STEP 39 FILE REFERENCE

Quick Reference for All Files and Functions

Generated: December 2, 2025
Purpose: Fast lookup for developers working with the scoring matrix feature

---

📁 File Structure Overview

```
nextjs_space/
├── lib/comparison/
│   ├── scoring-matrix-types.ts       [231 lines]  Type definitions
│   └── scoring-matrix.ts              [515 lines]  Core scoring engine
│
├── app/api/dashboard/rfps/[id]/comparison/matrix/
│   ├── route.ts                       [117 lines]  GET matrix
│   ├── recompute/route.ts             [143 lines]  POST recompute
│   └── export/route.ts                [118 lines]  GET export
│
├── app/dashboard/rfps/[id]/
│   ├── scoring-matrix/page.tsx        [600 lines]  UI page
│   └── compare/page.tsx              [1084 lines]  Comparison page
│
├── lib/
│   ├── activity-types.ts              [212 lines]  Activity event types
│   └── demo/scenario.ts              [1050 lines]  Demo data
│
├── prisma/
│   └── schema.prisma                  [596 lines]  Database schema
│
└── docs/
    ├── STEP_39_SCORING_MATRIX.md      [711 lines]  Documentation
    └── STEP_39_SCORING_MATRIX.pdf     [71 KB]      PDF documentation
```

1️⃣ Type Definitions

File: `lib/comparison/scoring-matrix-types.ts`

Purpose: TypeScript type definitions for the scoring matrix system
Lines: 231
Size: 5.4 KB

Exported Types

```
// Enums
export type RequirementCategoryId =
  "functional" | "commercial" | "legal" | "security" | "operational" | "other";

export type RequirementImportance =
  "must_have" | "should_have" | "nice_to_have";

export type RequirementScoreLevel =
  "pass" | "partial" | "fail" | "not_applicable" | "missing";

// Core Interfaces
export interface ScoringMatrixRequirement {
  requirementId: string;
  sourceType: "template_question" | "clause";
  referenceKey: string;
  shortLabel: string;
  longDescription: string;
  category: RequirementCategoryId;
  importance: RequirementImportance;
  defaultWeight: number;  // 0-1
}

export interface ScoringMatrixCell {
  requirementId: string;
  supplierId: string;
  scoreLevel: RequirementScoreLevel;
  numericScore: number;  // 0-1
  justification?: string;
}

export interface ScoringMatrixSupplierSummary {
  supplierId: string;
  supplierName: string;
  overallScore: number;       // 0-100
  weightedScore: number;      // 0-100
  categoryScores: Array<{
    category: RequirementCategoryId;
    score: number;
    weightedScore: number;
  }>;
  mustHaveCompliance: {
    total: number;
    passed: number;
    failed: number;
  };
}

export interface ScoringConfig {
  defaultWeights: {
    [category in RequirementCategoryId]?: number;
  };
  mustHavePenalty: number;
  partialFactor: number;
}

export interface ScoringMatrixSnapshot {
  rfpId: string;
  generatedAt: Date;
  generatedByUserId?: string;
  requirements: ScoringMatrixRequirement[];
  cells: ScoringMatrixCell[];
  supplierSummaries: ScoringMatrixSupplierSummary[];
  scoringConfig: ScoringConfig;
  meta: {
    totalRequirements: number;
    totalSuppliers: number;
    version: number;
  };
}

// Helper Types
export interface BuildMatrixOptions {
  forceRecompute?: boolean;
  scoringConfigOverrides?: Partial<ScoringConfig>;
  userId?: string;
}

export interface MatrixFilters {
  category?: RequirementCategoryId | "all";
  onlyDifferentiators?: boolean;
  onlyFailedOrPartial?: boolean;
  searchTerm?: string;
}
```

Constants

```
export const DEFAULT_SCORING_CONFIG: ScoringConfig = {
  defaultWeights: {
    functional: 1.0,
    commercial: 0.9,
    legal: 0.95,
    security: 1.0,
    operational: 0.8,
    other: 0.6
  },
  mustHavePenalty: 10,
  partialFactor: 0.5
};
```

Import in your code:

```
import {
  ScoringMatrixSnapshot,
  ScoringMatrixRequirement,
  RequirementScoreLevel,
  DEFAULT_SCORING_CONFIG,
} from '@/lib/comparison/scoring-matrix-types';
```

2️⃣ Scoring Engine

File: `lib/comparison/scoring-matrix.ts`

Purpose: Core business logic for matrix generation and scoring
Lines: 515
Size: 16.7 KB

Public Functions (3)

`buildScoringMatrix()`

```
async function buildScoringMatrix(
  rfpId: string,
  options?: BuildMatrixOptions
): Promise<ScoringMatrixSnapshot>
```

Purpose: Build a complete scoring matrix from scratch
When to use: Force fresh computation, initial matrix creation
Flow:
1. Fetch RFP with template, clauses, responses
2. Extract requirements via `extractRequirements()`
3. Build cells via `buildScoringCells()`
4. Calculate summaries via `calculateSupplierSummaries()`
5. Persist snapshot to `RFP.scoringMatrixSnapshot`
6. Return snapshot

Example:

```
const matrix = await buildScoringMatrix('rfp-123', {
  forceRecompute: true,
  userId: 'user-456',
  scoringConfigOverrides: {
    mustHavePenalty: 15
  }
});
```

`getScoringMatrix()`

```
async function getScoringMatrix(
  rfpId: string,
  fromCache?: boolean
): Promise<ScoringMatrixSnapshot | null>
```

Purpose: Retrieve matrix with caching support
When to use: Display matrix, prefer cached version
Flow:
1. If `fromCache` is true, try to load from `RFP.scoringMatrixSnapshot`
2. If not found or `fromCache` is false, call `buildScoringMatrix()`
3. Return snapshot or null on error

Example:

```
// Use cached version if available
const matrix = await getScoringMatrix('rfp-123', true);

// Force fresh computation
const freshMatrix = await getScoringMatrix('rfp-123', false);
```

`exportMatrixToCSV()`

```
async function exportMatrixToCSV(
  rfpId: string,
  filters?: MatrixFilters
): Promise<string>
```

Purpose: Export matrix to CSV format
When to use: Download/export functionality
Returns: CSV string ready for download
Flow:
1. Get matrix via `getScoringMatrix()`
2. Apply filters via `applyFilters()`
3. Build CSV header and rows
4. Return formatted CSV string

Example:

```
const csv = await exportMatrixToCSV('rfp-123', {
  category: 'security',
  onlyFailedOrPartial: true,
  searchTerm: 'compliance'
});
```

Private Helper Functions (8)

`extractRequirements()`

```
async function extractRequirements(
  rfp: RFP & { template: any }
): Promise<ScoringMatrixRequirement[]>
```

Purpose: Extract requirements from template and clauses
Sources:
- `rfp.appliedTemplateSnapshot.sections[].subsections[].questions[]`
- `rfp.appliedClausesSnapshot.clauses[]`

`buildScoringCells()`

```
async function buildScoringCells(
  rfp: RFP & { supplierResponses: any[] },
  requirements: ScoringMatrixRequirement[]
): Promise<ScoringMatrixCell[]>
```

Purpose: Create scoring cells (requirements × suppliers)
Algorithm: For each supplier × requirement pair, call `scoreSupplierOnRequirement()`

`scoreSupplierOnRequirement()`

```
async function scoreSupplierOnRequirement(
  supplierResponse: SupplierResponse,
  requirement: ScoringMatrixRequirement
): Promise<{
  scoreLevel: RequirementScoreLevel;
  numericScore: number;
  justification?: string
}>
```

Purpose: Score single supplier on single requirement
Logic:
- Looks up requirement in `supplierResponse.extractedRequirementsCoverage`
- Maps status to scoreLevel:
- `fully_addressed` → pass (1.0)
- `partially_addressed` → partial (0.5)
- `not_applicable` → not_applicable (0)
- `missing/fail` → fail (0)

`calculateSupplierSummaries()`

```
function calculateSupplierSummaries(
  requirements: ScoringMatrixRequirement[],
  cells: ScoringMatrixCell[],
  config: ScoringConfig
): ScoringMatrixSupplierSummary[]
```

Purpose: Calculate aggregated scores for all suppliers
Computes:
- Overall score (unweighted average)
- Weighted score (with category weights and importance)
- Category breakdown
- Must-have compliance stats

`applyFilters()`

```
function applyFilters(
  requirements: ScoringMatrixRequirement[],
  cells: ScoringMatrixCell[],
  filters: MatrixFilters
): ScoringMatrixRequirement[]
```

Purpose: Filter requirements based on UI filters
Supports:
- Category filter
- Only differentiators (where suppliers differ)
- Only failed/partial
- Search term

Mapping Helpers (4)

```
function mapSectionToCategory(sectionCode: string): RequirementCategoryId
function mapClauseTypeToCategory(clauseType: string): RequirementCategoryId
function mapWeightToImportance(weight: number): RequirementImportance
function mapMandatoryToImportance(isMandatory: boolean): RequirementImportance
```

Purpose: Intelligent mapping of template/clause metadata to scoring matrix types

Import in your code:

```
import {
  buildScoringMatrix,
  getScoringMatrix,
  exportMatrixToCSV
} from '@/lib/comparison/scoring-matrix';
```

3️⃣ API Endpoints

File: `app/api/dashboard/rfps/[id]/comparison/matrix/route.ts`

Purpose: GET endpoint to retrieve scoring matrix
Lines: 117
Size: 3.3 KB

Endpoint: `GET /api/dashboard/rfps/[id]/comparison/matrix`

Authentication: Required (NextAuth session)
Authorization: Buyer only, RFP owner
Response:

```
{
  "success": true,
  "matrix": {
    "rfpId": "...",
    "generatedAt": "2025-12-02T...",
    "requirements": [...],
    "cells": [...],
    "supplierSummaries": [...],
    "scoringConfig": {...},
    "meta": {...}
  }
}
```

Security Checks:
1. ✅ `getServerSession()` - Authentication
2. ✅ `user.role === 'buyer'` - Role check
3. ✅ `rfp.userId === session.user.id` - Ownership

Usage:

```
const response = await fetch(`/api/dashboard/rfps/${rfpId}/comparison/matrix`);
const { matrix } = await response.json();
```

File: `app/api/dashboard/rfps/[id]/comparison/matrix/recompute/route.ts`

Purpose: POST endpoint to force recomputation
Lines: 143
Size: 4.0 KB

Endpoint: `POST /api/dashboard/rfps/[id]/comparison/matrix/recompute`

Authentication: Required
Authorization: Buyer only, RFP owner
Request Body:

```json
{
  "scoringConfigOverrides": {
    "mustHavePenalty": 15,
    "partialFactor": 0.6
  }
}
```

Response:

```json
{
  "success": true,
  "message": "Scoring matrix recomputed successfully",
  "matrix": { ... }
}
```

Side Effects:
- Updates `RFP.scoringMatrixSnapshot`
- Logs activity event: `comparison_matrix_recomputed`

Usage:

```
const response = await fetch(
  `/api/dashboard/rfps/${rfpId}/comparison/matrix/recompute`,
  {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ scoringConfigOverrides })
  }
);
```

File: `app/api/dashboard/rfps/[id]/comparison/matrix/export/route.ts`

Purpose: GET endpoint to export matrix to CSV
Lines: 118
Size: 3.7 KB

Endpoint: `GET /api/dashboard/rfps/[id]/comparison/matrix/export`

Authentication: Required
Authorization: Buyer only, RFP owner
Query Parameters:
- `category` : Filter by category
- `onlyDifferentiators` : true/false
- `onlyFailedOrPartial` : true/false
- `searchTerm` : search string

Response: CSV file download

Side Effects:
- Logs activity event: `comparison_matrix_exported`

Usage:

```
const url = `/api/dashboard/rfps/${rfpId}/comparison/matrix/export?` +
  `category=security&onlyFailedOrPartial=true`;
window.location.href = url; // Trigger download
```

4️⃣ UI Component

File: `app/dashboard/rfps/[id]/scoring-matrix/page.tsx`

Purpose: React page component for scoring matrix UI
Lines: 600
Size: 22.5 KB
Route: `/dashboard/rfps/[id]/scoring-matrix`

Component Structure

```
export default function ScoringMatrixPage({
  params
}: {
  params: { id: string }
}) {
  // ... implementation
}
```

State Management

```
const [matrix, setMatrix] = useState<ScoringMatrixSnapshot | null>(null);
const [loading, setLoading] = useState(true);
const [recomputing, setRecomputing] = useState(false);
const [exporting, setExporting] = useState(false);
const [error, setError] = useState<string | null>(null);

const [filters, setFilters] = useState<MatrixFilters>({
  category: 'all',
  onlyDifferentiators: false,
  onlyFailedOrPartial: false,
  searchTerm: '',
});

const [showFilters, setShowFilters] = useState(false);
const [selectedTab, setSelectedTab] = useState<'matrix' | 'summaries'>('matrix');
```

Key Functions

`fetchMatrix()`

```
const fetchMatrix = async () => {
  // Fetches matrix from GET endpoint
  // Updates state with matrix data
}
```

`handleRecompute()`

```
const handleRecompute = async () => {
  // Calls POST recompute endpoint
  // Updates matrix state
  // Shows success/error toast
}
```

`handleExport()`

```
const handleExport = async () => {
  // Builds export URL with filters
  // Triggers file download
}
```

UI Sections

Header Section:
- Title and subtitle
- Recompute button (with loading spinner)
- Export CSV button (with loading spinner)

Filter Bar:
- Category dropdown
- Differentiators toggle
- Failed/partial toggle
- Search input

Matrix Table:
- Sticky header with supplier names
- Frozen left column with requirements
- Color-coded score badges
- Tooltips on hover

Summary Cards:
- Overall scores
- Weighted scores
- Category breakdown
- Must-have compliance

Empty States:
- No data available
- No requirements
- No supplier responses

Constants

```
const CATEGORY_ICONS: Record<RequirementCategoryId, any> = {
  functional: TrendingUp,
  commercial: DollarSign,
  legal: FileText,
  security: Shield,
  operational: SettingsIcon,
  other: Briefcase,
};

const SCORE_LEVEL_STYLES: Record<RequirementScoreLevel, {
  bg: string;
  text: string;
  icon: any;
}> = {
  pass: { bg: 'bg-green-100', text: 'text-green-700', icon: CheckCircle2 },
  partial: { bg: 'bg-amber-100', text: 'text-amber-700', icon: MinusCircle },
  fail: { bg: 'bg-red-100', text: 'text-red-700', icon: XCircle },
  not_applicable: { bg: 'bg-gray-100', text: 'text-gray-500', icon: HelpCircle },
  missing: { bg: 'bg-gray-50', text: 'text-gray-400', icon: AlertCircle },
};
```

Navigation:

```
// Access from RFP detail page
<Link href={`/dashboard/rfps/${rfpId}/scoring-matrix`}>
  View Scoring Matrix
</Link>
```

5️⃣ Database Schema

File: `prisma/schema.prisma`

Purpose: Database schema definition
Lines: 596
Size: 18.9 KB

RFP Model Extension

```
model RFP {
  // ... existing fields ...

  scoringMatrixSnapshot Json?  // ← Added for STEP 39

  // Relationships
  template          RfpTemplate?      @relation(fields: [templateId], references:
[id])
  supplierContacts  SupplierContact[]
  supplierResponses SupplierResponse[]
  // ... other relations ...
}
```

Data Type: `Json?` (optional JSON field)
Stores: Complete `ScoringMatrixSnapshot` object
Purpose: Caching computed matrix for performance

Example Query:

```
const rfp = await prisma.rFP.findUnique({
  where: { id: rfpId },
  select: { scoringMatrixSnapshot: true }
});

const snapshot = rfp.scoringMatrixSnapshot as ScoringMatrixSnapshot;
```

6️⃣ Activity Types

File: `lib/activity-types.ts`

Purpose: Activity logging type definitions
Lines: 212
Size: 8.0 KB

New Event Types

```
export type ActivityEventType =
  // ... existing types ...
  | "comparison_matrix_recomputed"
  | "comparison_matrix_exported";

export const ActivityEventTypes = {
  // ... existing events ...
  COMPARISON_MATRIX_RECOMPUTED: "comparison_matrix_recomputed" as ActivityEventType,
  COMPARISON_MATRIX_EXPORTED: "comparison_matrix_exported" as ActivityEventType,
};

export const ActivityEventLabels: Record<ActivityEventType, string> = {
  // ... existing labels ...
  comparison_matrix_recomputed: "Comparison Matrix Recomputed",
  comparison_matrix_exported: "Comparison Matrix Exported",
};
```

Usage:

```
import { logActivity } from '@/lib/activity-log';
import { ActivityEventTypes } from '@/lib/activity-types';

await logActivity({
  eventType: ActivityEventTypes.COMPARISON_MATRIX_RECOMPUTED,
  userId: session.user.id,
  actorRole: 'BUYER',
  summary: `Recomputed scoring matrix for RFP ${rfpId}`,
  rfpId: rfpId,
  details: {
    totalRequirements: matrix.meta.totalRequirements,
    totalSuppliers: matrix.meta.totalSuppliers,
  },
});
```

7️⃣ Demo Data

File: `lib/demo/scenario.ts`

Purpose: Demo mode scenario data
Lines: 1,050
Size: 45.1 KB

Scoring Matrix Integration

```
// Line ~794
const demoRFP = {
  // ... other demo RFP fields ...
  scoringMatrixSnapshot: demoScoringMatrixSnapshot as any
};
```

What to look for:
- `demoScoringMatrixSnapshot` constant definition

- Precomputed matrix with 3-4 suppliers
- 15-25 requirements with mixed scores
- Pass/Partial/Fail variations for visual interest

Testing:

```
// In demo mode, access the demo RFP
const demoRfpId = "demo-rfp-1";
const matrix = await getScoringMatrix(demoRfpId);
// Should return precomputed snapshot
```

8️⃣ Documentation

File: `docs/STEP_39_SCORING_MATRIX.md`

Purpose: Comprehensive technical documentation
Lines: 711
Size: 20.7 KB

Sections

1. Overview and objectives
2. Data model and types
3. Scoring engine architecture
4. API endpoints specification
5. UI components and behavior
6. Security model
7. Demo mode integration
8. Usage examples and best practices

Access: Read directly or convert to PDF

File: `docs/STEP_39_SCORING_MATRIX.pdf`

Purpose: PDF version for stakeholder distribution
Size: 71 KB
Format: Professional PDF document

🔍 Quick Lookup Table

Need	File	Function/Component
Build matrix from scratch	`scoring-matrix.ts`	`buildScoringMatrix()`
Get cached matrix	`scoring-matrix.ts`	`getScoringMatrix(rfpId, true)`
Export to CSV	`scoring-matrix.ts`	`exportMatrixToCSV()`
Type definitions	`scoring-matrix-types.ts`	All exports
Default config	`scoring-matrix-types.ts`	`DEFAULT_SCORING_CONFIG`
Retrieve via API	API endpoint	`GET /api/.../matrix`
Force recompute	API endpoint	`POST /api/.../matrix/recompute`
Export via API	API endpoint	`GET /api/.../matrix/export`
Display UI	`scoring-matrix/page.tsx`	`ScoringMatrixPage`
Activity logging	`activity-types.ts`	Event type constants
Database field	`schema.prisma`	`RFP.scoringMatrixSnapshot`
Demo data	`demo/scenario.ts`	`demoScoringMatrixSnapshot`

🛠️ Common Code Patterns

Pattern 1: Fetch and Display Matrix

```
import { getScoringMatrix } from '@/lib/comparison/scoring-matrix';
import { ScoringMatrixSnapshot } from '@/lib/comparison/scoring-matrix-types';

// In your component or API route
const matrix: ScoringMatrixSnapshot | null = await getScoringMatrix(rfpId, true);

if (!matrix) {
  // Handle no matrix case
  return <EmptyState />;
}

// Display matrix
return <MatrixTable matrix={matrix} />;
```

Pattern 2: Recompute Matrix

```typescript
import { buildScoringMatrix } from '@/lib/comparison/scoring-matrix';
import { logActivity } from '@/lib/activity-log';

// Force fresh computation
const matrix = await buildScoringMatrix(rfpId, {
  forceRecompute: true,
  userId: session.user.id
});

// Log activity
await logActivity({
  eventType: 'comparison_matrix_recomputed',
  userId: session.user.id,
  rfpId: rfpId,
  // ... other fields
});
```

Pattern 3: Export with Filters

```typescript
import { exportMatrixToCSV } from '@/lib/comparison/scoring-matrix';
import { MatrixFilters } from '@/lib/comparison/scoring-matrix-types';

const filters: MatrixFilters = {
  category: 'security',
  onlyFailedOrPartial: true,
  searchTerm: 'compliance'
};

const csvData = await exportMatrixToCSV(rfpId, filters);

// Return as download
return new Response(csvData, {
  headers: {
    'Content-Type': 'text/csv',
    'Content-Disposition': `attachment; filename="matrix-${rfpId}.csv"`
  }
});
```

Pattern 4: Access Control Check

```
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth-options';

// In API route
const session = await getServerSession(authOptions);

if (!session?.user?.id) {
  return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
}

const user = await prisma.user.findUnique({
  where: { id: session.user.id },
  select: { role: true }
});

if (user?.role !== 'buyer') {
  return NextResponse.json({ error: 'Buyers only' }, { status: 403 });
}

// Verify RFP ownership
const rfp = await prisma.rFP.findFirst({
  where: { id: rfpId, userId: session.user.id }
});

if (!rfp) {
  return NextResponse.json({ error: 'Access denied' }, { status: 403 });
}

// Proceed with operation
```

Pattern 5: Filter Requirements in UI

```
import { MatrixFilters, ScoringMatrixRequirement } from '@/lib/comparison/scoring-
matrix-types';

function filterRequirements(
  requirements: ScoringMatrixRequirement[],
  filters: MatrixFilters
): ScoringMatrixRequirement[] {
  let filtered = [...requirements];

  // Category filter
  if (filters.category && filters.category !== 'all') {
    filtered = filtered.filter(r => r.category === filters.category);
  }

  // Search filter
  if (filters.searchTerm) {
    const term = filters.searchTerm.toLowerCase();
    filtered = filtered.filter(r =>
      r.shortLabel.toLowerCase().includes(term) ||
      r.longDescription.toLowerCase().includes(term)
    );
  }

  return filtered;
}
```

📊 Scoring Algorithm Reference

Score Level Mapping

Status (from response)	Score Level	Numeric Score	Display
`fully_addressed`	`pass`	1.0	🟢 Green
`partially_addressed`	`partial`	0.5	🟡 Yellow
`not_applicable`	`not_applicable`	0	⚪ Gray
`missing` / not found	`missing`	0	⚫ Light Gray
`not_addressed`	`fail`	0	🔴 Red

Score Calculations

Overall Score (Unweighted):

```
overallScore = (sum of all numericScores) / (total requirements) * 100
```

Weighted Score:

```
weightedScore = (sum of (numericScore × requirementWeight × categoryWeight)) / (sum of
weights) * 100
```

Category Weights (Default):
- Functional: 1.0
- Security: 1.0
- Legal: 0.95
- Commercial: 0.9
- Operational: 0.8
- Other: 0.6

Importance Weights:
- Must-have: 1.0
- Should-have: 0.7
- Nice-to-have: 0.4

Must-Have Penalty:
- Default: -10 points per failed must_have requirement

🔐 Security Checklist

For any new code touching scoring matrix:

- [ ] Check authentication (`getServerSession()`)
- [ ] Verify role === ‘buyer’
- [ ] Validate RFP ownership (`rfp.userId === session.user.id`)
- [ ] Return 401 for unauthenticated
- [ ] Return 403 for unauthorized/suppliers
- [ ] Return 404 for not found
- [ ] Log activity events for auditing
- [ ] Validate and sanitize user input
- [ ] Use Prisma parameterized queries (no SQL injection)
- [ ] Don’t expose supplier responses to other suppliers

🚀 Performance Tips

1. **Use Caching:** Always call `getScoringMatrix(rfpId, true)` unless you need fresh data
2. **Avoid N+1 Queries:** Use Prisma `include` to fetch related data in single query
3. **Filter Client-Side:** When possible, filter in UI rather than regenerating matrix
4. **Debounce Search:** Debounce search input to avoid excessive filtering
5. **Lazy Load Details:** Load requirement details on-demand (tooltips) rather than upfront
6. **Paginate Large Matrices:** Consider pagination for 200+ requirements

---

📝 Testing Helpers

Test Data Factory

```
import { ScoringMatrixSnapshot } from '@/lib/comparison/scoring-matrix-types';

function createMockMatrix(): ScoringMatrixSnapshot {
  return {
    rfpId: 'test-rfp-1',
    generatedAt: new Date(),
    requirements: [
      {
        requirementId: 'req-1',
        sourceType: 'template_question',
        referenceKey: 'TECH:SEC:Q1',
        shortLabel: 'Data encryption',
        longDescription: 'Describe your data encryption approach',
        category: 'security',
        importance: 'must_have',
        defaultWeight: 1.0,
      },
      // ... more requirements
    ],
    cells: [
      {
        requirementId: 'req-1',
        supplierId: 'sup-1',
        scoreLevel: 'pass',
        numericScore: 1.0,
        justification: 'AES-256 encryption implemented',
      },
      // ... more cells
    ],
    supplierSummaries: [
      {
        supplierId: 'sup-1',
        supplierName: 'Supplier A',
        overallScore: 85.5,
        weightedScore: 88.2,
        categoryScores: [],
        mustHaveCompliance: { total: 10, passed: 9, failed: 1 },
      },
      // ... more summaries
    ],
    scoringConfig: DEFAULT_SCORING_CONFIG,
    meta: {
      totalRequirements: 20,
      totalSuppliers: 3,
      version: 1,
    },
  };
}
```

API Test Example

```
import { GET } from '@/app/api/dashboard/rfps/[id]/comparison/matrix/route';

describe('GET /api/dashboard/rfps/[id]/comparison/matrix', () => {
  it('returns matrix for authorized buyer', async () => {
    const response = await GET(
      new NextRequest('http://localhost/api/...'),
      { params: { id: 'rfp-123' } }
    );

    expect(response.status).toBe(200);
    const data = await response.json();
    expect(data.success).toBe(true);
    expect(data.matrix).toBeDefined();
  });

  it('returns 403 for supplier', async () => {
    // Mock session with supplier role
    const response = await GET(...);
    expect(response.status).toBe(403);
  });
});
```

🔗 Related Files

Files You May Need to Modify Together

When adding new requirement sources:
- `scoring-matrix.ts` → `extractRequirements()`
- `scoring-matrix-types.ts` → Add new `sourceType`

When changing scoring algorithm:
- `scoring-matrix.ts` → `scoreSupplierOnRequirement()`
- `scoring-matrix.ts` → `calculateSupplierSummaries()`
- `scoring-matrix-types.ts` → Update `ScoringConfig`

When adding UI filters:
- `scoring-matrix/page.tsx` → Add filter state
- `scoring-matrix.ts` → Update `applyFilters()`
- `scoring-matrix-types.ts` → Update `MatrixFilters`

When adding export formats:
- Create new export function in `scoring-matrix.ts`
- Update `export/route.ts` to handle new format
- Add button in `scoring-matrix/page.tsx`

📚 Further Reading

- **Full Specification:** `/home/ubuntu/Uploads/user_message_2025-12-02_04-35-12.txt`
- **Completion Report:** `/home/ubuntu/fyndr/STEP_39_COMPLETION_REPORT.md`

- **Executive Summary:** `/home/ubuntu/fyndr/STEP_39_EXECUTIVE_SUMMARY.md`
- **Technical Documentation:** `/home/ubuntu/fyndr/nextjs_space/docs/STEP_39_SCORING_MATRIX.md`
- **PDF Documentation:** `/home/ubuntu/fyndr/nextjs_space/docs/STEP_39_SCORING_MATRIX.pdf`

🆘 Troubleshooting

Common Issues

Issue: Matrix returns empty requirements
Solution: Check that RFP has `appliedTemplateSnapshot` or `appliedClausesSnapshot`

Issue: Scores all showing as "missing"
Solution: Verify supplier responses have `extractedRequirementsCoverage` field

Issue: 403 Forbidden on API calls
Solution: Ensure user is buyer and owns the RFP

Issue: Cached matrix is stale
Solution: Call recompute endpoint or use `getScoringMatrix(rfpId, false)`

Issue: Export not including filtered data
Solution: Ensure filters are passed as query parameters to export endpoint

---

File Reference Version: 1.0
Last Updated: December 2, 2025
Maintained By: Fyndr Development Team

For complete implementation details, see STEP_39_COMPLETION_REPORT.md