STEP 64: Admin Analytics Dashboard - Data Plan

Overview

This document defines all metrics, data sources, and computation strategies for the Admin Analytics Dashboard using only existing Prisma models.

Available Data Models

Primary Models (All scoped by companyId)

1. **RFP** - Core RFP data with stages, dates, status, awards
2. **SupplierContact** - Supplier invitations and participation
3. **SupplierResponse** - Submissions, scores, readiness data
4. **ActivityLog** - All system events with eventType categorization
5. **User** - Buyer/supplier users, role info
6. **ExecutiveSummaryDocument** - AI-generated summaries
7. **StageHistory** - RFP stage transitions

Key Fields for Analytics

- **RFP**: id, companyId, userId (buyer), createdAt, updatedAt, stage, status, awardStatus, awardedSupplierId, awardDecidedAt, isArchived, archivedAt
- **SupplierContact**: id, rfpId, invitationStatus, invitedAt, totalRFPsParticipated, totalWins, avgScore
- **SupplierResponse**: id, rfpId, supplierContactId, status, submittedAt, finalScore, readinessScore, autoScoreJson, autoScoreGeneratedAt
- **ActivityLog**: id, rfpId, userId, eventType, actorRole, createdAt, details (JSON metadata)
- **User**: id, companyId, role, name, email
- **StageHistory**: id, rfpId, oldStage, newStage, changedBy, createdAt

Admin Role Detection Strategy

Since the schema doesn’t have explicit admin role, we’ll use this logic:
- Check if User has role === “buyer” (suppliers cannot be admins)
- Optional: Future enhancement could check for user.preferences JSON field for admin flag
- All analytics are strictly scoped to user’s companyId

Metrics Definition

A. Global Filters

Filter	Type	Values	Default
dateRange	enum	“last_30_days” | “last_90_days” | “last_180_days” | “last_365_days” | “custom”	“last_90_days”
startDate	ISO string	Any date (required if dateRange=”custom”)	-
endDate	ISO string	Any date (required if dateRange=”custom”)	-
buyerId	string (optional)	User.id where role=”buyer”	null (all buyers)
stageFilter	RFPStage (optional)	Any RFPStage enum value	null (all stages)
statusFilter	string (optional)	“active” | “closed” | “all”	“all”

B. KPI Tiles (Top Row)

1. Active RFPs

- **Data Source**: RFP model
- **Query**: `COUNT(*)` WHERE `companyId = X AND isArchived = false AND status NOT IN ('cancelled')`
- **Notes**: Consider “active” as non-archived, non-cancelled RFPs

2. RFPs Closed (in date range)

- **Data Source**: RFP model
- **Query**: `COUNT(*)` WHERE `companyId = X AND (awardDecidedAt BETWEEN startDate AND endDate OR archivedAt BETWEEN startDate AND endDate)`
- **Notes**: Closed = awarded or archived within date range

3. Average Cycle Time

- **Data Source**: RFP model
- **Computation**: For RFPs closed in date range, `AVG(DATEDIFF(awardDecidedAt OR archivedAt, createdAt))`
- **Units**: Days
- **Notes**: Only include RFPs with valid closure dates

4. Win Rate

- **Data Source**: RFP model
- **Computation**: `COUNT(awardStatus = 'awarded') / COUNT(awardStatus IN ('awarded', 'cancelled', 'not_awarded')) * 100`
- **Units**: Percentage
- **Notes**: Filter by date range on awardDecidedAt or archivedAt

5. Supplier Participation

- **Data Source**: SupplierContact model
- **Computation**:
- avgInvited: `AVG(COUNT(supplierContacts) per RFP)` in date range
- participationRate: `COUNT(invitationStatus = 'ACCEPTED') / COUNT(invitationStatus != 'PENDING') * 100`
- **Units**: Count and percentage

6. Automation & AI Usage

- **Data Source**: ActivityLog model
- **Computation**:
- automationRunsCount: `COUNT(*) WHERE eventType = 'TIMELINE_AUTOMATION_RUN'` in date range
- aiScoringRunsCount: `COUNT(*) WHERE eventType IN ('AUTO_SCORE_RUN', 'AUTO_SCORE_REGENERATED')` in date range
- **Notes**: Scoped to companyId and date range

C. Charts & Detailed Metrics

1. RFP Volume Over Time

- **Data Source**: RFP model + ActivityLog
- **Query**: Group RFPs by week/month bucket based on:
- Created: `createdAt`
- Awarded: `awardDecidedAt WHERE awardStatus = 'awarded'`
- Cancelled: `archivedAt WHERE status = 'cancelled' OR awardStatus = 'cancelled'`
- **Output**: Array of `{ bucket: string, createdCount: number, awardedCount: number, cancelledCount: number }`
- **Bucket size**: Auto-detect based on date range (≤90 days = weekly, >90 days = monthly)

2. RFP Pipeline Stage Distribution

- **Data Source**: RFP model
- **Query**: `GROUP BY stage` for active (non-archived) RFPs as of current date
- **Output**: Array of `{ stage: RFPStage, count: number }`
- **Notes**: This is a snapshot, not time-ranged

3. Cycle Time by Stage

- **Data Source**: StageHistory model + RFP model
- **Computation**: For each stage, calculate average days spent by:
  1. Join StageHistory with RFP (closed in date range)
  2. For each RFP, compute time spent in each stage (difference between consecutive stage changes)
  3. Group by stage and average
- **Output**: Array of `{ stage: RFPStage, avgDays: number }`

- **Approximation**: If StageHistory is incomplete, use enteredStageAt field as fallback

4. Supplier Participation Funnel

- **Data Source**: SupplierContact + SupplierResponse models
- **Computation**: For RFPs in date range:
- avgInvited: `AVG(COUNT(SupplierContact per RFP))`
- avgSubmitted: `AVG(COUNT(SupplierResponse WHERE status = 'SUBMITTED' per RFP))`
- avgShortlisted: `AVG(COUNT(SupplierResponse WHERE awardOutcomeStatus IN ('recommended', 'shortlisted') per RFP))`
- **Output**: Object `{ avgInvited: number, avgSubmitted: number, avgShortlisted: number }`

5. Supplier Performance Overview (Top Suppliers)

- **Data Source**: SupplierContact model (has aggregated performance metrics)
- **Query**:
- Join SupplierContact with RFPs in company
- GROUP BY supplier (by email or organization)
- Aggregate: awardsWon (SUM(totalWins)), participationCount (SUM(totalRFPsParticipated)), avgScore (AVG(avgScore))
- **Output**: Array of `{ supplierId: string, supplierName: string, awardsWon: number, avgScore: number, participationCount: number }`
- **Limit**: Top 10 suppliers by awardsWon
- **Notes**: Use SupplierContact.totalWins and avgScore fields directly

6. Scoring Variance & Evaluation Quality

- **Data Source**: SupplierResponse.autoScoreJson + overrides (Step 61)
- **Computation**: For each RFP with multiple supplier scores:
  1. Extract scores per requirement from autoScoreJson
  2. For each RFP, compute variance = MAX(score) - MIN(score) across suppliers
  3. Flag high variance (>30% of scale) as potential quality issues
- **Output**: Array of `{ rfpId: string, rfpTitle: string, varianceValue: number, highScore: number, lowScore: number }`
- **Limit**: Top 10 RFPs with highest variance
- **Approximation**: If detailed scoring not available, use finalScore from SupplierResponse

7. Must-Have Violations & Risk Flags

- **Data Source**: SupplierResponse.autoScoreJson (must-have requirements)
- **Computation**:
  1. Parse autoScoreJson for must-have requirements
  2. Count suppliers per RFP that failed must-haves but still have status = ‘SUBMITTED’
- **Output**: Array of
  `{ rfpId: string, rfpTitle: string, supplierCount: number, violationsCount: number }`
- **Notes**: Only available for RFPs with auto-scoring enabled

8. Automation Impact

- **Data Source**: ActivityLog (TIMELINE_AUTOMATION_RUN) + RFP model
- **Computation**:
  1. Identify RFPs with at least one TIMELINE_AUTOMATION_RUN event (Group A)

2. Identify RFPs with zero TIMELINE_AUTOMATION_RUN events (Group B)
3. For RFPs closed in date range:
   - Group A: avgCycleTime, rfpsCount
   - Group B: avgCycleTime, rfpsCount

- **Output**: Object `{ withAutomation: { avgCycleTime: number, rfpsCount: number }, withoutAutomation: { avgCycleTime: number, rfpsCount: number } }`

9. AI Usage & Coverage

- **Data Source**: ActivityLog model
- **Computation**: Count events in date range by type:
- aiSummariesCount: `eventType = 'EXECUTIVE_SUMMARY_GENERATED'`
- aiDecisionBriefsCount: `eventType = 'DECISION_BRIEF_AI_GENERATED'`
- aiScoringEventsCount: `eventType IN ('AUTO_SCORE_RUN', 'AUTO_SCORE_REGENERATED')`
- rfpsWithAIUsageCount: `DISTINCT rfpId` from above events
- **Output**: Object `{ aiSummariesCount: number, aiDecisionBriefsCount: number, aiScoringEventsCount: number, rfpsWithAIUsageCount: number }`
- **Percentage**: `(rfpsWithAIUsageCount / totalRFPsInRange) * 100`

10. Export Usage

- **Data Source**: ActivityLog (EXPORT_GENERATED events from Step 63)
- **Query**: `GROUP BY details->>'exportId' OR details->>'exportTitle'` WHERE `eventType = 'EXPORT_GENERATED'` in date range
- **Output**: Array of `{ exportId: string, exportTitle: string, count: number }`
- **Sort**: By count descending
- **Limit**: Top 10 export types

11. Workload Distribution (Per Buyer)

- **Data Source**: RFP model + User model
- **Query**:
- JOIN RFP with User on userId
- GROUP BY userId
- Count: activeRfps (isArchived = false), closedRfps (archivedAt or awardDecidedAt in range)
- **Output**: Array of `{ buyerId: string, buyerName: string, activeRfps: number, closedRfps: number }`
- **Sort**: By activeRfps descending

12. Outcome Trends

- **Data Source**: RFP model
- **Query**: Group by week/month bucket based on awardDecidedAt or archivedAt:
- Awarded: `awardStatus = 'awarded'`
- Cancelled: `awardStatus = 'cancelled'` or `status = 'cancelled'`
- **Output**: Array of `{ bucket: string, awardedCount: number, cancelledCount: number }`
- **Bucket size**: Same as RFP Volume Over Time

Data Integrity & Edge Cases

Missing Data Handling

1. **No closed RFPs in range**: Return avgCycleTime = 0, winRate = 0

2. **No suppliers invited**: Return participation metrics = 0
3. **No activity logs**: Return automation/AI usage = 0
4. **No scoring data**: Return empty arrays for variance/violations
5. **No StageHistory**: Use RFP.enteredStageAt as fallback for cycle time

Performance Considerations

1. **Date range scoping**: Always apply WHERE clauses on date ranges to limit dataset
2. **Indexing**: Rely on existing indexes: rfpId, userId, companyId, createdAt, eventType
3. **Aggregation**: Use Prisma aggregations (_count, _avg, groupBy) where possible
4. **Caching**: Consider 5-minute server-side cache for dashboard data
5. **Pagination**: Limit large result sets (top 10 suppliers, top 10 exports, etc.)

Security & Scoping

1. **Company isolation**: ALL queries MUST include `companyId = user.companyId`
2. **No cross-company**: Never aggregate or compare across companies
3. **Admin-only**: Validate user role = “buyer” and access level before executing
4. **Read-only**: No writes except activity log entries (ADMIN_ANALYTICS_VIEWED)

Implementation Notes

Prisma Query Patterns

```
// Example: Active RFPs count
const activeRfps = await prisma.rFP.count({
  where: {
    companyId,
    isArchived: false,
    status: { not: 'cancelled' },
    ...(buyerId && { userId: buyerId }),
    ...(stageFilter && { stage: stageFilter }),
  }
});

// Example: Automation events count
const automationRuns = await prisma.activityLog.count({
  where: {
    eventType: 'TIMELINE_AUTOMATION_RUN',
    rfp: { companyId },
    createdAt: { gte: startDate, lte: endDate },
  }
});

// Example: Supplier performance aggregation
const topSuppliers = await prisma.supplierContact.groupBy({
  by: ['organization', 'name'],
  where: {
    rfp: { companyId },
  },
  _sum: {
    totalWins: true,
    totalRFPsParticipated: true,
  },
  _avg: {
    avgScore: true,
  },
  orderBy: {
    _sum: { totalWins: 'desc' },
  },
  take: 10,
});
```

Type Definitions

```
export interface AdminAnalyticsDashboard {
  kpis: AdminAnalyticsKPIs;
  charts: AdminAnalyticsCharts;
}

export interface AdminAnalyticsKPIs {
  activeRfps: number;
  closedRfps: number;
  avgCycleTimeDays: number;
  winRatePercent: number;
  avgSuppliersPerRfp: number;
  participationRate: number;
  automationRunsCount: number;
  aiScoringRunsCount: number;
}

export interface AdminAnalyticsCharts {
  rfpVolumeOverTime: Array<{ bucket: string; createdCount: number; awardedCount: num-
ber; cancelledCount: number }>;
  stageDistribution: Array<{ stage: string; count: number }>;
  cycleTimeByStage: Array<{ stage: string; avgDays: number }>;
  supplierParticipationFunnel: { avgInvited: number; avgSubmitted: number; avgShortl-
isted: number };
  supplierPerformance: Array<{ supplierId: string; supplierName: string; awardsWon: nu
mber; avgScore: number; participationCount: number }>;
  scoringVariance: Array<{ rfpId: string; rfpTitle: string; varianceValue: number; hig
hScore: number; lowScore: number }>;
  mustHaveViolations: Array<{ rfpId: string; rfpTitle: string; supplierCount: number;
violationsCount: number }>;
  automationImpact: { withAutomation: { avgCycleTime: number; rfpsCount: number }; wit
houtAutomation: { avgCycleTime: number; rfpsCount: number } };
  aiUsage: { aiSummariesCount: number; aiDecisionBriefsCount: number; aiS-
coringEventsCount: number; rfpsWithAIUsageCount: number; percentageRFPsWithAI:
number };
  exportUsage: Array<{ exportId: string; exportTitle: string; count: number }>;
  workloadByBuyer: Array<{ buyerId: string; buyerName: string; activeRfps: number; clo
sedRfps: number }>;
  outcomeTrends: Array<{ bucket: string; awardedCount: number; cancelledCount:
number }>;
}
```

Conclusion

All metrics are computable using existing Prisma models with no schema changes. The analytics service will be read-only (except for activity logging), company-scoped, and admin-only. Performance is maintained through proper indexing and query optimization.