Phase 6: Advanced Modules - TWIST ERP

Implementation Guide

Duration: 12–16 weeks
Version: 1.0
Date: October 2025
Project: TWIST ERP - Visual Drag-and-Drop Multi-Company ERP

1. Phase Overview

Phase 6 implements advanced business modules that complete the TWIST ERP ecosystem: Asset Management, Cost Centers & Budgeting, HR & Payroll, and Project Management. These modules build on the foundation from previous phases and provide comprehensive business management capabilities.

Modules Covered

1. **Asset Management** - Fixed assets, depreciation, maintenance
2. **Cost Centers & Budgeting** - Budget planning, monitoring, controls
3. **Human Resources & Payroll** - Employee management, attendance, payroll
4. **Project Management** - Project planning, tracking, resource allocation

Key Objectives

- Complete asset lifecycle management
- Implement multi-level budget hierarchy
- Automate payroll processing
- Enable project-based costing
- Integrate all modules with Finance
- Support multi-company consolidation

2. Asset Management Module

2.1 Data Models

```
# backend/apps/assets/models/asset.py
from django.db import models
from shared.models import CompanyAwareModel
from decimal import Decimal
```

```
class AssetCategory(CompanyAwareModel):
    """Asset categorization"""
    code = models.CharField(max_length=20)
    name = models.CharField(max_length=255)
    parent_category = models.ForeignKey(
        'self',
        on_delete=models.PROTECT,
        null=True,
        blank=True
    )

    # Depreciation defaults
    default_depreciation_method = models.CharField(
        max_length=20,
        choices=[
            ('STRAIGHT', 'Straight Line'),
            ('DECLINING', 'Declining Balance'),
            ('DOUBLE_DECLINING', 'Double Declining'),
            ('UNITS', 'Units of Production'),
        ],
        default='STRAIGHT'
    )
    default_useful_life = models.IntegerField(
        help_text="Default useful life in months"
    )

    # GL Accounts
    asset_account = models.ForeignKey(
        'finance.Account',
        on_delete=models.PROTECT,
        related_name='asset_categories'
    )
    accumulated_depreciation_account = models.ForeignKey(
        'finance.Account',
        on_delete=models.PROTECT,
        related_name='depreciation_categories'
    )
    depreciation_expense_account = models.ForeignKey(
        'finance.Account',
        on_delete=models.PROTECT,
        related_name='expense_categories'
    )

    is_active = models.BooleanField(default=True)

    class Meta:
        unique_together = [['company', 'code']]
        verbose_name_plural = 'Asset Categories'

class Asset(CompanyAwareModel):
    """Fixed asset register"""
    asset_number = models.CharField(max_length=50)
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True)

    category = models.ForeignKey(
```

```
        AssetCategory,
        on_delete=models.PROTECT
    )

    # Acquisition
    acquisition_date = models.DateField()
    acquisition_cost = models.DecimalField(max_digits=20, decimal_places=2)
    supplier = models.ForeignKey(
        'procurement.Supplier',
        on_delete=models.PROTECT,
        null=True,
        blank=True
    )
    purchase_order = models.ForeignKey(
        'procurement.PurchaseOrder',
        on_delete=models.SET_NULL,
        null=True,
        blank=True
    )

    # Location
    location = models.ForeignKey(
        'inventory.Warehouse',
        on_delete=models.PROTECT,
        related_name='assets'
    )
    custodian = models.ForeignKey(
        'users.User',
        on_delete=models.PROTECT,
        related_name='assigned_assets'
    )

    # Depreciation
    depreciation_method = models.CharField(max_length=20)
    useful_life_months = models.IntegerField()
    salvage_value = models.DecimalField(
        max_digits=20,
        decimal_places=2,
        default=0
    )

    # Calculated values
    accumulated_depreciation = models.DecimalField(
        max_digits=20,
        decimal_places=2,
        default=0
    )
    book_value = models.DecimalField(
        max_digits=20,
        decimal_places=2,
        default=0
    )

    # Status
    status = models.CharField(
        max_length=20,
```

```
        choices=[
            ('ACTIVE', 'Active'),
            ('MAINTENANCE', 'Under Maintenance'),
            ('DISPOSED', 'Disposed'),
            ('SOLD', 'Sold'),
        ],
        default='ACTIVE'
    )

    # Barcode/Tag
    barcode = models.CharField(max_length=100, blank=True)

    # Disposal
    disposal_date = models.DateField(null=True, blank=True)
    disposal_value = models.DecimalField(
        max_digits=20,
        decimal_places=2,
        null=True,
        blank=True
    )

    class Meta:
        unique_together = [['company', 'asset_number']]
        ordering = ['asset_number']

    def calculate_depreciation(self, as_of_date=None):
        """Calculate depreciation amount"""
        from datetime import datetime
        from dateutil.relativedelta import relativedelta

        if not as_of_date:
            as_of_date = datetime.now().date()

        # Months since acquisition
        months_used = relativedelta(as_of_date, self.acquisition_date).months
        months_used += relativedelta(as_of_date, self.acquisition_date).years * 12

        if months_used &lt;= 0:
            return Decimal('0')

        depreciable_amount = self.acquisition_cost - self.salvage_value

        if self.depreciation_method == 'STRAIGHT':
            # Straight line
            monthly_depreciation = depreciable_amount / self.useful_life_months
            total_depreciation = monthly_depreciation * min(months_used, self.useful_life

        elif self.depreciation_method == 'DECLINING':
            # Declining balance (20% per year example)
            rate = Decimal('0.20') / 12
            remaining_value = self.acquisition_cost
            total_depreciation = Decimal('0')

            for _ in range(min(months_used, self.useful_life_months)):
                month_dep = remaining_value * rate
                total_depreciation += month_dep
```

```
                remaining_value -= month_dep

        else:
            total_depreciation = Decimal('0')

        return min(total_depreciation, depreciable_amount)

    def update_book_value(self):
        """Update accumulated depreciation and book value"""
        self.accumulated_depreciation = self.calculate_depreciation()
        self.book_value = self.acquisition_cost - self.accumulated_depreciation
        self.save()

class AssetMaintenance(models.Model):
    """Asset maintenance tracking"""
    asset = models.ForeignKey(
        Asset,
        on_delete=models.CASCADE,
        related_name='maintenance_records'
    )

    maintenance_type = models.CharField(
        max_length=20,
        choices=[
            ('PREVENTIVE', 'Preventive'),
            ('CORRECTIVE', 'Corrective'),
            ('INSPECTION', 'Inspection'),
        ]
    )

    scheduled_date = models.DateField()
    completed_date = models.DateField(null=True, blank=True)

    description = models.TextField()
    cost = models.DecimalField(max_digits=20, decimal_places=2, default=0)

    performed_by = models.CharField(max_length=255, blank=True)

    status = models.CharField(
        max_length=20,
        choices=[
            ('SCHEDULED', 'Scheduled'),
            ('IN_PROGRESS', 'In Progress'),
            ('COMPLETED', 'Completed'),
            ('CANCELLED', 'Cancelled'),
        ],
        default='SCHEDULED'
    )

    class Meta:
        ordering = ['-scheduled_date']
```

3. Cost Centers & Budgeting Module

3.1 Data Models

```
# backend/apps/budgeting/models/cost_center.py
from django.db import models
from shared.models import CompanyAwareModel

class CostCenter(CompanyAwareModel):
    """
    Cost center hierarchy
    """
    code = models.CharField(max_length=20)
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True)

    # Hierarchy
    parent_cost_center = models.ForeignKey(
        'self',
        on_delete=models.PROTECT,
        null=True,
        blank=True,
        related_name='sub_centers'
    )

    # Manager
    manager = models.ForeignKey(
        'users.User',
        on_delete=models.PROTECT,
        related_name='managed_cost_centers'
    )

    # Type
    center_type = models.CharField(
        max_length=20,
        choices=[
            ('DEPARTMENT', 'Department'),
            ('PROJECT', 'Project'),
            ('LOCATION', 'Location'),
            ('PRODUCT', 'Product Line'),
        ]
    )

    is_active = models.BooleanField(default=True)

    class Meta:
        unique_together = [['company', 'code']]
        ordering = ['code']

class Budget(CompanyAwareModel):
    """
    Budget master
    """
    name = models.CharField(max_length=255)
    fiscal_year = models.CharField(max_length=4)
```

```
    budget_type = models.CharField(
        max_length=20,
        choices=[
            ('OPERATIONAL', 'Operational Budget'),
            ('CAPEX', 'Capital Expenditure'),
            ('REVENUE', 'Revenue Budget'),
        ]
    )

    # Period
    start_date = models.DateField()
    end_date = models.DateField()

    # Status
    status = models.CharField(
        max_length=20,
        choices=[
            ('DRAFT', 'Draft'),
            ('SUBMITTED', 'Submitted'),
            ('APPROVED', 'Approved'),
            ('ACTIVE', 'Active'),
            ('CLOSED', 'Closed'),
        ],
        default='DRAFT'
    )

    # Approval
    approved_by = models.ForeignKey(
        'users.User',
        on_delete=models.SET_NULL,
        null=True,
        blank=True
    )
    approved_at = models.DateTimeField(null=True, blank=True)

    class Meta:
        ordering = ['-fiscal_year', 'name']

class BudgetLine(models.Model):
    """
    Budget line item
    """
    budget = models.ForeignKey(
        Budget,
        on_delete=models.CASCADE,
        related_name='lines'
    )

    cost_center = models.ForeignKey(
        CostCenter,
        on_delete=models.PROTECT
    )
    account = models.ForeignKey(
        'finance.Account',
        on_delete=models.PROTECT
```

```
    )

    # Amounts
    budgeted_amount = models.DecimalField(max_digits=20, decimal_places=2)
    committed_amount = models.DecimalField(
        max_digits=20,
        decimal_places=2,
        default=0,
        help_text="PO commitments"
    )
    actual_amount = models.DecimalField(
        max_digits=20,
        decimal_places=2,
        default=0,
        help_text="Actual spend"
    )

    # Monthly breakdown
    monthly_breakdown = models.JSONField(
        default=dict,
        help_text="Month-by-month budget"
    )

    # Variance tracking
    variance_amount = models.DecimalField(
        max_digits=20,
        decimal_places=2,
        default=0
    )
    variance_percent = models.DecimalField(
        max_digits=5,
        decimal_places=2,
        default=0
    )

    notes = models.TextField(blank=True)

    class Meta:
        unique_together = [['budget', 'cost_center', 'account']]

    @property
    def available_amount(self):
        """Calculate available budget"""
        return self.budgeted_amount - self.committed_amount - self.actual_amount

    def check_budget_available(self, amount):
        """Check if budget available for amount"""
        return self.available_amount &gt;= amount

    def update_variance(self):
        """Calculate budget variance"""
        self.variance_amount = self.budgeted_amount - self.actual_amount
        if self.budgeted_amount &gt; 0:
            self.variance_percent = (
                self.variance_amount / self.budgeted_amount
            ) * 100
```

```
        self.save()

class BudgetAlert(models.Model):
    """
    Budget threshold alerts
    """
    budget_line = models.ForeignKey(
        BudgetLine,
        on_delete=models.CASCADE,
        related_name='alerts'
    )

    alert_type = models.CharField(
        max_length=20,
        choices=[
            ('THRESHOLD', 'Threshold Reached'),
            ('EXCEEDED', 'Budget Exceeded'),
            ('FORECAST', 'Forecast Overrun'),
        ]
    )

    threshold_percent = models.IntegerField()
    message = models.TextField()

    notify_users = models.ManyToManyField('users.User')

    triggered_at = models.DateTimeField(auto_now_add=True)
    is_resolved = models.BooleanField(default=False)

    class Meta:
        ordering = ['-triggered_at']
```

4. HR & Payroll Module

4.1 Data Models

```
# backend/apps/hr/models/employee.py
from django.db import models
from shared.models import CompanyAwareModel

class Department(CompanyAwareModel):
    """Department master"""
    code = models.CharField(max_length=20)
    name = models.CharField(max_length=255)
    manager = models.ForeignKey(
        'Employee',
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='managed_departments'
    )

    class Meta:
```

```
        unique_together = [['company', 'code']]

class Employee(CompanyAwareModel):
    """Employee master"""
    employee_number = models.CharField(max_length=50)

    # Personal
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    date_of_birth = models.DateField()
    gender = models.CharField(
        max_length=10,
        choices=[
            ('MALE', 'Male'),
            ('FEMALE', 'Female'),
            ('OTHER', 'Other'),
        ]
    )

    # Contact
    email = models.EmailField()
    phone = models.CharField(max_length=20)
    address = models.TextField()

    # Employment
    date_of_joining = models.DateField()
    date_of_leaving = models.DateField(null=True, blank=True)

    department = models.ForeignKey(
        Department,
        on_delete=models.PROTECT
    )
    designation = models.CharField(max_length=100)

    # Reporting
    reports_to = models.ForeignKey(
        'self',
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name='subordinates'
    )

    # Salary
    basic_salary = models.DecimalField(max_digits=20, decimal_places=2)

    # Status
    employment_status = models.CharField(
        max_length=20,
        choices=[
            ('ACTIVE', 'Active'),
            ('ON_LEAVE', 'On Leave'),
            ('RESIGNED', 'Resigned'),
            ('TERMINATED', 'Terminated'),
        ],
        default='ACTIVE'
```

```
    )

    # User account
    user = models.OneToOneField(
        'users.User',
        on_delete=models.SET_NULL,
        null=True,
        blank=True
    )

    class Meta:
        unique_together = [['company', 'employee_number']]
        ordering = ['employee_number']

    def get_full_name(self):
        return f"{self.first_name} {self.last_name}"

class Attendance(models.Model):
    """Daily attendance"""
    employee = models.ForeignKey(
        Employee,
        on_delete=models.CASCADE,
        related_name='attendance_records'
    )

    date = models.DateField()

    check_in = models.TimeField(null=True, blank=True)
    check_out = models.TimeField(null=True, blank=True)

    status = models.CharField(
        max_length=20,
        choices=[
            ('PRESENT', 'Present'),
            ('ABSENT', 'Absent'),
            ('HALF_DAY', 'Half Day'),
            ('LEAVE', 'On Leave'),
            ('HOLIDAY', 'Holiday'),
        ]
    )

    hours_worked = models.DecimalField(
        max_digits=5,
        decimal_places=2,
        default=0
    )

    remarks = models.TextField(blank=True)

    class Meta:
        unique_together = [['employee', 'date']]
        ordering = ['-date']

class LeaveType(CompanyAwareModel):
    """Leave types"""
    code = models.CharField(max_length=20)
```

```
    name = models.CharField(max_length=100)
    annual_allocation = models.IntegerField(
        help_text="Days per year"
    )
    is_paid = models.BooleanField(default=True)

    class Meta:
        unique_together = [['company', 'code']]

class LeaveApplication(models.Model):
    """Leave requests"""
    employee = models.ForeignKey(
        Employee,
        on_delete=models.CASCADE,
        related_name='leave_applications'
    )

    leave_type = models.ForeignKey(
        LeaveType,
        on_delete=models.PROTECT
    )

    from_date = models.DateField()
    to_date = models.DateField()
    days = models.IntegerField()

    reason = models.TextField()

    status = models.CharField(
        max_length=20,
        choices=[
            ('PENDING', 'Pending'),
            ('APPROVED', 'Approved'),
            ('REJECTED', 'Rejected'),
            ('CANCELLED', 'Cancelled'),
        ],
        default='PENDING'
    )

    approved_by = models.ForeignKey(
        'users.User',
        on_delete=models.SET_NULL,
        null=True,
        blank=True
    )

    applied_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-applied_at']

class PayrollComponent(CompanyAwareModel):
    """Salary components (earnings/deductions)"""
    code = models.CharField(max_length=20)
    name = models.CharField(max_length=100)
```

```
    component_type = models.CharField(
        max_length=20,
        choices=[
            ('EARNING', 'Earning'),
            ('DEDUCTION', 'Deduction'),
        ]
    )

    # Calculation
    calculation_type = models.CharField(
        max_length=20,
        choices=[
            ('FIXED', 'Fixed Amount'),
            ('PERCENTAGE', 'Percentage of Basic'),
            ('FORMULA', 'Custom Formula'),
        ]
    )

    amount = models.DecimalField(
        max_digits=20,
        decimal_places=2,
        null=True,
        blank=True
    )
    percentage = models.DecimalField(
        max_digits=5,
        decimal_places=2,
        null=True,
        blank=True
    )
    formula = models.TextField(blank=True)

    # GL Account
    account = models.ForeignKey(
        'finance.Account',
        on_delete=models.PROTECT
    )

    is_taxable = models.BooleanField(default=True)
    is_active = models.BooleanField(default=True)

    class Meta:
        unique_together = [['company', 'code']]

class Payroll(CompanyAwareModel):
    """Monthly payroll"""
    month = models.CharField(max_length=7)  # YYYY-MM
    payroll_date = models.DateField()

    status = models.CharField(
        max_length=20,
        choices=[
            ('DRAFT', 'Draft'),
            ('CALCULATED', 'Calculated'),
            ('APPROVED', 'Approved'),
            ('PAID', 'Paid'),
```

```
        ],
        default='DRAFT'
    )

    total_gross = models.DecimalField(max_digits=20, decimal_places=2, default=0)
    total_deductions = models.DecimalField(max_digits=20, decimal_places=2, default=0)
    total_net = models.DecimalField(max_digits=20, decimal_places=2, default=0)

    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        unique_together = [['company', 'month']]
        ordering = ['-month']

class PayrollEntry(models.Model):
    """Individual employee payroll"""
    payroll = models.ForeignKey(
        Payroll,
        on_delete=models.CASCADE,
        related_name='entries'
    )
    employee = models.ForeignKey(
        Employee,
        on_delete=models.PROTECT
    )

    # Attendance
    days_worked = models.IntegerField()

    # Amounts
    basic_salary = models.DecimalField(max_digits=20, decimal_places=2)
    total_earnings = models.DecimalField(max_digits=20, decimal_places=2)
    total_deductions = models.DecimalField(max_digits=20, decimal_places=2)
    net_salary = models.DecimalField(max_digits=20, decimal_places=2)

    # Component breakdown
    earnings_breakdown = models.JSONField(default=dict)
    deductions_breakdown = models.JSONField(default=dict)

    # Payment
    payment_mode = models.CharField(
        max_length=20,
        choices=[
            ('BANK', 'Bank Transfer'),
            ('CASH', 'Cash'),
            ('CHEQUE', 'Cheque'),
        ],
        default='BANK'
    )

    payment_status = models.CharField(
        max_length=20,
        choices=[
            ('PENDING', 'Pending'),
            ('PAID', 'Paid'),
        ],
```

```
        default='PENDING'
    )

    class Meta:
        unique_together = [['payroll', 'employee']]
```

5. Project Management Module

5.1 Data Models

```
# backend/apps/projects/models/project.py
from django.db import models
from shared.models import CompanyAwareModel

class Project(CompanyAwareModel):
    """Project master"""
    project_number = models.CharField(max_length=50)
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True)

    # Timeline
    start_date = models.DateField()
    end_date = models.DateField()

    # Customer/Client
    customer = models.ForeignKey(
        'sales.Customer',
        on_delete=models.PROTECT,
        null=True,
        blank=True
    )

    # Manager
    project_manager = models.ForeignKey(
        'hr.Employee',
        on_delete=models.PROTECT,
        related_name='managed_projects'
    )

    # Budget
    total_budget = models.DecimalField(max_digits=20, decimal_places=2)
    budget_consumed = models.DecimalField(max_digits=20, decimal_places=2, default=0)

    # Status
    status = models.CharField(
        max_length=20,
        choices=[
            ('PLANNING', 'Planning'),
            ('ACTIVE', 'Active'),
            ('ON_HOLD', 'On Hold'),
            ('COMPLETED', 'Completed'),
            ('CANCELLED', 'Cancelled'),
        ],
```

```
        default='PLANNING'
    )

    # Progress
    progress_percent = models.IntegerField(default=0)

    class Meta:
        unique_together = [['company', 'project_number']]
        ordering = ['-start_date']

class Task(models.Model):
    """Project task"""
    project = models.ForeignKey(
        Project,
        on_delete=models.CASCADE,
        related_name='tasks'
    )

    task_number = models.CharField(max_length=50)
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True)

    # Dependencies
    depends_on = models.ManyToManyField(
        'self',
        symmetrical=False,
        blank=True,
        related_name='dependent_tasks'
    )

    # Assignment
    assigned_to = models.ForeignKey(
        'hr.Employee',
        on_delete=models.PROTECT,
        related_name='assigned_tasks'
    )

    # Timeline
    start_date = models.DateField()
    due_date = models.DateField()
    completed_date = models.DateField(null=True, blank=True)

    # Effort
    estimated_hours = models.DecimalField(max_digits=10, decimal_places=2)
    actual_hours = models.DecimalField(max_digits=10, decimal_places=2, default=0)

    # Status
    status = models.CharField(
        max_length=20,
        choices=[
            ('TODO', 'To Do'),
            ('IN_PROGRESS', 'In Progress'),
            ('REVIEW', 'In Review'),
            ('COMPLETED', 'Completed'),
            ('BLOCKED', 'Blocked'),
        ],
```

```
        default='TODO'
    )

    # Priority
    priority = models.CharField(
        max_length=20,
        choices=[
            ('LOW', 'Low'),
            ('MEDIUM', 'Medium'),
            ('HIGH', 'High'),
            ('URGENT', 'Urgent'),
        ],
        default='MEDIUM'
    )

    progress_percent = models.IntegerField(default=0)

    class Meta:
        ordering = ['project', 'due_date']

class TimeSheet(models.Model):
    """Time tracking"""
    employee = models.ForeignKey(
        'hr.Employee',
        on_delete=models.CASCADE,
        related_name='timesheets'
    )

    project = models.ForeignKey(
        Project,
        on_delete=models.PROTECT,
        null=True,
        blank=True
    )
    task = models.ForeignKey(
        Task,
        on_delete=models.PROTECT,
        null=True,
        blank=True
    )

    date = models.DateField()
    hours = models.DecimalField(max_digits=5, decimal_places=2)

    description = models.TextField()

    # Billing
    is_billable = models.BooleanField(default=True)
    hourly_rate = models.DecimalField(
        max_digits=10,
        decimal_places=2,
        null=True,
        blank=True
    )

    status = models.CharField(
```

```
        max_length=20,
        choices=[
            ('DRAFT', 'Draft'),
            ('SUBMITTED', 'Submitted'),
            ('APPROVED', 'Approved'),
            ('REJECTED', 'Rejected'),
        ],
        default='DRAFT'
    )

    class Meta:
        ordering = ['-date']

class ProjectExpense(models.Model):
    """Project expenses"""
    project = models.ForeignKey(
        Project,
        on_delete=models.PROTECT,
        related_name='expenses'
    )

    expense_date = models.DateField()
    amount = models.DecimalField(max_digits=20, decimal_places=2)

    expense_type = models.CharField(max_length=100)
    description = models.TextField()

    # Link to finance
    invoice = models.ForeignKey(
        'finance.Invoice',
        on_delete=models.SET_NULL,
        null=True,
        blank=True
    )

    incurred_by = models.ForeignKey(
        'hr.Employee',
        on_delete=models.PROTECT
    )

    status = models.CharField(
        max_length=20,
        choices=[
            ('SUBMITTED', 'Submitted'),
            ('APPROVED', 'Approved'),
            ('PAID', 'Paid'),
            ('REJECTED', 'Rejected'),
        ],
        default='SUBMITTED'
    )

    class Meta:
        ordering = ['-expense_date']
```

6. Integration Services

6.1 Budget Control Service

```
# backend/apps/budgeting/services/budget_control.py
from apps.budgeting.models import BudgetLine, BudgetAlert
from apps.ai_companion.models import AIAlert

class BudgetControlService:
    """
    Budget checking and enforcement
    """

    def check_budget_availability(
        self,
        company,
        cost_center,
        account,
        amount,
        fiscal_year=None
    ):
        """
        Check if budget available for transaction
        """
        # Find active budget
        budget_line = BudgetLine.objects.filter(
            budget__company=company,
            budget__status='ACTIVE',
            cost_center=cost_center,
            account=account
        ).first()

        if not budget_line:
            return {
                'allowed': True,
                'message': 'No budget control configured'
            }

        # Check availability
        available = budget_line.available_amount

        if available &gt;= amount:
            return {
                'allowed': True,
                'available': float(available),
                'message': 'Budget available'
            }
        else:
            # Check if over-budget allowed with approval
            overrun_percent = ((amount - available) / budget_line.budgeted_amount) * 100

            if overrun_percent &lt;= 10:
                # Allow with warning
                return {
                    'allowed': True,
```

```
                    'warning': True,
                    'message': f'Budget overrun by {overrun_percent:.1f}%. Approval may b
                    'requires_approval': True
                }
            else:
                # Block transaction
                return {
                    'allowed': False,
                    'message': f'Insufficient budget. Available: {available}, Requested:
                }

    def commit_budget(self, budget_line, amount):
        """
        Commit budget for pending transaction (e.g., PO)
        """
        budget_line.committed_amount += amount
        budget_line.save()

        # Check thresholds
        self._check_thresholds(budget_line)

    def consume_budget(self, budget_line, amount, release_commitment=True):
        """
        Consume budget when transaction complete
        """
        if release_commitment:
            budget_line.committed_amount -= amount

        budget_line.actual_amount += amount
        budget_line.update_variance()

        # Check thresholds
        self._check_thresholds(budget_line)

    def _check_thresholds(self, budget_line):
        """
        Check budget thresholds and create alerts
        """
        consumed_percent = (
            budget_line.actual_amount / budget_line.budgeted_amount
        ) * 100

        # 80% threshold
        if consumed_percent &gt;= 80 and consumed_percent &lt; 100:
            AIAlert.objects.get_or_create(
                company=budget_line.budget.company,
                alert_type='THRESHOLD',
                target_module='budgeting',
                target_record_id=budget_line.id,
                defaults={
                    'title': 'Budget Threshold Reached',
                    'message': f"Budget for {budget_line.account.name} is {consumed_perce
                    'severity': 'WARNING',
                    'assigned_to': budget_line.cost_center.manager
                }
            )
```

```
        # Exceeded
        elif consumed_percent &gt;= 100:
            AIAlert.objects.create(
                company=budget_line.budget.company,
                alert_type='THRESHOLD',
                title='Budget Exceeded',
                message=f"Budget for {budget_line.account.name} has been exceeded",
                severity='CRITICAL',
                target_module='budgeting',
                target_record_id=budget_line.id,
                assigned_to=budget_line.cost_center.manager
            )
```

7. Implementation Checklist

Weeks 1-4: Asset Management

- ☐ Create asset models (Asset, Category, Maintenance)
- ☐ Implement depreciation calculation service
- ☐ Build asset lifecycle workflows
- ☐ Create barcode scanning support
- ☐ Write unit tests

Weeks 5-8: Cost Centers & Budgeting

- ☐ Create cost center and budget models
- ☐ Implement budget control service
- ☐ Build budget monitoring dashboards
- ☐ Create alert system
- ☐ Write integration tests

Weeks 9-12: HR & Payroll

- ☐ Create employee and attendance models
- ☐ Implement leave management
- ☐ Build payroll calculation engine
- ☐ Create payslip generation
- ☐ Write tests

Weeks 13-16: Project Management

- ☐ Create project and task models
- ☐ Implement timesheet tracking
- ☐ Build Gantt chart visualization
- ☐ Create project costing reports
- ☐ End-to-end testing

Document Control:

- **Version:** 1.0
- **Dependencies:** Phase 0-5 complete
- **Completion:** Full TWIST ERP System