GCS JSON to Database Storage Pipeline

Complete Technical Guide: From Normalized JSON to Database Records

Table of Contents

1. Overview

1.1 Process Summary

```
Normalized JSON (in GCS)
        ↓
Extract Metadata & Biomarkers
        ↓
Create Report Record (reports table)
        ↓
Get report_id
        ↓
Create Biomarker Records (biomarkers table)
        ↓
Return Success
```

1.2 Key Components

Component	Purpose	Location
Normalized JSON	Source data	GCS: `users/{nic}/processed/{file_id}_normalized.json`
`store_normalized_report_to_db()`	Main function	`app/services/reportService.py`
`reports` table	Report metadata	Supabase PostgreSQL
`biomarkers` table	Biomarker values	Supabase PostgreSQL

1.3 When This Happens

This process is triggered **automatically** by the background worker after OCR processing:

```
# app/workers/ocr_worker.py
def process_document_worker(gcs_uri, nic, patient_id, file_id):
    # ... OCR and normalization ...

    # 1. Save normalized JSON to GCS
    store_json(nic, f"{file_id}_normalized", normalized_json)

    # 2. Save to Supabase database
    store_normalized_report_to_db(
        patient_id=UUID(patient_id),
        file_id=file_id,
        gcs_path=gcs_uri,
        normalized_json=normalized_json  # ← This is the input
    )
```

---

2. JSON Structure from GCS

2.1 Complete Structure

The normalized JSON stored in GCS has this structure:

```
{
  "patient": {
    "name": "MR S KUMAR",
    "age_years": 62,
    "gender": "Male",
    "ref_doctor": "Dr. Silva",
    "service_ref_no": "CHL000735039"
  },
  "report": {
    "type": "Full Blood Count",
    "sample_collected_at": "2025-06-03T09:10:00",
    "printed_at": "2025-06-03T18:43:00"
  },
  "biomarkers": [
    {
      "name": "WBC",
      "value": 7970.0,
      "unit": "per cu mm",
      "absolute": null,
      "ref_range": [4000.0, 11000.0]
    },
    {
      "name": "Hemoglobin",
      "value": 13.5,
      "unit": "g/dL",
      "absolute": null,
      "ref_range": [13.0, 17.0]
    },
    {
      "name": "Platelets",
      "value": 250000.0,
      "unit": "per cu mm",
      "absolute": null,
      "ref_range": [150000.0, 450000.0]
    }
  ]
}
```

2.2 Field Mapping to Database

JSON Path	Database Table	Database Field	Notes
`report.type`	reports	report_type	Required
`report.sample_collected_at`	reports	sample_collected_at	Optional, ISO-8601
`biomarkers[].name`	biomarkers	name	Required
`biomarkers[].value`	biomarkers	value	Required, numeric
`biomarkers[].unit`	biomarkers	unit	Optional
`biomarkers[].ref_range[0]`	biomarkers	ref_min	Optional
`biomarkers[].ref_range[1]`	biomarkers	ref_max	Optional
`biomarkers[].flag`	biomarkers	flag	Optional (Lipid reports)

Note: Patient data from JSON is **NOT** stored in database (already exists from registration).

---

3. Database Schema

3.1 Tables Involved

```
-- Already exists (from patient registration)
patients (
  id uuid PRIMARY KEY,
  full_name text,
  email text,
  nic text,
  ...
)

-- Created by our process
reports (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  patient_id uuid REFERENCES patients(id) ON DELETE CASCADE,
  file_id text NOT NULL,
  report_type text NOT NULL,
  sample_collected_at timestamptz,
  gcs_path text,
  created_at timestamptz DEFAULT now()
)

-- Created by our process (multiple rows)
biomarkers (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  report_id uuid REFERENCES reports(id) ON DELETE CASCADE,
  name text NOT NULL,
  value numeric NOT NULL,
  unit text,
  ref_min numeric,
  ref_max numeric,
  flag text
)
```

3.2 Relationship Flow

```
Patient (already exists)
    │
    │ patient_id (UUID)
    ↓
Report (created from JSON)
    │
    │ report_id (UUID)
    ↓
Biomarkers (created from JSON array)
    ├─ Biomarker 1 (WBC)
    ├─ Biomarker 2 (Hemoglobin)
    ├─ Biomarker 3 (Platelets)
    └─ ... (N biomarkers)
```

4. Data Extraction Process

4.1 Step-by-Step Extraction

Step 1: Extract Report Metadata

```
# Input: normalized_json
normalized_json = {
  "report": {
    "type": "Full Blood Count",
    "sample_collected_at": "2025-06-03T09:10:00"
  },
  ...
}

# Extract
report_type = normalized_json.get("report_type", "Unknown")
```

```
# Result: "Full Blood Count"

sample_date_str = normalized_json.get("sample_collection_date")
# Result: "2025-06-03T09:10:00"

# Parse to datetime
sample_date = datetime.fromisoformat(sample_date_str)
# Result: datetime object
```

Important: The JSON uses `report.type` and `report.sample_collected_at`, but we extract using different keys for compatibility.

Step 2: Create Report Record

```
# Prepare data
report_data = ReportCreate(
    patient_id=UUID("550e8400-e29b-41d4-a716-446655440000"),
    file_id="abc-123-def-456",
    report_type="Full Blood Count",
    sample_collected_at=datetime(2025, 6, 3, 9, 10, 0),
    gcs_path="gs://bucket/users/199512345678/reports/abc-123.pdf"
)

# Insert to database
response = supabase.table("reports").insert({
    "patient_id": "550e8400-e29b-41d4-a716-446655440000",
    "file_id": "abc-123-def-456",
    "report_type": "Full Blood Count",
    "sample_collected_at": "2025-06-03T09:10:00",
    "gcs_path": "gs://bucket/users/199512345678/reports/abc-123.pdf"
}).execute()

# Get generated report_id
report_id = response.data[0]["id"]
# Result: "a1b2c3d4-e5f6-7890-abcd-ef1234567890"
```

Step 3: Extract Biomarkers Array

```
# Input: normalized_json
biomarkers_data = normalized_json.get("biomarkers", [])

# Result: Array of biomarker objects
[
  {
    "name": "WBC",
    "value": 7970.0,
    "unit": "per cu mm",
    "ref_range": [4000.0, 11000.0]
  },
  {
    "name": "Hemoglobin",
    "value": 13.5,
    "unit": "g/dL",
    "ref_range": [13.0, 17.0]
  }
]
```

Step 4: Process Each Biomarker

```
for bm in biomarkers_data:
    # Extract fields
    name = bm.get("name", "")          # "WBC"
    value = bm.get("value", 0)         # 7970.0
    unit = bm.get("unit")              # "per cu mm"

    # Handle ref_range array
    ref_range = bm.get("ref_range")    # [4000.0, 11000.0]

    if isinstance(ref_range, list) and len(ref_range) >= 2:
        ref_min = float(ref_range[0])  # 4000.0
        ref_max = float(ref_range[1])  # 11000.0
    else:
        ref_min = None
        ref_max = None

    # Create biomarker object
    biomarker = BiomarkerCreate(
        report_id=UUID(report_id),
        name=name,
        value=float(value),
        unit=unit,
        ref_min=ref_min,
        ref_max=ref_max,
        flag=bm.get("flag")
    )
```

Step 5: Bulk Insert Biomarkers

```
# Prepare array of biomarker data
biomarker_data = [
    {
        "report_id": "a1b2c3d4-...",
        "name": "WBC",
        "value": 7970.0,
        "unit": "per cu mm",
        "ref_min": 4000.0,
        "ref_max": 11000.0,
        "flag": None
    },
    {
        "report_id": "a1b2c3d4-...",
        "name": "Hemoglobin",
        "value": 13.5,
        "unit": "g/dL",
        "ref_min": 13.0,
        "ref_max": 17.0,
        "flag": None
    }
]

# Bulk insert
response = supabase.table("biomarkers").insert(biomarker_data).execute()

# Result: Array of created biomarker records with auto-generated IDs
```

---

5. Storage Implementation

5.1 Main Function (`reportService.py`)

```python
def store_normalized_report_to_db(
    patient_id: UUID,
    file_id: str,
    gcs_path: str,
    normalized_json: dict
) -> dict:
    """
    Store normalized report and biomarkers to Supabase.

    Args:
        patient_id: Patient's UUID (from patients table)
        file_id: Unique file identifier (from upload)
        gcs_path: Path to PDF in GCS (gs://bucket/...)
        normalized_json: Normalized medical report JSON (from GCS)

    Returns:
        {
            "success": True/False,
            "data": {
                "report": {...},
                "biomarkers": [...]
            }
        }
    """
    try:
        # STEP 1: Extract report metadata
        report_type = normalized_json.get("report_type", "Unknown")
        sample_date_str = normalized_json.get("sample_collection_date")

        # Parse date if provided
        sample_date = None
        if sample_date_str:
            try:
                sample_date = datetime.fromisoformat(sample_date_str)
            except:
                sample_date = None

        # STEP 2: Create report record
        report_data = ReportCreate(
            patient_id=patient_id,
            file_id=file_id,
            report_type=report_type,
            sample_collected_at=sample_date,
            gcs_path=gcs_path
        )

        report_result = create_report(report_data)
        if not report_result.get("success"):
            return {"success": False, "error": f"Failed to create report:
{report_result.get('error')}"}

        report_id = report_result["data"]["id"]

        # STEP 3: Extract biomarkers array
        biomarkers_data = normalized_json.get("biomarkers", [])

        if biomarkers_data:
            biomarkers = []

            # STEP 4: Process each biomarker
            for bm in biomarkers_data:
                # Safe float conversion
                def safe_float(value):
                    if value is None or value == "":
                        return None
                    try:
                        return float(value)
                    except (ValueError, TypeError):
                        return None

                # Handle ref_range array format
                ref_min = None
                ref_max = None

                if "ref_range" in bm and bm["ref_range"] is not None:
                    ref_range = bm["ref_range"]
                    if isinstance(ref_range, list) and len(ref_range) >= 2:
                        ref_min = safe_float(ref_range[0])
                        ref_max = safe_float(ref_range[1])
                else:
                    ref_min = safe_float(bm.get("ref_min"))
                    ref_max = safe_float(bm.get("ref_max"))

                # Create biomarker
                biomarker = BiomarkerCreate(
                    report_id=UUID(report_id),
                    name=bm.get("name", ""),
                    value=float(bm.get("value", 0)),
                    unit=bm.get("unit"),
                    ref_min=ref_min,
                    ref_max=ref_max,
                    flag=bm.get("flag")
                )
                biomarkers.append(biomarker)

            # STEP 5: Bulk insert biomarkers
            biomarkers_result = create_biomarkers_bulk(biomarkers)

            if not biomarkers_result.get("success"):
                return {
                    "success": True,
                    "warning": f"Report created but biomarkers failed:
{biomarkers_result.get('error')}",
                    "data": {
                        "report": report_result["data"],
                        "biomarkers": []
                    }
                }

            return {
                "success": True,
                "data": {
                    "report": report_result["data"],
                    "biomarkers": biomarkers_result["data"]
                }
            }

        # No biomarkers in JSON
        return {
            "success": True,
            "data": {
                "report": report_result["data"],
                "biomarkers": []
            }
        }

    except Exception as e:
        return {"success": False, "error": str(e)}
```

5.2 Helper Functions

`create_report()`

```
def create_report(report: ReportCreate) -> dict:
    """Insert report into Supabase"""
    try:
        report_data = {
            "patient_id": str(report.patient_id),
            "file_id": report.file_id,
            "report_type": report.report_type,
            "gcs_path": report.gcs_path,
        }

        if report.sample_collected_at:
            report_data["sample_collected_at"] =
report.sample_collected_at.isoformat()

        response = supabase.table("reports").insert(report_data).execute()

        if response.data:
            return {"success": True, "data": response.data[0]}
        return {"success": False, "error": "Failed to create report"}
    except Exception as e:
        return {"success": False, "error": str(e)}
```

`create_biomarkers_bulk()`

```
def create_biomarkers_bulk(biomarkers: List[BiomarkerCreate]) -> dict:
    """Insert multiple biomarkers at once"""
    try:
        biomarker_data = []
        for biomarker in biomarkers:
            biomarker_data.append({
                "report_id": str(biomarker.report_id),
                "name": biomarker.name,
                "value": float(biomarker.value),
                "unit": biomarker.unit,
                "ref_min": float(biomarker.ref_min) if biomarker.ref_min is not None
else None,
                "ref_max": float(biomarker.ref_max) if biomarker.ref_max is not None
else None,
                "flag": biomarker.flag
            })

        response = supabase.table("biomarkers").insert(biomarker_data).execute()

        if response.data:
            return {"success": True, "data": response.data, "count":
len(response.data)}
        return {"success": False, "error": "Failed to create biomarkers"}
    except Exception as e:
        return {"success": False, "error": str(e)}
```

---

6. Code Flow Analysis

6.1 Complete Call Stack

```
1. User uploads PDF with NIC
   ↓
2. API endpoint (reports.py)
```

```
   - Looks up patient_id from NIC
   - Uploads to GCS
   - Triggers background worker
   ↓
3. Background worker (ocr_worker.py)
   - OCR with Document AI
   - Normalize data
   - Save JSON to GCS
   - Call store_normalized_report_to_db()
   ↓
4. store_normalized_report_to_db() (reportService.py)
   - Extract metadata
   - Create report record
   - Get report_id
   - Extract biomarkers
   - Create biomarker records
   ↓
5. Database (Supabase)
   - Insert into reports table
   - Insert into biomarkers table
   - Return created records
```

6.2 Data Flow Diagram

```
┌─────────────────────────┐
│ Normalized JSON (GCS)   │
│ users/{nic}/processed/  │
│ {file_id}_normalized.   │
│         json            │
└───────────┬─────────────┘
            │
            ↓ Read & Parse
┌───────────────────────────────────────────────┐
│ store_normalized_report_to_db()               │
│  ┌─────────────────────────────────────────┐  │
│  │ Extract:                                │  │
│  │  - report_type                          │  │
│  │  - sample_collected_at                  │  │
│  │  - biomarkers array                     │  │
│  └─────────────────────────────────────────┘  │
│                                               │
└───────────┬───────────────────────────────────┘
            │
            ↓
┌───────────────────────────┐
│ Create Report Record      │
│  ┌─────────────────────┐  │
│  │ reports table       │  │
│  │ ┌─────────────────┐ │  │
│  │ │ patient_id (FK) │ │  │
│  │ │ file_id         │ │  │
│  │ │ report_type     │ │  │
│  │ │ sample_date     │ │  │
│  │ │ gcs_path        │ │  │
│  │ └──────────────────┘ │  │
│  └──────────────────────┘  │
│                            │
│  Returns: report_id        │
└────────────────────────────┘
            │
            │
            ↓
┌───────────────────────────────────────┐
│  Process Biomarkers Array             │
│  ┌─────────────────────────────────┐  │
│  │ For each biomarker:             │  │
│  │  - Extract name                 │  │
│  │  - Extract value                │  │
│  │  - Extract unit                 │  │
│  │  - Extract ref_range[0,1]       │  │
│  │    → Convert to ref_min/max     │  │
│  │  - Extract flag (if exists)     │  │
│  └─────────────────────────────────┘  │
└───────────────────────────────────────┘
           │
           │
           ↓
┌───────────────────────────────┐
│  Bulk Insert Biomarkers       │
│  ┌─────────────────────────┐  │
│  │ biomarkers table        │  │
│  │ ┌─────────────────────┐ │  │
│  │ │ report_id (FK)      │ │  │
│  │ │ name                │ │  │
│  │ │ value               │ │  │
│  │ │ unit                │ │  │
│  │ │ ref_min             │ │  │
│  │ │ ref_max             │ │  │
│  │ │ flag                │ │  │
│  │ └─────────────────────┘ │  │
│  │                         │  │
│  │ (Multiple rows)         │  │
│  └─────────────────────────┘  │
└───────────────────────────────┘
           │
           │
           ↓
     Return Success
```

7. Example Walkthroughs

7.1 Example 1: FBC Report

Input JSON (from GCS)

```json
{
  "patient": {
    "name": "MR S KUMAR",
    "age_years": 62,
    "gender": "Male"
  },
  "report": {
    "type": "Full Blood Count",
    "sample_collected_at": "2025-06-03T09:10:00"
  },
  "biomarkers": [
    {
      "name": "WBC",
      "value": 7970.0,
      "unit": "per cu mm",
      "ref_range": [4000.0, 11000.0]
    },
    {
      "name": "Hemoglobin",
      "value": 13.5,
      "unit": "g/dL",
      "ref_range": [13.0, 17.0]
    },
    {
      "name": "Platelets",
      "value": 250000.0,
      "unit": "per cu mm",
      "ref_range": [150000.0, 450000.0]
    }
  ]
}
```

Processing Steps

Step 1: Extract Metadata

```
report_type = "Full Blood Count"
sample_date = datetime(2025, 6, 3, 9, 10, 0)
```

Step 2: Create Report

```
INSERT INTO reports (
  patient_id,
  file_id,
  report_type,
  sample_collected_at,
  gcs_path
) VALUES (
  '550e8400-e29b-41d4-a716-446655440000',
  'fbc-2025-06-03',
  'Full Blood Count',
  '2025-06-03 09:10:00',
  'gs://healix/users/199512345678/reports/fbc-2025-06-03.pdf'
) RETURNING id;

-- Returns: report_id = 'a1b2c3d4-e5f6-7890-abcd-ef1234567890'
```

Step 3: Process Biomarkers

```
# Biomarker 1: WBC
{
  "report_id": "a1b2c3d4-...",
  "name": "WBC",
  "value": 7970.0,
  "unit": "per cu mm",
  "ref_min": 4000.0,
  "ref_max": 11000.0,
  "flag": None
}

# Biomarker 2: Hemoglobin
{
  "report_id": "a1b2c3d4-...",
  "name": "Hemoglobin",
  "value": 13.5,
  "unit": "g/dL",
  "ref_min": 13.0,
  "ref_max": 17.0,
  "flag": None
}

# Biomarker 3: Platelets
{
  "report_id": "a1b2c3d4-...",
  "name": "Platelets",
  "value": 250000.0,
  "unit": "per cu mm",
  "ref_min": 150000.0,
  "ref_max": 450000.0,
  "flag": None
}
```

Step 4: Bulk Insert

```
INSERT INTO biomarkers (
  report_id, name, value, unit, ref_min, ref_max, flag
) VALUES
  ('a1b2c3d4-...', 'WBC', 7970.0, 'per cu mm', 4000.0, 11000.0, NULL),
  ('a1b2c3d4-...', 'Hemoglobin', 13.5, 'g/dL', 13.0, 17.0, NULL),
  ('a1b2c3d4-...', 'Platelets', 250000.0, 'per cu mm', 150000.0, 450000.0, NULL)
RETURNING *;
```

Database Result

reports table:

id	patient_id	file_id	report_type	sample_collected_at	gcs_path	created_at
a1b2...	550e...	fbc-2025...	Full Blood Count	2025-06-03 09:10:00	gs://...	2026-02-07 14:00:00

biomarkers table:

id	report_id	name	value	unit	ref_min	ref_max	flag
bio1...	a1b2...	WBC	7970	per cu mm	4000	11000	NULL
bio2...	a1b2...	Hemoglobin	13.5	g/dL	13	17	NULL
bio3...	a1b2...	Platelets	250000	per cu mm	150000	450000	NULL

7.2 Example 2: Lipid Profile

Input JSON

```json
{
  "report": {
    "type": "Serum Lipid Profile",
    "sample_collected_at": "2025-06-15T08:30:00"
  },
  "biomarkers": [
    {
      "name": "Total Cholesterol",
      "value": 220.0,
      "unit": "mg/dL",
      "flag": "High",
      "ref_range": [125.0, 200.0]
    },
    {
      "name": "HDL Cholesterol",
      "value": 45.0,
      "unit": "mg/dL",
      "flag": "Low",
      "ref_range": [40.0, 60.0]
    },
    {
      "name": "LDL Cholesterol",
      "value": 150.0,
      "unit": "mg/dL",
      "flag": "High",
      "ref_range": [0.0, 100.0]
    }
  ]
}
```

Database Result

biomarkers table:

id	report_id	name	value	unit	ref_min	ref_max	flag
bio1...	lipid1...	Total Cholesterol	220	mg/dL	125	200	High
bio2...	lipid1...	HDL Cholesterol	45	mg/dL	40	60	Low
bio3...	lipid1...	LDL Cholesterol	150	mg/dL	0	100	High

Note: Lipid profiles include `flag` field with clinical interpretation.

---

8. Error Handling

8.1 Error Scenarios

Scenario 1: Invalid JSON Format

```
# Missing biomarkers key
normalized_json = {
  "report": {"type": "FBC"}
  # No "biomarkers" key
}

# Handling
biomarkers_data = normalized_json.get("biomarkers", [])
# Result: [] (empty array)
# Process continues, creates report with 0 biomarkers
```

Scenario 2: Invalid Reference Range

```
# String instead of array
bm = {
  "name": "WBC",
  "value": 7970.0,
  "ref_range": "4000-11000"  # String, not array
}

# Handling
if isinstance(ref_range, list) and len(ref_range) >= 2:
    ref_min = float(ref_range[0])  # Skipped
else:
    ref_min = None  # Used instead

# Result: ref_min = None, ref_max = None
```

Scenario 3: Missing Value

```
bm = {
  "name": "WBC",
  # No "value" key
  "unit": "per cu mm"
}

# Handling
value = float(bm.get("value", 0))  # Uses default: 0
# Result: value = 0.0 (stored in database)
```

Scenario 4: Database Constraint Violation

```
# Duplicate file_id
report_data = {
  "file_id": "existing-file-id",  # Already exists
  ...
}

# Result
# Supabase raises exception
# Caught in try-except
return {"success": False, "error": "duplicate key value..."}
```

8.2 Error Recovery

```
# Report created but biomarkers failed
try:
    create_report(...)  # Success
    create_biomarkers_bulk(...)  # Fails
except:
    # Don't rollback report
    # Return partial success with warning
    return {
        "success": True,
        "warning": "Report created but biomarkers failed",
        "data": {
            "report": ...,
            "biomarkers": []
        }
    }
```

---

9. Testing & Validation

9.1 Manual Testing

```
-- 1. Check report was created
SELECT * FROM reports WHERE file_id = 'test-file-123';

-- 2. Check biomarkers were created
```

```sql
SELECT r.file_id, r.report_type, COUNT(b.id) as biomarker_count
FROM reports r
LEFT JOIN biomarkers b ON r.id = b.report_id
WHERE r.file_id = 'test-file-123'
GROUP BY r.file_id, r.report_type;

-- 3. Check reference ranges
SELECT name, value, unit, ref_min, ref_max, flag
FROM biomarkers
WHERE report_id = (
  SELECT id FROM reports WHERE file_id = 'test-file-123'
)
ORDER BY name;

-- 4. Check data integrity
SELECT
  'Missing ref_min' as issue,
  COUNT(*) as count
FROM biomarkers
WHERE ref_min IS NULL AND ref_max IS NOT NULL

UNION ALL

SELECT
  'Negative values' as issue,
  COUNT(*) as count
FROM biomarkers
WHERE value < 0;
```

9.2 Validation Queries

```sql
-- Verify all reports have biomarkers
SELECT r.id, r.file_id, r.report_type, COUNT(b.id) as biomarker_count
FROM reports r
LEFT JOIN biomarkers b ON r.id = b.report_id
GROUP BY r.id, r.file_id, r.report_type
HAVING COUNT(b.id) = 0;

-- Check for orphaned biomarkers (shouldn't exist due to FK)
SELECT b.* FROM biomarkers b
LEFT JOIN reports r ON b.report_id = r.id
WHERE r.id IS NULL;

-- Verify patient linkages
SELECT p.full_name, p.nic, COUNT(r.id) as report_count
FROM patients p
LEFT JOIN reports r ON p.id = r.patient_id
GROUP BY p.id, p.full_name, p.nic;
```

Summary

Key Takeaways

1. **Source**: Normalized JSON from GCS (`users/{nic}/processed/{file_id}_normalized.json`)
2. **Process**:
   - Extract report metadata → Create report record
   - Extract biomarkers array → Process each biomarker
   - Handle `ref_range` array → Convert to `ref_min`/`ref_max`
   - Bulk insert biomarkers
3. **Tables**:
   - `reports`: 1 row per report
   - `biomarkers`: N rows per report (linked via `report_id`)
4. **Error Handling**:
   - Graceful defaults for missing data
   - Partial success if biomarkers fail
   - Database constraints enforce integrity
5. **Performance**:
   - Bulk insert for biomarkers (efficient)
   - Single transaction per report
   - Indexed foreign keys for fast queries

This document explains the complete pipeline from GCS JSON to database storage.

For more details:

- Code: `app/services/reportService.py`
- Schema: See database section
- Examples: See walkthroughs above