بله، مدل‌های بودجه‌ای که ارائه دادید (و همچنین مدل‌هایی که به طور ضمنی در کد مانند BudgetItem به آن‌ها اشاره شد) ساختار نسبتاً کاملی برای مدیریت بودجه چند سطحی را فراهم می‌کنند. بیایید آن‌ها را مرور کنیم و روند کلی جریان بودجه تا رسیدن به فاکتور را توضیح دهیم:

توضیح مدل‌های بودجه:

BudgetPeriod (دوره بودجه کلان):

نقش: این مدل بالاترین سطح تعریف بودجه است. یک دوره زمانی مشخص (مثلاً سال مالی ۱۴۰۴) و یک مبلغ کل (total\_amount) را برای یک سازمان (organization - معمولاً دفتر مرکزی یا واحد اصلی) تعریف می‌کند. این منبع اصلی پول است.

فیلدهای کلیدی:

organization: سازمانی که این بودجه کلی به آن تعلق دارد.

name: نام دوره (مثلاً "بودجه سال ۱۴۰۴").

start\_date, end\_date: بازه زمانی اعتبار بودجه.

total\_amount: سقف کل بودجه در این دوره برای این سازمان.

total\_allocated: (محاسباتی یا ذخیره شده) مجموع مبالغی که از این دوره به ردیف‌ها/سازمان‌های دیگر تخصیص داده شده است.

returned\_amount: مجموع مبالغی که از تخصیص‌های این دوره برگشت داده شده است.

is\_active, is\_archived, is\_completed: وضعیت‌های دوره.

هدف: تعیین می‌کند که یک سازمان خاص در یک بازه زمانی چقدر بودجه کلی در اختیار دارد.

BudgetItem (ردیف بودجه - مدل ارائه نشده اما ضروری):

نقش: این مدل (که احتمالاً در پروژه شما وجود دارد یا باید وجود داشته باشد) برای دسته‌بندی انواع هزینه‌ها استفاده می‌شود. این مدل مشخص می‌کند که بودجه برای چه منظوری قابل خرج است.

فیلدهای احتمالی:

name: نام ردیف (مثلاً "هزینه‌های سفر"، "خرید تجهیزات IT"، "حقوق و دستمزد").

code: یک کد منحصر به فرد برای ردیف.

budget\_period: (اختیاری، یا شاید در مدل دیگری) ارتباط با دوره بودجه‌ای که این ردیف در آن تعریف شده.

parent: (اختیاری) برای ایجاد ساختار درختی از ردیف‌های بودجه (مثلاً "IT" -> "نرم‌افزار"، "سخت‌افزار").

هدف: طبقه‌بندی بودجه و امکان تخصیص بودجه به موارد هزینه‌ای خاص.

BudgetAllocation (تخصیص بودجه):

نقش: این مدل بسیار کلیدی است. این مدل، بخشی از بودجه یک BudgetPeriod را به یک BudgetItem خاص برای یک Organization (که می‌تواند همان سازمان اصلی دوره یا یک زیرمجموعه باشد) اختصاص می‌دهد. این همان "پاکت پول" برای یک منظور خاص در یک شعبه/اداره است.

فیلدهای کلیدی:

budget\_period: دوره‌ای که پول از آن می‌آید.

organization: سازمانی که این تخصیص را دریافت می‌کند.

budget\_item: ردیف بودجه‌ای که این مبلغ برای آن در نظر گرفته شده است (مثلاً چقدر برای "خرید تجهیزات IT" در "دفتر مرکزی").

allocated\_amount: مبلغی که برای این ردیف خاص به این سازمان تخصیص داده شده است. (این همان فیلدی است که در متد get\_remaining\_amount شما استفاده می‌شود).

project: (اختیاری در این مدل، اما در ProjectBudgetAllocation کلیدی است) می‌تواند مستقیماً به یک پروژه لینک شود.

هدف: مشخص می‌کند که یک سازمان چقدر بودجه برای یک ردیف هزینه خاص در اختیار دارد. تمام تراکنش‌های مصرف و بازگشت بودجه به این مدل لینک می‌شوند.

ProjectBudgetAllocation (تخصیص بودجه پروژه):

نقش: این مدل به عنوان پل ارتباطی عمل می‌کند. یک BudgetAllocation (که مثلاً بودجه کلی IT دفتر مرکزی است) را به یک Project (مثلاً پروژه اجرای اتاق سرور) و حتی SubProject خاص لینک می‌کند.

فیلدهای کلیدی:

budget\_allocation: تخصیص بودجه‌ای که منبع مالی این پروژه است.

project: پروژه‌ای که بودجه به آن مرتبط است.

subproject: (اختیاری) زیرپروژه مرتبط.

allocated\_amount: (احتمالاً اینجا نباید باشد!) معمولاً مبلغ در BudgetAllocation اصلی تعریف می‌شود و این مدل فقط لینک‌دهنده است. اگر اینجا هم مبلغی دارید، منطق تخصیص پیچیده‌تر می‌شود (مثلاً از مبلغ BudgetAllocation چقدرش سهم این پروژه است). در کد ویو شما، به نظر می‌رسد مبلغ کل از ProjectBudgetAllocation خوانده می‌شود که نشان می‌دهد شما این فیلد را دارید و از آن به عنوان بودجه اختصاصی پروژه استفاده می‌کنید.

هدف: مشخص می‌کند کدام "پاکت پول" (BudgetAllocation) برای کدام پروژه استفاده می‌شود.

BudgetTransaction (تراکنش بودجه):

نقش: دفتر کل یا تاریخچه تمام تغییرات مالی روی یک BudgetAllocation. هر بار که پولی مصرف (مثلاً از طریق فاکتور یا تنخواه) یا بازگشت داده می‌شود، یک رکورد اینجا ثبت می‌شود.

فیلدهای کلیدی:

allocation: به BudgetAllocation ای اشاره می‌کند که تحت تأثیر قرار گرفته.

transaction\_type: نوع تراکنش ('CONSUMPTION', 'RETURN', 'ALLOCATION', 'ADJUSTMENT\_INCREASE', 'ADJUSTMENT\_DECREASE').

amount: مبلغ این تراکنش خاص.

related\_tankhah, related\_factor, related\_factor\_item: (اختیاری، اما بسیار مفید) لینک به شیء خاصی که باعث این تراکنش شده است. (همانطور که دیدیم، شما این‌ها را در مدل BudgetTransaction خود ندارید، فقط related\_tankhah را دارید).

هدف: ثبت دقیق تمام مصارف و بازگشت‌ها برای محاسبه دقیق باقیمانده BudgetAllocation.

روند جریان ثبت بودجه کلان و تخصیص (قبل از فاکتور):

تعریف دوره کلان: مدیر مالی یا بودجه یک BudgetPeriod جدید برای یک سازمان (مثلاً دفتر مرکزی) و یک بازه زمانی (مثلاً سال ۱۴۰۴) با یک مبلغ کل (total\_amount) تعریف می‌کند.

تعریف ردیف‌های بودجه: ردیف‌های هزینه‌ای مختلف (BudgetItem) در سیستم تعریف می‌شوند (مثلاً IT، سفر، آموزش...).

تخصیص بودجه به ردیف/سازمان: مدیر بودجه، بخشی از total\_amount دوره کلان را به یک BudgetItem خاص برای یک Organization خاص تخصیص می‌دهد و یک رکورد BudgetAllocation ایجاد می‌کند. مبلغ این تخصیص در allocated\_amount ثبت می‌شود.

مثال: از بودجه ۱۰۰ میلیاردی سال ۱۴۰۴ دفتر مرکزی، ۵ میلیارد به ردیف "تجهیزات IT" برای خود "دفتر مرکزی" تخصیص داده می‌شود (BudgetAllocation ایجاد می‌شود).

تخصیص بودجه به پروژه: وقتی پروژه‌ای (مثل اجرای اتاق سرور) تعریف می‌شود، باید مشخص شود که بودجه آن از کدام BudgetAllocation تأمین می‌شود. یک رکورد ProjectBudgetAllocation ایجاد می‌شود که BudgetAllocation مربوط به "تجهیزات IT دفتر مرکزی" را به "پروژه اتاق سرور" لینک می‌کند.

نکته: اگر ProjectBudgetAllocation شما فیلد allocated\_amount دارد، در این مرحله مشخص می‌شود که از ۵ میلیارد تخصیص IT، چقدر (مثلاً ۲ میلیارد) برای پروژه اتاق سرور است. اگر ندارد، یعنی کل باقیمانده آن BudgetAllocation برای این پروژه قابل استفاده است (تا زمانی که برای پروژه دیگری لینک نشود). کد ویوی شما نشان می‌دهد که شما از allocated\_amount در ProjectBudgetAllocation استفاده می‌کنید.

ایجاد تنخواه (احتمالاً): ممکن است قبل از ایجاد فاکتور، یک Tankhah برای مجری پروژه (مثلاً مدیر IT) با مبلغ مشخصی از بودجه پروژه ایجاد شود. این خود می‌تواند یک نوع BudgetTransaction از نوع CONSUMPTION (یا نوع خاص TANKHAH\_ALLOCATION) روی BudgetAllocation مربوطه ثبت کند و از باقیمانده آن کم کند. مبلغ Tankhah سقف هزینه‌ای است که مجری می‌تواند مستقیماً خرج کند.

ارتباط با فاکتور:

حالا وقتی کاربر می‌خواهد یک فاکتور (Factor) جدید ثبت کند:

انتخاب تنخواه: کاربر ابتدا Tankhah مرتبط با آن هزینه را انتخاب می‌کند.

اتصال به پروژه و تخصیص: از طریق Tankhah، سیستم به Project و سپس به ProjectBudgetAllocation و نهایتاً به BudgetAllocation اصلی می‌رسد.

بررسی بودجه تنخواه: ویو (FactorCreateView) با استفاده از get\_tankhah\_remaining\_budget(tankhah) چک می‌کند که آیا مبلغ کل فاکتور (total\_items\_amount) از بودجه باقیمانده تنخواه بیشتر است یا خیر. (این تابع، همانطور که بحث شد، احتمالاً مبلغ اولیه تنخواه منهای فاکتورهای در جریان یا پرداخت شده آن تنخواه را برمی‌گرداند).

بررسی بودجه تخصیص (در create\_budget\_transaction): وقتی کاربر روی "ذخیره نهایی پیش‌نویس" کلیک می‌کند و اعتبارسنجی‌های اولیه موفقیت‌آمیز است، تابع create\_budget\_transaction فراخوانی می‌شود. این تابع:

BudgetAllocation اصلی مرتبط با پروژه/تنخواه را پیدا می‌کند.

باقیمانده واقعی آن BudgetAllocation را با استفاده از متد get\_remaining\_amount() (که تمام تراکنش‌های قبلی را جمع می‌زند) محاسبه می‌کند.

چک می‌کند که مبلغ فاکتور (amount) از این باقیمانده واقعی بیشتر نباشد.

اگر بودجه کافی بود، یک رکورد BudgetTransaction جدید با transaction\_type='CONSUMPTION' و مبلغ فاکتور ایجاد می‌کند که به آن BudgetAllocation لینک شده است.

نتیجه:

بنابراین، ثبت یک فاکتور نهایی باعث کاهش بودجه در دو سطح می‌شود (یا باید بشود):

سطح تنخواه: باقیمانده تنخواه کم می‌شود (که در تابع get\_tankhah\_remaining\_budget با در نظر گرفتن وضعیت فاکتورها محاسبه می‌شود).

سطح تخصیص کلان: با ثبت BudgetTransaction مصرف، باقیمانده BudgetAllocation اصلی نیز کاهش می‌یابد (که در متد get\_remaining\_amount مدل BudgetAllocation محاسبه می‌شود).

امیدوارم این توضیحات روند کار مدل‌های بودجه شما و ارتباط آن با ثبت فاکتور را روشن کرده باشد. اگر سوال دیگری در مورد بخش خاصی دارید، بپرسید.

بسیار خوب! حالا که سناریوی پایه تأیید شده است، سناریو را گسترش می‌دهیم تا شامل جابجایی بزرگ بودجه بین پروژه‌ها، حتی در سطح دو سازمان یا دو شعبه مختلف، باشد. سپس، سازوکار این جابجایی را توضیح می‌دهم، تأثیر آن روی فیلدهای returned\_amount در مدل‌ها را بررسی می‌کنم، و در نهایت کد نهایی بهینه‌شده را برای پیاده‌سازی این سناریو ارائه می‌کنم.

گسترش سناریو: جابجایی بودجه بین پروژه‌ها (درون یا بین سازمان‌ها/شعب)

1. تعریف سناریو

در سناریوی جدید، فرض می‌کنیم:

بودجه می‌تواند از یک پروژه (یا زیرپروژه) در یک تخصیص (ProjectBudgetAllocation) به پروژه دیگری در همان سازمان یا سازمان/شعبه دیگر منتقل شود.

این جابجایی ممکن است شامل:

انتقال درون‌سازمانی: بودجه از یک پروژه به پروژه دیگر در همان سازمان یا شعبه منتقل شود.

انتقال بین‌سازمانی: بودجه از یک پروژه در یک سازمان/شعبه به پروژه‌ای در سازمان/شعبه دیگر منتقل شود.

جابجایی بودجه باید:

ردیابی شود تا مشخص باشد چه مقدار بودجه از کجا به کجا منتقل شده است.

تأثیر آن روی بودجه کل سازمان (در BudgetPeriod)، تخصیص‌ها (BudgetAllocation)، و پروژه‌ها (ProjectBudgetAllocation) به‌درستی ثبت شود.

با فیلدهای موجود (مثل returned\_amount) و تاریخچه (BudgetHistory) هماهنگ باشد.

هدف: اطمینان از اینکه این جابجایی به‌صورت شفاف، قابل ردیابی، و بدون ناسازگاری در داده‌ها انجام شود.

مثال سناریو:

وضعیت اولیه:

سازمان مرکزی یک BudgetPeriod با total\_amount=1,000,000,000 ریال دارد.

دو شعبه (سازمان A و B) بودجه دریافت کرده‌اند:

شعبه A: BudgetAllocation با allocated\_amount=500,000,000 ریال.

پروژه X: ProjectBudgetAllocation با allocated\_amount=300,000,000 ریال.

پروژه Y: ProjectBudgetAllocation با allocated\_amount=200,000,000 ریال.

شعبه B: BudgetAllocation با allocated\_amount=400,000,000 ریال.

پروژه Z: ProjectBudgetAllocation با allocated\_amount=400,000,000 ریال.

پروژه X مقدار 100,000,000 ریال مصرف کرده است (CONSUMPTION).

جابجایی بودجه:

مدیر تصمیم می‌گیرد 50,000,000 ریال از بودجه پروژه X (شعبه A) به پروژه Z (شعبه B) منتقل کند.

سوالات کلیدی:

چگونه این جابجایی ثبت شود؟

چگونه تأثیر آن روی returned\_amount در مدل‌ها منعکس شود؟

چگونه ردیابی کنیم که این بودجه به پروژه Z اضافه شده و از پروژه X کم شده است؟

آیا این جابجایی روی بودجه کل سازمان (BudgetPeriod) تأثیری دارد؟

2. سازوکار جابجایی بودجه

برای مدیریت جابجایی بودجه، دو رویکرد اصلی وجود دارد:

رویکرد بازگشت و تخصیص مجدد:

بودجه از پروژه مبدأ (مثل پروژه X) به‌صورت یک تراکنش RETURN به تخصیص یا دوره بودجه بازمی‌گردد.

سپس بودجه به پروژه مقصد (مثل پروژه Z) از طریق یک تراکنش ALLOCATION یا TRANSFER تخصیص داده می‌شود.

این روش با ساختار فعلی (استفاده از returned\_amount) سازگار است.

رویکرد انتقال مستقیم:

یک تراکنش مستقیم (TRANSFER) بین دو پروژه ثبت می‌شود بدون بازگشت به تخصیص یا دوره بودجه.

این روش نیاز به نوع تراکنش جدید و احتمالاً تغییرات در مدل‌ها دارد.

مقایسه رویکردها:

بازگشت و تخصیص مجدد:

مزایا:

با مدل‌های موجود (BudgetTransaction, BudgetHistory, returned\_amount) کاملاً سازگار است.

نیازی به تغییر ساختار مدل‌ها یا افزودن نوع تراکنش جدید ندارد.

ردیابی بازگشت و تخصیص مجدد در BudgetHistory به‌راحتی انجام می‌شود.

معایب:

نیاز به دو تراکنش جداگانه (بازگشت و تخصیص) که ممکن است پیچیدگی بیشتری در ردیابی ایجاد کند.

احتمال ناسازگاری اگر یکی از تراکنش‌ها انجام نشود.

انتقال مستقیم:

مزایا:

ساده‌تر برای کاربر (یک تراکنش به جای دو تراکنش).

ردیابی مستقیم‌تر با یک نوع تراکنش TRANSFER.

معایب:

نیاز به افزودن نوع تراکنش جدید (TRANSFER) به BudgetTransaction.

نیاز به تغییرات در منطق به‌روزرسانی returned\_amount و سایر فیلدها.

ممکن است با منطق فعلی returned\_amount (که فرض می‌کند بازگشت به تخصیص یا دوره است) ناسازگار باشد.

پیشنهاد: با توجه به ساختار فعلی مدل‌های شما (وجود returned\_amount در BudgetPeriod و BudgetAllocation) و نیاز به ردیابی دقیق، رویکرد بازگشت و تخصیص مجدد مناسب‌تر است، زیرا:

با سیستم موجود هماهنگ است.

امکان ردیابی کامل با BudgetHistory را فراهم می‌کند.

نیازی به تغییرات اساسی در مدل‌ها ندارد.

جریان جابجایی بودجه (رویکرد بازگشت و تخصیص مجدد):

مرحله بازگشت:

از پروژه X (در شعبه A) مقدار 50,000,000 ریال به‌صورت تراکنش RETURN به BudgetAllocation شعبه A برگشت داده می‌شود.

به‌روزرسانی‌ها:

ProjectBudgetAllocation (پروژه X): returned\_amount += 50,000,000, allocated\_amount -= 50,000,000.

BudgetAllocation (شعبه A): returned\_amount += 50,000,000, allocated\_amount -= 50,000,000.

BudgetPeriod: returned\_amount += 50,000,000, total\_allocated -= 50,000,000.

ثبت در BudgetHistory: تراکنش RETURN با جزئیات «برگشت از پروژه X».

مرحله تخصیص مجدد:

مقدار 50,000,000 ریال از BudgetPeriod به BudgetAllocation شعبه B تخصیص داده می‌شود.

سپس این مقدار به ProjectBudgetAllocation پروژه Z تخصیص می‌یابد.

به‌روزرسانی‌ها:

BudgetAllocation (شعبه B): allocated\_amount += 50,000,000.

ProjectBudgetAllocation (پروژه Z): allocated\_amount += 50,000,000.

BudgetPeriod: total\_allocated += 50,000,000.

ثبت در BudgetHistory: تراکنش ALLOCATION با جزئیات «تخصیص به پروژه Z».

ردیابی جابجایی:

برای ردیابی اینکه این 50,000,000 ریال از پروژه X به پروژه Z منتقل شده است:

از BudgetHistory استفاده کنید. دو رکورد خواهید داشت:

بازگشت: action='RETURN', content\_type=ProjectBudgetAllocation, object\_id=پروژه X, amount=50,000,000.

تخصیص: action='ALLOCATION', content\_type=ProjectBudgetAllocation, object\_id=پروژه Z, amount=50,000,000.

می‌توانید این دو تراکنش را با transaction\_id مرتبط کنید (مثلاً با یک شناسه مشترک برای جابجایی).

گزارش‌گیری:

کوئری روی BudgetHistory برای یافتن تمام جابجایی‌ها در یک دوره بودجه.

نمایش اینکه بودجه از کدام پروژه (یا تخصیص) به کدام پروژه منتقل شده است.

تأثیر بر بودجه کل سازمان:

در این سناریو، بودجه کل سازمان (BudgetPeriod.total\_amount) تغییری نمی‌کند، زیرا جابجایی فقط تخصیص‌ها را تغییر می‌دهد.

اما بودجه قابل تخصیص (BudgetPeriod.get\_remaining\_amount) تحت تأثیر قرار می‌گیرد:

هنگام بازگشت: returned\_amount افزایش می‌یابد و total\_allocated کاهش می‌یابد، بنابراین get\_remaining\_amount افزایش می‌یابد.

هنگام تخصیص مجدد: total\_allocated دوباره افزایش می‌یابد، بنابراین get\_remaining\_amount به حالت اولیه برمی‌گردد.

نتیجه: جابجایی بودجه تأثیری روی کل بودجه سازمان ندارد، فقط توزیع آن بین پروژه‌ها و شعب را تغییر می‌دهد.

مدیریت جابجایی بین‌سازمانی:

اگر جابجایی بین دو شعبه (مثل شعبه A و B) باشد:

بازگشت به BudgetPeriod انجام می‌شود، زیرا بودجه باید از تخصیص شعبه A آزاد شده و به تخصیص شعبه B منتقل شود.

این فرآیند تضمین می‌کند که بودجه کل سازمان ثابت بماند.

برای جابجایی درون‌سازمانی (مثل پروژه X به Y در شعبه A):

می‌توان مستقیماً بودجه را بین دو ProjectBudgetAllocation در همان BudgetAllocation جابجا کرد، اما برای سازگاری با سیستم، بهتر است فرآیند بازگشت و تخصیص مجدد را دنبال کنیم.

چالش‌ها:

پیچیدگی ردیابی: نیاز به ثبت دقیق هر دو تراکنش (بازگشت و تخصیص) و مرتبط کردن آن‌ها.

یکپارچگی داده‌ها: باید تمام به‌روزرسانی‌ها در یک تراکنش اتمیک انجام شوند تا از ناسازگاری جلوگیری شود.

عملکرد: کوئری‌های مکرر روی BudgetHistory برای گزارش‌گیری ممکن است سنگین باشد، بنابراین کشینگ ضروری است.

3. تصمیم در مورد فیلد returned\_amount

با توجه به سناریوی جابجایی بودجه:

نیاز به returned\_amount در ProjectBudgetAllocation:

این فیلد برای ردیابی بودجه برگشتی از هر پروژه ضروری است، به‌خصوص در جابجایی‌ها، زیرا نشان می‌دهد چه مقدار از بودجه پروژه به تخصیص یا دوره برگشت داده شده است.

بدون این فیلد، باید برای هر محاسبه بودجه برگشتی پروژه، تراکنش‌های RETURN را کوئری کنید، که عملکرد را کاهش می‌دهد.

نتیجه: فیلد returned\_amount را به ProjectBudgetAllocation اضافه کنید.

حفظ returned\_amount در BudgetPeriod و BudgetAllocation:

این فیلدها برای ردیابی بازگشت‌ها در سطوح بالاتر (دوره و تخصیص) ضروری هستند.

در جابجایی بین‌سازمانی، BudgetPeriod.returned\_amount نقش کلیدی دارد، زیرا بودجه از یک شعبه به دوره برگشت داده شده و سپس به شعبه دیگر تخصیص می‌یابد.

نتیجه: این فیلدها را حفظ کنید.

نتیجه‌گیری:

فیلد returned\_amount در هر سه مدل (BudgetPeriod, BudgetAllocation, ProjectBudgetAllocation) نگه داشته شود.

این ساختار امکان ردیابی دقیق بازگشت‌ها و جابجایی‌ها را در تمام سطوح فراهم می‌کند.

4. کد پیشنهادی

با تأیید سناریو و تصمیم برای افزودن returned\_amount به ProjectBudgetAllocation و حفظ آن در مدل‌های دیگر، کدهای نهایی را ارائه می‌کنم. این کدها شامل:

مهاجرت برای افزودن returned\_amount به ProjectBudgetAllocation.

فرم و ویو برای ثبت جابجایی بودجه (بازگشت و تخصیص مجدد).

تمپلیت برای رابط کاربری.

تابع گزارش‌گیری برای ردیابی جابجایی‌ها.

سیگنال‌ها برای مدیریت کش و یکپارچگی داده‌ها.

4.1. مهاجرت برای افزودن returned\_amount

python

# budgets/migrations/0006\_add\_returned\_amount\_to\_projectbudgetallocation.py

from django.db import migrations, models

class Migration(migrations.Migration):

dependencies = [

('budgets', '0005\_previous\_migration'), # به‌روزرسانی با مهاجرت قبلی

]

operations = [

migrations.AddField(

model\_name='ProjectBudgetAllocation',

name='returned\_amount',

field=models.DecimalField(

decimal\_places=2,

default=0,

max\_digits=25,

verbose\_name='مجموع بودجه برگشتی'

),

),

]

اجرا:

bash

python manage.py makemigrations

python manage.py migrate

توجه: اگر فیلد returned\_amount قبلاً در دیتابیس وجود دارد، از دستور migrate --fake استفاده کنید یا مهاجرت را حذف کنید.

4.2. فرم (BudgetTransferForm)

برای جابجایی بودجه، فرمی طراحی می‌کنیم که هم بازگشت و هم تخصیص مجدد را مدیریت کند.

python

# budgets/forms.py

from django import forms

from django.utils.translation import gettext\_lazy as \_

from django.db.models import Sum, Q

from django.core.cache import cache

from django.db import transaction

from decimal import Decimal

from budgets.models import BudgetTransaction, ProjectBudgetAllocation, BudgetAllocation

from budgets.budget\_calculations import get\_project\_remaining\_budget

import logging

logger = logging.getLogger(\_\_name\_\_)

class BudgetTransferForm(forms.Form):

source\_allocation = forms.ModelChoiceField(

queryset=ProjectBudgetAllocation.objects.all(),

label=\_("تخصیص مبدأ"),

widget=forms.Select(attrs={'class': 'form-control'})

)

destination\_allocation = forms.ModelChoiceField(

queryset=ProjectBudgetAllocation.objects.all(),

label=\_("تخصیص مقصد"),

widget=forms.Select(attrs={'class': 'form-control'})

)

amount = forms.DecimalField(

min\_value=0.01,

decimal\_places=2,

label=\_("مبلغ جابجایی"),

widget=forms.NumberInput(attrs={'class': 'form-control', 'placeholder': \_('مبلغ (ریال)')})

)

description = forms.CharField(

required=False,

label=\_("توضیحات"),

widget=forms.Textarea(attrs={'class': 'form-control', 'rows': 3, 'placeholder': \_('توضیحات (اختیاری)')})

)

def \_\_init\_\_(self, \*args, user=None, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

self.user = user

# محدود کردن تخصیص‌ها به مواردی که کاربر دسترسی دارد

if user:

self.fields['source\_allocation'].queryset = ProjectBudgetAllocation.objects.filter(

budget\_allocation\_\_organization\_\_in=user.get\_authorized\_organizations(),

is\_active=True

)

self.fields['destination\_allocation'].queryset = ProjectBudgetAllocation.objects.filter(

budget\_allocation\_\_organization\_\_in=user.get\_authorized\_organizations(),

is\_active=True

)

def clean(self):

cleaned\_data = super().clean()

source\_allocation = cleaned\_data.get('source\_allocation')

destination\_allocation = cleaned\_data.get('destination\_allocation')

amount = cleaned\_data.get('amount')

if not all([source\_allocation, destination\_allocation, amount]):

raise forms.ValidationError(\_('تمام فیلدها الزامی هستند.'))

if source\_allocation == destination\_allocation:

raise forms.ValidationError(\_('تخصیص مبدأ و مقصد نمی‌توانند یکسان باشند.'))

# بررسی مصرف خالص مبدأ

cache\_key = f"net\_consumed\_{source\_allocation.pk}"

net\_consumed = cache.get(cache\_key)

if net\_consumed is None:

transactions = BudgetTransaction.objects.filter(

allocation=source\_allocation.budget\_allocation,

project=source\_allocation.project

).aggregate(

consumed=Sum('amount', filter=Q(transaction\_type='CONSUMPTION')),

returned=Sum('amount', filter=Q(transaction\_type='RETURN'))

)

consumed = transactions['consumed'] or Decimal('0')

returned = transactions['returned'] or Decimal('0')

net\_consumed = consumed - returned

cache.set(cache\_key, net\_consumed, timeout=300)

if amount > net\_consumed:

raise forms.ValidationError(

\_(

f"مبلغ جابجایی ({amount:,.0f} ریال) نمی‌تواند بیشتر از مصرف خالص "

f"({net\_consumed:,.0f} ریال) باشد."

)

)

# بررسی بودجه باقی‌مانده پروژه مبدأ

remaining\_budget = get\_project\_remaining\_budget(source\_allocation.project)

if amount > remaining\_budget:

raise forms.ValidationError(

\_(

f"مبلغ جابجایی ({amount:,.0f} ریال) نمی‌تواند بیشتر از بودجه باقی‌مانده پروژه "

f"({remaining\_budget:,.0f} ریال) باشد."

)

)

# بررسی ظرفیت تخصیص مقصد

budget\_period = destination\_allocation.budget\_allocation.budget\_period

remaining\_period\_budget = budget\_period.get\_remaining\_amount()

if amount > remaining\_period\_budget:

raise forms.ValidationError(

\_(

f"مبلغ جابجایی ({amount:,.0f} ریال) نمی‌تواند بیشتر از بودجه باقی‌مانده دوره "

f"({remaining\_period\_budget:,.0f} ریال) باشد."

)

)

return cleaned\_data

def save(self):

from django.utils import timezone

source\_allocation = self.cleaned\_data['source\_allocation']

destination\_allocation = self.cleaned\_data['destination\_allocation']

amount = self.cleaned\_data['amount']

description = self.cleaned\_data['description']

transfer\_id = f"TRF-{timezone.now().strftime('%Y%m%d%H%M%S%f')}"

with transaction.atomic():

# مرحله 1: بازگشت از مبدأ

return\_transaction = BudgetTransaction.objects.create(

allocation=source\_allocation.budget\_allocation,

project=source\_allocation.project,

transaction\_type='RETURN',

amount=amount,

description=f"جابجایی به پروژه {destination\_allocation.project.name}: {description}",

created\_by=self.user,

transaction\_id=f"{transfer\_id}-RETURN"

)

source\_allocation.returned\_amount = (

source\_allocation.returned\_amount or Decimal('0')

) + amount

source\_allocation.allocated\_amount -= amount

source\_allocation.budget\_allocation.returned\_amount = (

source\_allocation.budget\_allocation.returned\_amount or Decimal('0')

) + amount

source\_allocation.budget\_allocation.allocated\_amount -= amount

source\_allocation.budget\_allocation.budget\_period.returned\_amount = (

source\_allocation.budget\_allocation.budget\_period.returned\_amount or Decimal('0')

) + amount

source\_allocation.budget\_allocation.budget\_period.total\_allocated -= amount

source\_allocation.save(update\_fields=['returned\_amount', 'allocated\_amount'])

source\_allocation.budget\_allocation.save(

update\_fields=['returned\_amount', 'allocated\_amount']

)

source\_allocation.budget\_allocation.budget\_period.save(

update\_fields=['returned\_amount', 'total\_allocated']

)

# ثبت تاریخچه بازگشت

from budgets.models import BudgetHistory

from django.contrib.contenttypes.models import ContentType

BudgetHistory.objects.create(

content\_type=ContentType.objects.get\_for\_model(ProjectBudgetAllocation),

object\_id=source\_allocation.id,

action='RETURN',

amount=amount,

created\_by=self.user,

details=f"جابجایی به پروژه {destination\_allocation.project.name}: {description}",

transaction\_type='RETURN',

transaction\_id=f"{transfer\_id}-RETURN"

)

# مرحله 2: تخصیص به مقصد

allocation\_transaction = BudgetTransaction.objects.create(

allocation=destination\_allocation.budget\_allocation,

project=destination\_allocation.project,

transaction\_type='ALLOCATION',

amount=amount,

description=f"جابجایی از پروژه {source\_allocation.project.name}: {description}",

created\_by=self.user,

transaction\_id=f"{transfer\_id}-ALLOCATION"

)

destination\_allocation.allocated\_amount += amount

destination\_allocation.budget\_allocation.allocated\_amount += amount

destination\_allocation.budget\_allocation.budget\_period.total\_allocated += amount

destination\_allocation.save(update\_fields=['allocated\_amount Hermione

destination\_allocation.budget\_allocation.budget\_period.save(

update\_fields=['total\_allocated']

)

# ثبت تاریخچه تخصیص

BudgetHistory.objects.create(

content\_type=ContentType.objects.get\_for\_model(ProjectBudgetAllocation),

object\_id=destination\_allocation.id,

action='ALLOCATION',

amount=amount,

created\_by=self.user,

details=f"جابجایی از پروژه {source\_allocation.project.name}: {description}",

transaction\_type='ALLOCATION',

transaction\_id=f"{transfer\_id}-ALLOCATION"

)

# ارسال اعلان‌ها

from budgets.budget\_calculations import check\_budget\_status

status, message = check\_budget\_status(source\_allocation.budget\_allocation.budget\_period)

if status in ('warning', 'locked', 'completed', 'stopped'):

source\_allocation.budget\_allocation.send\_notification(status, message)

source\_allocation.budget\_allocation.send\_notification(

'return',

f"مبلغ {amount:,.0f} ریال از پروژه {source\_allocation.project.name} جابجا شد."

)

status, message = check\_budget\_status(destination\_allocation.budget\_allocation.budget\_period)

if status in ('warning', 'locked', 'completed', 'stopped'):

destination\_allocation.budget\_allocation.send\_notification(status, message)

destination\_allocation.budget\_allocation.send\_notification(

'allocation',

f"مبلغ {amount:,.0f} ریال از پروژه {source\_allocation.project.name} دریافت شد."

)

return return\_transaction, allocation\_transaction

4.3. ویو (BudgetTransferView)

python

# budgets/views.py

from django.contrib.auth.mixins import LoginRequiredMixin

from django.core.exceptions import PermissionDenied, ValidationError

from django.http import Http404

from django.shortcuts import redirect

from django.views.generic import FormView

from django.urls import reverse\_lazy

from django.contrib import messages

from django.utils.translation import gettext\_lazy as \_

from budgets.forms import BudgetTransferForm

from core.PermissionBase import PermissionBaseView

import logging

logger = logging.getLogger(\_\_name\_\_)

class BudgetTransferView(LoginRequiredMixin, PermissionBaseView, FormView):

form\_class = BudgetTransferForm

template\_name = 'budgets/budget\_transfer\_form.html'

permission\_codenames = ['budgets.budgetallocation\_adjust']

check\_organization = True

def get\_form\_kwargs(self):

kwargs = super().get\_form\_kwargs()

kwargs['user'] = self.request.user

return kwargs

def form\_valid(self, form):

try:

return\_transaction, allocation\_transaction = form.save()

messages.success(self.request, \_("جابجایی بودجه با موفقیت انجام شد."))

logger.info(

f"User {self.request.user.username} transferred {form.cleaned\_data['amount']} "

f"from allocation {form.cleaned\_data['source\_allocation'].id} "

f"to allocation {form.cleaned\_data['destination\_allocation'].id}"

)

return super().form\_valid(form)

except ValidationError as e:

logger.error(f"Error transferring budget: {str(e)}")

form.add\_error(None, e)

return self.form\_invalid(form)

def get\_success\_url(self):

return reverse\_lazy('budgets:budgetallocation\_list')

def \_get\_organization\_from\_object(self, obj):

# برای بررسی دسترسی، از سازمان تخصیص مبدأ استفاده می‌کنیم

form = self.get\_form()

source\_allocation = form.cleaned\_data.get('source\_allocation')

return source\_allocation.budget\_allocation.organization if source\_allocation else None

4.4. تمپلیت (budget\_transfer\_form.html)

html

{% extends 'base.html' %}

{% load i18n rcms\_custom\_filters %}

{% block title %}{% trans "جابجایی بودجه" %}{% endblock %}

{% block content %}

<div class="container mt-5">

<h1 class="mb-4 text-center">{% trans "جابجایی بودجه بین پروژه‌ها" %}</h1>

<div class="card shadow">

<div class="card-header bg-primary text-white">

<h3 class="mb-0">{% trans "فرم جابجایی بودجه" %}</h3>

</div>

<div class="card-body">

<form method="post">

{% csrf\_token %}

<div class="mb-3">

<label for="{{ form.source\_allocation.id\_for\_label }}" class="form-label">

{% trans "تخصیص مبدأ" %}

</label>

{{ form.source\_allocation }}

{% if form.source\_allocation.errors %}

<div class="text-danger">{{ form.source\_allocation.errors }}</div>

{% endif %}

</div>

<div class="mb-3">

<label for="{{ form.destination\_allocation.id\_for\_label }}" class="form-label">

{% trans "تخصیص مقصد" %}

</label>

{{ form.destination\_allocation }}

{% if form.destination\_allocation.errors %}

<div class="text-danger">{{ form.destination\_allocation.errors }}</div>

{% endif %}

</div>

<div class="mb-3">

<label for="{{ form.amount.id\_for\_label }}" class="form-label">

{% trans "مبلغ جابجایی" %}

</label>

{{ form.amount }}

{% if form.amount.errors %}

<div class="text-danger">{{ form.amount.errors }}</div>

{% endif %}

</div>

<div class="mb-3">

<label for="{{ form.description.id\_for\_label }}" class="form-label">

{% trans "توضیحات" %}

</label>

{{ form.description }}

{% if form.description.errors %}

<div class="text-danger">{{ form.description.errors }}</div>

{% endif %}

</div>

{% if form.non\_field\_errors %}

<div class="text-danger mb-3">{{ form.non\_field\_errors }}</div>

{% endif %}

<div class="text-center">

<button type="submit" class="btn btn-primary">{% trans "انجام جابجایی" %}</button>

<a href="{% url 'budgets:budgetallocation\_list' %}" class="btn btn-secondary">

{% trans "لغو" %}

</a>

</div>

</form>

</div>

</div>

</div>

{% endblock %}

4.5. گزارش‌گیری (get\_budget\_transfers)

python

# budgets/reports.py

from django.contrib.contenttypes.models import ContentType

from django.core.cache import cache

from django.db.models import Q

from budgets.models import BudgetHistory

import logging

logger = logging.getLogger(\_\_name\_\_)

def get\_budget\_transfers(budget\_period, filters=None):

"""

گزارش جابجایی‌های بودجه در یک دوره.

Args:

budget\_period: نمونه BudgetPeriod

filters (dict): فیلترهای اختیاری (مثل date\_from, date\_to)

Returns:

list: لیستی از جابجایی‌ها با جزئیات

"""

cache\_key = f"budget\_transfers\_{budget\_period.pk}\_{hash(str(filters)) if filters else 'no\_filters'}"

cached\_result = cache.get(cache\_key)

if cached\_result is not None:

logger.debug(f"Returning cached budget transfers: {cache\_key}")

return cached\_result

content\_types = [

ContentType.objects.get\_for\_model(ProjectBudgetAllocation),

ContentType.objects.get\_for\_model(BudgetAllocation),

]

# یافتن تراکنش‌های RETURN و ALLOCATION مرتبط

transfers = []

return\_transactions = BudgetHistory.objects.filter(

content\_type\_\_in=content\_types,

action='RETURN',

content\_object\_\_budget\_period=budget\_period,

transaction\_id\_\_endswith='-RETURN'

).select\_related('created\_by')

for return\_tx in return\_transactions:

transfer\_id = return\_tx.transaction\_id.replace('-RETURN', '')

allocation\_tx = BudgetHistory.objects.filter(

content\_type\_\_in=content\_types,

action='ALLOCATION',

content\_object\_\_budget\_period=budget\_period,

transaction\_id=f"{transfer\_id}-ALLOCATION"

).select\_related('created\_by').first()

if allocation\_tx:

transfers.append({

'transfer\_id': transfer\_id,

'source': {

'content\_type': return\_tx.content\_type.model,

'object\_id': return\_tx.object\_id,

'amount': return\_tx.amount,

'details': return\_tx.details,

'created\_at': return\_tx.created\_at,

'created\_by': return\_tx.created\_by.username,

},

'destination': {

'content\_type': allocation\_tx.content\_type.model,

'object\_id': allocation\_tx.object\_id,

'amount': allocation\_tx.amount,

'details': allocation\_tx.details,

'created\_at': allocation\_tx.created\_at,

'created\_by': allocation\_tx.created\_by.username,

}

})

if filters:

from budgets.budget\_calculations import apply\_filters

transfers = apply\_filters(transfers, filters)

cache.set(cache\_key, transfers, timeout=3600)

logger.debug(f"Cached budget transfers: {cache\_key}")

return transfers

4.6. سیگنال‌ها

سیگنال‌های قبلی (ارائه‌شده در پاسخ‌های قبل) برای باطل کردن کش‌ها کافی هستند، اما برای اطمینان، کد را بازنویسی می‌کنم:

python

# budgets/signals.py

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

from django.dispatch import receiver

from django.core.cache import cache

from budgets.models import BudgetTransaction, BudgetAllocation, ProjectBudgetAllocation

import logging

logger = logging.getLogger(\_\_name\_\_)

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

def invalidate\_transaction\_cache(sender, instance, \*\*kwargs):

cache\_keys = set()

try:

project\_allocations = instance.allocation.project\_allocations.all()

for alloc in project\_allocations:

if alloc.project:

cache\_keys.add(f"project\_remaining\_budget\_{alloc.project.pk}")

if alloc.subproject:

cache\_keys.add(f"subproject\_remaining\_budget\_{alloc.subproject.pk}")

if hasattr(instance.allocation, 'tankhah') and instance.allocation.tankhah:

cache\_keys.add(f"tankhah\_remaining\_budget\_{instance.allocation.tankhah.pk}")

cache\_keys.add(f"net\_consumed\_{instance.allocation.pk}")

cache\_keys.add(f"budget\_transfers\_{instance.allocation.budget\_period.pk}\_no\_filters")

for key in cache\_keys:

cache.delete(key)

logger.debug(f"Invalidated cache keys after BudgetTransaction change: {cache\_keys}")

except Exception as e:

logger.error(f"Error invalidating cache for BudgetTransaction {instance.pk}: {str(e)}", exc\_info=True)

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

def invalidate\_allocation\_cache(sender, instance, \*\*kwargs):

cache\_keys = set()

try:

cache\_keys.add(f"organization\_remaining\_budget\_{instance.organization.pk}")

cache\_keys.add(f"budget\_transfers\_{instance.budget\_period.pk}\_no\_filters")

project\_allocations = instance.project\_allocations.all()

for alloc in project\_allocations:

if alloc.project:

cache\_keys.add(f"project\_total\_budget\_{alloc.project.pk}")

if alloc.subproject:

cache\_keys.add(f"subproject\_total\_budget\_{alloc.subproject.pk}")

for key in cache\_keys:

cache.delete(key)

logger.debug(f"Invalidated cache keys after BudgetAllocation change: {cache\_keys}")

except Exception as e:

logger.error(f"Error invalidating cache for BudgetAllocation {instance.pk}: {str(e)}", exc\_info=True)

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

def invalidate\_project\_allocation\_cache(sender, instance, \*\*kwargs):

cache\_keys = set()

try:

if instance.project:

cache\_keys.add(f"project\_total\_budget\_{instance.project.pk}")

cache\_keys.add(f"project\_remaining\_budget\_{instance.project.pk}")

if instance.subproject:

cache\_keys.add(f"subproject\_total\_budget\_{instance.subproject.pk}")

cache\_keys.add(f"subproject\_remaining\_budget\_{instance.subproject.pk}")

cache\_keys.add(f"net\_consumed\_{instance.pk}")

cache\_keys.add(f"budget\_transfers\_{instance.budget\_allocation.budget\_period.pk}\_no\_filters")

for key in cache\_keys:

cache.delete(key)

logger.debug(f"Invalidated cache keys after ProjectBudgetAllocation change: {cache\_keys}")

except Exception as e:

logger.error(f"Error invalidating cache for ProjectBudgetAllocation {instance.pk}: {str(e)}", exc\_info=True)

4.7. URL‌ها

python

# budgets/urls.py

from django.urls import path

from budgets.views import BudgetTransferView

app\_name = 'budgets'

urlpatterns = [

path(

'budget-transfer/',

BudgetTransferView.as\_view(),

name='budget\_transfer'

),

path(

'project-budget-allocation/<int:pk>/detail/',

ProjectBudgetAllocationDetailView.as\_view(),

name='project\_budget\_allocation\_detail'

),

path(

'budget-allocations/',

BudgetAllocationListView.as\_view(),

name='budgetallocation\_list'

),

]

تست‌ها

برای اطمینان از صحت عملکرد، تست‌های زیر را پیشنهاد می‌کنم:

python

# budgets/tests.py

from django.test import TestCase

from django.contrib.auth import get\_user\_model

from budgets.models import BudgetPeriod, BudgetAllocation, ProjectBudgetAllocation, BudgetTransaction

from budgets.forms import BudgetTransferForm

from decimal import Decimal

User = get\_user\_model()

class BudgetTransferTests(TestCase):

def setUp(self):

self.user = User.objects.create\_user(username='testuser', password='testpass')

self.organization\_a = ... # سازمان A

self.organization\_b = ... # سازمان B

self.budget\_period = BudgetPeriod.objects.create(

organization=self.organization\_a,

total\_amount=Decimal('1000000000'),

is\_active=True

)

self.budget\_allocation\_a = BudgetAllocation.objects.create(

budget\_period=self.budget\_period,

organization=self.organization\_a,

allocated\_amount=Decimal('500000000')

)

self.budget\_allocation\_b = BudgetAllocation.objects.create(

budget\_period=self.budget\_period,

organization=self.organization\_b,

allocated\_amount=Decimal('400000000')

)

self.project\_x = ... # پروژه X

self.project\_z = ... # پروژه Z

self.source\_allocation = ProjectBudgetAllocation.objects.create(

budget\_allocation=self.budget\_allocation\_a,

project=self.project\_x,

allocated\_amount=Decimal('300000000')

)

self.destination\_allocation = ProjectBudgetAllocation.objects.create(

budget\_allocation=self.budget\_allocation\_b,

project=self.project\_z,

allocated\_amount=Decimal('400000000')

)

BudgetTransaction.objects.create(

allocation=self.budget\_allocation\_a,

project=self.project\_x,

transaction\_type='CONSUMPTION',

amount=Decimal('100000000'),

created\_by=self.user

)

def test\_budget\_transfer\_form\_valid(self):

form\_data = {

'source\_allocation': self.source\_allocation,

'destination\_allocation': self.destination\_allocation,

'amount': Decimal('50000000'),

'description': 'Test transfer'

}

form = BudgetTransferForm(data=form\_data, user=self.user)

self.assertTrue(form.is\_valid())

return\_tx, alloc\_tx = form.save()

self.assertEqual(return\_tx.amount, Decimal('50000000'))

self.assertEqual(return\_tx.transaction\_type, 'RETURN')

self.assertEqual(alloc\_tx.amount, Decimal('50000000'))

self.assertEqual(alloc\_tx.transaction\_type, 'ALLOCATION')

def test\_budget\_transfer\_view(self):

self.client.login(username='testuser', password='testpass')

response = self.client.post(

'/budgets/budget-transfer/',

{

'source\_allocation': self.source\_allocation.pk,

'destination\_allocation': self.destination\_allocation.pk,

'amount': '50000000',

'description': 'Test transfer'

}

)

self.assertRedirects(response, '/budgets/budget-allocations/')

self.assertTrue(

BudgetTransaction.objects.filter(

transaction\_type='RETURN',

amount=Decimal('50000000')

).exists()

)

self.assertTrue(

BudgetTransaction.objects.filter(

transaction\_type='ALLOCATION',

amount=Decimal('50000000')

).exists()

)

جمع‌بندی

سناریو:

جابجایی بودجه بین پروژه‌ها (درون‌سازمانی یا بین‌سازمانی) با استفاده از فرآیند بازگشت و تخصیص مجدد پیاده‌سازی شد.

بازگشت بودجه از پروژه مبدأ به BudgetPeriod و سپس تخصیص به پروژه مقصد انجام می‌شود.

تمام تغییرات در returned\_amount و allocated\_amount در مدل‌های ProjectBudgetAllocation, BudgetAllocation, و BudgetPeriod ثبت می‌شوند.

ردیابی جابجایی‌ها با BudgetHistory و تابع get\_budget\_transfers امکان‌پذیر است.

فیلد returned\_amount:

به ProjectBudgetAllocation اضافه شد برای ردیابی دقیق‌تر.

در BudgetPeriod و BudgetAllocation حفظ شد برای انعطاف‌پذیری و گزارش‌گیری.

مزایا:

سازگاری کامل با ساختار موجود.

ردیابی شفاف با BudgetHistory.

عملکرد بهینه با کشینگ.

یکپارچگی داده‌ها با تراکنش‌های اتمیک.

اقدام بعدی:

مهاجرت‌ها را اعمال کنید و تست‌ها را اجرا کنید.

اگر نیاز به قابلیت اضافی (مثل نمایش جابجایی‌ها در داشبورد یا فیلترهای پیشرفته‌تر در گزارش‌گیری) دارید، اطلاع دهید.

اگر خطایی در اجرا رخ داد، جزئیات را بفرستید تا رفع کنم.

لطفاً تأیید کنید که کدهای ارائه‌شده با نیازهای شما مطابقت دارند یا اگر تغییری لازم است، مشخص کنید.

سوال شما بسیار کلی است ("چرا؟")، اما با توجه به تحلیل‌های اخیر ما بر اساس منابع ارائه شده در خصوص سیستم بودجه‌بندی و تنخواه و همچنین مشکلات و پیشنهاداتی که مطرح شد، می‌توانم دلایل نیاز به برخی اصلاحات و رویکردهای پیشنهادی را بر اساس اطلاعات موجود در منابع توضیح دهم.

دلایل این پیشنهادها و نیاز به اصلاحات را می‌توان در موارد زیر خلاصه کرد:

1. **چرا اصلاح و استفاده صحیح از فیلد remaining\_amount در BudgetAllocation حیاتی است؟**
   * منابع نشان می‌دهند که محاسبه یا استفاده از فیلد remaining\_amount در مدل BudgetAllocation مشکل دارد.
   * این موضوع منجر به **عدم دقت و یکپارچگی داده‌های بودجه** می‌شود، به خصوص در محاسبات مربوط به مانده بودجه تخصیص یافته به شعب یا پروژه‌ها.
   * برای **ردیابی صحیح** اینکه چقدر از بودجه تخصیص یافته باقی مانده و قابل استفاده است، وجود این فیلد و **محاسبه دقیق** آن بر اساس تمام تراکنش‌های مرتبط (مصرف، برگشت، تعدیل) ضروری است.
   * این اصلاح پایه و اساس مکانیسم‌های دیگری مانند برگشت بودجه است.
2. **چرا نیاز به پیاده‌سازی مکانیسم "برگشت بودجه تخصیص یافته" وجود دارد؟**
   * سناریوهای مالی ایجاب می‌کنند که در برخی موارد، بودجه‌ای که به یک شعبه یا پروژه تخصیص یافته، به بودجه کلان سازمان برگردانده شود. این اتفاق می‌تواند به دلیل عدم نیاز به بودجه در آن واحد یا لغو پروژه رخ دهد.
   * پیاده‌سازی این فرآیند امکان **مدیریت مجدد منابع** در سطح دفتر مرکزی را فراهم می‌کند.
   * این مکانیسم باید با ثبت یک تراکنش نوع RETURN در BudgetTransaction و به‌روزرسانی فیلدهای مربوطه (مانده تخصیص، مبلغ تخصیست یافته و مبلغ برگشتی در BudgetAllocation و مبلغ برگشتی در BudgetPeriod و کاهش مبلغ تخصیص یافته کل در BudgetPeriod) انجام شود.
3. **چرا نیاز به پیاده‌سازی مکانیسم "برگشت هزینه تنخواه" وجود دارد؟**
   * ممکن است هزینه‌هایی که از طریق تنخواه ثبت شده‌اند (به عنوان مثال به یک مرکز هزینه یا پروژه لینک شده‌اند)، نیاز به لغو یا برگشت داشته باشند (مانند ثبت اشتباه).
   * بدون این مکانیسم، **داده‌های مصرف بودجه نادرست باقی می‌مانند** و امکان اصلاح یا معکوس کردن تراکنش وجود ندارد.
   * این فرآیند نیز مستلزم شناسایی تنخواه‌های لغوشده و ثبت تراکنش معکوس (RETURN) برای بازگرداندن مبلغ به تخصیص مربوطه و به‌روزرسانی مانده‌ها است.
4. **چرا پیشنهاد می‌شود منطق پیچیده کسب و کار (مانند محاسبات، گردش کار، اعتبارسنجی چندمدلی) به یک لایه سرویس منتقل شود؟**
   * منابع به وضوح اشاره می‌کنند که منطق پیچیده در حال حاضر در متدهای save و clean مدل‌ها و همچنین در ویوها پخش شده است.
   * این باعث می‌شود کد مدل‌ها و ویوها **پیچیده، سنگین و دشوار برای خواندن، تست و نگهداری** شود.
   * انتقال این منطق به یک لایه سرویس مجزا اصول **جداسازی مسئولیت‌ها (Separation of Concerns)** و **اصل مسئولیت واحد (Single Responsibility Principle)** را رعایت می‌کند.
   * این کار باعث **افزایش قابلیت تست (Testability)** می‌شود، زیرا می‌توان منطق سرویس را به صورت واحد (Unit Test) تست کرد بدون نیاز به وابستگی‌های پیچیده مدل یا ویو.
   * همچنین، **قابلیت نگهداری (Maintainability)** و **خوانایی (Readability)** کد را بهبود می‌بخشد.
5. **چرا نیاز به بازنگری و تست کامل منطق مدیریت دسترسی، به خصوص PermissionBaseView وجود دارد؟**
   * منطق دسترسی فعلی، به خصوص has\_organization\_access\_، برای پوشش انواع ویوها و روابط مدل‌ها **پیچیده** شده است.
   * پیچیدگی در منطق دسترسی می‌تواند منجر به **حفره‌های امنیتی یا رد دسترسی‌های مجاز** شود.
   * ساده‌سازی و تست بسیار دقیق این بخش برای اطمینان از **امنیت، قابلیت اطمینان و صحت** سیستم در کنترل دسترسی کاربران به داده‌های سازمانی حیاتی است.
   * منابع پیشنهاد می‌کنند که دسترسی باید **علاوه بر ویوها، در ابتدای توابع سرویس** نیز بررسی شود تا از اجرای منطق حساس توسط کاربران غیرمجاز جلوگیری شود (**اصل Defense in Depth**).
6. **چرا استفاده از Signals برای به‌روزرسانی مقادیر تجمعی و ثبت تاریخچه پیشنهاد می‌شود؟**
   * به‌روزرسانی فیلدهای تجمعی (مانند total\_allocated, returned\_amount در BudgetPeriod) و ثبت تاریخچه (مانند BudgetHistory) اقدامات جانبی هستند که معمولاً در متد save مدل اصلی انجام می‌شوند.
   * قرار دادن این منطق در save می‌تواند آن را پیچیده کند و از **اتمی بودن** عملیات اصلی بکاهد.
   * استفاده از سیگنال‌های جنگو (مثل post\_save) برای انجام این اقدامات **بعد از ذخیره موفق مدل اصلی**، منطق save را ساده‌تر نگه می‌دارد و اطمینان حاصل می‌کند که به‌روزرسانی‌های تجمعی و ثبت تاریخچه به صورت اتمی انجام شوند.
7. **چرا نوشتن تست‌های واحد (Unit Tests) و یکپارچگی (Integration Tests) ضروری است؟**
   * تست‌ها **صحت عملکرد سیستم** را در سناریوهای مختلف تضمین می‌کنند.
   * تست‌های واحد برای **تأیید عملکرد صحیح منطق موجود در سرویس‌ها و متدهای ساده مدل‌ها** (مانند clean, property) ضروری هستند. منطق پیچیده در مدل‌ها و ویوهای بدون تست واحد دشوار است.
   * تست‌های یکپارچگی برای **بررسی تعامل صحیح بین مدل‌ها، سرویس‌ها و پایگاه داده** در سناریوهای کلیدی (مانند ایجاد کامل بودجه، گردش کار تنخواه، برگشت بودجه) لازم هستند.
   * تست‌ها به **شناسایی زودهنگام خطاها** کمک کرده و باعث **افزایش پایداری و قابلیت اطمینان** سیستم می‌شوند.

این دلایل بر اساس تحلیل‌های صورت گرفته در منابع، ضرورت اعمال تغییرات و بهبودهای پیشنهادی را برای دستیابی به سیستمی **دقیق‌تر، پایدارتر، امن‌تر، قابل تست‌تر و نگهداری بهتر** مشخص می‌کنند.

بسیار خوب! با توجه به سوالات و نیازهای مطرح‌شده، سناریوی جابجایی و برگشت بودجه را با جزئیات بیشتری بررسی می‌کنم، به‌ویژه با تمرکز بر انتقال بودجه بین شعب/پروژه‌ها، برگشت بودجه، تأثیر روی تنخواه و فاکتورها، اعلان‌ها، گزارش‌گیری، تمپلیت قوی، و مدیریت قفل شدن. همچنین، به این نکته که ممکن است کل بودجه جابجا یا برگشت نشود و تأثیر آن بر تنخواه و فاکتورها پرداخته خواهد شد. ابتدا سناریو را به‌صورت کامل تحلیل می‌کنم، سپس تأیید شما را می‌گیرم و بعد کد نهایی را ارائه می‌دهم.

تحلیل و گسترش سناریو

1. جابجایی بودجه و برگشت بودجه

سوال: آیا با این سناریو می‌توان بودجه را از پروژه جابجا کرد یا برگشت بودجه داد؟

پاسخ: بله، سناریوی پیشنهادی (رویکرد بازگشت و تخصیص مجدد) امکان هر دو عملیات را فراهم می‌کند:

جابجایی بودجه: بودجه از یک پروژه (یا تخصیص) به پروژه دیگر (در همان شعبه یا شعبه دیگر) منتقل می‌شود. این فرآیند شامل:

بازگشت بودجه از پروژه مبدأ به تخصیص یا دوره بودجه (BudgetPeriod).

تخصیص مجدد بودجه به پروژه مقصد.

برگشت بودجه: بودجه از پروژه به تخصیص یا دوره بودجه بازمی‌گردد بدون تخصیص مجدد. این حالت زمانی رخ می‌دهد که بودجه‌ای به کل بودجه سازمان برگردانده می‌شود تا بعداً تخصیص یابد.

سازوکار:

جابجایی:

تراکنش RETURN برای پروژه مبدأ ثبت می‌شود (مثلاً پروژه X در شعبه A).

تراکنش ALLOCATION برای پروژه مقصد ثبت می‌شود (مثلاً پروژه Z در شعبه B).

هر دو تراکنش در BudgetHistory با یک transfer\_id مشترک ثبت می‌شوند تا قابل ردیابی باشند.

برگشت:

فقط تراکنش RETURN ثبت می‌شود.

بودجه به BudgetPeriod یا BudgetAllocation برگردانده می‌شود و در returned\_amount ثبت می‌شود.

مزایا:

هر دو عملیات با مدل‌های موجود (BudgetTransaction, BudgetHistory) سازگار هستند.

امکان جابجایی جزئی یا کامل بودجه (مثلاً 50,000,000 از 300,000,000 ریال).

ردیابی دقیق با BudgetHistory.

2. ثبت خودکار تراکنش‌ها در مدل‌های بالاتر

سوال: در صورت انتقال بودجه از شعبه به شعبه یا پروژه به پروژه، تراکنش‌ها در مدل‌های بالاتر (مثل BudgetAllocation و BudgetPeriod) خودکار ثبت می‌شوند؟

پاسخ: بله، تراکنش‌ها به‌صورت خودکار در مدل‌های بالاتر ثبت و به‌روزرسانی می‌شوند.

سازوکار:

برای جابجایی بودجه:

مرحله بازگشت:

در ProjectBudgetAllocation (پروژه مبدأ): returned\_amount افزایش می‌یابد و allocated\_amount کاهش می‌یابد.

در BudgetAllocation (شعبه مبدأ): returned\_amount افزایش می‌یابد و allocated\_amount کاهش می‌یابد.

در BudgetPeriod: returned\_amount افزایش می‌یابد و total\_allocated کاهش می‌یابد.

مرحله تخصیص مجدد:

در ProjectBudgetAllocation (پروژه مقصد): allocated\_amount افزایش می‌یابد.

در BudgetAllocation (شعبه مقصد): allocated\_amount افزایش می‌یابد.

در BudgetPeriod: total\_allocated افزایش می‌یابد.

تمام این به‌روزرسانی‌ها در یک تراکنش اتمیک انجام می‌شوند تا از ناسازگاری جلوگیری شود.

برای برگشت بودجه:

فقط مرحله بازگشت انجام می‌شود (به‌روزرسانی returned\_amount و کاهش allocated\_amount در ProjectBudgetAllocation, BudgetAllocation, و BudgetPeriod).

ثبت تاریخچه:

هر تغییر (بازگشت یا تخصیص) به‌صورت خودکار در BudgetHistory ثبت می‌شود با جزئیات کامل (مثل content\_type, object\_id, amount, details, created\_by).

مزایا:

یکپارچگی داده‌ها با تراکنش‌های اتمیک.

ردیابی خودکار تمام تغییرات در سطوح مختلف.

3. اعلان‌ها

سوال: اعلان‌ها چگونه مدیریت می‌شوند؟

پاسخ: اعلان‌ها به‌صورت خودکار برای اطلاع‌رسانی به کاربران مرتبط (مثل مدیران مالی یا مدیران بودجه) ارسال می‌شوند.

سازوکار:

انواع اعلان‌ها:

برای جابجایی بودجه:

اعلان بازگشت: به کاربران مرتبط با شعبه مبدأ (مثلاً شعبه A) اطلاع می‌دهد که بودجه‌ای از پروژه X برگشت داده شده است.

اعلان تخصیص: به کاربران مرتبط با شعبه مقصد (مثلاً شعبه B) اطلاع می‌دهد که بودجه‌ای به پروژه Z تخصیص داده شده است.

برای برگشت بودجه:

اعلان بازگشت: به کاربران مرتبط با شعبه و دوره بودجه اطلاع می‌دهد که بودجه‌ای به BudgetPeriod یا BudgetAllocation برگشت داده شده است.

اعلان‌های وضعیت:

اگر وضعیت بودجه (مثلاً warning, locked, completed, stopped) پس از جابجایی یا برگشت تغییر کند، اعلان مربوطه ارسال می‌شود.

گیرندگان اعلان:

کاربران با نقش‌های خاص (مثل Financial Manager, Budget Manager) در سازمان/شعبه مربوطه.

کاربرانی که در پروژه یا تخصیص مربوطه دخیل هستند (مثلاً از طریق userpost یا organizations).

محتوای اعلان:

شامل جزئیات مثل مقدار بودجه، پروژه مبدأ/مقصد، تاریخ، و توضیحات (در صورت وجود).

مثال: «مبلغ 50,000,000 ریال از پروژه X به پروژه Z جابجا شد.»

پیاده‌سازی:

از متد send\_notification در مدل‌های BudgetAllocation و BudgetPeriod استفاده می‌شود.

اعلان‌ها می‌توانند از طریق ایمیل، پیام داخل سیستم، یا اعلان‌های وب ارسال شوند (بسته به تنظیمات سیستم).

مزایا:

اطلاع‌رسانی شفاف و به‌موقع به ذی‌نفعان.

امکان شخصی‌سازی گیرندگان و محتوای اعلان‌ها.

4. گزارش‌گیری

سوال: گزارش‌ها چگونه مدیریت می‌شوند؟ وضعیت تنخواه و فاکتورها در گزارش‌ها چگونه است؟

پاسخ: گزارش‌ها برای ردیابی جابجایی‌ها، برگشت‌ها، تنخواه‌ها، و فاکتورها طراحی می‌شوند.

سازوکار:

گزارش جابجایی‌ها:

تابع get\_budget\_transfers (ارائه‌شده در پاسخ قبلی) تمام جابجایی‌ها را با جزئیات (مبدأ، مقصد، مقدار، تاریخ، کاربر) گزارش می‌دهد.

فیلترها: بر اساس دوره بودجه، تاریخ، پروژه، یا شعبه.

مثال خروجی:

json

[

{

"transfer\_id": "TRF-20250510120000",

"source": {

"content\_type": "projectbudgetallocation",

"object\_id": 1,

"amount": 50000000,

"details": "جابجایی به پروژه Z",

"created\_at": "2025-05-10T12:00:00",

"created\_by": "testuser"

},

"destination": {

"content\_type": "projectbudgetallocation",

"object\_id": 2,

"amount": 50000000,

"details": "جابجایی از پروژه X",

"created\_at": "2025-05-10T12:00:00",

"created\_by": "testuser"

}

}

]

گزارش برگشت‌ها:

تابع get\_returned\_budgets (ارائه‌شده در پاسخ قبلی) تمام بودجه‌های برگشتی را گزارش می‌دهد.

فیلترها: بر اساس دوره، پروژه، شعبه، یا نوع تخصیص.

گزارش تنخواه و فاکتورها:

تنخواه:

فرض می‌کنیم مدل Tankhah (تنخواه) وجود دارد و با BudgetAllocation یا ProjectBudgetAllocation مرتبط است.

گزارش تنخواه شامل:

وضعیت تنخواه (مثل PAID, APPROVED, PENDING).

بودجه تخصیص‌یافته به تنخواه.

مقدار مصرف‌شده و برگشتی.

تابع پیشنهادی:

python

def get\_tankhah\_report(budget\_period, filters=None):

cache\_key = f"tankhah\_report\_{budget\_period.pk}\_{hash(str(filters)) if filters else 'no\_filters'}"

cached\_result = cache.get(cache\_key)

if cached\_result is not None:

return cached\_result

queryset = Tankhah.objects.filter(

allocation\_\_budget\_period=budget\_period

).select\_related('allocation', 'project').values(

'id',

'status',

'allocated\_amount',

'consumed\_amount',

'returned\_amount',

'project\_\_name',

'allocation\_\_organization\_\_name'

)

if filters:

queryset = apply\_filters(queryset, filters)

cache.set(cache\_key, queryset, timeout=3600)

return queryset

فاکتورها:

فرض می‌کنیم مدل Invoice (فاکتور) وجود دارد و با BudgetTransaction یا Tankhah مرتبط است.

گزارش فاکتورها شامل:

شماره فاکتور، مقدار، وضعیت (پرداخت‌شده، در انتظار).

پروژه یا تنخواه مرتبط.

تاریخ ثبت و تأیید.

تابع پیشنهادی:

python

def get\_invoice\_report(budget\_period, filters=None):

cache\_key = f"invoice\_report\_{budget\_period.pk}\_{hash(str(filters)) if filters else 'no\_filters'}"

cached\_result = cache.get(cache\_key)

if cached\_result is not None:

return cached\_result

queryset = Invoice.objects.filter(

transaction\_\_allocation\_\_budget\_period=budget\_period

).select\_related('transaction', 'project').values(

'id',

'invoice\_number',

'amount',

'status',

'project\_\_name',

'transaction\_\_allocation\_\_organization\_\_name',

'created\_at'

)

if filters:

queryset = apply\_filters(queryset, filters)

cache.set(cache\_key, queryset, timeout=3600)

return queryset

مزایا:

گزارش‌های جامع برای تمام سطوح (جابجایی، برگشت، تنخواه، فاکتور).

پشتیبانی از فیلترهای پویا.

بهینه‌سازی با کشینگ.

5. تمپلیت قوی

سوال: تمپلیتی قوی برای انجام جابجایی یا برگشت بودجه می‌خواهید.

پاسخ: یک تمپلیت پیشرفته طراحی می‌کنم که:

امکان انتخاب جابجایی یا برگشت بودجه را فراهم کند.

رابط کاربری ساده و پاسخ‌گو با بوت‌استرپ.

نمایش اطلاعات پروژه/شعبه مبدأ و مقصد به‌صورت پویا.

پشتیبانی از اعتبارسنجی‌های سمت کلاینت.

ویژگی‌ها:

انتخاب نوع عملیات (جابجایی یا برگشت).

انتخاب پروژه/تخصیص مبدأ و مقصد (برای جابجایی).

نمایش اطلاعات بودجه باقی‌مانده پروژه/شعبه.

فیلدهای مبلغ و توضیحات با اعتبارسنجی.

دکمه‌های تأیید و لغو.

جزئیات پیاده‌سازی: در بخش کد ارائه می‌شود.

6. تنخواه و فاکتورها

سوال: فاکتورهای ثبت‌شده در تنخواه و خود تنخواه چه می‌شوند؟ وضعیت قفل شدن برای آن‌ها اعمال می‌شود؟ تأثیر جابجایی/برگشت بودجه روی تنخواه و فاکتورها چیست؟

پاسخ:

وضعیت تنخواه:

فرض می‌کنیم مدل Tankhah وجود دارد و به BudgetAllocation یا ProjectBudgetAllocation مرتبط است.

تنخواه‌ها معمولاً بودجه‌ای هستند که برای هزینه‌های خاص (مثل هزینه‌های عملیاتی پروژه) تخصیص داده می‌شوند.

تأثیر جابجایی/برگشت بودجه:

جابجایی:

اگر بودجه از پروژه‌ای که تنخواه دارد (مثل پروژه X) جابجا شود:

ابتدا بررسی می‌شود که آیا بودجه کافی در تنخواه باقی مانده است (مثلاً remaining\_amount در Tankhah).

اگر تنخواه مصرف‌شده باشد، جابجایی فقط از بودجه باقی‌مانده پروژه امکان‌پذیر است.

تنخواه پروژه مبدأ (X) ممکن است غیرفعال شود یا بودجه‌اش کاهش یابد.

تنخواه پروژه مقصد (Z) می‌تواند بودجه جدیدی دریافت کند یا تنخواه جدیدی ایجاد شود.

ثبت در BudgetHistory: تراکنش‌های RETURN و ALLOCATION برای تنخواه ثبت می‌شوند.

برگشت:

اگر بودجه پروژه‌ای که تنخواه دارد برگشت داده شود:

بودجه تنخواه کاهش می‌یابد (allocated\_amount در Tankhah).

اگر تنخواه کاملاً مصرف شده باشد، ممکن است برگشت امکان‌پذیر نباشد مگر اینکه بخشی از بودجه آزاد باشد.

ثبت در BudgetHistory: تراکنش RETURN برای تنخواه.

وضعیت قفل شدن:

اگر BudgetPeriod, BudgetAllocation, یا ProjectBudgetAllocation قفل شوند (مثلاً به دلیل lock\_condition یا warning\_action):

تنخواه‌های مرتبط نیز قفل می‌شوند (مثلاً با تنظیم is\_active=False در Tankhah).

تراکنش‌های جدید (مثل مصرف یا برگشت) در تنخواه محدود می‌شوند.

متد پیشنهادی برای قفل کردن تنخواه:

python

def check\_tankhah\_lock\_status(self):

if self.allocation.budget\_period.is\_locked or self.allocation.is\_locked:

self.is\_active = False

self.save(update\_fields=['is\_active'])

return True, \_("تنخواه به دلیل قفل شدن تخصیص یا دوره غیرفعال شد.")

return False, \_("تنخواه فعال است.")

فاکتورها:

فرض می‌کنیم مدل Invoice وجود دارد و به BudgetTransaction یا Tankhah مرتبط است.

تأثیر جابجایی/برگشت بودجه:

جابجایی:

فاکتورهای ثبت‌شده در پروژه مبدأ (X) به همان پروژه متصل باقی می‌مانند.

اگر بودجه جابجا شود، فاکتورهای جدید در پروژه مقصد (Z) ثبت می‌شوند.

اگر تنخواه درگیر باشد، فاکتورهای تنخواه باید به تخصیص جدید متصل شوند.

برگشت:

فاکتورهای موجود تغییری نمی‌کنند، اما بودجه برگشتی نمی‌تواند شامل مبالغ فاکتورهای پرداخت‌شده باشد.

بررسی می‌شود که بودجه برگشتی از بخش آزاد بودجه پروژه باشد.

وضعیت قفل شدن:

اگر تخصیص یا دوره قفل شود، ثبت فاکتورهای جدید محدود می‌شود.

فاکتورهای موجود (مثلاً پرداخت‌شده) تحت تأثیر قرار نمی‌گیرند.

تأثیر جابجایی/برگشت جزئی:

اگر فقط بخشی از بودجه جابجا یا برگشت شود (مثلاً 50,000,000 از 300,000,000):

تنخواه: بودجه تنخواه پروژه مبدأ کاهش می‌یابد، اما فقط به اندازه‌ای که آزاد است (غیرمصرف‌شده).

فاکتورها: فاکتورهای موجود دست‌نخورده باقی می‌مانند. فاکتورهای جدید در پروژه مقصد یا با بودجه باقی‌مانده ثبت می‌شوند.

اعتبارسنجی در فرم:

بررسی می‌شود که مقدار جابجایی/برگشت از بودجه آزاد (غیرمصرف‌شده در تنخواه یا فاکتورها) باشد.

مثال: اگر تنخواه پروژه X مقدار 80,000,000 ریال مصرف کرده باشد، حداکثر 220,000,000 ریال (از 300,000,000) قابل جابجایی است.

مزایا:

مدیریت دقیق تنخواه و فاکتورها با بررسی‌های خودکار.

هماهنگی با قفل شدن تخصیص‌ها و دوره‌ها.

امکان ردیابی در گزارش‌ها.

7. تأثیر جابجایی/برگشت بودجه روی تنخواه و فاکتورها

سوال: اگر کل بودجه جابجا یا برگشت نشود، تأثیر آن روی تنخواه و فاکتورها چیست؟

پاسخ:

جابجایی جزئی:

فقط بخشی از بودجه پروژه مبدأ (مثلاً 50,000,000 از 300,000,000) جابجا می‌شود.

تأثیر روی تنخواه:

اگر تنخواه پروژه مبدأ بودجه کافی داشته باشد (مثلاً remaining\_amount بیشتر از 50,000,000 باشد)، بودجه تنخواه کاهش می‌یابد.

تنخواه پروژه مقصد می‌تواند بودجه جدید دریافت کند یا تنخواه جدیدی ایجاد شود.

ثبت در BudgetHistory برای ردیابی تغییرات تنخواه.

تأثیر روی فاکتورها:

فاکتورهای موجود پروژه مبدأ دست‌نخورده باقی می‌مانند.

فاکتورهای جدید در پروژه مقصد با بودجه جابجاشده ثبت می‌شوند.

برگشت جزئی:

بخشی از بودجه پروژه به تخصیص یا دوره برگشت داده می‌شود.

تأثیر روی تنخواه:

بودجه تنخواه کاهش می‌یابد، اما فقط به اندازه بودجه آزاد.

اگر تنخواه کاملاً مصرف شده باشد، برگشت امکان‌پذیر نیست مگر اینکه بودجه آزاد دیگری در پروژه وجود داشته باشد.

تأثیر روی فاکتورها:

فاکتورهای پرداخت‌شده تغییری نمی‌کنند.

بودجه برگشتی از بخش غیرمصرف‌شده پروژه تأمین می‌شود.

اعتبارسنجی:

فرم جابجایی/برگشت بررسی می‌کند که مقدار درخواستی از بودجه آزاد (غیرمصرف‌شده در تنخواه یا فاکتورها) باشد.

متد get\_remaining\_amount در ProjectBudgetAllocation و Tankhah برای این منظور استفاده می‌شود.

مزایا:

انعطاف‌پذیری در جابجایی/برگشت جزئی.

حفاظت از فاکتورهای پرداخت‌شده.

هماهنگی با تنخواه.

8. جمع‌بندی سناریو

جابجایی و برگشت بودجه:

امکان جابجایی بودجه بین پروژه‌ها (درون یا بین شعب) و برگشت بودجه به تخصیص/دوره وجود دارد.

فرآیند بازگشت و تخصیص مجدد استفاده می‌شود.

ثبت تراکنش‌ها:

تمام تغییرات به‌صورت خودکار در ProjectBudgetAllocation, BudgetAllocation, و BudgetPeriod ثبت می‌شوند.

تاریخچه در BudgetHistory با transfer\_id مشترک.

اعلان‌ها:

اعلان‌های خودکار برای بازگشت، تخصیص، و تغییرات وضعیت.

گیرندگان: مدیران مالی، مدیران بودجه، و کاربران مرتبط.

گزارش‌گیری:

گزارش‌های جامع برای جابجایی‌ها، برگشت‌ها، تنخواه‌ها، و فاکتورها.

پشتیبانی از فیلترهای پویا و کشینگ.

تنخواه و فاکتورها:

جابجایی/برگشت بودجه فقط روی بودجه آزاد تأثیر می‌گذارد.

تنخواه‌ها و فاکتورها با قفل شدن تخصیص/دوره هماهنگ هستند.

گزارش‌های اختصاصی برای تنخواه و فاکتورها.

تمپلیت:

رابط کاربری پیشرفته برای انتخاب جابجایی یا برگشت.

نمایش اطلاعات پویا و اعتبارسنجی سمت کلاینت.

قفل شدن:

قفل شدن تخصیص‌ها و دوره‌ها روی تنخواه و فاکتورها اعمال می‌شود.

محدودیت ثبت تراکنش‌های جدید در حالت قفل.

تأییدیه: لطفاً تأیید کنید که این سناریو با نیازهای شما مطابقت دارد. اگر نکته‌ای نیاز به تغییر یا اضافه شدن دارد (مثلاً جزئیات بیشتر در گزارش‌ها، قابلیت خاص در تمپلیت، یا مدیریت متفاوت تنخواه/فاکتورها)، اطلاع دهید.

کد پیشنهادی (پس از تأیید)

با تأیید شما، کدهای زیر را ارائه می‌کنم:

مهاجرت‌ها: برای اطمینان از وجود returned\_amount در ProjectBudgetAllocation.

فرم‌ها:

BudgetTransferForm برای جابجایی بودجه.

BudgetReturnForm برای برگشت بودجه (به‌روزرسانی‌شده از پاسخ قبلی).

ویوها:

BudgetTransferView برای جابجایی.

BudgetReturnView برای برگشت.

تمپلیت:

یک تمپلیت قوی و یکپارچه که هر دو عملیات (جابجایی و برگشت) را پشتیبانی کند.

گزارش‌گیری:

توابع get\_budget\_transfers, get\_returned\_budgets, get\_tankhah\_report, get\_invoice\_report.

سیگنال‌ها:

برای مدیریت کش و یکپارچگی داده‌ها.

مدل‌های تنخواه و فاکتور (در صورت نیاز):

اگر مدل‌های Tankhah و Invoice وجود ندارند، ساختار پیشنهادی ارائه می‌شود.

سوالات برای تأیید:

آیا سناریوی پیشنهادی (جابجایی و برگشت با رویکرد بازگشت و تخصیص مجدد) مناسب است؟

آیا نیاز به قابلیت خاصی در تمپلیت (مثل نمودار، فیلترهای پیشرفته، یا پیش‌نمایش) دارید؟

آیا مدل‌های Tankhah و Invoice وجود دارند؟ اگر خیر، آیا باید ساختار پیشنهادی ارائه کنم؟

آیا فیلتر یا جزئیات خاصی برای گزارش‌های تنخواه و فاکتورها نیاز دارید؟

آیا قفل شدن تنخواه/فاکتورها باید رفتار متفاوتی داشته باشد؟

لطفاً پاسخ دهید تا کد نهایی را دقیقاً بر اساس نیازهای شما بنویسم.