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 BudgetPeriodرا به یک BudgetItemخاص 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_amountشما استفاده می‌شود.(

project: (اختیاری در این مدل، اما در ProjectBudgetAllocationکلیدی 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 SubProjectخاص 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 (مثلاًIT ، سفر، آموزش.(...

تخصیص بودجه به ردیف/سازمان :مدیر بودجه، بخشی از total_amountدوره کلان را به یک BudgetItem خاص برای یک Organizationخاص تخصیص می‌دهد و یک رکورد BudgetAllocationایجاد می‌کند. مبلغ این تخصیص در allocated_amountثبت 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 BudgetAllocationتأمین می‌شود. یک رکورد ProjectBudgetAllocationایجاد می‌شود که BudgetAllocationمربوط به "تجهیزات IT دفتر مرکزی" را به "پروژه اتاق سرور" لینک می‌کند.

نکته :اگر ProjectBudgetAllocationشما فیلد allocated_amountدارد، در این مرحله مشخص می‌شود که از ۵ میلیارد تخصیصIT ، چقدر (مثلاً ۲ میلیارد) برای پروژه اتاق سرور است. اگر ندارد، یعنی کل باقیمانده آن BudgetAllocationبرای این پروژه قابل استفاده است (تا زمانی که برای پروژه دیگری لینک نشود) .کد ویوی شما نشان می‌دهد که شما از allocated_amountدر ProjectBudgetAllocationاستفاده می‌کنید.

ایجاد تنخواه (احتمالاً) :ممکن است قبل از ایجاد فاکتور، یک Tankhahبرای مجری پروژه) مثلاً مدیر (IT با مبلغ مشخصی از بودجه پروژه ایجاد شود. این خود می‌تواند یک نوع BudgetTransactionاز نوع CONSUMPTION)یا نوع خاص (TANKHAH_ALLOCATIONروی BudgetAllocationمربوطه ثبت کند و از باقیمانده آن کم کند. مبلغ Tankhahسقف 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 Tankhahمرتبط با آن هزینه را انتخاب می‌کند.

اتصال به پروژه و تخصیص :از طریق Tankhah، سیستم به Projectو سپس به ProjectBudgetAllocation و نهایتاً به BudgetAllocationاصلی 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_transactionوقتی کاربر روی "ذخیره نهایی پیش‌نویس" کلیک می‌کند و اعتبارسنجی‌های اولیه موفقیت‌آمیز است، تابع create_budget_transactionفراخوانی می‌شود. این تابع:

BudgetAllocationاصلی مرتبط با پروژه/تنخواه را پیدا می‌کند.

باقیمانده واقعی آن BudgetAllocationرا 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 BudgetTransactionجدید با 'transaction_type='CONSUMPTION و مبلغ فاکتور ایجاد می‌کند که به آن BudgetAllocationلینک 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_budgetبا در نظر گرفتن وضعیت فاکتورها محاسبه می‌شود.(

سطح تخصیص کلان :با ثبت BudgetTransactionمصرف، باقیمانده BudgetAllocationاصلی نیز کاهش می‌یابد (که در متد get_remaining_amountمدل BudgetAllocationمحاسبه 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)

.1تعریف 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 ۱۰۰,۰۰۰,۰۰۰ ریال مصرف کرده است (CONSUMPTION).

جابجایی بودجه:

مدیر تصمیم می‌گیرد ۵۰,۰۰۰,۰۰۰ 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؟

.2سازوکار 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 ۵۰,۰۰۰,۰۰۰ 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 ۵۰,۰۰۰,۰۰۰ 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 ۵۰,۰۰۰,۰۰۰ 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.

.3تصمیم در مورد فیلدreturned_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.

.4کد 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 ۱: 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} "
                            f"{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 ۲: 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 %}

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

{% block content %}

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

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

<form method="post">

{% csrf_token %}

<label for="{{ form.source_allocation.id_for_label }}" class="form-label">

{% trans "تخصیص مبدأ" %}

</label>

{{ form.source_allocation }}

{% if form.source_allocation.errors %}

{{ form.source_allocation.errors }}

{% endif %}

<label for="{{ form.destination_allocation.id_for_label }}" class="form-label">

{% trans "تخصیص مقصد" %}

</label>

{{ form.destination_allocation }}

{% if form.destination_allocation.errors %}

{{ form.destination_allocation.errors }}

{% endif %}

<label for="{{ form.amount.id_for_label }}" class="form-label">

{% trans "مبلغ جابجایی" %}

</label>

{{ form.amount }}

{% if form.amount.errors %}

{{ form.amount.errors }}

```
{% 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_fil
ters")
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 = ...  سازمانA
# self.organization_b = ...  سازمانB
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 = ...  # پروژهX
        self.project_z = ...  # پروژهZ
        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:

۱. **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.
۲. **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 تخصیصت یافته و مبلغ برگشتی در `BudgetAllocation` و مبلغ برگشتی در `BudgetPeriod` و کاهش مبلغ تخصیص یافته کل در `BudgetPeriod`) انجام شود.
۳. **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.
٤. **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.

۵. **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**

۶. **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.

۷. **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

جابجایی بودجه و برگشت بودجه .1

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:

).BudgetPeriodبازگشت بودجه از پروژه مبدأ به تخصیص یا دوره بودجه (

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:

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

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

مشترک ثبت می‌شوند تا قابل ردیابی باشند.transfer_id با یک BudgetHistoryهر دو تراکنش در

برگشت:

ثبت می‌شود.RETURNفقط تراکنش

ثبت می‌شود.returned_amount برگردانده می‌شود و در BudgetAllocation یا BudgetPeriodبودجه به

مزایا:

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

امکان جابجایی جزئی یا کامل بودجه (مثلاً ۵۰,۰۰۰,۰۰۰ از ۳۰۰,۰۰۰,۰۰۰ ریال).

.BudgetHistoryردیابی دقیق با

ثبت خودکار تراکنش‌ها در مدل‌های بالاتر .2

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

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

سازوکار:

برای جابجایی بودجه:

مرحله بازگشت:

کاهش می‌یابد.allocated_amount افزایش می‌یابد و returned_amount (پروژه مبدأ): ProjectBudgetAllocationدر

کاهش می‌یابد.allocated_amount افزایش می‌یابد و returned_amount (شعبه مبدأ): BudgetAllocationدر

کاهش می‌یابد.total_allocated افزایش می‌یابد و returned_amount: BudgetPeriodدر

مرحله تخصیص مجدد:

افزایش می‌یابد.allocated_amount (پروژه مقصد): ProjectBudgetAllocationدر

افزایش می‌یابد.allocated_amount (شعبه مقصد): BudgetAllocationدر

افزایش می‌یابد.total_allocated: BudgetPeriodدر

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

برای برگشت بودجه:

, ProjectBudgetAllocation در allocated_amount و کاهش returned_amountفقط مرحله بازگشت انجام می‌شود (به‌روزرسانی).BudgetPeriod, و BudgetAllocation

ثبت تاریخچه:

, object_id ,content_type ثبت می‌شود با جزئیات کامل (مثل BudgetHistoryهر تغییر (بازگشت یا تخصیص) به‌صورت خودکار در).created_by, details ,amount

مزایا:

یکپارچگی داده‌ها با تراکنش‌های اتمیک.

ردیابی خودکار تمام تغییرات در سطوح مختلف.

اعلان‌ها .3

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:

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

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

برای برگشت بودجه:

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

اعلان‌های وضعیت:

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

گیرندگان اعلان:

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

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

محتوای اعلان:

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

جابجا شد.»Z به پروژه Xمثال: «مبلغ ۵۰,۰۰۰,۰۰۰ ریال از پروژه

پیاده‌سازی:

استفاده می‌شود.BudgetPeriod و BudgetAllocation در مدل‌های send_notificationاز متد

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:

تابعget_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:

تابعget_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:

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

گزارش تنخواه شامل:

).PENDING, APPROVED, PAIDوضعیت تنخواه (مثل

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:

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

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.

تمپلیت قوی .5

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.

تنخواه و فاکتورها .6

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:

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

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

تأثیر جابجایی/برگشت بودجه:

جابجایی:

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

).Tankhah در remaining_amountابتدا بررسی می‌شود که آیا بودجه کافی در تنخواه باقی مانده است (مثلاً

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

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

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

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

برگشت:

اگر بودجه پروژه‌ای که تنخواه دارد برگشت داده شود:

).Tankhah در allocated_amountبودجه تنخواه کاهش می‌یابد (

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

برای تنخواه.RETURN: تراکنش BudgetHistoryثبت در

وضعیت قفل شدن:

یا lock_condition قفل شوند (مثلاً به دلیل ProjectBudgetAllocation, یا BudgetAllocation, BudgetPeriodاگر
):warning_action

).Tankhah در is_active=Falseتنخواه‌های مرتبط نیز قفل می‌شوند (مثلاً با تنظیم

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'])

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

(".تنخواه فعال است"_ ,return False

فاکتورها:

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

تأثیر جابجایی/برگشت بودجه:

جابجایی:

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

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

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 ۵۰,۰۰۰,۰۰۰ از ۳۰۰,۰۰۰,۰۰۰):

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.

مقدار ۸۰,۰۰۰,۰۰۰ ریال مصرف کرده باشد، حداکثر ۲۲۰,۰۰۰,۰۰۰ ریال (از ۳۰۰,۰۰۰,۰۰۰) قابل جابجایی است.Xمثال: اگر تنخواه پروژه

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 ۵۰,۰۰۰,۰۰۰ از ۳۰۰,۰۰۰,۰۰۰) جابجا می‌شود.

تأثیر روی تنخواه:

اگر تنخواه پروژه مبدأ بودجه کافی داشته باشد (مثلاً remaining_amount بیشتر از ۵۰,۰۰۰,۰۰۰ باشد)، بودجه تنخواه کاهش می‌یابد.

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

برای ردیابی تغییرات تنخواه، BudgetHistory ثبت در

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.

برای این منظور استفاده می‌شود. Tankhah و ProjectBudgetAllocation در get_remaining_amount متد

مزایا:

انعطاف‌پذیری در جابجایی/برگشت جزئی.

حفاظت از فاکتورهای پرداخت‌شده.

هماهنگی با تنخواه.

جمع‌بندی سناریو .8

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:

ثبت می‌شوند.BudgetPeriod, و BudgetAllocation, ProjectBudgetAllocationتمام تغییرات به‌صورت خودکار در

مشترک.transfer_id با BudgetHistoryتاریخچه در

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:

برای جابجایی بودجه.BudgetTransferForm

برای برگشت بودجه (به‌روزرسانی‌شده از پاسخ قبلی).BudgetReturnForm

ویوها:

برای جابجایی.BudgetTransferView

برای برگشت.BudgetReturnView

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.