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!