success(request, \_(f'{entity\_type.capitalize()} با موفقیت رد شد.'))  
  
 *# ثبت در ApprovalLog* user\_post = user.userpost\_set.filter(is\_active=True, end\_date\_\_isnull=True).first()  
 ApprovalLog.objects.create(  
 content\_type=ContentType.objects.get\_for\_model(entity),  
 object\_id=entity.id,  
 action=action,  
 stage=entity.current\_stage,  
 user=user,  
 post=user\_post.post if user\_post else None,  
 comment=request.POST.get('comment', ''),  
 )  
  
 entity.save()  
  
 return HttpResponseRedirect(request.META.get('HTTP\_REFERER', 'dashboard'))

from django import forms  
from django.core.exceptions import ValidationError  
from django.utils.translation import gettext\_lazy as \_  
from budgets.models import BudgetItem, BudgetPeriod  
from core.models import Organization  
from decimal import Decimal  
import logging  
  
logger = logging.getLogger(\_\_name\_\_)  
  
class BudgetItemForm(forms.ModelForm):  
 class Meta:  
 model = BudgetItem  
 *# fields = ['budget\_period', 'organization', 'name', 'code', 'is\_active']* fields = ['budget\_period', 'organization', 'name', 'code', 'is\_active']  
 widgets = {  
 'budget\_period': forms.Select(attrs={  
 'class': 'form-select',  
 'required': 'required',  
 'data-control': 'select2',  
 }),  
 'organization': forms.Select(attrs={  
 'class': 'form-select',  
 'required': 'required',  
 'data-control': 'select2',  
 }),  
 'name': forms.TextInput(attrs={  
 'class': 'form-control',  
 'required': 'required',  
 'placeholder': \_('نام ردیف بودجه'),  
 }),  
 'code': forms.TextInput(attrs={  
 'class': 'form-control',  
 'required': 'required',  
 'placeholder': \_('کد ردیف بودجه'),  
 }),  
 'is\_active': forms.CheckboxInput(attrs={  
 'class': 'form-check-input',  
 'role': 'switch',  
 }),  
 }  
 labels = {  
 'budget\_period': \_('دوره بودجه'),  
 'organization': \_('سازمان/شعبه'),  
 'name': \_('نام ردیف بودجه'),  
 'code': \_('کد ردیف بودجه'),  
 'is\_active': \_('فعال'),  
 }  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 self.fields['budget\_period'].queryset = BudgetPeriod.objects.filter(is\_active=True).select\_related(  
 'organization')  
 self.fields['organization'].queryset = Organization.objects.filter(is\_active=True).select\_related('org\_type')  
 logger.debug("Initialized BudgetItemForm")  
  
 def clean\_code(self):  
 code = self.cleaned\_data.get('code')  
 instance = self.instance  
 if BudgetItem.objects.exclude(pk=instance.pk if instance else None).filter(code=code).exists():  
 raise forms.ValidationError(\_('این کد قبلاً استفاده شده است.'))  
 return code  
 *#* def clean\_total\_amount(self):  
 amount = self.cleaned\_data.get('total\_amount')  
 budget\_period = self.cleaned\_data.get('budget\_period')  
 if amount is None:  
 raise forms.ValidationError(\_('مبلغ کل نمی‌تواند خالی باشد.'))  
 if amount <= 0:  
 raise forms.ValidationError(\_('مبلغ کل باید مثبت باشد.'))  
 if budget\_period:  
 remaining = budget\_period.get\_remaining\_amount()  
 if amount > remaining:  
 raise forms.ValidationError(  
 f"مبلغ ردیف ({amount:,.0f} ریال) بیشتر از باقی‌مانده دوره ({remaining:,.0f} ریال) است."  
 )  
 return amount  
  
 def clean(self):  
 cleaned\_data = super().clean()  
 budget\_period = cleaned\_data.get('budget\_period')  
 organization = cleaned\_data.get('organization')  
 if budget\_period and organization and budget\_period.organization != organization:  
 raise forms.ValidationError(\_('سازمان باید با دوره بودجه مطابقت داشته باشد.'))  
 return cleaned\_data

from django.contrib import messages  
from django.db import models  
from django.db.models import Q  
from django.shortcuts import redirect  
from django.views.generic import ListView, CreateView, UpdateView, DeleteView, DetailView  
from django.urls import reverse\_lazy  
from django.contrib.messages.views import SuccessMessageMixin  
from django.utils.translation import gettext\_lazy as \_  
  
from budgets.Budget\_Items.frosm\_BudgetItem import BudgetItemForm  
from budgets.models import BudgetItem, BudgetPeriod  
  
from django.contrib.auth.mixins import PermissionRequiredMixin  
import logging  
  
logger = logging.getLogger(\_\_name\_\_)  
  
class BudgetItemListView(PermissionRequiredMixin, ListView):  
 model = BudgetItem  
 template\_name = 'budgets/budgetitem/budgetitem\_list.html'  
 context\_object\_name = 'budget\_items'  
 paginate\_by = 10  
 permission\_required = 'budgets.view\_budgetitem'  
  
 def get\_queryset(self):  
 queryset = BudgetItem.objects.select\_related('budget\_period', 'organization')*#.order\_by('-created\_at')* query = self.request.GET.get('q')  
 if query:  
 queryset = queryset.filter(  
 Q(name\_\_icontains=query) |  
 Q(code\_\_icontains=query) |  
 Q(budget\_period\_\_name\_\_icontains=query) |  
 Q(organization\_\_name\_\_icontains=query)  
 )  
 return queryset  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['title'] = \_('لیست ردیف‌های بودجه')  
 context['query'] = self.request.GET.get('q', '')  
 return context  
  
class BudgetItemCreateView(PermissionRequiredMixin, SuccessMessageMixin, CreateView):  
 model = BudgetItem  
 form\_class = BudgetItemForm  
 template\_name = 'budgets/budgetitem/budgetitem\_form.html'  
 success\_url = reverse\_lazy('budgetitem\_list')  
 success\_message = \_('ردیف بودجه با موفقیت ایجاد شد.')  
 permission\_required = 'budgets.add\_budgetitem'  
  
 def get\_context\_data(self, \*\*kwargs):  
 from core.models import Organization  
 context = super().get\_context\_data(\*\*kwargs)  
 context['title'] = \_('ایجاد ردیف بودجه جدید')  
 context['budget\_periods'] = BudgetPeriod.objects.filter(is\_active=True).select\_related('organization')  
 context['organizations'] = Organization.objects.filter(is\_active=True).select\_related('org\_type')  
 return context  
  
  
 def form\_invalid(self, form):  
 logger.error(f"Form invalid: errors={form.errors.as\_json()}")  
 messages.error(self.request, \_('لطفاً خطاهای فرم را بررسی کنید.'))  
 return super().form\_invalid(form)  
  
class BudgetItemUpdateView(PermissionRequiredMixin, SuccessMessageMixin, UpdateView):  
 model = BudgetItem  
 form\_class = BudgetItemForm  
 template\_name = 'budgets/budgetitem/budgetitem\_form.html'  
 success\_url = reverse\_lazy('budgetitem\_list')  
 success\_message = \_('ردیف بودجه با موفقیت ویرایش شد.')  
 permission\_required = 'budgets.change\_budgetitem'  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['title'] = \_('ویرایش ردیف بودجه')  
 return context  
  
class BudgetItemDeleteView(PermissionRequiredMixin, SuccessMessageMixin, DeleteView):  
 model = BudgetItem  
 template\_name = 'budgets/budgetitem/budgetitem\_confirm\_delete.html'  
 success\_url = reverse\_lazy('budgetitem\_list')  
 success\_message = \_('ردیف بودجه با موفقیت حذف شد.')  
 permission\_required = 'budgets.delete\_budgetitem'  
  
 def post(self, request, \*args, \*\*kwargs):  
 try:  
 return super().post(request, \*args, \*\*kwargs)  
 except models.ProtectedError:  
 messages.error(request, \_('نمی‌توان ردیف بودجه را حذف کرد چون تخصیص‌هایی به آن وابسته‌اند.'))  
 return redirect('budgetitem\_list')  
  
class BudgetItemDetailView(PermissionRequiredMixin, DetailView):  
 model = BudgetItem  
 template\_name = 'budgets/budgetitem/budgetitem\_detail.html'  
 context\_object\_name = 'budget\_item'  
 permission\_required = 'budgets.view\_budgetitem'  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['title'] = \_('جزئیات ردیف بودجه')  
 return context

import logging  
  
from decimal import Decimal  
import jdatetime  
from django import forms  
from django.db import transaction  
from django.db.models import Sum  
from django.utils.translation import gettext\_lazy as \_  
from django.core.exceptions import ValidationError *# Import ValidationError  
  
# Assuming models are imported correctly*from budgets.models import BudgetAllocation, BudgetPeriod, BudgetItem  
from core.models import Organization, OrganizationType, Project *# Assuming core models  
# Assuming utils are imported correctly*from Tanbakhsystem.utils import to\_english\_digits, format\_jalali\_date, parse\_jalali\_date  
  
logger = logging.getLogger(\_\_name\_\_)  
  
class BudgetAllocationForm(forms.ModelForm):  
 ALLOCATION\_TYPE\_CHOICES = [  
 ('amount', \_('مبلغ')),  
 ('percent', \_('درصد')),  
 ]  
 *# Assuming WARNING\_ACTION\_CHOICES are defined in the model now, use model's choices  
 # WARNING\_ACTION\_CHOICES = BudgetAllocation.WARNING\_ACTION\_CHOICES # Get choices from model* allocation\_type = forms.ChoiceField(  
 choices=ALLOCATION\_TYPE\_CHOICES,  
 widget=forms.RadioSelect(attrs={'class': 'form-check-input'}),  
 label=\_('نوع تخصیص'),  
 initial='amount',  
 required=True  
 )  
  
 *# Use CharField for date input to handle Jalali string* allocation\_date = forms.CharField(  
 label=\_('تاریخ تخصیص'), *# Changed label slightly* widget=forms.TextInput(attrs={  
 'data-jdp': '',  
 'class': 'form-control form-control-lg jalali-datepicker', *# Added jalali-datepicker class* 'autocomplete': 'off',  
 'placeholder': \_('مثال: 1404/01/26') *# Updated placeholder* }),  
 required=True  
 )  
  
 class Meta:  
 model = BudgetAllocation  
 fields = [  
 'budget\_period','budget\_item', 'organization', 'project', 'allocated\_amount',  
 'allocation\_date', 'description', 'is\_active', 'is\_stopped',  
 'allocation\_type', 'locked\_percentage', 'warning\_threshold',  
 'warning\_action', 'allocation\_number'  
 ]  
 widgets = {  
 'budget\_item': forms.Select(attrs={  
 'class': 'form-select form-select-lg',  
 'required': 'required',  
 'data-control': 'select2',  
 'data-placeholder': \_('انتخاب ردیف بودجه'),  
 'id': 'id\_budget\_item'  
 }),  
 'organization': forms.Select(attrs={  
 'class': 'form-select form-select-lg',  
 'required': 'required',  
 'data-control': 'select2',  
 'data-placeholder': \_('انتخاب سازمان'),  
 'id': 'id\_organization' *# برای دسترسی آسان در جاوااسکریپت* }),  
 'project': forms.Select(attrs={  
 'class': 'form-select form-select-lg',  
 'data-control': 'select2',  
 'data-placeholder': \_('انتخاب پروژه'),  
 'id': 'id\_project' *# برای دسترسی آسان در جاوااسکریپت* }),  
 'allocated\_amount': forms.TextInput(attrs={ *# Use TextInput for flexibility with formatting JS* 'class': 'form-control form-control-lg numeric-input',  
 'inputmode': 'decimal', *# Use decimal for amounts potentially with fractions* 'required': 'required',  
 'placeholder': \_('مقدار مبلغ یا درصد') *# Placeholder reflects type choice* }),  
 *# allocation\_date widget is defined above as CharField* 'description': forms.Textarea(attrs={  
 'class': 'form-control form-control-lg',  
 'rows': 3,  
 'placeholder': \_('توضیحات (اختیاری)')  
 }),  
 'is\_active': forms.CheckboxInput(attrs={'class': 'form-check-input', 'role': 'switch'}),  
 'is\_stopped': forms.CheckboxInput(attrs={'class': 'form-check-input', 'role': 'switch'}),  
 'locked\_percentage': forms.NumberInput(attrs={  
 'class': 'form-control form-control-lg', 'min': '0', 'max': '100', 'step': '0.01'  
 }),  
 'warning\_threshold': forms.NumberInput(attrs={  
 'class': 'form-control form-control-lg', 'min': '0', 'max': '100', 'step': '0.01'  
 }),  
 'warning\_action': forms.Select(attrs={'class': 'form-select form-select-lg'}), *# Use model choices* 'allocation\_number': forms.NumberInput(attrs={  
 'class': 'form-control form-control-lg', 'min': '0', 'step': '1'  
 }),  
 'budget\_period': forms.HiddenInput(),  
 }  
 labels = {  
 *# Labels are mostly defined by model verbose\_name, override if needed* 'budget\_item': \_(' ردیف بودجه '),  
 'organization': \_('سازمان/شعبه'),  
 'project': \_('پروژه (اختیاری)'),  
 'allocated\_amount': \_('مقدار (مبلغ/درصد)'),  
 'allocation\_date': \_('تاریخ تخصیص'),  
 'description': \_('شرح'),  
 'is\_active': \_('فعال'),  
 'is\_stopped': \_('متوقف شده'),  
 'locked\_percentage': \_('درصد قفل (%)'),  
 'warning\_threshold': \_('آستانه هشدار (%)'),  
 'warning\_action': \_('اقدام هشدار'),  
 'allocation\_number': \_('شماره تخصیص (اختیاری)'),  
 }  
 help\_texts = {  
 'locked\_percentage': \_('درصدی از بودجه که تخصیص دیگر امکان‌پذیر نیست.'),  
 'warning\_threshold': \_('درصدی از بودجه که باعث نمایش/ارسال هشدار می‌شود.'),  
 'project': \_('پروژه‌ای که این بودجه به آن مرتبط است (در صورت وجود).')  
 }  
  
 def \_\_init\_\_(self, \*args, user=None, budget\_period=None, \*\*kwargs):  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 self.user = user  
 self.budget\_period = budget\_period  
  
 *# تنظیم budget\_period* if self.budget\_period:  
 self.fields['budget\_period'].queryset = BudgetPeriod.objects.filter(id=self.budget\_period.id,  
 is\_active=True)  
 self.fields['budget\_item'].queryset = BudgetItem.objects.filter(budget\_period=self.budget\_period,  
 is\_active=True)  
 else:  
 self.fields['budget\_period'].queryset = BudgetPeriod.objects.filter(is\_active=True)  
 self.fields['budget\_item'].queryset = BudgetItem.objects.none()  
  
 self.fields['organization'].queryset = Organization.objects.filter(  
 org\_type\_\_is\_budget\_allocatable=True, is\_active=True  
 ).select\_related('org\_type').order\_by('name')  
 self.fields['project'].queryset = Project.objects.filter(is\_active=True).order\_by('name')  
 self.fields['project'].required = False  
 self.fields['project'].empty\_label = \_("--------- (بدون پروژه)")  
  
 if not self.initial.get('allocation\_date'):  
 if self.instance and self.instance.pk and self.instance.allocation\_date:  
 try:  
 self.initial['allocation\_date'] = jdatetime.date.fromgregorian(  
 date=self.instance.allocation\_date  
 ).strftime('%Y/%m/%d')  
 except Exception as e:  
 logger.error(f"Error formatting existing allocation date: {e}")  
 self.initial['allocation\_date'] = jdatetime.date.today().strftime('%Y/%m/%d')  
 else:  
 self.initial['allocation\_date'] = jdatetime.date.today().strftime('%Y/%m/%d')  
  
 if self.budget\_period and not self.instance.pk:  
 self.initial['locked\_percentage'] = self.budget\_period.locked\_percentage  
 self.initial['warning\_threshold'] = self.budget\_period.warning\_threshold  
 self.initial['warning\_action'] = self.budget\_period.warning\_action  
  
 *# تنظیم کوئری‌ست سازمان‌ها* try:  
 allowed\_org\_types = OrganizationType.objects.filter(  
 is\_budget\_allocatable=True  
 ).values\_list('id', flat=True)  
 self.fields['organization'].queryset = Organization.objects.filter(  
 org\_type\_\_in=allowed\_org\_types,  
 is\_active=True  
 ).select\_related('org\_type').order\_by('name')  
 logger.debug("Organization queryset set for active, allocatable orgs.")  
 except Exception as e:  
 logger.error(f"Error setting organization queryset: {e}")  
 self.fields['organization'].queryset = Organization.objects.none()  
  
 def clean\_organization(self):  
 organization = self.cleaned\_data.get('organization')  
 if organization and not organization.is\_active:  
 raise ValidationError(\_('سازمان انتخاب شده فعال نیست و نمی‌توان به آن بودجه تخصیص داد.'))  
 if organization and not organization.org\_type.is\_budget\_allocatable:  
 raise ValidationError(\_('این نوع سازمان قابلیت دریافت تخصیص بودجه ندارد.'))  
 return organization  
  
 def clean\_allocation\_date(self):  
 date\_str = self.cleaned\_data.get('allocation\_date')  
 if not date\_str:  
 raise ValidationError(\_('تاریخ تخصیص اجباری است.'))  
 try:  
 parsed\_date = parse\_jalali\_date(str(date\_str), field\_name=\_('تاریخ تخصیص'))  
 return parsed\_date  
 except ValueError as e:  
 logger.warning(f"Could not parse allocation\_date '{date\_str}': {e}")  
 raise ValidationError(e)  
 except Exception as e:  
 logger.error(f"Unexpected error parsing allocation\_date '{date\_str}': {e}", exc\_info=True)  
 raise ValidationError(\_('فرمت تاریخ تخصیص نامعتبر است.'))  
  
 def clean\_allocated\_amount(self):  
 amount\_input = self.cleaned\_data.get('allocated\_amount')  
 allocation\_type = self.cleaned\_data.get('allocation\_type')  
 budget\_period = self.cleaned\_data.get('budget\_period') or self.budget\_period  
  
 if amount\_input is None:  
 raise ValidationError(\_('مقدار تخصیص نمی‌تواند خالی باشد.'))  
  
 try:  
 amount\_decimal = Decimal(str(amount\_input))  
 except Exception:  
 raise ValidationError(\_('مقدار وارد شده عدد معتبری نیست.'))  
  
 if amount\_decimal <= 0:  
 raise ValidationError(\_('مقدار تخصیص باید مثبت باشد.'))  
  
 if allocation\_type == 'percent' and not (0 <= amount\_decimal <= 100):  
 raise ValidationError(\_('درصد تخصیص باید بین ۰ تا ۱۰۰ باشد.'))  
  
 return amount\_decimal  
  
 def clean(self):  
 cleaned\_data = super().clean()  
 organization = cleaned\_data.get('organization')  
 budget\_item = cleaned\_data.get('budget\_item')  
 project = cleaned\_data.get('project')  
 allocated\_amount = cleaned\_data.get('allocated\_amount')  
 allocation\_date = cleaned\_data.get('allocation\_date')  
 budget\_period = cleaned\_data.get('budget\_period') or self.budget\_period  
 allocation\_type = cleaned\_data.get('allocation\_type')  
  
 if budget\_item and organization:  
 if budget\_item.organization != organization:  
 self.add\_error('budget\_item', \_('ردیف بودجه باید متعلق به شعبه انتخاب‌شده باشد.'))  
  
 if not budget\_period:  
 raise ValidationError(\_('دوره بودجه مشخص نیست.'))  
  
 if project and organization:  
 if not project.organizations.filter(id=organization.id).exists():  
 self.add\_error('project', \_('پروژه انتخاب‌شده به سازمان انتخاب‌شده تعلق ندارد.'))  
  
 *# محاسبه مبلغ موثر* if allocation\_type == 'percent' and allocated\_amount is not None:  
 effective\_amount = (allocated\_amount / Decimal('100')) \* budget\_period.total\_amount  
 cleaned\_data['effective\_amount'] = effective\_amount  
 elif allocated\_amount is not None:  
 cleaned\_data['effective\_amount'] = allocated\_amount  
  
 *# اعتبارسنجی بودجه دوره* if cleaned\_data.get('effective\_amount') and budget\_period:  
 used\_budget = BudgetAllocation.objects.filter(budget\_period=budget\_period).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
 if self.instance.pk:  
 current\_allocation = BudgetAllocation.objects.filter(pk=self.instance.pk).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
 used\_budget -= current\_allocation  
 remaining\_budget = budget\_period.total\_amount - used\_budget  
 if cleaned\_data['effective\_amount'] > remaining\_budget:  
 self.add\_error('allocated\_amount', \_(  
 f'مبلغ تخصیص ({cleaned\_data["effective\_amount"]:,.0f} ریال) بیشتر از بودجه باقی‌مانده ({remaining\_budget:,.0f} ریال) است.'  
 ))  
  
 if allocation\_date and budget\_period:  
 if not (budget\_period.start\_date <= allocation\_date <= budget\_period.end\_date):  
 self.add\_error('allocation\_date', \_(  
 f'تاریخ تخصیص باید در بازه {budget\_period.start\_date} تا {budget\_period.end\_date} باشد.'  
 ))  
  
 if cleaned\_data.get('warning\_threshold') is not None and not (  
 0 <= cleaned\_data.get('warning\_threshold') <= 100):  
 self.add\_error('warning\_threshold', \_('آستانه اخطار باید بین ۰ تا ۱۰۰ باشد.'))  
  
 return cleaned\_data  
  
 def save(self, commit=True):  
 instance = super().save(commit=False)  
 if not self.cleaned\_data.get('budget\_period') and self.budget\_period:  
 instance.budget\_period = self.budget\_period  
 elif not instance.budget\_period\_id:  
 raise ValidationError(\_('دوره بودجه اجباری است.'))  
  
 instance.allocated\_amount = self.cleaned\_data['effective\_amount']  
 if not instance.pk and self.user and self.user.is\_authenticated:  
 instance.created\_by = self.user  
  
 with transaction.atomic():  
 instance = super().save(commit=commit)  
 if commit and instance.budget\_period\_id:  
 total\_allocated = BudgetAllocation.objects.filter(  
 budget\_period=instance.budget\_period  
 ).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 instance.budget\_period.total\_allocated = total\_allocated  
 instance.budget\_period.save(update\_fields=['total\_allocated'])  
  
 return instance  
 *# # تنظیم کوئری‌ست پروژه‌ها  
 # try:  
 # self.fields['project'].queryset = Project.objects.filter(is\_active=True).order\_by('name')  
 # self.fields['project'].empty\_label = \_("--------- (بدون پروژه)")  
 # self.fields['project'].required = False  
 # logger.debug("Initial project queryset set for all active projects.")  
 # except Exception as e:  
 # logger.error(f"Error setting project queryset: {e}")  
 # self.fields['project'].queryset = Project.objects.none()  
 #  
 # # تنظیم تاریخ اولیه  
 # if not self.initial.get('allocation\_date'):  
 # if self.instance and self.instance.pk and self.instance.allocation\_date:  
 # try:  
 # self.initial['allocation\_date'] = jdatetime.date.fromgregorian(  
 # date=self.instance.allocation\_date  
 # ).strftime('%Y/%m/%d')  
 # logger.debug(f"Set initial allocation\_date from instance: {self.initial['allocation\_date']}")  
 # except Exception as e:  
 # logger.error(f"Error formatting existing allocation date: {e}")  
 # self.initial['allocation\_date'] = jdatetime.date.today().strftime('%Y/%m/%d')  
 # else:  
 # self.initial['allocation\_date'] = jdatetime.date.today().strftime('%Y/%m/%d')  
 # logger.debug(f"Set default allocation\_date for new instance: {self.initial['allocation\_date']}")  
 #  
 # # تنظیم مقادیر اولیه برای هشدار و قفل  
 # if self.budget\_period and not self.instance.pk:  
 # self.initial['locked\_percentage'] = self.budget\_period.locked\_percentage  
 # self.initial['warning\_threshold'] = self.budget\_period.warning\_threshold  
 # self.initial['warning\_action'] = self.budget\_period.warning\_action  
 # logger.debug("Set initial warning/lock values from budget period.")  
 #  
 # def clean\_organization(self):  
 # """Ensure the selected organization is active."""  
 # organization = self.cleaned\_data.get('organization')  
 # if organization and not organization.is\_active:  
 # logger.error(f"Selected organization '{organization.name}' (ID: {organization.id}) is not active.")  
 # raise ValidationError(\_('سازمان انتخاب شده فعال نیست و نمی‌توان به آن بودجه تخصیص داد.'))  
 # # Also check if it's allocatable type (redundant if queryset is correct, but safe)  
 # if organization and not organization.org\_type.is\_budget\_allocatable:  
 # logger.error(f"Selected organization '{organization.name}' type '{organization.org\_type.name}' is not budget allocatable.")  
 # raise ValidationError(\_('این نوع سازمان قابلیت دریافت تخصیص بودجه ندارد.'))  
 # return organization  
 #  
 # def clean\_allocation\_date(self):  
 # """Parse Jalali date string to Python date object."""  
 # date\_str = self.cleaned\_data.get('allocation\_date')  
 # logger.debug(f"Cleaning allocation\_date: input='{date\_str}'")  
 # if not date\_str:  
 # # This might be redundant if field is required=True  
 # raise ValidationError(\_('تاریخ تخصیص اجباری است.'))  
 # try:  
 # # Use the utility function to parse  
 # parsed\_date = parse\_jalali\_date(str(date\_str), field\_name=\_('تاریخ تخصیص'))  
 # logger.debug(f"Parsed allocation\_date: {parsed\_date}")  
 # return parsed\_date # Return the Python date object  
 # except ValueError as e: # Catch specific parsing errors  
 # logger.warning(f"Could not parse allocation\_date '{date\_str}': {e}")  
 # raise ValidationError(e) # Show the specific error from parse\_jalali\_date  
 # except Exception as e: # Catch other unexpected errors  
 # logger.error(f"Unexpected error parsing allocation\_date '{date\_str}': {e}", exc\_info=True)  
 # raise ValidationError(\_('فرمت تاریخ تخصیص نامعتبر است.'))  
 #  
 # def clean\_allocated\_amount(self):  
 # amount\_input = self.cleaned\_data.get('allocated\_amount')  
 # allocation\_type = self.cleaned\_data.get('allocation\_type')  
 # budget\_item = self.cleaned\_data.get('budget\_item')  
 # budget\_period = self.cleaned\_data.get('budget\_period') or self.budget\_period  
 # logger.debug(f"Cleaning allocated\_amount: input={amount\_input}, allocation\_type={allocation\_type}")  
 #  
 # if amount\_input is None:  
 # raise ValidationError(\_('مقدار تخصیص نمی‌تواند خالی باشد.'))  
 #  
 # try:  
 # amount\_decimal = Decimal(str(amount\_input))  
 # except Exception:  
 # raise ValidationError(\_('مقدار وارد شده عدد معتبری نیست.'))  
 #  
 # if amount\_decimal <= 0:  
 # raise ValidationError(\_('مقدار تخصیص باید مثبت باشد.'))  
 #  
 # if allocation\_type == 'percent' and not (0 <= amount\_decimal <= 100):  
 # raise ValidationError(\_('درصد تخصیص باید بین ۰ تا ۱۰۰ باشد.'))  
 #  
 # if allocation\_type == 'percent' and budget\_item:  
 # total\_period\_amount = budget\_item.total\_amount  
 # calculated\_amount = (amount\_decimal / Decimal('100')) \* total\_period\_amount  
 # else:  
 # calculated\_amount = amount\_decimal  
 #  
 # # if budget\_item:  
 # # remaining\_budget = budget\_item.get\_remaining\_amount()  
 # # if self.instance.pk:  
 # # current\_allocation = BudgetAllocation.objects.filter(pk=self.instance.pk).aggregate(  
 # # total=Sum('allocated\_amount')  
 # # )['total'] or Decimal('0')  
 # # remaining\_budget += current\_allocation  
 # # if calculated\_amount > remaining\_budget:  
 # # raise ValidationError(\_(  
 # # f"مبلغ تخصیص ({calculated\_amount:,.0f} ریال) بیشتر از باقی‌مانده ردیف بودجه ({remaining\_budget:,.0f} ریال) است."  
 # # ))  
 #  
 # logger.debug(f"Validated allocated\_amount: {amount\_decimal}")  
 # return amount\_decimal  
 #  
 # def clean(self):  
 # cleaned\_data = super().clean()  
 # logger.debug(f"Form clean: cleaned\_data keys={list(cleaned\_data.keys())}")  
 # organization = cleaned\_data.get('organization')  
 # budget\_item = cleaned\_data.get('budget\_item')  
 # project = cleaned\_data.get('project')  
 # allocated\_amount = cleaned\_data.get('allocated\_amount')  
 # allocation\_date = cleaned\_data.get('allocation\_date')  
 # budget\_period = cleaned\_data.get('budget\_period') or self.budget\_period  
 # allocation\_type = cleaned\_data.get('allocation\_type')  
 #  
 # if budget\_item and organization:  
 # if budget\_item.organization != organization:  
 # self.add\_error('budget\_item', \_('ردیف بودجه باید متعلق به شعبه انتخاب‌شده باشد.'))  
 #  
 # if not budget\_period:  
 # raise ValidationError(\_('دوره بودجه مشخص نیست.'))  
 #  
 # if project and organization:  
 # if not project.organizations.filter(id=organization.id).exists():  
 # self.add\_error('project', \_('پروژه انتخاب‌شده به سازمان انتخاب‌شده تعلق ندارد.'))  
 #  
 # if allocation\_type == 'percent' and allocated\_amount is not None:  
 # effective\_amount = (allocated\_amount / Decimal('100')) \* budget\_period.total\_amount  
 # logger.info(f"Calculated percent: {allocated\_amount}% = {effective\_amount} Riyals")  
 # cleaned\_data['effective\_amount'] = effective\_amount  
 # elif allocated\_amount is not None:  
 # cleaned\_data['effective\_amount'] = allocated\_amount  
 # logger.debug(f"Effective amount: {allocated\_amount} Riyals")  
 #  
 # if cleaned\_data.get('effective\_amount') and budget\_period:  
 # used\_budget = BudgetAllocation.objects.filter(budget\_period=budget\_period).aggregate(  
 # total=Sum('allocated\_amount')  
 # )['total'] or Decimal('0')  
 # remaining\_budget = budget\_period.total\_amount - used\_budget  
 # logger.debug(f"Budget period {budget\_period.id}: total={budget\_period.total\_amount}, used={used\_budget}, remaining={remaining\_budget}")  
 # if cleaned\_data['effective\_amount'] > remaining\_budget:  
 # self.add\_error('allocated\_amount', \_(  
 # f'مبلغ تخصیص ({cleaned\_data["effective\_amount"]:,.0f} ریال) بیشتر از بودجه باقی‌مانده ({remaining\_budget:,.0f} ریال) است.'  
 # ))  
 #  
 # if allocation\_date and budget\_period:  
 # if not (budget\_period.start\_date <= allocation\_date <= budget\_period.end\_date):  
 # self.add\_error('allocation\_date', \_(  
 # f'تاریخ تخصیص باید در بازه {budget\_period.start\_date} تا {budget\_period.end\_date} باشد.'  
 # ))  
 #  
 # effective\_amount = cleaned\_data.get('effective\_amount')  
 # if budget\_item and effective\_amount and hasattr(budget\_item, 'total\_amount'):  
 # remaining = budget\_item.get\_remaining\_amount()  
 # if effective\_amount > remaining:  
 # self.add\_error('budget\_item', \_(  
 # f"مبلغ تخصیص ({effective\_amount:,.0f} ریال) بیشتر از باقی‌مانده ردیف ({remaining:,.0f} ریال) است."  
 # ))  
 #  
 # if cleaned\_data.get('warning\_threshold') is not None and not (0 <= cleaned\_data.get('warning\_threshold') <= 100):  
 # self.add\_error('warning\_threshold', \_('آستانه اخطار باید بین ۰ تا ۱۰۰ باشد.'))  
 #  
 # logger.debug("Form clean completed")  
 # return cleaned\_data  
 #  
 # def save(self, commit=True):  
 # instance = super().save(commit=False)  
 # logger.debug(f"Saving BudgetAllocationForm (commit={commit}, user={self.user})")  
 # try:  
 # if not self.cleaned\_data.get('budget\_period') and self.budget\_period:  
 # self.instance.budget\_period = self.budget\_period  
 # logger.debug(f"Set budget\_period: {self.budget\_period.id}")  
 # elif not self.instance.budget\_period\_id:  
 # logger.error("No budget\_period provided")  
 # raise ValidationError("Budget period is required.")  
 # #  
 # # allocation\_type = self.cleaned\_data.get('allocation\_type')  
 # # allocated\_amount = self.cleaned\_data.get('allocated\_amount')  
 # # budget\_item = self.cleaned\_data.get('budget\_item')  
 # #  
 # # if allocation\_type == 'percent' and allocated\_amount is not None:  
 # # total\_period\_amount = self.instance.budget\_period.total\_amount  
 # # calculated\_amount = (allocated\_amount / Decimal('100')) \* total\_period\_amount  
 # # self.instance.allocated\_amount = calculated\_amount  
 # # logger.info(f"Set allocated\_amount from percent: {allocated\_amount}% = {calculated\_amount} Riyals")  
 # # else:  
 # # self.instance.allocated\_amount = allocated\_amount  
 # # logger.debug(f"Set allocated\_amount: {allocated\_amount} Riyals")  
 #  
 # instance.allocated\_amount = self.cleaned\_data['effective\_amount']  
 # if not self.instance.pk and self.user and self.user.is\_authenticated:  
 # self.instance.created\_by = self.user  
 # logger.debug(f"Set created\_by: {self.user}")  
 #  
 # with transaction.atomic():  
 # instance = super().save(commit=commit)  
 # logger.info(f"Form saved BudgetAllocation with ID: {instance.pk}")  
 # if commit and instance.budget\_period\_id:  
 # total\_allocated = BudgetAllocation.objects.filter(  
 # budget\_period=instance.budget\_period  
 # ).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 # instance.budget\_period.total\_allocated = total\_allocated  
 # instance.budget\_period.save(update\_fields=['total\_allocated'])  
 # logger.info(f"Updated BudgetPeriod {instance.budget\_period\_id} with total\_allocated={total\_allocated}")  
 #  
 # return instance  
 # except Exception as e:  
 # logger.error(f"Error in BudgetAllocationForm.save: {str(e)}", exc\_info=True)  
 # self.add\_error(None, str(e))  
 # raise  
 #  
 # def get(self, request, \*args, \*\*kwargs):  
 # budget\_period = self.\_get\_budget\_period()  
 # if not budget\_period:  
 # from django.contrib import messages  
 # messages.error(request, \_('دوره بودجه معتبر انتخاب نشده است.'))  
 # from django.shortcuts import redirect  
 # return redirect('budgetperiod\_list')  
 # return super().get(request, \*args, \*\*kwargs)*

from django.http import JsonResponse  
from django.views.decorators.http import require\_GET  
from core.models import Project, Organization  
from django.contrib.auth.decorators import login\_required  
from django.http import JsonResponse  
from budgets.models import BudgetItem  
from django.db.models import Sum  
from decimal import Decimal  
  
import logging  
  
logger = logging.getLogger(\_\_name\_\_)  
  
@require\_GET  
@login\_required  
def get\_projects\_by\_organization(request):  
 organization\_id = request.GET.get('organization\_id')  
 if not organization\_id:  
 return JsonResponse({'projects': []}, status=400)  
  
 try:  
 organization = Organization.objects.get(id=organization\_id)  
 *# فرض بر این است که Project رابطه ManyToMany با Organization دارد* projects = Project.objects.filter(  
 organizations=organization, *# فیلتر پروژه‌های مرتبط* is\_active=True  
 ).values('id', 'name').order\_by('name')  
 return JsonResponse({'projects': list(projects)})  
 except Organization.DoesNotExist:  
 return JsonResponse({'projects': []}, status=404)  
 except Exception as e:  
 logger.error(f"Error fetching projects for organization {organization\_id}: {e}")  
 return JsonResponse({'projects': []}, status=500)  
  
def get\_budget\_items\_by\_organization(request):  
 organization\_id = request.GET.get('organization\_id')  
 budget\_period\_id = request.GET.get('budget\_period\_id')  
 if not organization\_id or not budget\_period\_id:  
 return JsonResponse({'budget\_items': [], 'total\_budget': 0})  
  
 budget\_items = BudgetItem.objects.filter(  
 organization\_id=organization\_id,  
 budget\_period\_id=budget\_period\_id,  
 is\_active=True  
 ).select\_related('organization')  
  
 total\_budget = budget\_items.aggregate(total=Sum('total\_amount'))['total'] or Decimal('0')  
  
 data = [{  
 'id': item.id,  
 'name': item.name,  
 'organization\_name': item.organization.name  
 } for item in budget\_items]  
  
 return JsonResponse({'budget\_items': data, 'total\_budget': float(total\_budget)})  
  
def get\_budget\_item\_remaining(request):  
 budget\_item\_id = request.GET.get('budget\_item\_id')  
 if not budget\_item\_id:  
 return JsonResponse({'remaining\_amount': 0})  
  
 try:  
 budget\_item = BudgetItem.objects.get(id=budget\_item\_id, is\_active=True)  
 remaining = budget\_item.get\_remaining\_amount()  
 return JsonResponse({'remaining\_amount': float(remaining)})  
 except BudgetItem.DoesNotExist:  
 return JsonResponse({'remaining\_amount': 0})  
  
def get\_budget\_item\_details(request):  
 budget\_item\_id = request.GET.get('budget\_item\_id')  
 if not budget\_item\_id:  
 return JsonResponse({'total\_amount': 0, 'remaining\_amount': 0})  
  
 try:  
 budget\_item = BudgetItem.objects.get(id=budget\_item\_id, is\_active=True)  
 return JsonResponse({  
 'total\_amount': float(budget\_item.total\_amount),  
 'remaining\_amount': float(budget\_item.get\_remaining\_amount())  
 })  
 except BudgetItem.DoesNotExist:  
 return JsonResponse({'total\_amount': 0, 'remaining\_amount': 0})

from django.core.exceptions import ValidationError  
from django.views.generic.edit import CreateView  
from django.urls import reverse\_lazy  
from django.utils.translation import gettext\_lazy as \_  
from django.contrib import messages  
from django.db.models import Sum, Q  
from django.db import transaction  
  
from Tanbakhsystem.utils import parse\_jalali\_date, to\_english\_digits  
from budgets.BudgetAllocation.forms\_BudgetAllocation import BudgetAllocationForm  
from django.views.generic import ListView, DetailView, UpdateView, DeleteView, CreateView, TemplateView  
  
from decimal import Decimal  
import logging  
from django.http import JsonResponse  
  
from budgets.budget\_calculations import calculate\_allocation\_percentages  
from budgets.models import BudgetPeriod, BudgetAllocation, BudgetTransaction, BudgetItem, ProjectBudgetAllocation  
from core.PermissionBase import PermissionBaseView  
from core.models import Organization, Project, OrganizationType  
  
logger = logging.getLogger(\_\_name\_\_)  
def get\_projects\_by\_organization(request):  
 *"""ویو برای دریافت پروژه‌های یک سازمان به صورت AJAX"""* org\_id = request.GET.get('organization\_id')  
  
 if org\_id:  
 try:  
 projects = Project.objects.filter(  
 organizations\_\_id=org\_id,  
 is\_active=True  
 ).values('id', 'name', 'code')  
 return JsonResponse({'projects': list(projects)})  
 except ValueError:  
 pass  
  
 return JsonResponse({'projects': []})  
 *# ---------------------------------  
# --- BudgetAllocation CRUD ---*class BudgetAllocationCreateView(PermissionBaseView, CreateView):  
 model = BudgetAllocation  
 form\_class = BudgetAllocationForm  
 template\_name = 'budgets/budget/budgetallocation\_form.html'  
 success\_url = reverse\_lazy('budgetallocation\_list')  
 permission\_required = 'budgets.add\_budgetallocation'  
  
 def \_get\_budget\_period(self):  
 budget\_period\_id = self.request.GET.get('budget\_period') or self.request.POST.get('budget\_period')  
 logger.debug(f"Getting budget\_period with ID: {budget\_period\_id}")  
 if not budget\_period\_id:  
 logger.warning("No budget\_period\_id provided")  
 return None  
 try:  
 budget\_period = BudgetPeriod.objects.get(id=budget\_period\_id, is\_active=True)  
 logger.info(f"Retrieved budget\_period: ID={budget\_period.id}, name={budget\_period.name}")  
 return budget\_period  
 except (BudgetPeriod.DoesNotExist, ValueError) as e:  
 logger.warning(f"Invalid budget\_period\_id: {budget\_period\_id}, error: {str(e)}")  
 return None  
  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 budget\_period\_id = self.request.GET.get('budget\_period')  
 budget\_period = self.\_get\_budget\_period()  
  
 context['title'] = \_('ایجاد تخصیص بودجه جدید')  
  
 *# سازمان‌های مجاز* allowed\_org\_types = OrganizationType.objects.filter(is\_budget\_allocatable=True).values\_list('id', flat=True)  
 context['organizations'] = Organization.objects.filter(  
 org\_type\_\_in=allowed\_org\_types, is\_active=True  
 ).select\_related('org\_type')  
  
 if budget\_period:  
 context['budget\_period'] = budget\_period  
 context['budget\_items'] = BudgetItem.objects.filter(  
 budget\_period=budget\_period,  
 is\_active=True  
 ).select\_related('organization')  
 *# بودجه کل از BudgetPeriod* context['total\_amount'] = budget\_period.total\_amount or Decimal('0')  
 *# باقی‌مانده دوره* context['remaining\_amount'] = budget\_period.get\_remaining\_amount() or Decimal('0')  
 context['remaining\_percent'] = (  
 (context['remaining\_amount'] / context['total\_amount'] \* 100)  
 if context['total\_amount'] else 0  
 )  
 context['locked\_percentage'] = budget\_period.locked\_percentage or 0  
 context['warning\_threshold'] = budget\_period.warning\_threshold or 10  
 logger.info(f"Budget period data: total={context['total\_amount']}, remaining={context['remaining\_amount']}")  
 else:  
 context['budget\_period'] = None  
 context['budget\_items'] = BudgetItem.objects.none()  
 context['total\_amount'] = Decimal('0')  
 context['remaining\_amount'] = Decimal('0')  
 context['remaining\_percent'] = 0  
 context['locked\_percentage'] = 0  
 context['warning\_threshold'] = 10  
 messages.warning(self.request, \_('دوره بودجه انتخاب‌شده نامعتبر است.'))  
  
 context['projects'] = Project.objects.filter(is\_active=True).select\_related('category')  
 logger.debug(f"Loaded {context['projects'].count()} projects")  
  
 return context  
  
  
 def get\_form\_kwargs(self):  
 kwargs = super().get\_form\_kwargs()  
 budget\_period = self.\_get\_budget\_period()  
 if not budget\_period:  
 logger.error("No valid budget period for form")  
 messages.error(self.request, \_("دوره بودجه معتبر انتخاب نشده است."))  
 kwargs['budget\_period'] = None  
 else:  
 kwargs['budget\_period'] = budget\_period  
 kwargs['user'] = self.request.user  
 return kwargs  
  
  
 def form\_valid(self, form):  
 logger.debug("Form is valid, starting save process")  
 try:  
 with transaction.atomic():  
 response = super().form\_valid(form)  
 logger.info(f"BudgetAllocation created with ID: {self.object.pk}")  
 messages.success(self.request, \_('تخصیص بودجه با موفقیت ثبت شد.'))  
 return response  
 except ValidationError as e:  
 logger.error(f"Validation error saving budget allocation: {str(e)}")  
 form.add\_error(None, str(e))  
 return self.form\_invalid(form)  
 except Exception as e:  
 logger.error(f"Unexpected error saving budget allocation: {str(e)}", exc\_info=True)  
 form.add\_error(None, \_('خطایی در ثبت تخصیص بودجه رخ داد: ') + str(e))  
 return self.form\_invalid(form)  
  
  
 def form\_invalid(self, form):  
 logger.error(f"Form invalid: errors={form.errors.as\_json()}")  
 messages.error(self.request, \_('لطفاً خطاهای فرم را بررسی کنید.'))  
 return self.render\_to\_response(self.get\_context\_data(form=form))  
  
class BudgetAllocationUpdateView(PermissionBaseView, UpdateView):  
 model = BudgetAllocation  
 form\_class = BudgetAllocationForm  
 template\_name = 'budgets/budget/budgetallocation\_form.html'  
 success\_url = reverse\_lazy('budgetallocation\_list')  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['title'] = \_('ویرایش تخصیص بودجه')  
 context['budget\_period'] = self.object.budget\_period  
 context['total\_amount'] = self.object.budget\_period.total\_amount or Decimal('0')  
 context['remaining\_amount'] = self.object.budget\_period.get\_remaining\_amount() + self.object.allocated\_amount  
 context['remaining\_percent'] = (  
 (context['remaining\_amount'] / context['total\_amount'] \* 100)  
 if context['total\_amount'] else 0  
 )  
 context['locked\_percentage'] = self.object.budget\_period.locked\_percentage or 0  
 context['warning\_threshold'] = self.object.budget\_period.warning\_threshold or 10  
 context['budget\_items'] = BudgetItem.objects.filter(  
 budget\_period=self.object.budget\_period,  
 is\_active=True  
 ).select\_related('organization')  
 context['organizations'] = Organization.objects.filter(is\_active=True).select\_related('org\_type')  
 context['projects'] = Project.objects.filter(is\_active=True).select\_related('category')  
 return context  
  
 @transaction.atomic  
 def form\_valid(self, form):  
 old\_amount = self.get\_object().allocated\_amount  
 new\_amount = form.instance.allocated\_amount  
 difference = new\_amount - old\_amount  
 remaining = self.get\_object().budget\_period.get\_remaining\_amount() + old\_amount  
 if new\_amount > remaining:  
 messages.error(self.request, f'مبلغ تخصیص بیشتر از باقی‌مانده بودجه ({remaining:,.0f} ریال) است.')  
 return self.form\_invalid(form)  
  
 response = super().form\_valid(form)  
 if difference != 0:  
 transaction\_type = 'ADJUSTMENT\_INCREASE' if difference > 0 else 'ADJUSTMENT\_DECREASE'  
 BudgetTransaction.objects.create(  
 allocation=form.instance,  
 transaction\_type=transaction\_type,  
 amount=abs(difference),  
 created\_by=self.request.user,  
 description=f'تغییر تخصیص بودجه به {form.instance.organization.name}'  
 )  
 messages.success(self.request, f'تخصیص بودجه به {form.instance.organization.name} با موفقیت به‌روزرسانی شد.')  
 return response  
  
class BudgetAllocationListView(PermissionBaseView, ListView):  
 model = BudgetAllocation  
 template\_name = 'budgets/budget/budgetallocation\_list.html'  
 context\_object\_name = 'budget\_allocations'  
 paginate\_by = 10  
 permission\_required = 'budgets.view\_budgetallocation'  
  
 def get\_queryset(self):  
 *# همان کد قبلی بدون تغییر* queryset = BudgetAllocation.objects.all().select\_related('organization', 'project', 'budget\_period',  
 'budget\_item')  
 logger.info(f"BudgetAllocationListView queryset: {queryset.count()} records")  
  
 query = self.request.GET.get('q', '')  
 date\_from = self.request.GET.get('date\_from', '')  
 date\_to = self.request.GET.get('date\_to', '')  
 budget\_item\_id = self.request.GET.get('budget\_item')  
 organization\_id = self.request.GET.get('organization')  
  
 if query:  
 queryset = queryset.filter(  
 Q(budget\_period\_\_name\_\_icontains=query) |  
 Q(budget\_item\_\_name\_\_icontains=query) |  
 Q(organization\_\_name\_\_icontains=query) |  
 Q(project\_\_name\_\_icontains=query)  
 )  
  
 if budget\_item\_id:  
 queryset = queryset.filter(budget\_item\_id=budget\_item\_id)  
 if organization\_id:  
 queryset = queryset.filter(organization\_id=organization\_id)  
  
 if date\_from:  
 try:  
 date\_from = parse\_jalali\_date(to\_english\_digits(date\_from))  
 queryset = queryset.filter(allocation\_date\_\_gte=date\_from)  
 except ValueError:  
 logger.warning(f"Invalid date\_from format: {date\_from}")  
  
 if date\_to:  
 try:  
 date\_to = parse\_jalali\_date(to\_english\_digits(date\_to))  
 queryset = queryset.filter(allocation\_date\_\_lte=date\_to)  
 except ValueError:  
 logger.warning(f"Invalid date\_to format: {date\_to}")  
  
 return queryset.order\_by('-allocation\_date')  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 organization\_id = self.request.GET.get('organization')  
 budget\_period\_id = self.request.GET.get('budget\_period')  
 budget\_item\_id = self.request.GET.get('budget\_item')  
 project\_id = self.request.GET.get('project') *# اضافه کردن فیلتر پروژه  
  
 # مقداردهی اولیه* total\_budget = Decimal('0')  
 remaining\_budget = Decimal('0')  
 allocated\_percentage = Decimal('0')  
 remaining\_percentage = Decimal('0')  
 total\_allocated = Decimal('0')  
  
 *# محاسبه بودجه دوره* if budget\_period\_id:  
 try:  
 budget\_period = BudgetPeriod.objects.get(id=budget\_period\_id, is\_active=True)  
 total\_budget = budget\_period.total\_amount or Decimal('0')  
 remaining\_budget = budget\_period.get\_remaining\_amount() or Decimal('0')  
 total\_allocated = BudgetAllocation.objects.filter(budget\_period=budget\_period).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
  
 if total\_budget > 0:  
 allocated\_percentage = (total\_allocated / total\_budget) \* Decimal('100')  
 remaining\_percentage = (remaining\_budget / total\_budget) \* Decimal('100')  
  
 logger.info(  
 f"Budget period {budget\_period.id}: total={total\_budget}, "  
 f"remaining={remaining\_budget}, allocated={total\_allocated}, "  
 f"allocated\_percentage={allocated\_percentage:.2f}%, "  
 f"remaining\_percentage={remaining\_percentage:.2f}%"  
 )  
 except BudgetPeriod.DoesNotExist:  
 logger.warning(f"Invalid budget\_period\_id: {budget\_period\_id}")  
 context['status\_message'] = \_('دوره بودجه انتخاب‌شده نامعتبر است.')  
 else:  
 total\_allocated = context['budget\_allocations'].aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
 total\_budget = BudgetPeriod.objects.filter(is\_active=True).aggregate(  
 total=Sum('total\_amount')  
 )['total'] or Decimal('0')  
 remaining\_budget = BudgetPeriod.objects.filter(is\_active=True).aggregate(  
 total=Sum('total\_amount') - Sum('total\_allocated')  
 )['total'] or Decimal('0')  
  
 if total\_budget > 0:  
 allocated\_percentage = (total\_allocated / total\_budget) \* Decimal('100')  
 remaining\_percentage = (remaining\_budget / total\_budget) \* Decimal('100')  
  
 *# محاسبه اطلاعات ردیف بودجه* budget\_item\_data = {}  
 if budget\_item\_id:  
 try:  
 budget\_item = BudgetItem.objects.get(id=budget\_item\_id, is\_active=True)  
 budget\_period = budget\_item.budget\_period  
 budget\_item\_total = budget\_period.total\_amount or Decimal('0')  
 budget\_item\_allocated = BudgetAllocation.objects.filter(budget\_item=budget\_item).aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
 budget\_item\_remaining = budget\_item\_total - budget\_item\_allocated  
 budget\_item\_allocated\_percentage = (  
 (budget\_item\_allocated / budget\_item\_total \* Decimal('100'))  
 if budget\_item\_total > 0 else Decimal('0')  
 )  
 budget\_item\_remaining\_percentage = (  
 (budget\_item\_remaining / budget\_item\_total \* Decimal('100'))  
 if budget\_item\_total > 0 else Decimal('0')  
 )  
  
 budget\_item\_data = {  
 'total': budget\_item\_total,  
 'remaining': budget\_item\_remaining,  
 'allocated': budget\_item\_allocated,  
 'allocated\_percentage': budget\_item\_allocated\_percentage,  
 'remaining\_percentage': budget\_item\_remaining\_percentage,  
 }  
 except BudgetItem.DoesNotExist:  
 logger.warning(f"Invalid budget\_item\_id: {budget\_item\_id}")  
 context['status\_message'] = \_('ردیف بودجه انتخاب‌شده نامعتبر است.')  
  
 *# محاسبه اطلاعات پروژه* project\_data = {}  
 if project\_id:  
 try:  
 project = Project.objects.get(id=project\_id, is\_active=True)  
 project\_allocations = ProjectBudgetAllocation.objects.filter(project=project)  
 project\_total = project\_allocations.aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
 project\_remaining = project.get\_remaining\_budget() or Decimal('0')  
 project\_allocated\_percentage = (  
 (project\_total / project\_total \* Decimal('100'))  
 if project\_total > 0 else Decimal('0')  
 )  
 project\_remaining\_percentage = (  
 (project\_remaining / project\_total \* Decimal('100'))  
 if project\_total > 0 else Decimal('0')  
 )  
  
 project\_data = {  
 'total': project\_total,  
 'remaining': project\_remaining,  
 'allocated': project\_total,  
 'allocated\_percentage': project\_allocated\_percentage,  
 'remaining\_percentage': project\_remaining\_percentage,  
 }  
 logger.info(  
 f"Project {project.id}: total={project\_total}, "  
 f"remaining={project\_remaining}, allocated\_percentage={project\_allocated\_percentage:.2f}%, "  
 f"remaining\_percentage={project\_remaining\_percentage:.2f}%"  
 )  
 except Project.DoesNotExist:  
 logger.warning(f"Invalid project\_id: {project\_id}")  
 context['status\_message'] = \_('پروژه انتخاب‌شده نامعتبر است.')  
  
 context.update({  
 'total\_budget': total\_budget,  
 'remaining\_budget': remaining\_budget,  
 'total\_allocated': total\_allocated,  
 'allocated\_percentage': allocated\_percentage,  
 'remaining\_percentage': remaining\_percentage,  
 'budget\_item\_data': budget\_item\_data,  
 'project\_data': project\_data, *# اضافه کردن اطلاعات پروژه* 'budget\_items': BudgetItem.objects.filter(is\_active=True).select\_related('organization', 'budget\_period'),  
 'organizations': Organization.objects.filter(is\_active=True),  
 'budget\_periods': BudgetPeriod.objects.filter(is\_active=True),  
 'projects': Project.objects.filter(is\_active=True), *# اضافه کردن پروژه‌ها برای فیلتر* 'status': 'filtered',  
 'status\_message': context.get('status\_message', \_('نمایش تخصیص‌های فیلترشده')),  
 'query': self.request.GET.get('q', ''),  
 'date\_from': self.request.GET.get('date\_from', ''),  
 'date\_to': self.request.GET.get('date\_to', ''),  
 'selected\_budget\_item': budget\_item\_id,  
 'selected\_organization': organization\_id,  
 'selected\_budget\_period': budget\_period\_id,  
 'selected\_project': project\_id,  
 })  
  
 logger.debug(f"BudgetAllocationListView context: {context}")  
 return context  
  
class BudgetAllocationDetailView(PermissionBaseView, DetailView):  
 model = BudgetAllocation  
 template\_name = 'budgets/budget/budgetallocation\_detail.html'  
 context\_object\_name = 'budget\_allocation'  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 from budgets.budget\_calculations import get\_budget\_details  
 context['budget\_details'] = get\_budget\_details(self.object)  
 logger.debug(f"BudgetAllocationDetailView context: {context}")  
 return context  
class BudgetAllocationDeleteView(PermissionBaseView, DeleteView):  
 model = BudgetAllocation  
 template\_name = 'budgets/budget/budgetallocation\_confirm\_delete.html'  
 success\_url = reverse\_lazy('budgetallocation\_list')  
  
 @transaction.atomic  
 def post(self, request, \*args, \*\*kwargs):  
 allocation = self.get\_object()  
 allocation.delete()  
 messages.success(request, f'تخصیص بودجه به {allocation.organization.name} با موفقیت حذف شد.')  
 from django.shortcuts import redirect  
 return redirect(self.success\_url)

import logging  
  
import jdatetime  
from django import forms  
from django.utils.translation import gettext\_lazy as \_  
  
from Tanbakhsystem.utils import format\_jalali\_date, parse\_jalali\_date  
from budgets.models import BudgetPeriod, SystemSettings  
from core.templatetags.rcms\_custom\_filters import number\_to\_farsi\_words  
  
logger = logging.getLogger(\_\_name\_\_)  
  
  
def to\_english\_digits(text):  
 *"""تبدیل اعداد فارسی به انگلیسی"""* farsi\_to\_english = str.maketrans('۰۱۲۳۴۵۶۷۸۹', '0123456789')  
 return str(text).translate(farsi\_to\_english)  
  
class BudgetPeriodForm(forms.ModelForm):  
 start\_date = forms.CharField(  
 label=\_('تاریخ شروع'),  
 widget=forms.TextInput(attrs={  
 'data-jdp': '',  
 'class': 'form-control jalali-datepicker',  
 'autocomplete': 'off'  
 }),  
 required=True  
 )  
 end\_date = forms.CharField(  
 label=\_('تاریخ پایان'),  
 widget=forms.TextInput(attrs={  
 'data-jdp': '',  
 'class': 'form-control jalali-datepicker',  
 'autocomplete': 'off'  
 }),  
 required=True  
 )  
  
 class Meta:  
 model = BudgetPeriod  
 fields = [  
 'organization', 'name', 'start\_date', 'end\_date', 'total\_amount', 'is\_active',  
 'is\_archived', 'is\_completed', 'lock\_condition', 'locked\_percentage',  
 'warning\_threshold', 'warning\_action', 'description',  
 *# 'total\_allocated', 'returned\_amount', 'allocation\_phase' <-- حذف شدند* ]  
 widgets = {  
 'organization': forms.Select(attrs={'class': 'form-select', 'required': True}),  
 *# استفاده از form-select برای سازگاری با بوت استرپ 5+* 'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder': \_('نام دوره مثل ردیف "عملیاتی"، "سرمایه‌ای" یا "پژوهشی" بهمراه مثلا سال 1404')}),  
 'total\_amount': forms.NumberInput(attrs={'class': 'form-control', 'min': 1}),  
 'is\_active': forms.CheckboxInput(attrs={'class': 'form-check-input'}),  
 'is\_archived': forms.CheckboxInput(attrs={'class': 'form-check-input'}),  
 'is\_completed': forms.CheckboxInput(attrs={'class': 'form-check-input'}),  
 'lock\_condition': forms.Select(attrs={'class': 'form-select'}), *# استفاده از form-select* 'locked\_percentage': forms.NumberInput(attrs={'class': 'form-control', 'min': 0, 'max': 100, 'step': 0.01}),  
 'warning\_threshold': forms.NumberInput(attrs={'class': 'form-control', 'min': 0, 'max': 100, 'step': 0.01}),  
 'warning\_action': forms.Select(attrs={'class': 'form-select'}), *# استفاده از form-select* 'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 2}),  
 *# تغییر rows به 2 برای فضای بیشتر* }  
 labels = {  
 'organization': \_('دفتر مرکزی'),  
 'name': \_('نام دوره بودجه'),  
 'total\_amount': \_('مبلغ کل (ریال)'),  
 'is\_active': \_('فعال'),  
 'is\_archived': \_('بایگانی شده'),  
 'is\_completed': \_('تمام‌شده'),  
 'lock\_condition': \_('شرط قفل'),  
 'locked\_percentage': \_('درصد قفل (%)'), *# اضافه کردن واحد* 'warning\_threshold': \_('آستانه اخطار (%)'), *# اضافه کردن واحد* 'warning\_action': \_('اقدام هشدار'),  
 'description': \_('توضیحات'), *# اضافه کردن لیبل برای توضیحات* }  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 self.user = kwargs.pop('user', None)  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 logger.debug("Initializing BudgetPeriodForm")  
 *# محدود کردن سازمان‌ها به دفاتر مرکزی فعال* from core.models import Organization  
 self.fields['organization'].queryset = Organization.objects.filter(is\_core=True, is\_active=True).select\_related('org\_type')  
  
 *# تنظیم مقادیر پیش‌فرض از SystemSettings* system\_settings = SystemSettings.objects.first()  
 if system\_settings:  
 self.fields['locked\_percentage'].initial = system\_settings.budget\_locked\_percentage\_default  
 self.fields['warning\_threshold'].initial = system\_settings.budget\_warning\_threshold\_default  
 self.fields['warning\_action'].initial = system\_settings.budget\_warning\_action\_default  
 *# تنظیم تاریخ‌های اولیه* if self.instance and self.instance.pk:  
 self.initial['start\_date'] = format\_jalali\_date(self.instance.start\_date)  
 self.initial['end\_date'] = format\_jalali\_date(self.instance.end\_date)  
 logger.debug(f"Set initial dates: start\_date={self.initial['start\_date']}, end\_date={self.initial['end\_date']}")  
 else:  
 current\_jalali\_year = jdatetime.date.today().year  
 self.initial['start\_date'] = f"{current\_jalali\_year}/01/01"  
 self.initial['end\_date'] = f"{current\_jalali\_year}/12/29"  
 logger.debug(f"Set default dates for new instance: start={self.initial['start\_date']}, end={self.initial['end\_date']}")  
  
 *# مقدار حروف برای total\_amount* self.total\_amount\_words = ""  
 if self.instance and self.instance.total\_amount:  
 self.total\_amount\_words = number\_to\_farsi\_words(self.instance.total\_amount)  
 logger.info(f' self.total\_amount\_words IS {self.total\_amount\_words}')  
  
 def clean\_start\_date(self):  
 date\_str = self.cleaned\_data.get('start\_date')  
 logger.debug(f"Cleaning start\_date: input={date\_str}")  
 if not date\_str:  
 logger.error("start\_date is empty")  
 raise forms.ValidationError(\_('تاریخ شروع اجباری است.'))  
 try:  
 date\_str = to\_english\_digits(date\_str)  
 parsed\_date = parse\_jalali\_date(date\_str, field\_name=\_('تاریخ شروع'))  
 logger.debug(f"Parsed start\_date: {parsed\_date}")  
 return parsed\_date  
 except Exception as e:  
 logger.error(f"Error parsing start\_date: {str(e)}")  
 raise forms.ValidationError(\_('فرمت تاریخ شروع نامعتبر است.'))  
  
 def clean\_end\_date(self):  
 date\_str = self.cleaned\_data.get('end\_date')  
 logger.debug(f"Cleaning end\_date: input={date\_str}")  
 if not date\_str:  
 logger.error("end\_date is empty")  
 raise forms.ValidationError(\_('تاریخ پایان اجباری است.'))  
 try:  
 date\_str = to\_english\_digits(date\_str)  
 parsed\_date = parse\_jalali\_date(date\_str, field\_name=\_('تاریخ پایان'))  
 logger.debug(f"Parsed end\_date: {parsed\_date}")  
 return parsed\_date  
 except Exception as e:  
 logger.error(f"Error parsing end\_date: {str(e)}")  
 raise forms.ValidationError(\_('فرمت تاریخ پایان نامعتبر است.'))  
  
 def clean\_total\_amount(self):  
 amount = self.cleaned\_data.get('total\_amount')  
 logger.debug(f"Cleaning total\_amount: input={amount}")  
 if amount is None:  
 logger.error("total\_amount is None")  
 raise forms.ValidationError(\_('مبلغ کل نمی‌تواند خالی باشد.'))  
 if amount <= 0:  
 logger.error(f"Invalid total\_amount: {amount} (must be positive)")  
 raise forms.ValidationError(\_('مبلغ کل باید بزرگ‌تر از صفر باشد.'))  
 *# به‌روزرسانی مقدار حروف* self.total\_amount\_words = number\_to\_farsi\_words(amount)  
 logger.debug(f"Validated total\_amount: {amount}, words: {self.total\_amount\_words}")  
 return amount  
  
 def clean\_locked\_percentage(self):  
 percentage = self.cleaned\_data.get('locked\_percentage')  
 logger.debug(f"Cleaning locked\_percentage: input={percentage}")  
 if percentage is None:  
 logger.error("locked\_percentage is None")  
 raise forms.ValidationError(\_('درصد قفل‌شده نمی‌تواند خالی باشد.'))  
 if not (0 <= percentage <= 100):  
 logger.error(f"Invalid locked\_percentage: {percentage} (must be 0-100)")  
 raise forms.ValidationError(\_('درصد قفل‌شده باید بین ۰ تا ۱۰۰ باشد.'))  
 logger.debug(f"Validated locked\_percentage: {percentage}")  
 return percentage  
  
 def clean\_warning\_threshold(self):  
 threshold = self.cleaned\_data.get('warning\_threshold')  
 logger.debug(f"Cleaning warning\_threshold: input={threshold}")  
 if threshold is None:  
 logger.error("warning\_threshold is None")  
 raise forms.ValidationError(\_('آستانه اخطار نمی‌تواند خالی باشد.'))  
 if not (0 <= threshold <= 100):  
 logger.error(f"Invalid warning\_threshold: {threshold} (must be 0-100)")  
 raise forms.ValidationError(\_('آستانه اخطار باید بین ۰ تا ۱۰۰ باشد.'))  
 locked\_percentage = self.cleaned\_data.get('locked\_percentage')  
 if locked\_percentage is not None and threshold <= locked\_percentage:  
 logger.error(f"warning\_threshold ({threshold}) <= locked\_percentage ({locked\_percentage})")  
 raise forms.ValidationError(\_('آستانه اخطار باید بزرگ‌تر از درصد قفل‌شده باشد.'))  
 logger.debug(f"Validated warning\_threshold: {threshold}")  
 return threshold  
  
 def clean\_organization(self):  
 organization = self.cleaned\_data.get('organization')  
 logger.debug(f"Cleaning organization: input={organization}")  
 if organization and not organization.is\_core:  
 logger.error(f"Selected organization {organization} is not core")  
 raise forms.ValidationError(\_('فقط دفاتر مرکزی می‌توانند برای بودجه کلان انتخاب شوند.'))  
 return organization  
  
 def clean(self):  
 cleaned\_data = super().clean()  
 logger.debug(f"Running clean method: cleaned\_data={cleaned\_data}")  
 start\_date = cleaned\_data.get('start\_date')  
 end\_date = cleaned\_data.get('end\_date')  
 is\_completed = cleaned\_data.get('is\_completed')  
 is\_active = cleaned\_data.get('is\_active')  
 organization = cleaned\_data.get('organization')  
 name = cleaned\_data.get('name')  
  
 if start\_date and end\_date:  
 if end\_date <= start\_date:  
 logger.error(f"end\_date ({end\_date}) <= start\_date ({start\_date})")  
 raise forms.ValidationError(\_('تاریخ پایان باید بعد از تاریخ شروع باشد.'))  
 overlapping\_periods = BudgetPeriod.objects.filter(  
 organization=organization,  
 start\_date\_\_lte=end\_date,  
 end\_date\_\_gte=start\_date  
 ).exclude(pk=self.instance.pk)  
 if overlapping\_periods.exists():  
 logger.error("Overlapping budget period detected")  
 *# raise forms.ValidationError(\_('دوره بودجه با تاریخ‌های متداخل در این سازمان وجود دارد.'))  
 # from django.contrib import messages  
 # messages.success(self.request, \_('دوره بودجه با موفقیت ایجاد شد.👍') + \_(  
 # '💸دوره بودجه با تاریخ‌های متداخل در این سازمان وجود دارد.'))* logger.debug("Validated date comparison")  
 else:  
 logger.warning(f"Missing dates: start\_date={start\_date}, end\_date={end\_date}")  
  
 if is\_completed and is\_active:  
 logger.error("is\_completed and is\_active both True")  
 raise forms.ValidationError(\_('دوره تمام‌شده نمی‌تواند فعال باشد.'))  
  
 if organization and name:  
 if BudgetPeriod.objects.filter(organization=organization, name=name).exclude(pk=self.instance.pk).exists():  
 logger.error(f"Duplicate name '{name}' for organization {organization}")  
 raise forms.ValidationError(\_('نام دوره در این سازمان قبلاً استفاده شده است.'))  
 logger.debug("Validated name uniqueness")  
  
 logger.debug("Clean method completed successfully")  
 return cleaned\_data  
  
 def save(self, commit=True):  
 logger.debug("Saving BudgetPeriodForm")  
 instance = super().save(commit=False)  
 if not self.user or not self.user.is\_authenticated:  
 logger.error("No authenticated user provided")  
 raise forms.ValidationError(\_('کاربر معتبر برای ایجاد دوره بودجه لازم است.'))  
 instance.created\_by = self.user  
 logger.debug(f"Set created\_by: {instance.created\_by}")  
 if commit:  
 try:  
 instance.save()  
 logger.info(f"BudgetPeriod saved: {instance}")  
 except Exception as e:  
 logger.error(f"Error saving BudgetPeriod: {str(e)}")  
 raise  
 return instance

from decimal import Decimal  
  
from django.db.models import Q, Sum  
from django.shortcuts import redirect  
from django.views.generic import CreateView, UpdateView, DeleteView,ListView,DetailView  
from django.urls import reverse\_lazy  
from django.contrib import messages  
from django.db import transaction  
  
from Tanbakhsystem.utils import parse\_jalali\_date\_jdate  
from budgets.budget\_calculations import get\_budget\_details, check\_budget\_status, calculate\_remaining\_budget  
from core.PermissionBase import PermissionBaseView  
from budgets.models import BudgetPeriod, BudgetTransaction, BudgetHistory  
from budgets.BudgetPeriod.Forms\_BudgetPeriod import BudgetPeriodForm  
from django.utils.translation import gettext\_lazy as \_  
import logging  
  
logger = logging.getLogger(\_\_name\_\_)  
  
class BudgetPeriodCreateView(PermissionBaseView,CreateView):  
 model = BudgetPeriod  
 form\_class = BudgetPeriodForm  
 template\_name = 'budgets/budget/budgetperiod\_form.html'  
 success\_url = reverse\_lazy('budgetperiod\_list')  
 permission\_required = 'budgets.add\_budgetperiod'  
 check\_object\_permission = 'budgets.add\_budgetperiod'  
 check\_organization = True  
  
 def get\_form\_kwargs(self):  
 kwargs = super().get\_form\_kwargs()  
 kwargs['initial']['created\_by'] = self.request.user  
 kwargs['user'] = self.request.user  
 return kwargs  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['title'] = \_('ایجاد دوره بودجه جدید')  
 from core.models import Organization  
 context['organizations'] = Organization.objects.filter(  
 is\_core=True, is\_active=True  
 ).select\_related('org\_type')  
 return context  
  
 @transaction.atomic  
 def form\_valid(self, form):  
 try:  
 response = super().form\_valid(form)  
 messages.success(self.request, \_('دوره بودجه با موفقیت ایجاد شد.'))  
 return response  
 except Exception as e:  
 messages.error(self.request, \_('خطایی در ایجاد دوره بودجه رخ داد: ') + str(e))  
 return self.form\_invalid(form)  
  
 def form\_invalid(self, form):  
 logger.error(f"Form invalid: errors={form.errors.as\_json()}")  
 messages.error(self.request, \_('لطفاً خطاهای فرم را بررسی کنید.'))  
 return self.render\_to\_response(self.get\_context\_data(form=form))  
  
  
class BudgetPeriodUpdateView(PermissionBaseView,UpdateView):  
 model = BudgetPeriod  
 form\_class = BudgetPeriodForm  
 template\_name = 'budgets/budget/budgetperiod\_form.html'  
 success\_url = reverse\_lazy('budgetperiod\_list')  
  
 def get\_form\_kwargs(self):  
 kwargs = super().get\_form\_kwargs()  
 kwargs['user'] = self.request.user  
 return kwargs  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['title'] = \_('ویرایش دوره بودجه')  
 context['total\_allocated'] = self.object.total\_allocated  
 context['remaining\_amount'] = self.object.get\_remaining\_amount()  
 return context  
  
 @transaction.atomic  
 def form\_valid(self, form):  
 try:  
 response = super().form\_valid(form)  
 messages.success(self.request, \_('دوره بودجه با موفقیت به‌روزرسانی شد.'))  
 return response  
 except Exception as e:  
 messages.error(self.request, \_('خطایی در به‌روزرسانی دوره بودجه رخ داد: ') + str(e))  
 return self.form\_invalid(form)  
  
 def form\_invalid(self, form):  
 messages.error(self.request, \_('لطفاً خطاهای فرم را بررسی کنید.'))  
 return self.render\_to\_response(self.get\_context\_data(form=form))  
  
class BudgetPeriodDeleteView(PermissionBaseView, DeleteView):  
 model = BudgetPeriod  
 template\_name = 'budgets/budget/budgetperiod\_confirm\_delete.html'  
 success\_url = reverse\_lazy('budgetperiod\_list')  
  
 def post(self, request, \*args, \*\*kwargs):  
 budget\_period = self.get\_object()  
 with transaction.atomic():  
 budget\_period.delete()  
 messages.success(request, f'دوره بودجه {budget\_period.name} با موفقیت حذف شد.')  
 return redirect(self.success\_url)  
  
*# --- BudgetPeriod CRUD ---*class BudgetPeriodListView(PermissionBaseView, ListView):  
 model = BudgetPeriod  
 template\_name = 'budgets/budget/budgetperiod\_list.html'  
 context\_object\_name = 'budget\_periods'  
 paginate\_by = 10  
 from Tanbakhsystem.utils import parse\_jalali\_date  
  
 def get\_queryset(self):  
 queryset = super().get\_queryset()  
 q = self.request.GET.get('q', '')  
 status = self.request.GET.get('status', '')  
 date\_from = self.request.GET.get('date\_from', '')  
 date\_to = self.request.GET.get('date\_to', '')  
  
 if q:  
 queryset = queryset.filter(Q(name\_\_icontains=q) | Q(organization\_\_name\_\_icontains=q))  
 if status:  
 status\_map = {  
 'active': Q(is\_active=True),  
 'inactive': Q(is\_active=False),  
 'locked': Q(lock\_condition='MANUAL'),  
 'completed': Q(is\_completed=True),  
 }  
 queryset = queryset.filter(status\_map.get(status, Q()))  
  
 try:  
 if date\_from:  
 queryset = queryset.filter(start\_date\_\_gte=parse\_jalali\_date\_jdate(date\_from))  
 if date\_to:  
 queryset = queryset.filter(end\_date\_\_lte=parse\_jalali\_date\_jdate(date\_to))  
 except Exception as e:  
 logger.error(f"Date filtering error: {str(e)}")  
  
 return queryset.order\_by('-start\_date')  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 queryset = self.get\_queryset() *# برای محاسبه status\_summary* for period in context['budget\_periods']:  
 period.remaining\_amount = max(period.total\_amount - (period.total\_allocated or Decimal('0')), Decimal('0'))  
 period.status, period.status\_message = check\_budget\_status(period)  
 if period.status in ('warning', 'locked', 'completed'):  
 messages.warning(self.request, f"{period.name}: {period.status\_message}")  
 from django.contrib.contenttypes.models import ContentType  
 BudgetHistory.objects.create(  
 content\_type=ContentType.objects.get\_for\_model(BudgetPeriod),  
 object\_id=period.pk,  
 action='STATUS\_CHECK',  
 details=f"وضعیت: {period.status} - {period.status\_message}",  
 created\_by=self.request.user  
 )  
  
 context['query'] = self.request.GET.get('q', '')  
 context['status'] = self.request.GET.get('status', '')  
 context['date\_from'] = self.request.GET.get('date\_from', '')  
 context['date\_to'] = self.request.GET.get('date\_to', '')  
 context['status\_summary'] = {  
 'active': queryset.filter(is\_active=True).count(),  
 'locked': queryset.filter(lock\_condition='MANUAL').count(),  
 'completed': queryset.filter(is\_completed=True).count(),  
 'total': queryset.count(),  
 }  
 logger.debug(f"BudgetPeriodListView context: {context}")  
 return context  
  
  
class BudgetPeriodDetailView(PermissionBaseView, DetailView):  
 model = BudgetPeriod  
 template\_name = 'budgets/budget/budgetperiod\_detail.html'  
 context\_object\_name = 'budget\_period'  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 *# صفحه‌بندی تراکنش‌ها* transactions = BudgetTransaction.objects.filter(  
 allocation\_\_budget\_period=self.object  
 ).order\_by('-timestamp')  
 from django.core.paginator import Paginator  
 paginator = Paginator(transactions, 10)  
 page\_number = self.request.GET.get('page')  
 context['transactions'] = paginator.get\_page(page\_number)  
 *# جزئیات بودجه* try:  
 context['budget\_details'] = get\_budget\_details(self.object)  
 except Exception as e:  
 logger.error(f"Error in get\_budget\_details: {str(e)}")  
 messages.error(self.request, \_('خطایی در محاسبه جزئیات بودجه رخ داد.'))  
 context['budget\_details'] = {  
 'total\_budget': Decimal('0'),  
 'total\_allocated': Decimal('0'),  
 'remaining\_budget': Decimal('0'),  
 'status': 'error',  
 'status\_message': 'خطا در محاسبه'  
 }  
 *# اعلان وضعیت* try:  
 status\_dict = check\_budget\_status(self.object)  
 status = status\_dict['status']  
 message = status\_dict['message']  
 if status in ('warning', 'locked', 'completed'):  
 messages.warning(self.request, message)  
 except Exception as e:  
 logger.error(f"Error in check\_budget\_status: {str(e)}")  
 messages.error(self.request, \_('خطایی در بررسی وضعیت بودجه رخ داد.'))  
 logger.debug(f"BudgetPeriodDetailView context: {context}")  
 return context

import logging  
from django import forms  
from django.utils.translation import gettext\_lazy as \_  
from django.utils import timezone  
from decimal import Decimal  
import jdatetime  
from Tanbakhsystem.utils import format\_jalali\_date, parse\_jalali\_date  
from budgets.budget\_calculations import get\_project\_remaining\_budget, check\_budget\_status  
from budgets.models import BudgetPeriod, SystemSettings, BudgetTransaction, BudgetAllocation  
from core.templatetags.rcms\_custom\_filters import number\_to\_farsi\_words  
logger = logging.getLogger(\_\_name\_\_)  
  
*# budgets/forms.py*from django import forms  
from django.utils.translation import gettext\_lazy as \_  
from django.db.models import Sum  
from decimal import Decimal  
  
class BudgetReturnForm(forms.ModelForm):  
 class Meta:  
 model = BudgetTransaction  
 fields = ['amount', 'description']  
 widgets = {  
 'amount': forms.NumberInput(attrs={'class': 'form-control', 'min': 0, 'placeholder': 'مبلغ (ریال)'}),  
 'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 3, 'placeholder': 'توضیحات'}),  
 }  
 labels = {  
 'amount': \_('مبلغ برگشتی'),  
 'description': \_('توضیحات'),  
 }  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 self.allocation = kwargs.pop('allocation', None)  
 self.user = kwargs.pop('user', None)  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 if self.allocation:  
 self.instance.allocation = self.allocation.budget\_allocation  
 if self.user:  
 self.instance.created\_by = self.user  
 self.instance.transaction\_type = 'RETURN'  
  
 def clean\_amount(self):  
 amount = self.cleaned\_data.get('amount')  
 if not amount:  
 raise forms.ValidationError(\_('مبلغ برگشتی الزامی است.'))  
 if amount <= 0:  
 raise forms.ValidationError(\_('مبلغ برگشتی باید مثبت باشد.'))  
  
 if self.allocation:  
 *# محاسبه مصرف خالص* consumed = BudgetTransaction.objects.filter(  
 allocation=self.allocation.budget\_allocation,  
 allocation\_\_project=self.allocation.project,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation=self.allocation.budget\_allocation,  
 allocation\_\_project=self.allocation.project,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 total\_consumed = consumed - returned  
  
 *# بررسی سقف بازگشت* if amount > total\_consumed:  
 raise forms.ValidationError(  
 \_(f"مبلغ برگشتی ({amount:,.0f} ریال) نمی‌تواند بیشتر از مصرف خالص ({total\_consumed:,.0f} ریال) باشد.")  
 )  
  
 *# بررسی بودجه باقی‌مانده پروژه* remaining\_budget = get\_project\_remaining\_budget(self.allocation.project)  
 if amount > remaining\_budget:  
 raise forms.ValidationError(  
 \_(f"مبلغ برگشتی ({amount:,.0f} ریال) نمی‌تواند بیشتر از بودجه باقی‌مانده پروژه ({remaining\_budget:,.0f} ریال) باشد.")  
 )  
  
 return amount  
  
 def save(self, commit=True):  
 instance = super().save(commit=False)  
 instance.transaction\_type = 'RETURN'  
 if commit:  
 instance.save()  
 *# بررسی وضعیت بودجه و ارسال اعلان* status, message = check\_budget\_status(instance.allocation.budget\_period)  
 if status in ('warning', 'locked', 'completed', 'stopped'):  
 instance.allocation.send\_notification(status, message)  
 instance.allocation.send\_notification(  
 'return',  
 f"مبلغ {instance.amount:,.0f} ریال از تخصیص {instance.allocation.id} برگشت داده شد."  
 )  
 return instance

from django.contrib.auth.mixins import LoginRequiredMixin  
from django.core.exceptions import PermissionDenied  
from django.http import Http404  
from django.shortcuts import redirect  
from django.views.generic import CreateView  
from django.urls import reverse\_lazy  
from django.contrib import messages  
from django.utils.translation import gettext\_lazy as \_  
  
from budgets.models import BudgetTransaction  
from budgets.BudgetReturn.forms\_BudgetReturm import BudgetReturnForm  
from budgets.budget\_calculations import (get\_project\_remaining\_budget,  
 calculate\_remaining\_budget,  
 get\_project\_remaining\_budget,  
 get\_project\_total\_budget,  
 get\_project\_used\_budget,  
 get\_budget\_details,  
 check\_budget\_status, get\_organization\_budget  
 )  
from core.PermissionBase import PermissionBaseView  
import logging  
logger = logging.getLogger(\_\_name\_\_)  
  
*#-----------------*class BudgetReturnView(PermissionBaseView, CreateView):  
 model = BudgetTransaction  
 form\_class = BudgetReturnForm  
 template\_name = 'budgets/budget\_return\_form.html'  
 permission\_codename = 'budgets.BudgetTransaction\_add'  
 permission\_denied\_message = \_('متاسفانه دسترسی مجاز ندارید')  
 check\_organization = True  
  
  
 def get\_allocation(self):  
 try:  
 allocation = ProjectBudgetAllocation.objects.get(pk=self.kwargs['allocation\_id'], is\_active=True)  
 user\_organizations = self.request.user.get\_authorized\_organizations()  
 if not user\_organizations.filter(pk=allocation.budget\_allocation.organization.pk).exists():  
 logger.warning(  
 f"User {self.request.user.username} attempted to access allocation {self.kwargs['allocation\_id']} "  
 f"without organization permission"  
 )  
 raise PermissionDenied(self.permission\_denied\_message)  
 return allocation  
 except ProjectBudgetAllocation.DoesNotExist:  
 logger.error(f"Allocation {self.kwargs['allocation\_id']} does not exist or is inactive")  
 messages.error(self.request, \_('تخصیص بودجه مورد نظر یافت نشد یا غیرفعال است.'))  
 return None  
  
  
 def dispatch(self, request, \*args, \*\*kwargs):  
 allocation = self.get\_allocation()  
 if not allocation:  
 return redirect('budgetallocation\_list')  
 return super().dispatch(request, \*args, \*\*kwargs)  
  
  
 def get\_initial(self):  
 initial = super().get\_initial()  
 allocation = self.get\_allocation()  
 if allocation:  
 initial['allocation'] = allocation.budget\_allocation  
 initial['transaction\_type'] = 'RETURN'  
 return initial  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 allocation = self.get\_allocation()  
 if not allocation:  
 return context  
  
 project = allocation.project  
 organization = allocation.budget\_allocation.organization  
 budget\_period = allocation.budget\_allocation.budget\_period  
  
 context.update({  
 'allocation': allocation,  
 'project': project,  
 'organization': organization,  
 'total\_budget': get\_project\_total\_budget(project),  
 'used\_budget': get\_project\_used\_budget(project),  
 'remaining\_budget': get\_project\_remaining\_budget(project),  
 'org\_budget': get\_organization\_budget(organization),  
 'budget\_details': get\_budget\_details(entity=project),  
 'budget\_status': check\_budget\_status(budget\_period)[0],  
 'budget\_status\_message': check\_budget\_status(budget\_period)[1],  
 })  
 return context  
  
 *# def get\_context\_data(self, \*\*kwargs):  
 # context = super().get\_context\_data(\*\*kwargs)  
 # allocation = self.get\_allocation()  
 # if not allocation:  
 # return context  
 #  
 # # محاسبه اطلاعات بودجه با استفاده از توابع  
 # project = allocation.project  
 # organization = allocation.budget\_allocation.organization  
 # budget\_period = allocation.budget\_allocation.budget\_period  
 #  
 # total\_budget = get\_project\_total\_budget(project)  
 # used\_budget = get\_project\_used\_budget(project)  
 # remaining\_budget = get\_project\_remaining\_budget(project)  
 # org\_budget = get\_organization\_budget(organization)  
 # budget\_details = get\_budget\_details(entity=project)  
 # budget\_status = check\_budget\_status(budget\_period)  
 #  
 # context.update({  
 # 'allocation': allocation,  
 # 'project': project,  
 # 'organization': organization,  
 # 'total\_budget': total\_budget,  
 # 'used\_budget': used\_budget,  
 # 'remaining\_budget': remaining\_budget,  
 # 'org\_budget': org\_budget,  
 # 'budget\_details': budget\_details,  
 # 'budget\_status': budget\_status[0],  
 # 'budget\_status\_message': budget\_status[1],  
 # })  
 # return context* def form\_valid(self, form):  
 allocation = self.get\_allocation()  
 if not allocation:  
 return redirect('budgetallocation\_list')  
  
 *# اعتبارسنجی اضافی با استفاده از توابع* project = allocation.project  
 remaining\_budget = get\_project\_remaining\_budget(project)  
 consumed = get\_project\_used\_budget(project)  
 if form.instance.amount > consumed:  
 form.add\_error(  
 'amount',  
 \_(f"مبلغ بازگشت ({form.instance.amount:,.0f} ریال) نمی‌تواند بیشتر از مصرف‌شده ({consumed:,.0f} ریال) باشد.")  
 )  
 return self.form\_invalid(form)  
 if form.instance.amount > remaining\_budget:  
 form.add\_error(  
 'amount',  
 \_(f"مبلغ بازگشت ({form.instance.amount:,.0f} ریال) نمی‌تواند بیشتر از بودجه باقی‌مانده ({remaining\_budget:,.0f} ریال) باشد.")  
 )  
 return self.form\_invalid(form)  
  
 *# if form.instance.amount > remaining\_budget:  
 # form.add\_error('amount', \_(f"مبلغ بازگشت ({form.instance.amount:,.0f} ریال) نمی‌تواند بیشتر از بودجه باقی‌مانده ({remaining\_budget:,.0f} ریال) باشد."))  
 # return self.form\_invalid(form)* form.instance.created\_by = self.request.user  
 logger.info(f"User {self.request.user.username} created return transaction for allocation {allocation.id}")  
 messages.success(self.request, \_("تراکنش بازگشت بودجه با موفقیت ثبت شد."))  
  
 *# ثبت تراکنش و ارسال اعلان* instance = form.save(commit=False)  
 instance.transaction\_type = 'RETURN'  
 instance.save()  
  
 *# بررسی وضعیت بودجه و ارسال اعلان* status, message = check\_budget\_status(allocation.budget\_allocation.budget\_period)  
 if status in ('warning', 'locked', 'completed', 'stopped'):  
 allocation.budget\_allocation.send\_notification(status, message)  
 allocation.budget\_allocation.send\_notification(  
 'return',  
 f"مبلغ {instance.amount:,.0f} ریال از تخصیص {allocation.id} برگشت داده شد."  
 )  
  
 return super().form\_valid(form)  
  
 def get\_form\_kwargs(self):  
 kwargs = super().get\_form\_kwargs()  
 allocation = self.get\_allocation()  
 if allocation:  
 kwargs['allocation'] = allocation  
 kwargs['user'] = self.request.user  
 return kwargs  
  
 def get\_success\_url(self):  
 return reverse\_lazy('project\_budget\_allocation\_detail', kwargs={'pk': self.kwargs['allocation\_id']})  
*#-----------------*""""گزارش‌گیری بودجه برگشتی:"""  
from budgets.models import BudgetHistory, ProjectBudgetAllocation, BudgetTransaction  
  
  
def get\_returned\_budgets(budget\_period):  
 from django.contrib.contenttypes.models import ContentType  
 from budgets.models import BudgetAllocation  
 return BudgetHistory.objects.filter(  
 content\_type=ContentType.objects.get\_for\_model(BudgetAllocation),  
 action='RETURN',  
 content\_object\_\_budget\_period=budget\_period  
 ).values('amount', 'details', 'created\_at', 'created\_by\_\_username')

*# budgets/views.py*from decimal import Decimal  
  
from django.db.models import Q, Sum  
from django.views.generic import ListView, DetailView  
import datetime  
import logging  
  
from budgets.models import BudgetTransaction, ProjectBudgetAllocation  
from core.PermissionBase import PermissionBaseView  
from core.models import Project  
  
logger = logging.getLogger(\_\_name\_\_)  
*# --- BudgetTransaction CRUD ---*class BudgetTransactionListView\_2D(PermissionBaseView, ListView):  
 model = BudgetTransaction  
 template\_name = 'budgets/BudgetTransaction/budget\_transaction\_list.html'  
 context\_object\_name = 'transactions'  
 permission\_codename = 'budgets.BudgetTransaction\_view'  
  
 def get\_queryset(self):  
 allocation = ProjectBudgetAllocation.objects.get(pk=self.kwargs['allocation\_id'])  
 return BudgetTransaction.objects.filter(  
 allocation=allocation.budget\_allocation,  
 allocation\_\_project=allocation.project  
 ).order\_by('-timestamp')  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 allocation = ProjectBudgetAllocation.objects.get(pk=self.kwargs['allocation\_id'])  
 context['allocation'] = allocation  
 return context  
  
class BudgetTransactionListView(PermissionBaseView, ListView):  
 model = BudgetTransaction  
 template\_name = 'budgets/budget/budgettransaction\_list.html'  
 context\_object\_name = 'budget\_transactions'  
 paginate\_by = 10  
  
 def get\_queryset(self):  
 queryset = super().get\_queryset()  
 query = self.request.GET.get('q', '')  
 if query:  
 queryset = queryset.filter(  
 Q(allocation\_\_budget\_period\_\_name\_\_icontains=query) |  
 Q(allocation\_\_organization\_\_name\_\_icontains=query) |  
 Q(description\_\_icontains=query)  
 )  
 trans\_type = self.request.GET.get('transaction\_type')  
 if trans\_type:  
 queryset = queryset.filter(transaction\_type=trans\_type)  
 return queryset.order\_by('-timestamp')  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['query'] = self.request.GET.get('q', '')  
 context['transaction\_type'] = self.request.GET.get('transaction\_type', '')  
 context['transaction\_types'] = BudgetTransaction.TRANSACTION\_TYPES  
 logger.debug(f"BudgetTransactionListView context: {context}")  
 return context  
  
class BudgetTransactionDetailView(PermissionBaseView, DetailView):  
 model = BudgetTransaction  
 template\_name = 'budgets/budget/budgettransaction\_detail.html'  
 context\_object\_name = 'budget\_transaction'

*# budgets/forms.py*from django import forms  
from django.db.models import Sum  
from django.utils.translation import gettext\_lazy as \_  
from django.core.exceptions import ValidationError  
import jdatetime  
import logging  
from decimal import Decimal  
from Tanbakhsystem.utils import parse\_jalali\_date  
from budgets.models import BudgetAllocation, BudgetTransaction, ProjectBudgetAllocation  
  
logger = logging.getLogger(\_\_name\_\_)  
  
class \_\_Project\_Budget\_Allocation\_Update\_Form(forms.ModelForm):  
 allocation\_date = forms.CharField(  
 label=\_('تاریخ تخصیص'),  
 widget=forms.TextInput(attrs={  
 'data-jdp': '',  
 'class': 'form-control form-control-lg jalali-datepicker',  
 'autocomplete': 'off',  
 'placeholder': \_('مثال: 1404/01/26')  
 }),  
 required=True  
 )  
  
 class Meta:  
 model = ProjectBudgetAllocation  
 fields = ['project', 'subproject', 'allocated\_amount', 'allocation\_date', 'is\_active', 'description']  
 widgets = {  
 'project': forms.Select(attrs={  
 'class': 'form-select select2',  
 'data-placeholder': \_('انتخاب پروژه'),  
 }),  
 'subproject': forms.Select(attrs={  
 'class': 'form-select select2',  
 'data-placeholder': \_('انتخاب زیرپروژه'),  
 }),  
 'allocated\_amount': forms.TextInput(attrs={  
 'class': 'form-control numeric-input',  
 'placeholder': \_('مبلغ به ریال'),  
 'dir': 'ltr',  
 'inputmode': 'numeric',  
 }),  
 'is\_active': forms.CheckboxInput(attrs={  
 'class': 'form-check-input',  
 }),  
 'description': forms.Textarea(attrs={  
 'class': 'form-control',  
 'rows': 2,  
 'placeholder': \_('توضیحات مربوط به تخصیص بودجه'),  
 }),  
 }  
 labels = {  
 'project': \_('پروژه'),  
 'subproject': \_('زیرپروژه'),  
 'allocated\_amount': \_('مبلغ تخصیص'),  
 'allocation\_date': \_('تاریخ تخصیص'),  
 'is\_active': \_('فعال'),  
 'description': \_('توضیحات'),  
 }  
 help\_texts = {  
 'allocated\_amount': \_('مبلغ را به ریال وارد کنید. مثال: 1,000,000'),  
 'is\_active': \_('تخصیص را فعال یا غیرفعال کنید'),  
 'allocation\_date': \_('تاریخ را به فرمت 1404/01/17 وارد کنید'),  
 }  
  
 def \_\_init\_\_(self, \*args, organization\_id=None, user=None, \*\*kwargs):  
 self.user = user  
 self.organization\_id = organization\_id  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
  
 if organization\_id:  
 *# فیلتر پروژه‌ها بر اساس سازمان* self.fields['project'].queryset = self.fields['project'].queryset.filter(  
 organization\_id=organization\_id  
 )  
 *# فیلتر زیرپروژه‌ها بر اساس پروژه انتخاب‌شده* if self.instance and self.instance.project:  
 self.fields['subproject'].queryset = self.fields['subproject'].queryset.filter(  
 project=self.instance.project  
 )  
  
 *# غیرفعال کردن پروژه و زیرپروژه در ویرایش* if self.instance and self.instance.pk:  
 self.fields['project'].disabled = True  
 self.fields['subproject'].disabled = True  
  
 *# تنظیم تاریخ اولیه* if self.instance and self.instance.pk and self.instance.allocation\_date:  
 try:  
 j\_date = jdatetime.date.fromgregorian(date=self.instance.allocation\_date)  
 self.initial['allocation\_date'] = j\_date.strftime('%Y/%m/%d')  
 logger.debug(f"Set initial allocation\_date: {self.initial['allocation\_date']}")  
 except Exception as e:  
 logger.error(f"Error formatting allocation\_date: {e}")  
 self.initial['allocation\_date'] = jdatetime.date.today().strftime('%Y/%m/%d')  
 else:  
 self.initial['allocation\_date'] = jdatetime.date.today().strftime('%Y/%m/%d')  
  
 def clean\_allocated\_amount(self):  
 allocated\_amount = self.cleaned\_data.get('allocated\_amount')  
 if not allocated\_amount:  
 raise forms.ValidationError(\_('مبلغ تخصیص اجباری است.'))  
  
 *# تبدیل مقدار ورودی به عدد* try:  
 if isinstance(allocated\_amount, str):  
 allocated\_amount = allocated\_amount.replace(',', '').replace('،', '')  
 allocated\_amount = float(allocated\_amount)  
 allocated\_amount = Decimal(allocated\_amount)  
 except (ValueError, TypeError):  
 raise forms.ValidationError(\_('مقدار تخصیص نامعتبر است.'))  
  
 if allocated\_amount <= 0:  
 raise forms.ValidationError(\_('مبلغ تخصیص باید مثبت باشد.'))  
  
 *# بررسی بودجه مصرف‌شده* if self.instance and self.instance.pk:  
 consumed = BudgetTransaction.objects.filter(  
 allocation=self.instance.budget\_allocation,  
 project=self.instance.project,  
 subproject=self.instance.subproject,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation=self.instance.budget\_allocation,  
 project=self.instance.project,  
 subproject=self.instance.subproject,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 net\_consumed = consumed - returned  
 if allocated\_amount < net\_consumed:  
 raise forms.ValidationError(  
 f'مبلغ تخصیص نمی‌تواند کمتر از مبلغ مصرف‌شده ({net\_consumed:,} ریال) باشد.'  
 )  
  
 *# بررسی بودجه باقی‌مانده* if self.instance and self.instance.budget\_allocation:  
 total\_budget = self.instance.budget\_allocation.allocated\_amount  
 other\_allocations = ProjectBudgetAllocation.objects.filter(  
 budget\_allocation=self.instance.budget\_allocation,  
 is\_active=True  
 ).exclude(id=self.instance.id).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 consumed = BudgetTransaction.objects.filter(  
 allocation=self.instance.budget\_allocation,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation=self.instance.budget\_allocation,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 remaining\_budget = total\_budget - consumed + returned - other\_allocations  
 if allocated\_amount > remaining\_budget:  
 raise forms.ValidationError(  
 f'مبلغ تخصیص نمی‌تواند بیشتر از بودجه باقی‌مانده ({remaining\_budget:,} ریال) باشد.'  
 )  
  
 return allocated\_amount  
  
 def clean\_allocation\_date(self):  
 date\_str = self.cleaned\_data.get('allocation\_date')  
 logger.debug(f"Cleaning allocation\_date: input='{date\_str}'")  
 if not date\_str:  
 raise forms.ValidationError(\_('تاریخ تخصیص اجباری است.'))  
 try:  
 parsed\_date = parse\_jalali\_date(str(date\_str), field\_name=\_('تاریخ تخصیص'))  
 logger.debug(f"Parsed allocation\_date: {parsed\_date}")  
 return parsed\_date  
 except ValueError as e:  
 logger.warning(f"Could not parse allocation\_date '{date\_str}': {e}")  
 raise forms.ValidationError(e)  
 except Exception as e:  
 logger.error(f"Unexpected error parsing allocation\_date '{date\_str}': {e}", exc\_info=True)  
 raise forms.ValidationError(\_('فرمت تاریخ تخصیص نامعتبر است.'))  
  
 def clean(self):  
 cleaned\_data = super().clean()  
 allocation\_date = cleaned\_data.get('allocation\_date')  
 project = cleaned\_data.get('project')  
  
 if project and allocation\_date and self.instance.budget\_allocation:  
 start\_date = self.instance.budget\_allocation.budget\_period.start\_date  
 end\_date = self.instance.budget\_allocation.budget\_period.end\_date  
 if not (start\_date <= allocation\_date <= end\_date):  
 raise forms.ValidationError(\_('تاریخ تخصیص باید در بازه دوره بودجه باشد.'))  
  
 return cleaned\_data  
  
 def save(self, commit=True):  
 instance = super().save(commit=False)  
 if self.user and self.user.is\_authenticated:  
 instance.modified\_by = self.user  
 else:  
 raise forms.ValidationError(\_('کاربر معتبر برای ویرایش تخصیص لازم است.'))  
 if commit:  
 try:  
 instance.save()  
 logger.info(f"ProjectBudgetAllocation {instance.pk} updated by {self.user}")  
 except Exception as e:  
 logger.error(f"Error saving ProjectBudgetAllocation: {str(e)}")  
 raise  
 return instance  
*#--------------*def parse\_jalali\_date(date\_str, field\_name=''):  
 *"""تبدیل تاریخ جلالی به میلادی"""* try:  
 j\_date = jdatetime.datetime.strptime(date\_str, '%Y/%m/%d')  
 g\_date = j\_date.togregorian()  
 logger.debug(f"Parsed jalali date {date\_str} to gregorian {g\_date.date()}")  
 return g\_date.date()  
 except ValueError:  
 logger.warning(f"Invalid jalali date format: {date\_str}")  
 raise forms.ValidationError(\_(f'فرمت {field\_name} نامعتبر است. از فرمت 1404/01/17 استفاده کنید.'))  
  
class Project\_Budget\_Allocation\_Update\_Form(forms.ModelForm):  
 *# فیلد تاریخ تخصیص با ویجت تقویم جلالی* allocation\_date = forms.CharField(  
 label=\_('تاریخ تخصیص'),  
 widget=forms.TextInput(attrs={  
 'data-jdp': '',  
 'class': 'form-control form-control-lg jalali-datepicker',  
 'autocomplete': 'off',  
 'placeholder': \_('مثال: 1404/01/26')  
 }),  
 required=True  
 )  
  
 class Meta:  
 model = ProjectBudgetAllocation  
 fields = ['project', 'subproject', 'allocated\_amount', 'allocation\_date', 'is\_active', 'description']  
 widgets = {  
 'project': forms.Select(attrs={  
 'class': 'form-select select2',  
 'data-placeholder': \_('انتخاب پروژه'),  
 }),  
 'subproject': forms.Select(attrs={  
 'class': 'form-select select2',  
 'data-placeholder': \_('انتخاب زیرپروژه'),  
 }),  
 'allocated\_amount': forms.TextInput(attrs={  
 'class': 'form-control numeric-input',  
 'placeholder': \_('مبلغ به ریال'),  
 'dir': 'ltr',  
 'inputmode': 'numeric',  
 }),  
 'is\_active': forms.CheckboxInput(attrs={  
 'class': 'form-check-input',  
 }),  
 'description': forms.Textarea(attrs={  
 'class': 'form-control',  
 'rows': 2,  
 'placeholder': \_('توضیحات مربوط به تخصیص بودجه'),  
 }),  
 }  
 labels = {  
 'project': \_('پروژه'),  
 'subproject': \_('زیرپروژه'),  
 'allocated\_amount': \_('مبلغ تخصیص'),  
 'allocation\_date': \_('تاریخ تخصیص'),  
 'is\_active': \_('فعال'),  
 'description': \_('توضیحات'),  
 }  
 help\_texts = {  
 'allocated\_amount': \_('مبلغ را به ریال وارد کنید. مثال: 1,000,000'),  
 'is\_active': \_('تخصیص را فعال یا غیرفعال کنید'),  
 'allocation\_date': \_('تاریخ را به فرمت 1404/01/17 وارد کنید'),  
 }  
  
 def \_\_init\_\_(self, \*args, organization\_id=None, user=None, \*\*kwargs):  
 *"""  
 تنظیم اولیه فرم.  
 - organization\_id: برای فیلتر کردن پروژه‌ها.  
 - user: برای ثبت کاربر ویرایش‌کننده.  
 - غیرفعال کردن پروژه و زیرپروژه در ویرایش.  
 - تنظیم تاریخ اولیه به فرمت جلالی.  
 """* self.user = user  
 self.organization\_id = organization\_id  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
 logger.debug(f"Initializing form with organization\_id={organization\_id}, user={user}")  
  
 if organization\_id:  
 *# فیلتر پروژه‌ها بر اساس سازمان* self.fields['project'].queryset = self.fields['project'].queryset.filter(  
 organization\_id=organization\_id  
 )  
 logger.debug(f"Filtered projects for organization\_id={organization\_id}")  
 *# فیلتر زیرپروژه‌ها بر اساس پروژه انتخاب‌شده* if self.instance and self.instance.project:  
 self.fields['subproject'].queryset = self.fields['subproject'].queryset.filter(  
 project=self.instance.project  
 )  
 logger.debug(f"Filtered subprojects for project={self.instance.project}")  
  
 *# غیرفعال کردن پروژه و زیرپروژه در ویرایش* if self.instance and self.instance.pk:  
 self.fields['project'].disabled = True  
 self.fields['subproject'].disabled = True  
 logger.debug("Disabled project and subproject fields for editing")  
  
 *# تنظیم تاریخ اولیه* if self.instance and self.instance.pk and self.instance.allocation\_date:  
 try:  
 j\_date = jdatetime.date.fromgregorian(date=self.instance.allocation\_date)  
 self.initial['allocation\_date'] = j\_date.strftime('%Y/%m/%d')  
 logger.debug(f"Set initial allocation\_date: {self.initial['allocation\_date']}")  
 except Exception as e:  
 logger.error(f"Error formatting allocation\_date: {str(e)}")  
 self.initial['allocation\_date'] = jdatetime.date.today().strftime('%Y/%m/%d')  
 else:  
 self.initial['allocation\_date'] = jdatetime.date.today().strftime('%Y/%m/%d')  
 logger.debug(f"Set default allocation\_date: {self.initial['allocation\_date']}")  
  
 def clean\_allocated\_amount(self):  
 *"""  
 اعتبارسنجی مبلغ تخصیص.  
 - مطمئن می‌شه مبلغ معتبر و مثبت باشه.  
 - چک می‌کنه که مبلغ از مصرف‌شده کمتر نباشه.  
 - چک می‌کنه که مبلغ از بودجه باقی‌مانده بیشتر نباشه.  
 """* allocated\_amount = self.cleaned\_data.get('allocated\_amount')  
 logger.debug(f"Cleaning allocated\_amount: {allocated\_amount}")  
 if not allocated\_amount:  
 logger.warning("Allocated amount is empty")  
 raise forms.ValidationError(\_('مبلغ تخصیص اجباری است.'))  
  
 *# تبدیل مقدار ورودی به عدد* try:  
 if isinstance(allocated\_amount, str):  
 allocated\_amount = allocated\_amount.replace(',', '').replace('،', '')  
 allocated\_amount = float(allocated\_amount)  
 allocated\_amount = Decimal(allocated\_amount)  
 logger.debug(f"Converted allocated\_amount to Decimal: {allocated\_amount}")  
 except (ValueError, TypeError):  
 logger.warning(f"Invalid allocated\_amount format: {allocated\_amount}")  
 raise forms.ValidationError(\_('مقدار تخصیص نامعتبر است.'))  
  
 if allocated\_amount <= 0:  
 logger.warning(f"Allocated amount is non-positive: {allocated\_amount}")  
 raise forms.ValidationError(\_('مبلغ تخصیص باید مثبت باشد.'))  
  
 *# بررسی بودجه مصرف‌شده* if self.instance and self.instance.pk:  
 consumed = BudgetTransaction.objects.filter(  
 allocation=self.instance.budget\_allocation,  
 project=self.instance.project,  
 subproject=self.instance.subproject,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation=self.instance.budget\_allocation,  
 project=self.instance.project,  
 subproject=self.instance.subproject,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 net\_consumed = consumed - returned  
 logger.debug(f"Net consumed: {net\_consumed}")  
 if allocated\_amount < net\_consumed:  
 logger.warning(f"Allocated amount {allocated\_amount} is less than net consumed {net\_consumed}")  
 raise forms.ValidationError(  
 f'مبلغ تخصیص نمی‌تواند کمتر از مبلغ مصرف‌شده ({net\_consumed:,} ریال) باشد.'  
 )  
  
 *# بررسی بودجه باقی‌مانده* if self.instance and self.instance.budget\_allocation:  
 total\_budget = self.instance.budget\_allocation.allocated\_amount  
 other\_allocations = ProjectBudgetAllocation.objects.filter(  
 budget\_allocation=self.instance.budget\_allocation,  
 is\_active=True  
 ).exclude(id=self.instance.id).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 consumed = BudgetTransaction.objects.filter(  
 allocation=self.instance.budget\_allocation,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation=self.instance.budget\_allocation,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 remaining\_budget = total\_budget - consumed + returned - other\_allocations  
 logger.debug(f"Remaining budget: {remaining\_budget}")  
 if allocated\_amount > remaining\_budget:  
 logger.warning(f"Allocated amount {allocated\_amount} exceeds remaining budget {remaining\_budget}")  
 raise forms.ValidationError(  
 f'مبلغ تخصیص نمی‌تواند بیشتر از بودجه باقی‌مانده ({remaining\_budget:,} ریال) باشد.'  
 )  
  
 return allocated\_amount  
  
 def clean\_allocation\_date(self):  
 *"""  
 اعتبارسنجی تاریخ تخصیص.  
 - تاریخ جلالی رو به میلادی تبدیل می‌کنه.  
 - مطمئن می‌شه تاریخ معتبر باشه.  
 """* date\_str = self.cleaned\_data.get('allocation\_date')  
 logger.debug(f"Cleaning allocation\_date: input='{date\_str}'")  
 if not date\_str:  
 logger.warning("Allocation date is empty")  
 raise forms.ValidationError(\_('تاریخ تخصیص اجباری است.'))  
 try:  
 parsed\_date = parse\_jalali\_date(str(date\_str), field\_name=\_('تاریخ تخصیص'))  
 logger.debug(f"Parsed allocation\_date: {parsed\_date}")  
 return parsed\_date  
 except ValueError as e:  
 logger.warning(f"Could not parse allocation\_date '{date\_str}': {str(e)}")  
 raise forms.ValidationError(str(e))  
 except Exception as e:  
 logger.error(f"Unexpected error parsing allocation\_date '{date\_str}': {str(e)}", exc\_info=True)  
 raise forms.ValidationError(\_('فرمت تاریخ تخصیص نامعتبر است.'))  
  
 def clean(self):  
 *"""  
 اعتبارسنجی کلی فرم.  
 - چک می‌کنه که تاریخ تخصیص در بازه دوره بودجه باشه.  
 """* cleaned\_data = super().clean()  
 allocation\_date = cleaned\_data.get('allocation\_date')  
 project = cleaned\_data.get('project')  
 logger.debug(f"Cleaning form, allocation\_date={allocation\_date}, project={project}")  
  
 if project and allocation\_date and self.instance.budget\_allocation:  
 start\_date = self.instance.budget\_allocation.budget\_period.start\_date  
 end\_date = self.instance.budget\_allocation.budget\_period.end\_date  
 logger.debug(f"Budget period: start={start\_date}, end={end\_date}")  
 if not (start\_date <= allocation\_date <= end\_date):  
 logger.warning(f"Allocation date {allocation\_date} is outside budget period {start\_date} to {end\_date}")  
 raise forms.ValidationError(\_('تاریخ تخصیص باید در بازه دوره بودجه باشد.'))  
  
 return cleaned\_data  
  
 def save(self, commit=True):  
 *"""  
 ذخیره تخصیص.  
 - کاربر ویرایش‌کننده رو ثبت می‌کنه.  
 - تخصیص رو در دیتابیس ذخیره می‌کنه.  
 """* instance = super().save(commit=False)  
 if self.user and self.user.is\_authenticated:  
 instance.modified\_by = self.user  
 logger.debug(f"Setting modified\_by to user {self.user}")  
 else:  
 logger.error("No authenticated user provided for saving allocation")  
 raise forms.ValidationError(\_('کاربر معتبر برای ویرایش تخصیص لازم است.'))  
 if commit:  
 try:  
 instance.save()  
 logger.info(f"ProjectBudgetAllocation {instance.pk} updated by {self.user}")  
 except Exception as e:  
 logger.error(f"Error saving ProjectBudgetAllocation: {str(e)}", exc\_info=True)  
 raise  
 return instance

from decimal import Decimal  
from django import forms  
from django.core.exceptions import ValidationError  
import jdatetime  
import re  
from django.utils.translation import gettext\_lazy as \_  
from budgets.models import ProjectBudgetAllocation, BudgetAllocation  
from core.models import Project, SubProject  
import logging  
  
logger = logging.getLogger(\_\_name\_\_)  
  
def to\_english\_digits(text):  
 persian\_digits = '۰۱۲۳۴۵۶۷۸۹'  
 english\_digits = '0123456789'  
 translation\_table = str.maketrans(persian\_digits, english\_digits)  
 return text.translate(translation\_table)  
  
class ProjectBudgetAllocationForm(forms.ModelForm):  
 allocation\_date = forms.CharField(  
 label=\_('تاریخ تخصیص'), *# Changed label slightly* widget=forms.TextInput(attrs={  
 'data-jdp': '',  
 'class': 'form-control form-control-lg jalali-datepicker', *# Added jalali-datepicker class* 'autocomplete': 'off',  
 'placeholder': \_('مثال: 1404/01/26') *# Updated placeholder* }),  
 required=True  
 )  
 class Meta:  
 model = ProjectBudgetAllocation  
 fields = ['budget\_allocation', 'project', 'subproject', 'allocated\_amount', 'allocation\_date', 'description']  
 widgets = {  
 'budget\_allocation': forms.Select(attrs={  
 'class': 'form-select select2',  
 'data-placeholder': \_('انتخاب تخصیص بودجه شعبه'),  
 }),  
 'project': forms.Select(attrs={  
 'class': 'form-select select2',  
 'data-placeholder': \_('انتخاب پروژه'),  
 }),  
 'subproject': forms.Select(attrs={  
 'class': 'form-select select2',  
 'data-placeholder': \_('انتخاب زیرپروژه (اختیاری)'),  
 }),  
 'allocated\_amount': forms.TextInput(attrs={ *# تغییر به TextInput برای پشتیبانی از اعداد پارسی* 'class': 'form-control numeric-input',  
 'placeholder': \_('مبلغ به ریال یا درصد'),  
 'dir': 'ltr',  
 'inputmode': 'numeric',  
 }),  
 'allocation\_date': forms.DateInput(attrs={  
 'data-jdp': 'true',  
 'class': 'form-control form-control-lg jalali-datepicker',  
 'autocomplete': 'off',  
 'placeholder': \_('مثال: 1404/01/26'),  
 }),  
 'is\_active': forms.CheckboxInput(attrs={'class': 'form-check-input'}),  
 'description': forms.Textarea(attrs={  
 'class': 'form-control',  
 'rows': 2,  
 'placeholder': \_('توضیحات مربوط به تخصیص بودجه'),  
 }),  
  
 }  
 labels = {  
 'budget\_allocation': \_('تخصیص بودجه شعبه'),  
 'project': \_('پروژه'),  
 'subproject': \_('زیرپروژه (اختیاری)'),  
 'allocated\_amount': \_('مبلغ تخصیص'),  
 'allocation\_date': \_('تاریخ تخصیص'),  
 'is\_active': \_('فعال'),  
 'description': \_('توضیحات'),  
 }  
 help\_texts = {  
 'allocated\_amount': \_('مبلغ یا درصد را وارد کنید. مثال: 1,000,000 یا 10%'),  
 'subproject': \_('در صورتی که بودجه به زیرپروژه خاصی تعلق دارد انتخاب کنید'),  
 'allocation\_date': \_('تاریخ را به فرمت 1404/01/17 وارد کنید'),  
 }  
  
 def \_\_init\_\_(self, \*args, organization\_id=None, user=None, \*\*kwargs):  
 self.user = user  
 self.organization\_id = organization\_id  
 super().\_\_init\_\_(\*args, \*\*kwargs)  
  
 *# اگر نمونه تخصیص وجود دارد* if self.instance and self.instance.pk:  
 if not self.instance.is\_active:  
 logger.warning(f"Editing inactive allocation {self.instance.id}")  
 *# اختیاری: غیرفعال کردن برخی فیلدها  
 # self.fields['allocated\_amount'].disabled = True* if not self.instance.budget\_allocation.budget\_period.is\_active:  
 logger.warning(f"Editing allocation {self.instance.id} with inactive budget period")  
 *# اختیاری: نمایش پیام یا غیرفعال کردن فیلدها* self.fields['allocated\_amount'].disabled = True  
  
 *# فیلتر کردن تخصیص‌های بودجه* if organization\_id:  
 budget\_allocations = BudgetAllocation.objects.filter(  
 organization\_id=organization\_id,  
 is\_active=True,  
 budget\_period\_\_is\_active=True  
 )  
 self.fields['budget\_allocation'].queryset = budget\_allocations  
  
 self.fields['project'].queryset = Project.objects.filter(  
 organizations\_\_id=organization\_id,  
 is\_active=True  
 )  
 self.fields['subproject'].queryset = SubProject.objects.filter(  
 project\_\_organizations\_\_id=organization\_id,  
 is\_active=True  
 )  
  
 if budget\_allocations.exists() and not self.initial.get('budget\_allocation'):  
 self.initial['budget\_allocation'] = budget\_allocations.first().id  
 logger.info(f"Set default budget\_allocation to ID {self.initial['budget\_allocation']}")  
  
 *# تنظیم تاریخ اولیه* if not self.initial.get('allocation\_date'):  
 if self.instance and self.instance.pk and self.instance.allocation\_date:  
 try:  
 self.initial['allocation\_date'] = jdatetime.date.fromgregorian(  
 date=self.instance.allocation\_date  
 ).strftime('%Y/%m/%d')  
 logger.debug(f"Set initial allocation\_date from instance: {self.initial['allocation\_date']}")  
 except Exception as e:  
 logger.error(f"Error formatting existing allocation date: {e}")  
 self.initial['allocation\_date'] = jdatetime.date.today().strftime('%Y/%m/%d')  
 else:  
 self.initial['allocation\_date'] = jdatetime.date.today().strftime('%Y/%m/%d')  
 logger.debug(f"Set default allocation\_date for new instance: {self.initial['allocation\_date']}")  
  
  
 def clean\_allocated\_amount(self):  
 allocated\_amount = self.cleaned\_data.get('allocated\_amount')  
 input\_mode = self.data.get('input\_mode', 'amount')  
 budget\_allocation = self.cleaned\_data.get('budget\_allocation')  
  
 if not allocated\_amount:  
 raise forms.ValidationError(\_('مبلغ تخصیص اجباری است.'))  
  
 *# تبدیل مقدار ورودی به عدد* try:  
 if isinstance(allocated\_amount, str):  
 allocated\_amount = allocated\_amount.replace(',', '').replace('،', '')  
 allocated\_amount = float(allocated\_amount)  
 allocated\_amount = Decimal(allocated\_amount)  
 except (ValueError, TypeError):  
 raise forms.ValidationError(\_('مقدار تخصیص نامعتبر است.'))  
  
 if input\_mode == 'percent':  
 if allocated\_amount < 0 or allocated\_amount > 100:  
 raise forms.ValidationError(\_('درصد باید بین 0 تا 100 باشد.'))  
 if budget\_allocation:  
 allocated\_amount = (allocated\_amount / 100) \* budget\_allocation.allocated\_amount  
 allocated\_amount = allocated\_amount.quantize(Decimal('1.')) *# گرد کردن به عدد صحیح* if allocated\_amount <= 0:  
 raise forms.ValidationError(\_('مبلغ تخصیص باید مثبت باشد.'))  
  
 if budget\_allocation:  
 remaining = budget\_allocation.get\_remaining\_amount()  
 if allocated\_amount > remaining:  
 raise forms.ValidationError(  
 f'مبلغ تخصیص بیشتر از باقی‌مانده بودجه ({remaining:,} ریال) است.'  
 )  
  
 return allocated\_amount  
  
 *# \* def clean\_allocation\_date(self):  
 # date\_input = self.cleaned\_data.get('allocation\_date')  
 # if not date\_input:  
 # raise forms.ValidationError(\_('تاریخ تخصیص اجباری است.'))  
 # if isinstance(date\_input, jdatetime.date):  
 # return date\_input.togregorian()  
 # raise forms.ValidationError(\_('تاریخ نامعتبر است.'))* def clean\_allocation\_date(self):  
 *"""Parse Jalali date string to Python date object."""* date\_str = self.cleaned\_data.get('allocation\_date')  
 logger.debug(f"Cleaning allocation\_date: input='{date\_str}'")  
 if not date\_str:  
 *# This might be redundant if field is required=True* raise ValidationError(\_('تاریخ تخصیص اجباری است.'))  
 try:  
 *# Use the utility function to parse* from Tanbakhsystem.utils import parse\_jalali\_date  
 parsed\_date = parse\_jalali\_date(str(date\_str), field\_name=\_('تاریخ تخصیص'))  
 logger.debug(f"Parsed allocation\_date: {parsed\_date}")  
 return parsed\_date *# Return the Python date object* except ValueError as e: *# Catch specific parsing errors* logger.warning(f"Could not parse allocation\_date '{date\_str}': {e}")  
 raise ValidationError(e) *# Show the specific error from parse\_jalali\_date* except Exception as e: *# Catch other unexpected errors* logger.error(f"Unexpected error parsing allocation\_date '{date\_str}': {e}", exc\_info=True)  
 raise ValidationError(\_('فرمت تاریخ تخصیص نامعتبر است.'))  
  
 def clean(self):  
 cleaned\_data = super().clean()  
 budget\_allocation = cleaned\_data.get('budget\_allocation')  
 project = cleaned\_data.get('project')  
 subproject = cleaned\_data.get('subproject')  
 allocation\_date = cleaned\_data.get('allocation\_date')  
  
 if not budget\_allocation:  
 raise forms.ValidationError(\_('لطفاً یک تخصیص بودجه انتخاب کنید.'))  
 if not project:  
 raise forms.ValidationError(\_('لطفاً یک پروژه انتخاب کنید.'))  
 if subproject and subproject.project != project:  
 raise forms.ValidationError(\_('زیرپروژه باید متعلق به پروژه انتخاب‌شده باشد.'))  
  
 if allocation\_date and budget\_allocation:  
 start\_date = budget\_allocation.budget\_period.start\_date  
 end\_date = budget\_allocation.budget\_period.end\_date  
 if not (start\_date <= allocation\_date <= end\_date):  
 raise forms.ValidationError(\_('تاریخ تخصیص باید در بازه دوره بودجه باشد.'))  
  
 return cleaned\_data  
  
 def save(self, commit=True):  
 instance = super().save(commit=False)  
 if self.user and self.user.is\_authenticated:  
 instance.created\_by = self.user  
 else:  
 raise forms.ValidationError(\_('کاربر معتبر برای ایجاد تخصیص لازم است.'))  
 if commit:  
 try:  
 instance.save()  
 instance.budget\_allocation.remaining\_amount = instance.budget\_allocation.get\_remaining\_amount()  
 instance.budget\_allocation.save(update\_fields=['remaining\_amount'])  
 except Exception as e:  
 logger.error(f"Error saving ProjectBudgetAllocation: {str(e)}")  
 raise  
 return instance

*# budgets/views.py*from django.views.generic import ListView  
from django.contrib import messages  
from django.utils.translation import gettext\_lazy as \_  
from django.db.models import Q, Sum  
from decimal import Decimal  
  
from Tanbakhsystem.utils import parse\_jalali\_date, to\_english\_digits  
from budgets.budget\_calculations import get\_project\_remaining\_budget, get\_project\_used\_budget, get\_budget\_details, \  
 get\_project\_total\_budget  
from budgets.models import ProjectBudgetAllocation, BudgetPeriod, BudgetItem  
from core.models import Project  
from core.models import Organization  
from core.views import PermissionBaseView  
  
import logging  
  
logger = logging.getLogger(\_\_name\_\_)  
  
class ProjectBudgetAllocationListView2D(PermissionBaseView, ListView):  
 model = ProjectBudgetAllocation  
 template\_name = 'budgets/budget/project\_budget\_allocation\_list\_2d.html'  
 context\_object\_name = 'project\_allocations'  
 paginate\_by = 10  
 permission\_required = 'budgets.view\_projectbudgetallocation'  
  
 def get\_queryset(self):  
 queryset = ProjectBudgetAllocation.objects.filter(is\_active=True).select\_related(  
 'budget\_allocation\_\_organization',  
 'budget\_allocation\_\_budget\_period',  
 'project',  
 )  
 logger.info(f"ProjectBudgetAllocationListView queryset: {queryset.count()} records")  
  
 *# فیلتر سازمان* organization\_id = self.request.GET.get('organization')  
 if organization\_id:  
 try:  
 organization = Organization.objects.get(id=organization\_id, is\_active=True)  
 queryset = queryset.filter(budget\_allocation\_\_organization=organization)  
 except Organization.DoesNotExist:  
 logger.warning(f"Invalid organization\_id: {organization\_id}")  
 messages.error(self.request, \_('سازمان انتخاب‌شده نامعتبر است.'))  
  
 *# فیلترهای جستجو* query = self.request.GET.get('q', '')  
 date\_from = self.request.GET.get('date\_from', '')  
 date\_to = self.request.GET.get('date\_to', '')  
 project\_id = self.request.GET.get('project')  
  
 if query:  
 queryset = queryset.filter(  
 Q(project\_\_name\_\_icontains=query) |  
 Q(project\_\_code\_\_icontains=query) |  
 Q(budget\_allocation\_\_organization\_\_name\_\_icontains=query)  
 )  
  
 if project\_id:  
 try:  
 project = Project.objects.get(id=project\_id, is\_active=True)  
 queryset = queryset.filter(project=project)  
 except Project.DoesNotExist:  
 logger.warning(f"Invalid project\_id: {project\_id}")  
 messages.error(self.request, \_('پروژه انتخاب‌شده نامعتبر است.'))  
  
 if date\_from:  
 try:  
 date\_from = parse\_jalali\_date(to\_english\_digits(date\_from))  
 queryset = queryset.filter(allocation\_date\_\_gte=date\_from)  
 except ValueError:  
 logger.warning(f"Invalid date\_from format: {date\_from}")  
 messages.error(self.request, \_('فرمت تاریخ شروع نامعتبر است.'))  
  
 if date\_to:  
 try:  
 date\_to = parse\_jalali\_date(to\_english\_digits(date\_to))  
 queryset = queryset.filter(allocation\_date\_\_lte=date\_to)  
 except ValueError:  
 logger.warning(f"Invalid date\_to format: {date\_to}")  
 messages.error(self.request, \_('فرمت تاریخ پایان نامعتبر است.'))  
  
 *# بررسی دسترسی سازمانی* user\_organizations = self.request.user.get\_authorized\_organizations()  
 queryset = queryset.filter(budget\_allocation\_\_organization\_\_in=user\_organizations)  
  
 return queryset.order\_by('-allocation\_date')  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 organization\_id = self.request.GET.get('organization')  
 project\_id = self.request.GET.get('project')  
  
 *# مقداردهی اولیه* total\_allocated = Decimal('0')  
 total\_used = Decimal('0')  
 total\_remaining = Decimal('0')  
  
 *# محاسبه اطلاعات کل* queryset = self.get\_queryset()  
 for allocation in queryset:  
 project = allocation.project  
 total\_allocated += allocation.allocated\_amount  
 total\_used += get\_project\_used\_budget(project)  
 total\_remaining += get\_project\_remaining\_budget(project)  
  
 *# اطلاعات سازمان* organization = None  
 if organization\_id:  
 try:  
 organization = Organization.objects.get(id=organization\_id, is\_active=True)  
 context['organization'] = organization  
 context['org\_budget'] = get\_budget\_details(entity=organization)  
 except Organization.DoesNotExist:  
 logger.warning(f"Invalid organization\_id: {organization\_id}")  
 context['status\_message'] = \_('سازمان انتخاب‌شده نامعتبر است.')  
  
 *# اطلاعات پروژه* project\_data = {}  
 if project\_id:  
 try:  
 project = Project.objects.get(id=project\_id, is\_active=True)  
 project\_total = get\_project\_total\_budget(project)  
 project\_used = get\_project\_used\_budget(project)  
 project\_remaining = get\_project\_remaining\_budget(project)  
 project\_data = {  
 'total': project\_total,  
 'used': project\_used,  
 'remaining': project\_remaining,  
 'allocated\_percentage': (project\_used / project\_total \* Decimal('100')) if project\_total > 0 else Decimal('0'),  
 'remaining\_percentage': (project\_remaining / project\_total \* Decimal('100')) if project\_total > 0 else Decimal('0'),  
 }  
 logger.info(  
 f"Project {project.id}: total={project\_total}, used={project\_used}, "  
 f"remaining={project\_remaining}"  
 )  
 except Project.DoesNotExist:  
 logger.warning(f"Invalid project\_id: {project\_id}")  
 context['status\_message'] = \_('پروژه انتخاب‌شده نامعتبر است.')  
  
 context.update({  
 'total\_allocated': total\_allocated,  
 'total\_used': total\_used,  
 'total\_remaining': total\_remaining,  
 'project\_data': project\_data,  
 'organizations': Organization.objects.filter(is\_active=True, id\_\_in=self.request.user.get\_authorized\_organizations()),  
 'projects': Project.objects.filter(is\_active=True),  
 'status': 'filtered',  
 'status\_message': context.get('status\_message', \_('نمایش تخصیص‌های فیلترشده')),  
 'query': self.request.GET.get('q', ''),  
 'date\_from': self.request.GET.get('date\_from', ''),  
 'date\_to': self.request.GET.get('date\_to', ''),  
 'selected\_organization': organization\_id,  
 'selected\_project': project\_id,  
 })  
  
 logger.debug(f"ProjectBudgetAllocationListView context: {context}")  
 return context

*# budgets/views.py*from django.views.generic.edit import UpdateView  
from django.shortcuts import get\_object\_or\_404, redirect  
from django.contrib import messages  
from django.urls import reverse\_lazy  
from django.http import JsonResponse, Http404  
from django.db import transaction  
from django.db.models import Sum, Q  
from django.core.exceptions import ValidationError  
from decimal import Decimal  
*# from .models import ProjectBudgetAllocation, BudgetTransaction  
# from .forms import ProjectBudgetAllocationForm # فرم هماهنگ با ProjectBudgetAllocation*import logging  
from django.utils.translation import gettext\_lazy as \_  
  
from budgets.ProjectBudgetAllocation.forms\_ProjectBudgetAllocation import ProjectBudgetAllocationForm  
from budgets.models import ProjectBudgetAllocation, BudgetTransaction  
from core.models import SubProject  
  
*# تنظیم لاگر برای ثبت جزئیات خطاها و رویدادها*logger = logging.getLogger(\_\_name\_\_)  
from core.PermissionBase import PermissionBaseView  
  
class Project\_\_Budget\_\_Allocation\_\_Edit\_\_View(PermissionBaseView, UpdateView):  
 *# مدل ویو: از ProjectBudgetAllocation استفاده می‌کنیم که تخصیص بودجه به پروژه/زیرپروژه رو مدیریت می‌کنه* model = ProjectBudgetAllocation  
 *# فرم: از فرم جدید ProjectBudgetAllocationForm استفاده می‌کنیم که برای ProjectBudgetAllocation طراحی شده* form\_class = ProjectBudgetAllocationForm  
 *# تمپلیت: همون تمپلیت قبلی که داده‌های تخصیص و سازمان رو نمایش می‌ده* template\_name = 'budgets/budget/project\_budget\_allocation\_edit.html'  
  
 def get\_object(self, queryset=None):  
 logger.debug(  
 f"Attempting to get ProjectBudgetAllocation with pk={self.kwargs['pk']} for user={self.request.user}")  
 try:  
 allocation = ProjectBudgetAllocation.objects.get(pk=self.kwargs['pk'])  
 except ProjectBudgetAllocation.DoesNotExist:  
 logger.error(f"ProjectBudgetAllocation with pk={self.kwargs['pk']} does not exist")  
 raise Http404(\_("تخصیص موردنظر یافت نشد"))  
  
 authorized\_orgs = self.request.user.get\_authorized\_organizations()  
 logger.debug(f"Authorized organizations: {list(authorized\_orgs.values('id', 'name'))}")  
 logger.debug(f"Allocation organization: {allocation.budget\_allocation.organization}")  
 if allocation.budget\_allocation.organization not in authorized\_orgs:  
 logger.error(  
 f"User {self.request.user} does not have access to organization {allocation.budget\_allocation.organization} (ID: {allocation.budget\_allocation.organization.id})")  
 raise Http404(  
 \_("شما به سازمان این تخصیص ({} - {}) دسترسی ندارید").format(  
 allocation.budget\_allocation.organization.id,  
 allocation.budget\_allocation.organization.name  
 )  
 )  
  
 if not allocation.budget\_allocation.budget\_period.is\_active:  
 logger.warning(f"Budget period {allocation.budget\_allocation.budget\_period.id} is inactive")  
 messages.warning(self.request, \_("دوره بودجه این تخصیص بسته شده است، اما می‌توانید ویرایش کنید."))  
  
 logger.info(  
 f"Successfully retrieved allocation👍💸 {allocation.id} for organization {allocation.budget\_allocation.organization}")  
 return allocation  
  
 def get\_form\_kwargs(self):  
 *"""  
 ارسال پارامترهای اضافی به فرم.  
 - user: برای ثبت کاربر ویرایش‌کننده.  
 - organization\_id: برای فیلتر کردن پروژه‌ها و زیرپروژه‌ها در فرم.  
 """* kwargs = super().get\_form\_kwargs()  
 kwargs['user'] = self.request.user  
 try:  
 kwargs['organization\_id'] = self.get\_object().budget\_allocation.organization.id  
 logger.debug(f"Passing organization\_id={kwargs['organization\_id']} to form")  
 except AttributeError as e:  
 logger.error(f"Failed to get organization\_id: {str(e)}")  
 raise  
 return kwargs  
  
 def get\_context\_data(self, \*\*kwargs):  
 *"""  
 ارسال داده‌های اضافی به تمپلیت برای نمایش خلاصه بودجه.  
 - organization: برای نمایش نام سازمان و ریدایرکت‌ها.  
 - total\_org\_budget: بودجه کل تخصیص‌شده به سازمان.  
 - consumed\_amount: مبلغ مصرف‌شده برای این تخصیص.  
 - remaining\_org\_budget: بودجه باقی‌مانده.  
 - allocation: خود تخصیص برای نمایش جزئیات.  
 """* context = super().get\_context\_data(\*\*kwargs)  
 allocation = self.get\_object()  
 organization = allocation.budget\_allocation.organization  
 budget\_allocation = allocation.budget\_allocation  
 logger.debug(f"Preparing context for allocation {allocation.id}, organization {organization.name}")  
  
 *# محاسبه بودجه کل سازمان* total\_org\_budget = budget\_allocation.allocated\_amount  
 logger.debug(f"Total organization budget: {total\_org\_budget}")  
  
 *# پیدا کردن subproject‌های مرتبط با این تخصیص* subprojects = SubProject.objects.filter(allocations=allocation)  
  
 *# محاسبه مبلغ مصرف‌شده برای این تخصیص* consumed = BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 allocation\_\_project=allocation.project,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
  
 returned = BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 allocation\_\_project=allocation.project,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
  
 consumed\_amount = consumed - returned  
 logger.debug(f"Consumed: {consumed}, Returned: {returned}, Net consumed: {consumed\_amount}")  
  
 *# محاسبه بودجه باقی‌مانده* remaining\_org\_budget = (Decimal(total\_org\_budget) - Decimal(BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')) +  
 Decimal(BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')))  
 logger.debug(f"Remaining organization budget: {remaining\_org\_budget}")  
  
 context.update({  
 'organization': organization,  
 'total\_org\_budget': total\_org\_budget,  
 'consumed\_amount': consumed\_amount,  
 'remaining\_org\_budget': remaining\_org\_budget,  
 'allocation': allocation,  
 'subprojects': subprojects,  
 })  
 return context  
  
 def validate\_allocation\_amount(self, allocation, new\_amount):  
 *"""  
 اعتبارسنجی مبلغ تخصیص.  
 - چک می‌کنه که مبلغ جدید از مبلغ مصرف‌شده کمتر نباشه.  
 - چک می‌کنه که مبلغ جدید از بودجه باقی‌مانده بیشتر نباشه.  
 - اگه خطایی رخ بده، ValidationError پرت می‌کنه.  
 """* logger.debug(f"Validating new amount {new\_amount} for allocation {allocation.id}")  
 budget\_allocation = allocation.budget\_allocation  
  
 consumed = BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 allocation\_\_project=allocation.project,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 allocation\_\_project=allocation.project,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 total\_consumed = consumed - returned  
 logger.debug(f"Total consumed for validation: {total\_consumed}")  
  
 if new\_amount < total\_consumed:  
 logger.warning(f"Validation failed: New amount {new\_amount} is less than consumed {total\_consumed}")  
 raise ValidationError(  
 f"مبلغ تخصیص ({new\_amount:,.0f} ریال) نمی‌تواند کمتر از مبلغ مصرف‌شده ({total\_consumed:,.0f} ریال) باشد."  
 )  
  
 remaining\_budget = budget\_allocation.allocated\_amount - BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0') + BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 logger.debug(f"Remaining budget for validation: {remaining\_budget}")  
  
 if new\_amount > remaining\_budget:  
 logger.warning(f"Validation failed: New amount {new\_amount} exceeds remaining budget {remaining\_budget}")  
 raise ValidationError(  
 f"مبلغ تخصیص ({new\_amount:,.0f} ریال) نمی‌تواند بیشتر از بودجه باقی‌مانده ({remaining\_budget:,.0f} ریال) باشد."  
 )  
  
 def form\_valid(self, form):  
 *"""  
 ذخیره فرم با اعتبارسنجی و پاسخ AJAX.  
 - مبلغ جدید رو اعتبارسنجی می‌کنه.  
 - تخصیص رو ذخیره می‌کنه و کاربر ویرایش‌کننده رو ثبت می‌کنه.  
 - درصد باقی‌مانده رو محاسبه می‌کنه.  
 - برای درخواست‌های AJAX پاسخ JSON برمی‌گردونه.  
 - برای درخواست‌های معمولی پیام موفقیت نمایش می‌ده و ریدایرکت می‌کنه.  
 """* try:  
 with transaction.atomic():  
 old\_amount = self.object.allocated\_amount  
 new\_amount = form.cleaned\_data['allocated\_amount']  
 logger.debug(f"Form valid, processing update for allocation {self.object.id}, old\_amount={old\_amount}, new\_amount={new\_amount}")  
  
 *# اعتبارسنجی مبلغ تخصیص* self.validate\_allocation\_amount(self.object, new\_amount)  
  
 allocation = form.save(commit=False)  
 allocation.modified\_by = self.request.user  
 allocation.save()  
 logger.info(f"Allocation {allocation.id} updated successfully by user {self.request.user}")  
  
 *# محاسبه درصد باقی‌مانده* consumed = BudgetTransaction.objects.filter(  
 allocation=allocation.budget\_allocation,  
 project=allocation.project,  
 subproject=allocation.subproject,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation=allocation.budget\_allocation,  
 project=allocation.project,  
 subproject=allocation.subproject,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 remaining = allocation.allocated\_amount - consumed + returned  
 remaining\_percent = (  
 (Decimal(remaining )/ Decimal(allocation.allocated\_amount \* 100)) if allocation.allocated\_amount > 0 else Decimal('0')  
 )  
 logger.debug(f"Remaining amount: {remaining}, Remaining percent: {remaining\_percent}")  
  
 if self.request.headers.get('X-Requested-With') == 'XMLHttpRequest':  
 logger.debug("Returning AJAX response")  
 return JsonResponse({  
 'success': True,  
 'allocated\_amount': f"{allocation.allocated\_amount:,.0f}",  
 'remaining\_percent': f"{remaining\_percent:.1f}",  
 'is\_active': allocation.is\_active,  
 'organization\_id': allocation.budget\_allocation.organization.id  
 })  
  
 messages.success(self.request, \_("تخصیص بودجه با موفقیت ویرایش شد"))  
 logger.debug("Redirecting to success URL")  
 return super().form\_valid(form)  
  
 except ValidationError as e:  
 logger.error(f"Validation error: {str(e)}")  
 form.add\_error('allocated\_amount', str(e))  
 return self.form\_invalid(form)  
 except Exception as e:  
 logger.error(f"Unexpected error in form\_valid: {str(e)}", exc\_info=True)  
 form.add\_error(None, \_("خطایی در ذخیره تخصیص رخ داد."))  
 return self.form\_invalid(form)  
  
 def form\_invalid(self, form):  
 *"""  
 مدیریت خطاهای فرم.  
 - برای درخواست‌های AJAX خطاها رو به‌صورت JSON برمی‌گردونه.  
 - برای درخواست‌های معمولی پیام خطا نمایش می‌ده و فرم رو رندر می‌کنه.  
 """* logger.error(f"Form invalid, errors: {form.errors}")  
 if self.request.headers.get('X-Requested-With') == 'XMLHttpRequest':  
 errors = {field: errors for field, errors in form.errors.items()}  
 logger.debug(f"Returning AJAX error response: {errors}")  
 return JsonResponse({  
 'success': False,  
 'errors': errors  
 }, status=400)  
  
 messages.error(self.request, \_("لطفاً خطاهای فرم را بررسی کنید"))  
 return super().form\_invalid(form)  
  
 def get\_success\_url(self):  
 *"""  
 URL ریدایرکت پس از ذخیره موفق.  
 - به لیست تخصیص‌ها با organization\_id درست هدایت می‌کنه.  
 """* try:  
 organization\_id = self.object.budget\_allocation.organization.id  
 logger.debug(f"Redirecting to project\_budget\_allocation\_list with organization\_id={organization\_id}")  
 return reverse\_lazy('project\_budget\_allocation\_list', kwargs={'organization\_id': organization\_id})  
 except AttributeError as e:  
 logger.error(f"Failed to get organization\_id for redirect: {str(e)}")  
 return reverse\_lazy('organization\_list')  
  
 def dispatch(self, request, \*args, \*\*kwargs):  
 *"""  
 مدیریت دسترسی و خطاها.  
 - تخصیص رو با get\_object می‌گیره.  
 - اگه خطایی رخ بده (مثل تخصیص ناموجود یا عدم دسترسی)، ریدایرکت به organization\_list می‌کنه.  
 """* logger.debug(f"Dispatching request for allocation edit, pk={self.kwargs.get('pk')}, user={self.request.user}")  
 try:  
 allocation = self.get\_object()  
 logger.info(f"Accessing edit view for allocation {allocation.id}")  
 return super().dispatch(request, \*args, \*\*kwargs)  
 except Exception as e:  
 logger.error(f"Error accessing allocation: {str(e)}", exc\_info=True)  
 messages.error(request, \_("تخصیص موردنظر یافت نشد یا شما دسترسی ندارید"))  
 return redirect('organization\_list')

import json  
  
from django.core.serializers.json import DjangoJSONEncoder  
from django.http import Http404  
from django.utils.translation import gettext\_lazy as \_  
*# --- ProjectBudgetAllocation CRUD ---*from decimal import Decimal  
  
from django.contrib import messages  
from django.core.exceptions import ValidationError, PermissionDenied  
from django.db import transaction  
from django.db.models import Sum  
from django.shortcuts import get\_object\_or\_404, redirect  
from django.urls import reverse\_lazy  
from django.views.generic import ListView, DetailView, UpdateView, DeleteView, CreateView, TemplateView  
  
from budgets.ProjectBudgetAllocation.forms\_ProjectBudgetAllocation import ProjectBudgetAllocationForm  
from budgets.models import ProjectBudgetAllocation, BudgetPeriod, BudgetTransaction, BudgetAllocation  
from core.PermissionBase import PermissionBaseView  
from budgets.budget\_calculations import (  
 get\_budget\_details, get\_organization\_budget,  
 get\_project\_remaining\_budget, get\_subproject\_remaining\_budget, get\_subproject\_total\_budget,  
 get\_project\_total\_budget, get\_project\_used\_budget  
)  
import logging  
  
from core.models import Organization, Project, SubProject  
  
logger = logging.getLogger(\_\_name\_\_)  
  
class ProjectBudgetAllocationListView(ListView):  
 model = ProjectBudgetAllocation  
 template\_name = 'budgets/budget/project\_budget\_allocation\_list.html'  
 context\_object\_name = 'allocations'  
 paginate\_by = 10  
  
 def get\_queryset(self):  
 organization\_id = self.kwargs['organization\_id']  
 organization = get\_object\_or\_404(Organization, id=organization\_id)  
 project\_id = self.request.GET.get('project\_id')  
 queryset = ProjectBudgetAllocation.objects.filter(  
 budget\_allocation\_\_organization=organization,  
 budget\_allocation\_\_is\_active=True  
 ).select\_related(  
 'project', 'subproject', 'budget\_allocation\_\_budget\_period', 'budget\_allocation\_\_organization'  
 ).order\_by('-allocation\_date')  
 if project\_id:  
 queryset = queryset.filter(project\_id=project\_id)  
 logger.info(f"Allocations for org {organization\_id}: {queryset.count()} records")  
 return queryset  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 organization = get\_object\_or\_404(Organization, id=self.kwargs['organization\_id'])  
  
 *# دریافت تخصیص‌های بودجه سازمان* budget\_allocations = BudgetAllocation.objects.filter(  
 organization=organization, is\_active=True  
 ).select\_related('budget\_period')  
  
 *# محاسبه بودجه کل و باقی‌مانده سازمان* total\_org\_budget = budget\_allocations.aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
  
 consumed = BudgetTransaction.objects.filter(  
 allocation\_\_in=budget\_allocations,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation\_\_in=budget\_allocations,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 remaining\_org\_budget = total\_org\_budget - consumed + returned  
  
 *# داده‌های پروژه‌ها* project\_data = []  
 projects = Project.objects.filter(  
 organizations=organization, is\_active=True  
 ).prefetch\_related('subprojects')  
  
 for project in projects:  
 project\_allocations = ProjectBudgetAllocation.objects.filter(  
 project=project, subproject\_\_isnull=True  
 )  
 total\_budget = project\_allocations.aggregate(  
 total=Sum('allocated\_amount')  
 )['total'] or Decimal('0')  
  
 *# استفاده از related\_name صحیح: project\_allocations* project\_budget\_allocations = BudgetAllocation.objects.filter(  
 project\_allocations\_\_project=project  
 )  
 consumed = BudgetTransaction.objects.filter(  
 allocation\_\_in=project\_budget\_allocations,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation\_\_in=project\_budget\_allocations,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
  
 remaining\_budget = total\_budget - consumed + returned  
 remaining\_percentage = (  
 (remaining\_budget / total\_budget \* 100) if total\_budget > 0 else Decimal('0')  
 )  
 allocated\_percentage = (  
 (total\_budget / total\_org\_budget \* 100) if total\_org\_budget > 0 else Decimal('0')  
 )  
  
 project\_data.append({  
 'project': project,  
 'total\_budget': total\_budget,  
 'remaining\_budget': remaining\_budget,  
 'remaining\_percentage': remaining\_percentage,  
 'allocated\_percentage': allocated\_percentage  
 })  
  
 *# بررسی شرایط خاص* if not budget\_allocations.exists():  
 messages.warning(self.request, \_("هیچ تخصیص بودجه فعالی برای این سازمان یافت نشد"))  
  
 context.update({  
 'organization': organization,  
 'total\_org\_budget': total\_org\_budget,  
 'remaining\_org\_budget': remaining\_org\_budget,  
 'project\_data': project\_data,  
 })  
 logger.info(f"Context prepared for organization {organization.id}: {context}")  
 return context  
  
  
class ProjectBudgetAllocationDetailView(PermissionBaseView, DetailView):  
 model = ProjectBudgetAllocation  
 template\_name = 'budgets/budget/project\_budget\_allocation\_detail.html'  
 context\_object\_name = 'allocation'  
 permission\_codename = 'budgets.ProjectBudgetAllocation\_view'  
 permission\_denied\_message = \_('متاسفانه دسترسی مجاز ندارید')  
 check\_organization = True  
  
 def get\_object(self, queryset=None):  
 try:  
 obj = super().get\_object(queryset)  
 if not obj.is\_active:  
 logger.error(f"Allocation {obj.pk} is inactive")  
 raise Http404(\_('تخصیص بودجه غیرفعال است.'))  
 *# بررسی دسترسی سازمانی* try:  
 user\_organizations = self.request.user.get\_authorized\_organizations()  
 except AttributeError:  
 user\_organizations = self.request.user.organizations.all()  
 if not user\_organizations.filter(pk=obj.budget\_allocation.organization.pk).exists():  
 logger.warning(  
 f"User {self.request.user.username} attempted to access allocation {obj.pk} "  
 f"without organization permission"  
 )  
 raise PermissionDenied(self.permission\_denied\_message)  
 return obj  
 except ProjectBudgetAllocation.DoesNotExist:  
 logger.error(f"Allocation {self.kwargs['pk']} does not exist")  
 messages.error(self.request, \_('تخصیص بودجه مورد نظر یافت نشد.'))  
 return None  
  
 def dispatch(self, request, \*args, \*\*kwargs):  
 obj = self.get\_object()  
 if not obj:  
 return redirect('budgetallocation\_list')  
 return super().dispatch(request, \*args, \*\*kwargs)  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 allocation = self.get\_object()  
 if not allocation:  
 return context  
  
 organization = allocation.budget\_allocation.organization  
 budget\_allocation = allocation.budget\_allocation  
 logger.debug(f"Preparing context for allocation {allocation.id}, organization {organization.name}")  
  
 *# بودجه کل سازمان* total\_org\_budget = budget\_allocation.allocated\_amount  
 logger.debug(f"Total organization budget: {total\_org\_budget}")  
  
 *# پیدا کردن زیرپروژه‌های مرتبط* subprojects = SubProject.objects.filter(allocations=allocation)  
  
 *# محاسبه تراکنش‌ها* consumed = BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 allocation\_\_project=allocation.project,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
  
 returned = BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 allocation\_\_project=allocation.project,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
  
 consumed\_amount = consumed - returned  
 logger.debug(f"Consumed: {consumed}, Returned: {returned}, Net consumed: {consumed\_amount}")  
  
 *# بودجه باقی‌مانده* remaining\_org\_budget = total\_org\_budget - BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0') + BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 logger.debug(f"Remaining organization budget: {remaining\_org\_budget}")  
  
 *# تراکنش‌های اخیر* recent\_transactions = BudgetTransaction.objects.filter(  
 allocation=budget\_allocation,  
 allocation\_\_project=allocation.project  
 ).order\_by('-timestamp')[:10]  
  
 context.update({  
 'organization': organization,  
 'total\_org\_budget': total\_org\_budget,  
 'consumed\_amount': consumed\_amount,  
 'remaining\_org\_budget': remaining\_org\_budget,  
 'subprojects': subprojects,  
 'recent\_transactions': recent\_transactions,  
 'project': allocation.project,  
 })  
 *# اضافه کردن اطلاعات بودجه* project = allocation.project  
 context.update({  
 'total\_budget': get\_project\_total\_budget(project),  
 'used\_budget': get\_project\_used\_budget(project),  
 'remaining\_budget': get\_project\_remaining\_budget(project),  
 'budget\_details': get\_budget\_details(entity=project),  
 })  
  
 return context  
  
 def get(self, request, \*args, \*\*kwargs):  
 try:  
 return super().get(request, \*args, \*\*kwargs)  
 except Exception as e:  
 logger.error(f"Error accessing allocation {self.kwargs.get('pk')}: {str(e)}")  
 messages.error(request, \_("خطایی در دسترسی به جزئیات تخصیص بودجه رخ داد."))  
 return self.redirect\_to\_organizations()  
  
 def redirect\_to\_organizations(self):  
 return redirect('organization\_list')  
  
  
class ProjectBudgetAllocationCreateView(PermissionBaseView,CreateView):  
 model = ProjectBudgetAllocation  
 form\_class = ProjectBudgetAllocationForm  
 template\_name = 'budgets/budget/project\_budget\_allocation.html'  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 organization = get\_object\_or\_404(Organization, id=self.kwargs['organization\_id'])  
 context['organization'] = organization  
  
 *# تخصیص‌های بودجه فعال* budget\_allocations = BudgetAllocation.objects.filter(  
 organization=organization,  
 is\_active=True,  
 budget\_period\_\_is\_active=True  
 ).select\_related('budget\_period', 'budget\_item').order\_by('-allocation\_date')  
  
 if not budget\_allocations.exists():  
 logger.warning(f"No active BudgetAllocation found for organization {organization.id}")  
 messages.warning(self.request, \_("هیچ تخصیص بودجه فعالی برای این سازمان یافت نشد"))  
 context.update({  
 'form': None,  
 'projects': [],  
 'budget\_allocation': None,  
 'budget\_period': None,  
 'total\_org\_budget': Decimal('0'),  
 'remaining\_amount': Decimal('0'),  
 'remaining\_percent': Decimal('0'),  
 'warning\_threshold': 50,  
 'project\_data': [],  
 'budget\_allocations\_json': '[]',  
 'subprojects\_json': '[]',  
 })  
 return context  
  
 *# داده‌های JSON برای جاوااسکریپت* from django\_jalali.templatetags.jformat import jformat  
 allocations\_list\_for\_json = [  
 {  
 'id': alloc.id,  
 'name': f"{alloc.budget\_item.name} - {alloc.budget\_period.name} ({alloc.allocation\_date })",  
 'allocated\_amount': float(alloc.allocated\_amount),  
 'remaining\_amount': float(alloc.get\_remaining\_amount()),  
 } for alloc in budget\_allocations  
 ]  
 context['budget\_allocations\_json'] = json.dumps(allocations\_list\_for\_json, cls=DjangoJSONEncoder)  
  
 subprojects\_list = list(SubProject.objects.filter(  
 project\_\_organizations=organization,  
 is\_active=True  
 ).values('id', 'name', 'project\_id'))  
 context['subprojects\_json'] = json.dumps(subprojects\_list, cls=DjangoJSONEncoder)  
  
 *# تخصیص اولیه* form = context.get('form')  
 initial\_allocation\_id = None  
 if form and form.initial.get('budget\_allocation'):  
 initial\_allocation\_id = form.initial['budget\_allocation']  
 elif form and form.is\_bound and form.data.get('budget\_allocation'):  
 initial\_allocation\_id = form.data['budget\_allocation']  
  
 budget\_allocation = budget\_allocations.filter(id=initial\_allocation\_id).first() or budget\_allocations.first()  
 context['budget\_allocation'] = budget\_allocation  
  
 *# محاسبه بودجه کل و باقی‌مانده شعبه* total\_org\_budget = budget\_allocations.aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 consumed = BudgetTransaction.objects.filter(  
 allocation\_\_in=budget\_allocations,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation\_\_in=budget\_allocations,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 remaining\_amount = total\_org\_budget - consumed + returned  
 remaining\_percent = (remaining\_amount / total\_org\_budget \* 100) if total\_org\_budget > 0 else Decimal('0')  
  
 *# محاسبه برای تخصیص انتخاب‌شده* allocation\_total = budget\_allocation.allocated\_amount if budget\_allocation else Decimal('0')  
 allocation\_remaining = budget\_allocation.get\_remaining\_amount() if budget\_allocation else Decimal('0')  
 allocation\_remaining\_percent = (  
 (allocation\_remaining / allocation\_total \* 100) if allocation\_total > 0 else Decimal('0')  
 )  
  
 *# پروژه‌ها* projects = Project.objects.filter(  
 organizations=organization,  
 is\_active=True  
 ).prefetch\_related('subprojects', 'budget\_allocations').order\_by('name')  
  
 project\_data = []  
 for project in projects:  
 total\_budget = ProjectBudgetAllocation.objects.filter(  
 project=project,  
 subproject\_\_isnull=True  
 ).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 project\_budget\_allocations = BudgetAllocation.objects.filter(  
 project\_allocations\_\_project=project  
 )  
 consumed = BudgetTransaction.objects.filter(  
 allocation\_\_in=project\_budget\_allocations,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 returned = BudgetTransaction.objects.filter(  
 allocation\_\_in=project\_budget\_allocations,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 remaining\_budget = total\_budget - consumed + returned  
 remaining\_percentage = (  
 (remaining\_budget / total\_budget \* 100) if total\_budget > 0 else Decimal('0')  
 )  
  
 project.total\_budget = total\_budget  
 project.remaining\_budget = remaining\_budget  
 project.total\_percent = (  
 (total\_budget / total\_org\_budget \* 100) if total\_org\_budget > 0 else Decimal('0')  
 )  
 project.remaining\_percent = remaining\_percentage  
  
 for subproject in project.subprojects.filter(is\_active=True):  
 subproject\_budget = ProjectBudgetAllocation.objects.filter(  
 subproject=subproject  
 ).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 subproject\_budget\_allocations = BudgetAllocation.objects.filter(  
 project\_allocations\_\_subproject=subproject  
 )  
 sub\_consumed = BudgetTransaction.objects.filter(  
 allocation\_\_in=subproject\_budget\_allocations,  
 transaction\_type='CONSUMPTION'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 sub\_returned = BudgetTransaction.objects.filter(  
 allocation\_\_in=subproject\_budget\_allocations,  
 transaction\_type='RETURN'  
 ).aggregate(total=Sum('amount'))['total'] or Decimal('0')  
 subproject.total\_budget = subproject\_budget  
 subproject.remaining\_budget = subproject\_budget - sub\_consumed + sub\_returned  
 subproject.remaining\_percent = (  
 (subproject.remaining\_budget / subproject\_budget \* 100) if subproject\_budget > 0 else Decimal('0')  
 )  
 subproject.total\_percent = (  
 (subproject\_budget / total\_budget \* 100) if total\_budget > 0 else Decimal('0')  
 )  
  
 project\_data.append({  
 'project': project,  
 'total\_budget': total\_budget,  
 'remaining\_budget': remaining\_budget,  
 'remaining\_percentage': remaining\_percentage  
 })  
  
 context.update({  
 'budget\_period': budget\_allocation.budget\_period if budget\_allocation else None,  
 'total\_org\_budget': total\_org\_budget,  
 'remaining\_amount': remaining\_amount,  
 'remaining\_percent': remaining\_percent,  
 'allocation\_total': allocation\_total,  
 'allocation\_remaining': allocation\_remaining,  
 'allocation\_remaining\_percent': allocation\_remaining\_percent,  
 'warning\_threshold': budget\_allocation.warning\_threshold if budget\_allocation else 50,  
 'projects': projects,  
 'project\_data': project\_data,  
 })  
 logger.info(f"Context prepared for organization {organization.id}")  
 return context  
  
 def get\_form\_kwargs(self):  
 kwargs = super().get\_form\_kwargs()  
 kwargs['organization\_id'] = self.kwargs['organization\_id']  
 kwargs['user'] = self.request.user  
 return kwargs  
  
 def form\_invalid(self, form):  
 logger.error(f"Form errors: {form.errors.as\_json()}")  
 messages.error(self.request, \_('لطفاً خطاهای فرم را بررسی کنید.'))  
 return self.render\_to\_response(self.get\_context\_data(form=form))  
  
 @transaction.atomic  
 def form\_valid(self, form):  
 form.instance.created\_by = self.request.user  
 remaining = form.instance.budget\_allocation.get\_remaining\_amount()  
 if form.instance.allocated\_amount > remaining:  
 messages.error(self.request, f'مبلغ تخصیص بیشتر از باقی‌مانده بودجه شعبه ({remaining:,} ریال) است.')  
 return self.form\_invalid(form)  
 try:  
 response = super().form\_valid(form)  
 logger.info(f"New allocation saved: {self.object.pk} - {self.object.allocated\_amount}")  
 messages.success(self.request, \_("تخصیص بودجه با موفقیت ثبت شد"))  
 return response  
 except ValidationError as e:  
 logger.error(f"Validation error on create: {str(e)}")  
 form.add\_error('allocated\_amount', str(e))  
 return self.form\_invalid(form)  
  
 def get\_success\_url(self):  
 return reverse\_lazy('project\_budget\_allocation\_list', kwargs={'organization\_id': self.kwargs['organization\_id']})  
  
 def dispatch(self, request, \*args, \*\*kwargs):  
 try:  
 Organization.objects.get(id=self.kwargs['organization\_id'])  
 except Organization.DoesNotExist:  
 logger.error(f"Organization with id={self.kwargs['organization\_id']} not found")  
 messages.error(request, \_("سازمان موردنظر یافت نشد"))  
 return redirect('budgetperiod\_list')  
 return super().dispatch(request, \*args, \*\*kwargs)  
  
"""BudgetAllocationUpdateView داخل فایل دیگه ای """  
class ProjectBudgetAllocationDeleteView(PermissionBaseView, DeleteView):  
 model = ProjectBudgetAllocation  
 template\_name = 'budgets/budget/project\_budget\_allocation\_delete.html'  
 context\_object\_name = 'allocation'  
  
 @transaction.atomic  
 def delete(self, request, \*args, \*\*kwargs):  
 self.object = self.get\_object()  
 budget\_allocation = self.object.budget\_allocation  
 logger.info(f"Allocation {self.object.pk} deleted")  
 success\_url = self.get\_success\_url()  
 self.object.delete()  
 *# به‌روزرسانی remaining\_amount* budget\_allocation.remaining\_amount = budget\_allocation.get\_remaining\_amount()  
 budget\_allocation.save(update\_fields=['remaining\_amount'])  
 messages.success(self.request, \_("تخصیص بودجه با موفقیت حذف شد"))  
 from django.shortcuts import redirect  
 return redirect(success\_url)  
  
 def get\_success\_url(self):  
 return reverse\_lazy('project\_budget\_allocation\_list',  
 kwargs={'organization\_id': self.object.budget\_allocation.organization\_id})  
  
*# Reports*class BudgetRealtimeReportView(PermissionBaseView, TemplateView):  
 template\_name = 'reports/realtime\_report.html'  
 permission\_required = 'budgets.view\_budgetallocation'  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 organization\_id = self.request.GET.get('organization\_id')  
 project\_id = self.request.GET.get('project\_id')  
  
 organizations = Organization.objects.filter(is\_active=True)  
 data = []  
  
 for org in organizations:  
 if organization\_id and org.id != int(organization\_id):  
 continue  
 org\_budget = get\_organization\_budget(org) or Decimal('0')  
 org\_remaining = sum(  
 alloc.get\_remaining\_amount() for alloc in BudgetAllocation.objects.filter(  
 organization=org, is\_active=True  
 )  
 ) or Decimal('0')  
 projects = Project.objects.filter(organizations=org, is\_active=True)  
 project\_data = []  
  
 for project in projects:  
 if project\_id and project.id != int(project\_id):  
 continue  
 total\_budget = project.get\_total\_budget() or Decimal('0')  
 remaining\_budget = project.get\_remaining\_budget() or Decimal('0')  
 transactions = BudgetTransaction.objects.filter(  
 allocation\_\_project\_allocations\_\_project=project  
 ).order\_by('-timestamp')[:10] *# محدود به 10 تراکنش اخیر* project\_data.append({  
 'project': project,  
 'total\_budget': total\_budget,  
 'remaining\_budget': remaining\_budget,  
 'allocated\_percentage': (  
 (total\_budget / org\_budget \* 100) if org\_budget > 0 else Decimal('0')  
 ),  
 'remaining\_percentage': (  
 (remaining\_budget / total\_budget \* 100) if total\_budget > 0 else Decimal('0')  
 ),  
 'transactions': transactions,  
 })  
  
 data.append({  
 'organization': org,  
 'total\_budget': org\_budget,  
 'remaining\_budget': org\_remaining,  
 'projects': project\_data,  
 })  
  
 context.update({  
 'report\_data': data,  
 'organizations': organizations,  
 'selected\_organization': organization\_id,  
 'selected\_project': project\_id,  
 })  
 logger.debug(f"BudgetRealtimeReportView context: {context}")  
 return context

محتوای سیگنال   
from decimal import Decimal  
  
from django.db.models import Sum  
from django.db.models.signals import post\_save, post\_delete  
from django.dispatch import receiver  
  
from budgets.models import BudgetAllocation  
  
  
@receiver([post\_save, post\_delete], sender=BudgetAllocation)  
def update\_total\_allocated(sender, instance, \*\*kwargs):  
  
 budget\_period = instance.budget\_period  
 total\_allocated = BudgetAllocation.objects.filter(  
 budget\_period=budget\_period  
 ).aggregate(total=Sum('allocated\_amount'))['total'] or Decimal('0')  
 budget\_period.total\_allocated = total\_allocated  
 budget\_period.save(update\_fields=['total\_allocated'])

*# budgets/urls.py*from django.urls import path  
from . import views  
from budgets.views import BudgetDashboardView , \  
 OrganizationBudgetAllocationListView, \  
 PaymentOrderListView, PaymentOrderCreateView, \  
 PaymentOrderUpdateView, PaymentOrderDeleteView, PayeeListView, PayeeCreateView, PayeeUpdateView, PayeeDeleteView, \  
 TransactionTypeListView, TransactionTypeCreateView, TransactionTypeUpdateView, TransactionTypeDeleteView, \  
 NumberToWordsView, budget\_Help  
from .ApproveReject.view\_ApproveReject import ApproveRejectView  
from .BudgetAllocation.views\_BudgetAllocation import BudgetAllocationDetailView, BudgetAllocationDeleteView, BudgetAllocationUpdateView  
from budgets.BudgetPeriod.views\_BudgetPeriod import BudgetPeriodListView, BudgetPeriodDetailView,BudgetPeriodCreateView,BudgetPeriodUpdateView,\  
 BudgetPeriodDeleteView  
from .BudgetAllocation.get\_projects\_by\_organization import get\_projects\_by\_organization  
from .BudgetAllocation.views\_BudgetAllocation import BudgetAllocationCreateView, BudgetAllocationListView  
from .BudgetReturn.view\_BudgetReturn import BudgetReturnView  
from .BudgetTransaction.view\_budgetTransaction import BudgetTransactionListView\_2D, BudgetTransactionDetailView, \  
 BudgetTransactionListView  
from .ProjectBudgetAllocation.view\_Project\_budget\_list import ProjectBudgetAllocationListView2D  
from .ProjectBudgetAllocation.view\_Update\_Project\_Budget\_Allocation import Project\_\_Budget\_\_Allocation\_\_Edit\_\_View  
*# from .BudgetAllocation.views\_BudgetAllocation import BudgetAllocationCreateView*from .ProjectBudgetAllocation.views\_ProjectBudgetAllocation import ProjectBudgetAllocationListView, \  
 ProjectBudgetAllocationCreateView, \  
 ProjectBudgetAllocationDetailView, ProjectBudgetAllocationDeleteView, \  
 BudgetRealtimeReportView  
  
urlpatterns = [  
 path('budgets-dashboard',BudgetDashboardView.as\_view(), name='budgets\_dashboard'), *# داشبورد  
  
 # BudgetPeriod* path('budgetperiod/',BudgetPeriodListView.as\_view(), name='budgetperiod\_list'),  
 path('budgetperiod/<int:pk>/',BudgetPeriodDetailView.as\_view(), name='budgetperiod\_detail'),  
 path('budgetperiod/add/',BudgetPeriodCreateView.as\_view(), name='budgetperiod\_create'),  
 path('budgetperiod/<int:pk>/edit/',BudgetPeriodUpdateView.as\_view(), name='budgetperiod\_update'),  
 path('budgetperiod/<int:pk>/delete/',BudgetPeriodDeleteView.as\_view(), name='budgetperiod\_delete'),  
  
 *# BudgetAllocation* path('budgetallocation/',BudgetAllocationListView.as\_view(), name='budgetallocation\_list'),  
 path('budgetallocation/<int:pk>/',BudgetAllocationDetailView.as\_view(), name='budgetallocation\_detail'),  
 *# path('budgetallocation/add/',BudgetAllocationCreateView.as\_view(), name='budgetallocation\_add'),* path('budgetallocation/<int:pk>/edit/',BudgetAllocationUpdateView.as\_view(), name='budgetallocation\_update'),  
 path('budgetallocation/<int:pk>/delete/',BudgetAllocationDeleteView.as\_view(),  
 name='budgetallocation\_delete'),  
 path('budgetallocations/create/',BudgetAllocationCreateView.as\_view(), name='budgetallocation\_create'),  
  
 *# path('budget/allocation/<int:pk>/detail/', ProjectBudgetAllocationDetailView.as\_view(),  
 # name='project\_budget\_allocation\_detail'),  
  
  
 # Organization Budget* path('organization/<int:org\_id>/allocations/', OrganizationBudgetAllocationListView.as\_view(), name='organization\_budgetallocation\_list'),  
  
 *# ProjectBudgetAllocation* path('organization/<int:organization\_id>/project-budget-allocations/',ProjectBudgetAllocationListView.as\_view(), name='project\_budget\_allocation\_list'),  
  
  
 path('organization/<int:organization\_id>/project-budget-allocation/',ProjectBudgetAllocationCreateView.as\_view(), name='project\_budget\_allocation'),  
 path('project-budget-allocation/<int:pk>/detail/',ProjectBudgetAllocationDetailView.as\_view(), name='project\_budget\_allocation\_detail'),  
  
 *# path('project\_\_budget\_\_allocation\_\_edit\_\_view/<int:pk>/edit/',Project\_\_Budget\_\_Allocation\_\_Edit\_\_View.as\_view(), name='project\_\_budget\_\_allocation\_\_edit\_\_view'),* path('budget/allocation/<int:pk>/edit/', Project\_\_Budget\_\_Allocation\_\_Edit\_\_View.as\_view(), name='project\_\_budget\_\_allocation\_\_edit\_\_view'),  
  
  
 path('project-budget-allocation/<int:pk>/delete/',ProjectBudgetAllocationDeleteView.as\_view(), name='project\_budget\_allocation\_delete'),  
  
 *# BudgetTransaction* path('budgettransaction/',BudgetTransactionListView.as\_view(), name='budgettransaction\_list'),  
 path('budget/allocation/<int:allocation\_id>/transactions/', BudgetTransactionListView\_2D.as\_view(),  
 name='budget\_transaction\_list\_2d'),  
 path('budgettransaction/<int:pk>/',BudgetTransactionDetailView.as\_view(), name='budgettransaction\_detail'),  
  
 *# PaymentOrder* path('paymentorder/',PaymentOrderListView.as\_view(), name='paymentorder\_list'),  
 path('paymentorder/add/',PaymentOrderCreateView.as\_view(), name='paymentorder\_add'),  
 path('paymentorder/<int:pk>/edit/',PaymentOrderUpdateView.as\_view(), name='paymentorder\_edit'),  
 path('paymentorder/<int:pk>/delete/',PaymentOrderDeleteView.as\_view(), name='paymentorder\_delete'),  
 *# Payee* path('payee/',PayeeListView.as\_view(), name='payee\_list'),  
 path('payee/add/',PayeeCreateView.as\_view(), name='payee\_add'),  
 path('payee/<int:pk>/edit/',PayeeUpdateView.as\_view(), name='payee\_edit'),  
 path('payee/<int:pk>/delete/',PayeeDeleteView.as\_view(), name='payee\_delete'),  
  
 *# TransactionType* path('transactiontype/',TransactionTypeListView.as\_view(), name='transactiontype\_list'),  
 path('transactiontype/add/',TransactionTypeCreateView.as\_view(), name='transactiontype\_add'),  
 path('transactiontype/<int:pk>/edit/',TransactionTypeUpdateView.as\_view(), name='transactiontype\_edit'),  
 path('transactiontype/<int:pk>/delete/',TransactionTypeDeleteView.as\_view(), name='transactiontype\_delete'),  
  
  
 *# path('budgetallocations/<int:pk>/update/', BudgetAllocationUpdateView.as\_view(), name='budgetallocation\_update'),  
  
 # path('get\_projects\_by\_organization/',get\_projects\_by\_organization, name='get\_projects\_by\_organization'),# گرفتن پروژه‌های سازمان* path('api/projects-by-organization/', get\_projects\_by\_organization, name='get\_projects\_by\_organization'),  
  
 path('convert\_number\_to\_words/', NumberToWordsView.as\_view(), name='convert\_number\_to\_words'),  
  
]  
  
urlpatterns += [  
 path('budget\_Help/',budget\_Help,name='budget\_Help')  
]  
from .Budget\_Items.views\_Budget\_item import BudgetItemListView, BudgetItemCreateView, BudgetItemUpdateView, BudgetItemDeleteView,BudgetItemDetailView  
urlpatterns += [  
 path('budgetitems/', BudgetItemListView.as\_view(), name='budgetitem\_list'),  
 path('budgetitems/create/', BudgetItemCreateView.as\_view(), name='budgetitem\_create'),  
 path('budgetitems/<int:pk>/update/', BudgetItemUpdateView.as\_view(), name='budgetitem\_update'),  
 path('budgetitems/<int:pk>/delete/', BudgetItemDeleteView.as\_view(), name='budgetitem\_delete'),  
 path('budgetitems/<int:pk>/', BudgetItemDetailView.as\_view(), name='budgetitem\_detail'),  
]  
from core.views import BudgetAllocationView  
urlpatterns += [  
 path('budget\_allocation\_view/', BudgetAllocationView.as\_view(),name='budget\_allocation\_view'),  
]  
  
urlpatterns += [  
 path('budgetrealtimeReportView/', BudgetRealtimeReportView.as\_view(),name='budgetrealtimeReportView'),  
] *# Reports*urlpatterns += [  
 path('approve-reject/<str:entity\_type>/<int:entity\_id>/<str:action>/', ApproveRejectView.as\_view(),  
 name='approve\_reject'),  
] *# رد بودجه*urlpatterns += [  
 path('budget/allocation/<int:allocation\_id>/return/', BudgetReturnView.as\_view(), name='budget\_return'),  
 ] *# برگشت بودجه  
  
# #/  
# urlpatterns+ = [  
# path('budgetperiods/', BudgetPeriodListView.as\_view(), name='budgetperiod\_list'), # مسیر لیست دوره‌های بودجه (برای دکمه انصراف)  
# path('budgetperiods/<int:pk>/', BudgetPeriodDetailView.as\_view(), name='budgetperiod\_detail'),  
# path('budgetperiods/create/', BudgetPeriodCreateView.as\_view(), name='budgetperiod\_create'),  
# path('budgetperiods/<int:pk>/update/', BudgetPeriodUpdateView.as\_view(), name='budgetperiod\_update'),  
# path('budgetperiods/<int:pk>/delete/', BudgetPeriodDeleteView.as\_view(), name='budgetperiod\_delete'),  
# ]#دوره بودجه کلان  
# urlpatterns += [  
# path('budgetallocations/', BudgetAllocationListView.as\_view(), name='budgetallocation\_list'),  
# path('budgetallocations/<int:pk>/', BudgetAllocationDetailView.as\_view(), name='budgetallocation\_detail'),  
# path('budgetallocations/create/', BudgetAllocationCreateView.as\_view(), name='budgetallocation\_create'),  
# path('budgetallocations/<int:pk>/update/', BudgetAllocationUpdateView.as\_view(), name='budgetallocation\_update'),  
# path('budgetallocations/<int:pk>/delete/', BudgetAllocationDeleteView.as\_view(), name='budgetallocation\_delete'),  
#  
# path('organization/<int:org\_id>/allocations/', OrganizationBudgetAllocationListView.as\_view(), name='organization\_budgetallocation\_list'),  
#  
# ]#تخصیص بودجه به هر سطح از Organization  
# urlpatterns += [  
# path('budgettransactions/', BudgetTransactionListView.as\_view(), name='budgettransaction\_list'),  
# path('budgettransactions/<int:pk>/', BudgetTransactionDetailView.as\_view(), name='budgettransaction\_detail'),  
# ]# هر تغییر در بودجه  
# urlpatterns += [  
# path('paymentorders/', PaymentOrderListView.as\_view(), name='paymentorder\_list'),  
# path('paymentorders/<int:pk>/', PaymentOrderDetailView.as\_view(), name='paymentorder\_detail'),  
# path('paymentorders/create/', PaymentOrderCreateView.as\_view(), name='paymentorder\_create'),  
# path('paymentorders/<int:pk>/update/', PaymentOrderUpdateView.as\_view(), name='paymentorder\_update'),  
# path('paymentorders/<int:pk>/delete/', PaymentOrderDeleteView.as\_view(), name='paymentorder\_delete'),  
# ]# دستور پرداخت  
# urlpatterns += [  
# path('payees/', PayeeListView.as\_view(), name='payee\_list'),  
# path('payees/<int:pk>/', PayeeDetailView.as\_view(), name='payee\_detail'),  
# path('payees/create/', PayeeCreateView.as\_view(), name='payee\_create'),  
# path('payees/<int:pk>/update/', PayeeUpdateView.as\_view(), name='payee\_update'),  
# path('payees/<int:pk>/delete/', PayeeDeleteView.as\_view(), name='payee\_delete'),  
# ]# اطلاعات دریافت‌کننده  
# urlpatterns += [  
# path('transactiontypes/', TransactionTypeListView.as\_view(), name='transactiontype\_list'),  
# path('transactiontypes/<int:pk>/', TransactionTypeDetailView.as\_view(), name='transactiontype\_detail'),  
# path('transactiontypes/create/', TransactionTypeCreateView.as\_view(), name='transactiontype\_create'),  
# path('transactiontypes/<int:pk>/update/', TransactionTypeUpdateView.as\_view(), name='transactiontype\_update'),  
# path('transactiontypes/<int:pk>/delete/', TransactionTypeDeleteView.as\_view(), name='transactiontype\_delete'),  
# ] # تعریف پویا نوع تراکنش‌ها  
  
  
#  
# urlpatterns += [  
# #صفحه، پروژه‌ها و زیرپروژه‌ها رو با بودجه تخصیص‌یافته، باقی‌مانده، و درصد نشون می‌ده.  
# # مسیر تخصیص بودجه به پروژه‌ها و زیرپروژه‌ها  
#  
# path('organization/<int:organization\_id>/project-budget-allocations/',  
# ProjectBudgetAllocationListView.as\_view(),  
# name='project\_budget\_allocation\_list'),  
#  
# path('project-budget-allocation/<int:pk>/detail/',  
# ProjectBudgetAllocationDetailView.as\_view(),  
# name='project\_budget\_allocation\_detail'),  
#  
# path('project-budget-allocation/<int:pk>/edit/',  
# ProjectBudgetAllocationEditView.as\_view(),  
# name='project\_budget\_allocation\_edit'),  
#  
# path('project-budget-allocation/<int:pk>/delete/',  
# ProjectBudgetAllocationDeleteView.as\_view(),  
# name='project\_budget\_allocation\_delete'),  
#  
# path('organization/<int:organization\_id>/project-budget-allocation/',  
# ProjectBudgetAllocationCreateView.as\_view(),  
# name='project\_budget\_allocation'),  
# ]*

*# budgets/views.py*import json  
import logging  
from decimal import Decimal  
  
from django.contrib import messages  
from django.core.exceptions import ValidationError  
from django.db import transaction  
from django.db.models import Q, Sum  
from django.shortcuts import get\_object\_or\_404, redirect, render  
from django.urls import reverse\_lazy  
from django.utils.translation import gettext\_lazy as \_  
from django.views.generic import ListView, DetailView, UpdateView, DeleteView, CreateView, TemplateView  
  
*# from budgets.ProjectBudgetAllocation.forms\_ProjectBudgetAllocation import ProjectBudgetAllocationForm*from budgets.forms import PaymentOrderForm, \  
 PayeeForm, TransactionTypeForm  
from budgets.models import BudgetPeriod, BudgetAllocation, ProjectBudgetAllocation, BudgetTransaction, PaymentOrder, \  
 Payee, TransactionType  
from core.PermissionBase import PermissionBaseView  
from core.models import Organization, Project, SubProject, OrganizationType  
from budgets.budget\_calculations import (  
 get\_budget\_details, calculate\_allocation\_percentages, calculate\_total\_allocated,  
 calculate\_remaining\_budget, get\_budget\_status, get\_organization\_budget,  
 get\_project\_remaining\_budget, get\_subproject\_remaining\_budget, get\_subproject\_total\_budget, get\_project\_total\_budget  
)  
from core.templatetags.rcms\_custom\_filters import number\_to\_farsi\_words  
  
  
from django.http import JsonResponse  
from django.views.decorators.csrf import csrf\_protect  
from django.utils.decorators import method\_decorator  
from django.views import View  
  
logger = logging.getLogger(\_\_name\_\_)  
*# Dashboard*class BudgetDashboardView(PermissionBaseView, TemplateView):  
 template\_name = 'budgets/budgets\_dashboard.html'  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['organizations'] = Organization.objects.all()  
 return context  
  
*# استفاده در تمپلیت‌ها  
# جزئیات تخصیص: {% url 'project\_budget\_allocation\_detail' allocation.pk %}  
# ویرایش تخصیص: {% url 'project\_budget\_allocation\_edit' allocation.pk %}  
# لیست تراکنش‌ها: {% url 'budget\_transaction\_list' allocation\_id=allocation.id %}  
# گزارش بودجه پروژه: {% url 'project\_budget\_report' project.pk %}  
# گزارش هشدارها: {% url 'budget\_warning\_report' %}  
  
  
# --- Organization Budget ---*class OrganizationBudgetAllocationListView(PermissionBaseView, ListView):  
 model = BudgetAllocation  
 template\_name = 'budgets/budget/organization\_budgetallocation\_list.html'  
 context\_object\_name = 'budget\_allocations'  
 paginate\_by = 10  
  
 def get\_queryset(self):  
 org\_id = self.kwargs.get('org\_id')  
 return BudgetAllocation.objects.filter(organization\_id=org\_id).order\_by('-allocation\_date')  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 org\_id = self.kwargs.get('org\_id')  
 organization = get\_object\_or\_404(Organization, id=org\_id)  
 context['organization'] = organization  
 context['budget\_details'] = get\_budget\_details(organization)  
 logger.debug(f"OrganizationBudgetAllocationListView context: {context}")  
 return context  
  
*# --- PaymentOrder CRUD ---*class PaymentOrderListView(PermissionBaseView, ListView):  
 model = PaymentOrder  
 template\_name = 'budgets/paymentorder/paymentorder\_list.html'  
 context\_object\_name = 'payment\_orders'  
 paginate\_by = 10  
  
 def get\_queryset(self):  
 queryset = super().get\_queryset()  
 query = self.request.GET.get('q', '')  
 if query:  
 queryset = queryset.filter(  
 Q(order\_number\_\_icontains=query) |  
 Q(payee\_\_name\_\_icontains=query) |  
 Q(tankhah\_\_project\_\_name\_\_icontains=query)  
 )  
 status = self.request.GET.get('status')  
 if status:  
 queryset = queryset.filter(status=status)  
 return queryset.order\_by('-issue\_date')  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['query'] = self.request.GET.get('q', '')  
 context['status'] = self.request.GET.get('status', '')  
 context['status\_choices'] = PaymentOrder.STATUS\_CHOICES  
 logger.debug(f"PaymentOrderListView context: {context}")  
 return context  
class PaymentOrderCreateView(PermissionBaseView, CreateView):  
 model = PaymentOrder  
 form\_class = PaymentOrderForm  
 template\_name = 'budgets/paymentorder/paymentorder\_form.html'  
 success\_url = reverse\_lazy('paymentorder\_list')  
  
 def form\_valid(self, form):  
 form.instance.created\_by = self.request.user  
 with transaction.atomic():  
 response = super().form\_valid(form)  
 messages.success(self.request, f'دستور پرداخت {form.instance.order\_number} با موفقیت ایجاد شد.')  
 return response  
class PaymentOrderUpdateView(PermissionBaseView, UpdateView):  
 model = PaymentOrder  
 form\_class = PaymentOrderForm  
 template\_name = 'budgets/paymentorder/paymentorder\_form.html'  
 success\_url = reverse\_lazy('paymentorder\_list')  
  
 def form\_valid(self, form):  
 with transaction.atomic():  
 response = super().form\_valid(form)  
 messages.success(self.request, f'دستور پرداخت {form.instance.order\_number} با موفقیت به‌روزرسانی شد.')  
 return response  
class PaymentOrderDeleteView(PermissionBaseView, DeleteView):  
 model = PaymentOrder  
 template\_name = 'budgets/paymentorder/paymentorder\_confirm\_delete.html'  
 success\_url = reverse\_lazy('paymentorder\_list')  
  
 def post(self, request, \*args, \*\*kwargs):  
 payment\_order = self.get\_object()  
 with transaction.atomic():  
 payment\_order.delete()  
 messages.success(request, f'دستور پرداخت {payment\_order.order\_number} با موفقیت حذف شد.')  
 return redirect(self.success\_url)  
class PaymentOrderDetailView(PermissionBaseView, DetailView):  
 model = PaymentOrder  
 template\_name = 'budgets/paymentorder/paymentorder\_detail.html'  
 permission\_codenames = ['budgets.PaymentOrder\_view']  
 context\_object\_name = 'payment\_order'  
  
*# --- Payee CRUD ---*class PayeeListView(PermissionBaseView, ListView):  
 model = Payee  
 template\_name = 'budgets/payee/payee\_list.html'  
 context\_object\_name = 'payees'  
 paginate\_by = 10  
  
 def get\_queryset(self):  
 queryset = super().get\_queryset()  
 query = self.request.GET.get('q', '')  
 if query:  
 queryset = queryset.filter(  
 Q(name\_\_icontains=query) |  
 Q(national\_id\_\_icontains=query) |  
 Q(iban\_\_icontains=query)  
 )  
 payee\_type = self.request.GET.get('payee\_type')  
 if payee\_type:  
 queryset = queryset.filter(payee\_type=payee\_type)  
 return queryset.order\_by('name')  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['query'] = self.request.GET.get('q', '')  
 context['payee\_type'] = self.request.GET.get('payee\_type', '')  
 context['payee\_types'] = Payee.PAYEE\_TYPES  
 logger.debug(f"PayeeListView context: {context}")  
 return context  
class PayeeDetailView(PermissionBaseView, DetailView):  
 model = Payee  
 template\_name = 'budgets/payee/payee\_detail.html'  
 context\_object\_name = 'payee'  
class PayeeCreateView(PermissionBaseView, CreateView):  
 model = Payee  
 form\_class = PayeeForm  
 template\_name = 'budgets/payee/payee\_form.html'  
 success\_url = reverse\_lazy('payee\_list')  
  
 def form\_valid(self, form):  
 form.instance.created\_by = self.request.user  
 with transaction.atomic():  
 response = super().form\_valid(form)  
 messages.success(self.request, f'دریافت‌کننده {form.instance.name} با موفقیت ایجاد شد.')  
 return response  
class PayeeUpdateView(PermissionBaseView, UpdateView):  
 model = Payee  
 form\_class = PayeeForm  
 template\_name = 'budgets/payee/payee\_form.html'  
 success\_url = reverse\_lazy('payee\_list')  
  
 def form\_valid(self, form):  
 with transaction.atomic():  
 response = super().form\_valid(form)  
 messages.success(self.request, f'دریافت‌کننده {form.instance.name} با موفقیت به‌روزرسانی شد.')  
 return response  
class PayeeDeleteView(PermissionBaseView, DeleteView):  
 model = Payee  
 template\_name = 'budgets/payee/payee\_confirm\_delete.html'  
 success\_url = reverse\_lazy('payee\_list')  
  
 def post(self, request, \*args, \*\*kwargs):  
 payee = self.get\_object()  
 with transaction.atomic():  
 payee.delete()  
 messages.success(request, f'دریافت‌کننده {payee.name} با موفقیت حذف شد.')  
 return redirect(self.success\_url)  
  
*# --- TransactionType CRUD ---*class TransactionTypeListView(PermissionBaseView, ListView):  
 model = TransactionType  
 template\_name = 'budgets/transactiontype/transactiontype\_list.html'  
 context\_object\_name = 'transaction\_types'  
 paginate\_by = 10  
  
 def get\_queryset(self):  
 queryset = super().get\_queryset()  
 query = self.request.GET.get('q', '')  
 if query:  
 queryset = queryset.filter(Q(name\_\_icontains=query) | Q(description\_\_icontains=query))  
 approval = self.request.GET.get('requires\_extra\_approval')  
 if approval == 'yes':  
 queryset = queryset.filter(requires\_extra\_approval=True)  
 elif approval == 'no':  
 queryset = queryset.filter(requires\_extra\_approval=False)  
 return queryset.order\_by('name')  
  
 def get\_context\_data(self, \*\*kwargs):  
 context = super().get\_context\_data(\*\*kwargs)  
 context['query'] = self.request.GET.get('q', '')  
 context['requires\_extra\_approval'] = self.request.GET.get('requires\_extra\_approval', '')  
 logger.debug(f"TransactionTypeListView context: {context}")  
 return context  
class TransactionTypeDetailView(PermissionBaseView, DetailView):  
 model = TransactionType  
 template\_name = 'budgets/transactiontype/transactiontype\_detail.html'  
 context\_object\_name = 'transaction\_type'  
class TransactionTypeCreateView(PermissionBaseView, CreateView):  
 model = TransactionType  
 form\_class = TransactionTypeForm  
 template\_name = 'budgets/transactiontype/transactiontype\_form.html'  
 success\_url = reverse\_lazy('transactiontype\_list')  
  
 def form\_valid(self, form):  
 form.instance.created\_by = self.request.user  
 with transaction.atomic():  
 response = super().form\_valid(form)  
 messages.success(self.request, f'نوع تراکنش {form.instance.name} با موفقیت ایجاد شد.')  
 return response  
class TransactionTypeUpdateView(PermissionBaseView, UpdateView):  
 model = TransactionType  
 form\_class = TransactionTypeForm  
 template\_name = 'budgets/transactiontype/transactiontype\_form.html'  
 success\_url = reverse\_lazy('transactiontype\_list')  
  
 def form\_valid(self, form):  
 with transaction.atomic():  
 response = super().form\_valid(form)  
 messages.success(self.request, f'نوع تراکنش {form.instance.name} با موفقیت به‌روزرسانی شد.')  
 return response  
class TransactionTypeDeleteView(PermissionBaseView, DeleteView):  
 model = TransactionType  
 template\_name = 'budgets/transactiontype/transactiontype\_confirm\_delete.html'  
 success\_url = reverse\_lazy('transactiontype\_list')  
  
 def post(self, request, \*args, \*\*kwargs):  
 transaction\_type = self.get\_object()  
 with transaction.atomic():  
 transaction\_type.delete()  
 messages.success(request, f'نوع تراکنش {transaction\_type.name} با موفقیت حذف شد.')  
 return redirect(self.success\_url)  
  
@method\_decorator(csrf\_protect, name='dispatch')  
class NumberToWordsView(View):  
 *"""پیاده‌سازی API برای تبدیل پویا"""* def post(self, request):  
 try:  
 data = json.loads(request.body)  
 number = data.get('number')  
 words = number\_to\_farsi\_words(number)  
 logger.debug(f"Converted number {number} to words: {words}")  
 return JsonResponse({'words': words})  
 except Exception as e:  
 logger.error(f"Error converting number to words: {str(e)}")  
 return JsonResponse({'error': 'Invalid input'}, status=400)  
  
  
def budget\_Help(request):  
 return render(request, template\_name='help/budget\_Period\_help.html')