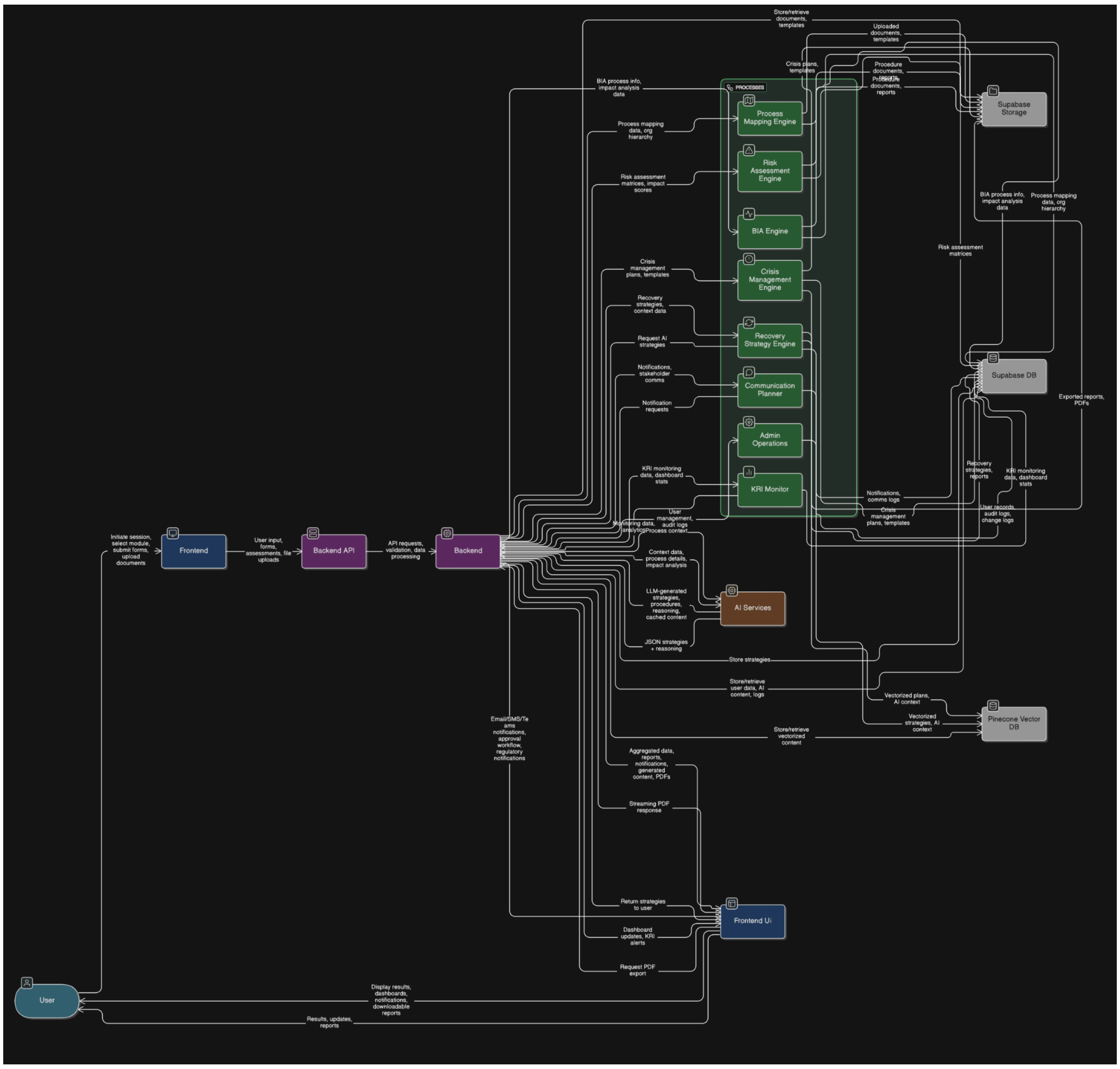
Level-1 Data Flow Diagram for the BCM System


User
Initiate session, select module, submit forms, upload documents
Frontend
User input, forms, assessments, file uploads
Backend API
API requests, validation, data processing
Backend
PROCESSES
Process Mapping Engine
Risk Assessment Engine
BIA Engine
Crisis Management Engine
Recovery Strategy Engine
Communication Planner
Admin Operations
KRI Monitor
Supabase Storage
Supabase DB
Pinecone Vector DB
AI Services
Frontend Ui
Store/retrieve documents, templates
Uploaded documents, templates
Crisis plans, templates
Procedure documents, reports
Procedure documents, reports
BIA process info, impact analysis data
Process mapping data, org hierarchy
Risk assessment matrices, impact scores
Crisis management plans, templates
Recovery strategies, context data
Request AI strategies
Notifications, stakeholder comms
Notification requests
KRI monitoring data, dashboard stats
User management, audit logs
Monitoring data, analytics
Process context
BIA process info, impact analysis data
Process mapping data, org hierarchy
Risk assessment matrices
Exported reports, PDFs
Recovery strategies, reports
KRI monitoring data, dashboard stats
Notifications, comms logs
Crisis management plans, templates
User records, audit logs, change logs
Context data, process details, impact analysis
LLM-generated strategies, procedures, reasoning, cached content
JSON strategies + reasoning
Store strategies
Store/retrieve user data, AI content, logs
Vectorized plans, AI context
Vectorized strategies, AI context
Store/retrieve vectorized content
Email/SMS/Teams notifications, approval workflow, regulatory notifications
Aggregated data, reports, notifications, generated content, PDFs
Streaming PDF response
Return strategies to user
Dashboard updates, KRI alerts
Request PDF export
Display results, dashboards, notifications, downloadable reports
Results, updates, reports

Expanded Explanation of the High-Level BCM System Data Flow Diagram

The High-Level Business Continuity Management (BCM) System Data Flow Diagram provides a comprehensive overview of how information travels within the platform—from the moment a user interacts with the interface to the point where data gets processed, analyzed, and stored within the system's backend infrastructure. This diagram serves as a foundational reference point for understanding how each subsystem communicates, how processes are integrated, and how the architecture maintains reliability, scalability, and security across the entire lifecycle of BCM operations.

---

1. Frontend Layer – User Interaction & Data Capture

The data flow begins at the **Frontend**, which functions as the direct touchpoint between users and the BCM platform. This layer is responsible for collecting structured inputs, providing interactive dashboards, and allowing users to perform key activities such as:

- **User Authentication** – Login, access control validations, role-based permissions.
- **BIA (Business Impact Analysis) Form Submission** – Users provide detailed information on processes, criticality, dependencies, tolerable downtime, and resource requirements.

- **Risk Assessment Inputs** – Users submit risk likelihoods, severity scores, existing controls, and mitigation strategies.
- **Crisis Management Triggers/Updates** – Incident declarations, status updates, communication logs.
- **Recovery Strategy Inputs** – Selection or creation of recovery plans, RTO/RPO definitions, resource allocations, and escalation flows.

The frontend layer does **not** handle any business logic. Instead, it serves as the data acquisition and visualization interface. Its primary responsibility is to validate basic form data, ensure usability, and send structured, clean JSON payloads to the backend via API routes. This separation ensures a secure, reliable, and modular workflow.

---

2. Backend API Layer – Core Routing, Validation & Orchestration

Once a request leaves the frontend, it enters the **Backend API Layer**, which acts as the **traffic controller** for all communications. This layer contains:

- Authentication middleware
- Input validation
- Authorization checks
- Routing logic
- Request throttling & caching
- Error handling
- Orchestration for downstream modules

The API layer ensures that every incoming request is:

1. Authenticated (via JWT/session tokens)
2. Verified (data structure, types, missing fields)
3. Authorized (user permissions, roles)
4. Routed to the correct internal engine

It also formats consistent API responses, ensuring predictable output for all clients (frontend/UI, admin consoles, automation scripts).

In many ways, the API layer is the “nervous system” of the BCM platform—it coordinates which internal engine should process each request while enforcing compliance, security, and auditability.

---

3. Processing Engines – Modular Business Logic Execution

The system incorporates several specialized “Processing Engines,” each designed to handle a specific BCM domain. These engines encapsulate complex logic, formulas, risk calculations, decision matrices, and automated workflows.

a. BIA Engine

Handles:

- Business impact calculation
- RTO/RPO derivation
- Process criticality scoring
- Dependency mapping
- Auto-generation of impact heatmaps

b. Risk Assessment Engine

Includes:

- Threat likelihood/severity scoring
- Inherent vs. residual risk calculations
- Control efficiency evaluation
- Auto-mapped mitigation strategies

c. Crisis Management Engine

Processes:

- Incident creation & escalation rules
- Communication workflows
- Status updates
- Roles & responsibilities alignment
- Automated notifications

d. Recovery Strategy Engine

Responsible for:

- Strategy selection recommendations
- Recovery resource mapping
- Alternative site readiness checks
- Restoration workflow logic
- Gap analysis

Each engine operates in isolation to maintain modularity. They communicate with one another only through well-defined API contracts. This makes the architecture scalable and allows future engines or microservices to be added without disrupting existing functionality.

4. Data Storage & Persistence Layer – Structured and Unstructured Data Handling

After the processing engines complete their computations, the results are passed into the **data sinks**, which consist of both structured and unstructured storage repositories:

a. Supabase (Postgres-based Structured Storage)

Used for:

- BIA records
- Process inventory
- Risk registers
- Strategy matrices
- User accounts, roles, permissions
- Audit logs (structured entries)

This ensures ACID-compliant, relational data storage suitable for analytics and complex joins.

b. MongoDB (Unstructured & Semi-structured Storage)

Stores:

- Large JSON logs

- Crisis communication transcripts
- Workflow execution logs
- Historical data snapshots
- Attachments or multimedia references

The use of MongoDB ensures flexibility and scalability for unpredictable or evolving data formats.

---

5. End-to-End Data Movement Summary

Putting it all together, the data flow can be summarized as:

1. **User submits data through the frontend UI.**
2. The **Backend API** receives the request, performs security checks, and determines the appropriate engine.
3. The request is processed within **one or more specialized engines**, which apply BCM logic and computations.
4. Results are written either to **Supabase (structured)** or **MongoDB (unstructured)** depending on the data type.
5. The API returns computed results, update confirmations, or next steps back to the frontend.
6. The UI displays updated dashboards, visualizations, and recommendations.

This structured flow ensures that the BCM platform remains efficient, reliable, modular, and capable of handling complex, multi-layered business continuity operations in real time.

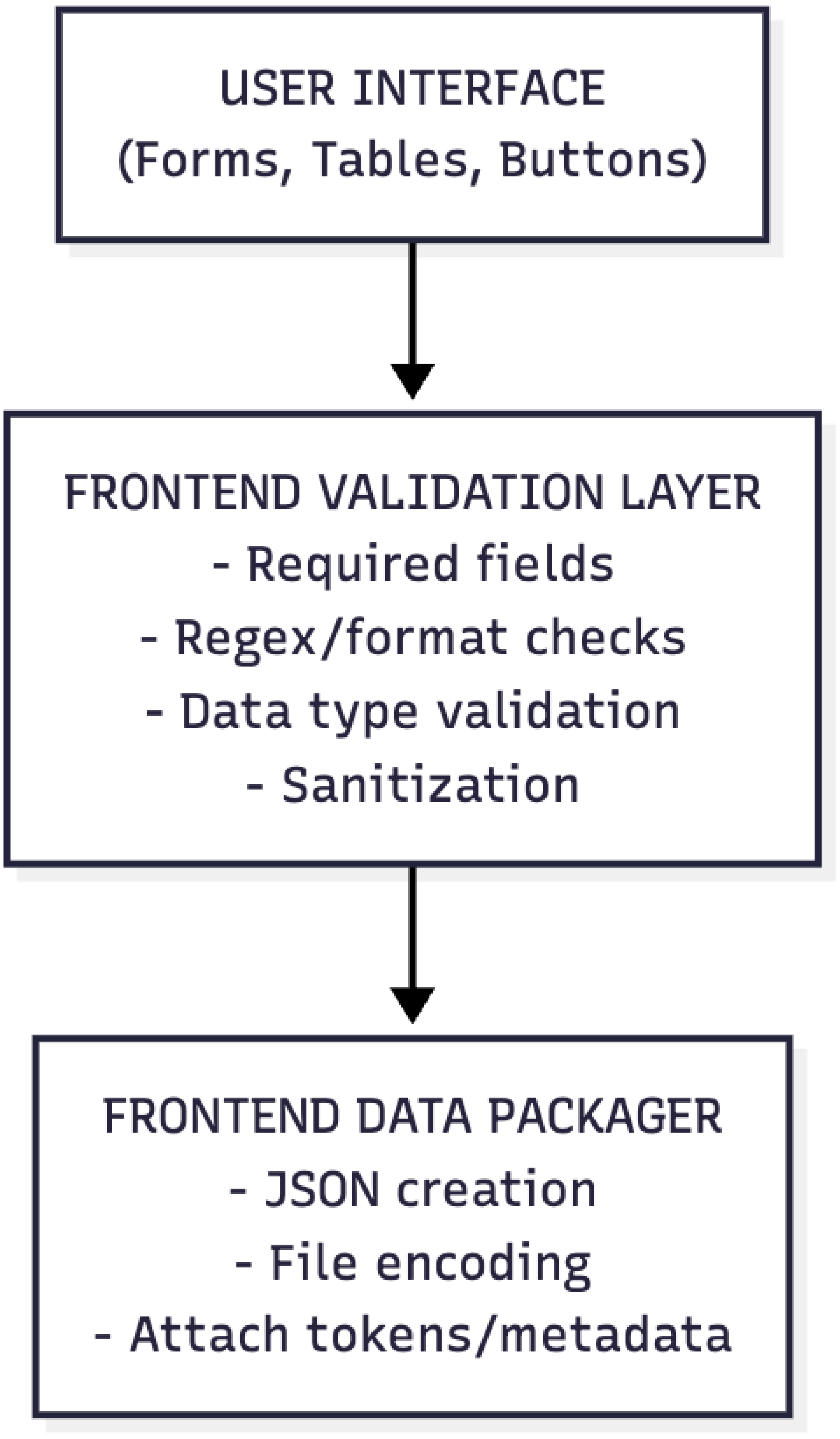
1. Frontend (Low-Level Breakdown)

The frontend is the first touchpoint for all user actions. It does preprocessing, client-side validation, and ensures only clean, structured data is sent to the backend.

1.1 Input Handling & Pre-Validation

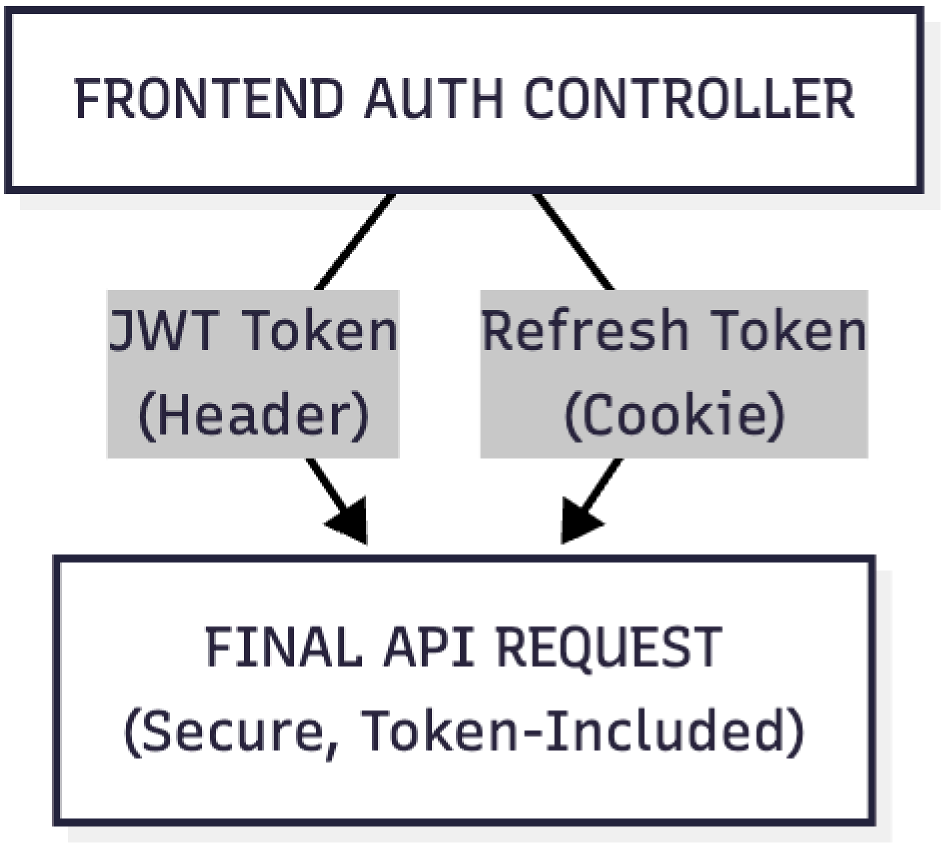
Before any request goes to the API, the frontend performs several checks:

- Required fields check (e.g., username/password, BIA numeric fields)
- Format checks (numbers only, date ranges, dropdown selections)
- Prevention of malformed submissions
- Pre-sanitization of input
    - trimming
    - escaping
    - removing dangerous characters


USER INTERFACE
(Forms, Tables, Buttons)
FRONTEND VALIDATION LAYER
- Required fields
- Regex/format checks
- Data type validation
- Sanitization
FRONTEND DATA PACKAGER
- JSON creation
- File encoding
- Attach tokens/metadata

1.2 Token Injection & Security Measures

- JWT tokens added to headers
- Refresh tokens retained in httpOnly cookies
- XSRF checks
- No sensitive data stored locally



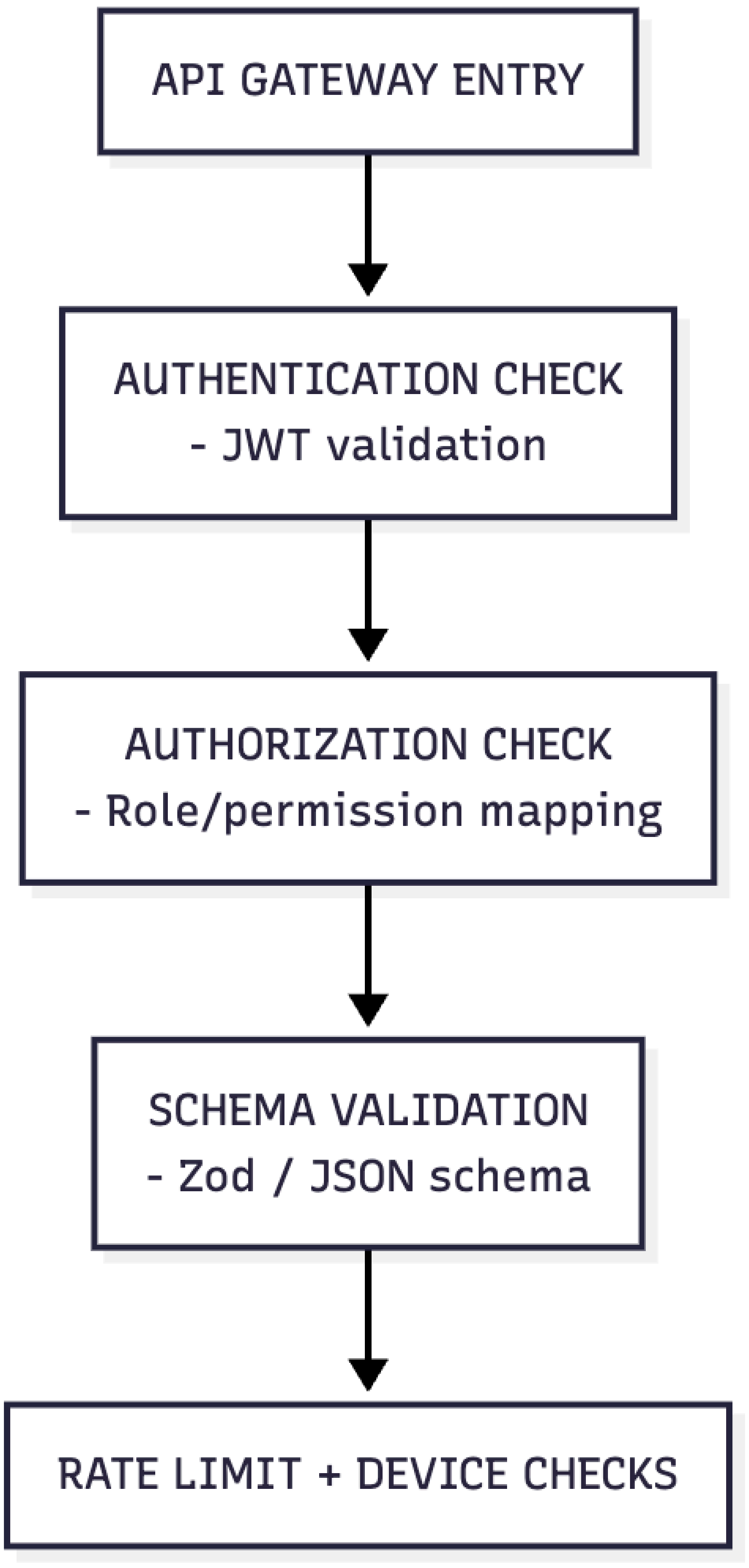
2. Backend API Gateway (Low-Level Breakdown)

The API acts as a routing, validation, transformation, and orchestration layer.

2.1 Request Verification Layer

Every request flows through:

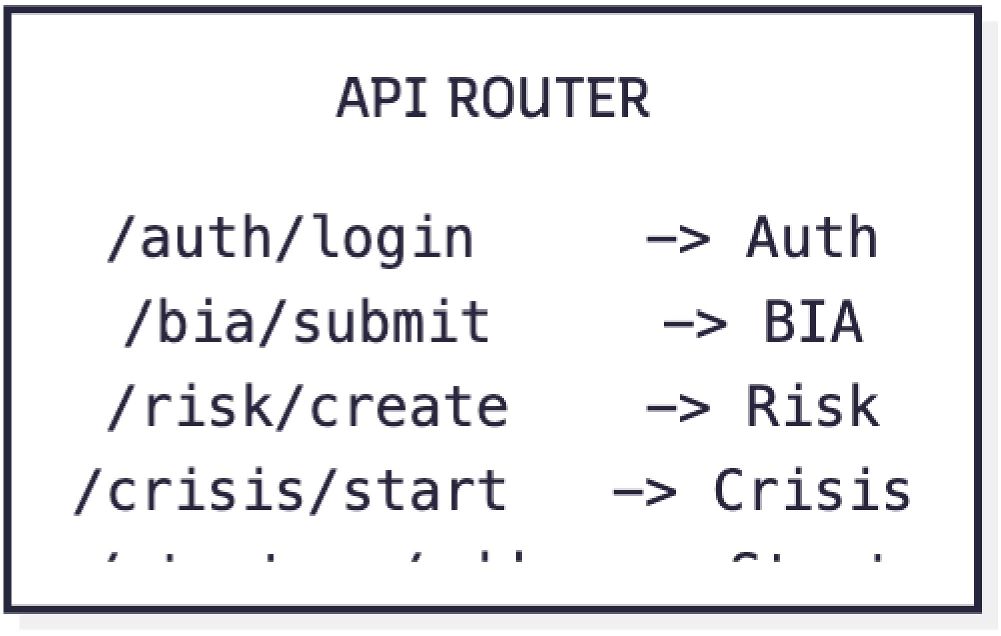
- JWT Verification
- Permission Authorization
- Schema Validation
- Rate Limiting
- IP/Device fingerprint checks


API GATEWAY ENTRY
AUTHENTICATION CHECK
- JWT validation
AUTHORIZATION CHECK
- Role/permission mapping
SCHEMA VALIDATION
- Zod / JSON schema
RATE LIMIT + DEVICE CHECKS

2.2 Routing and Dispatch

Based on the request URL, the API determines which engine should handle the request:

- /auth/* → Authentication Engine
- /bia/* → BIA Engine
- /risk/* → Risk Engine
- /crisis/* → Crisis Engine
- /strategy/* → Recovery Strategy Engine



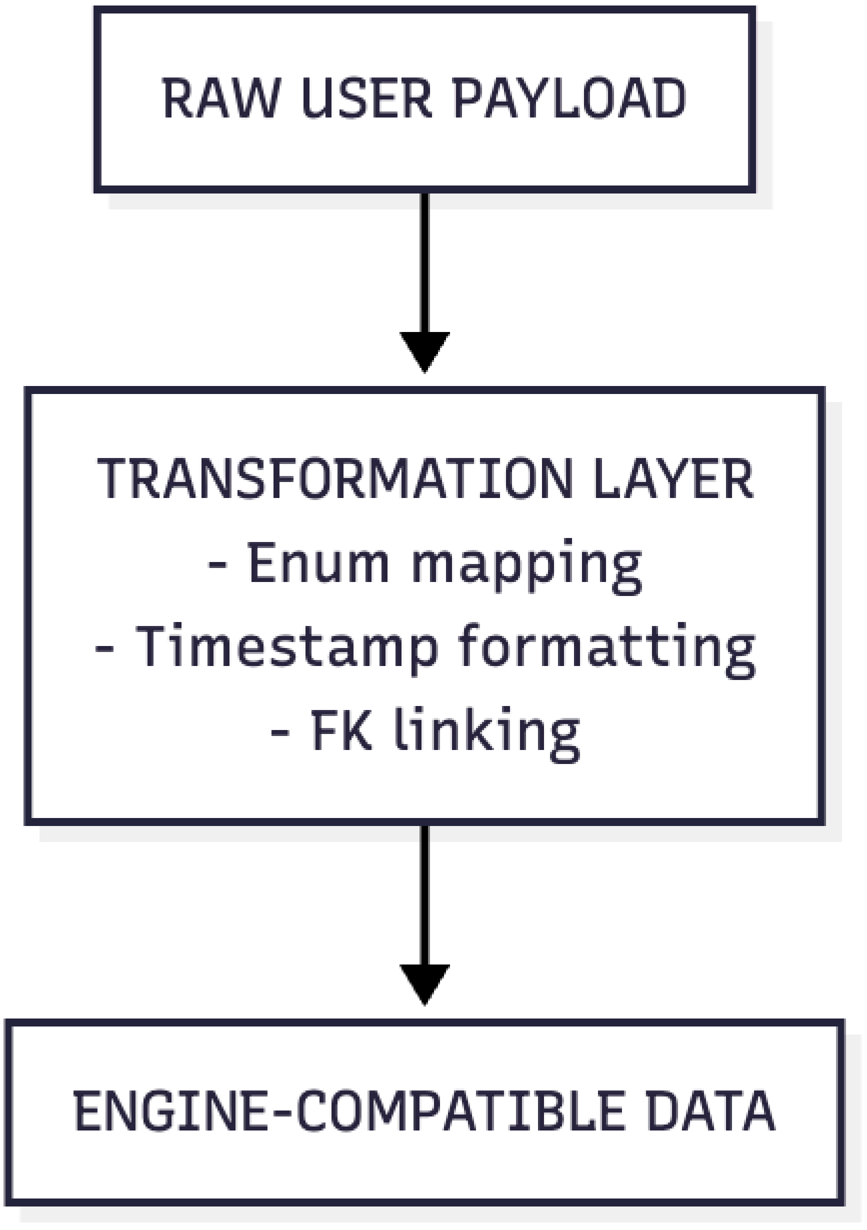
```
API ROUTER

/auth/login      -> Auth
/bia/submit      -> BIA
/risk/create     -> Risk
/crisis/start    -> Crisis
```

2.3 Data Transformation Layer

Before data reaches any engine:

- Does normalization
- Maps foreign keys

- Standardizes enums
- Converts timestamps
- Calculates data deltas

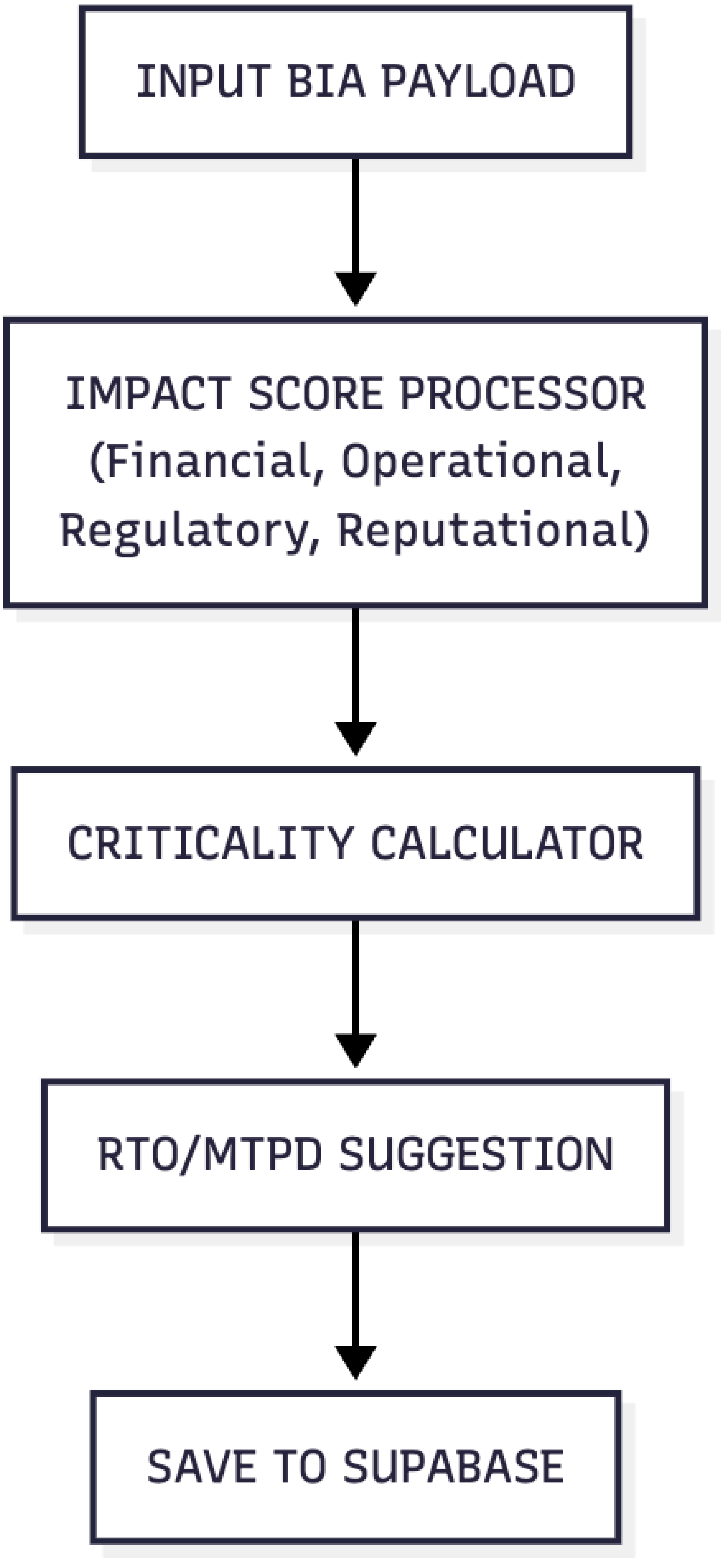


3. Processing Engines (Low-Level Breakdown)

Each engine performs computational or rule-based logic.

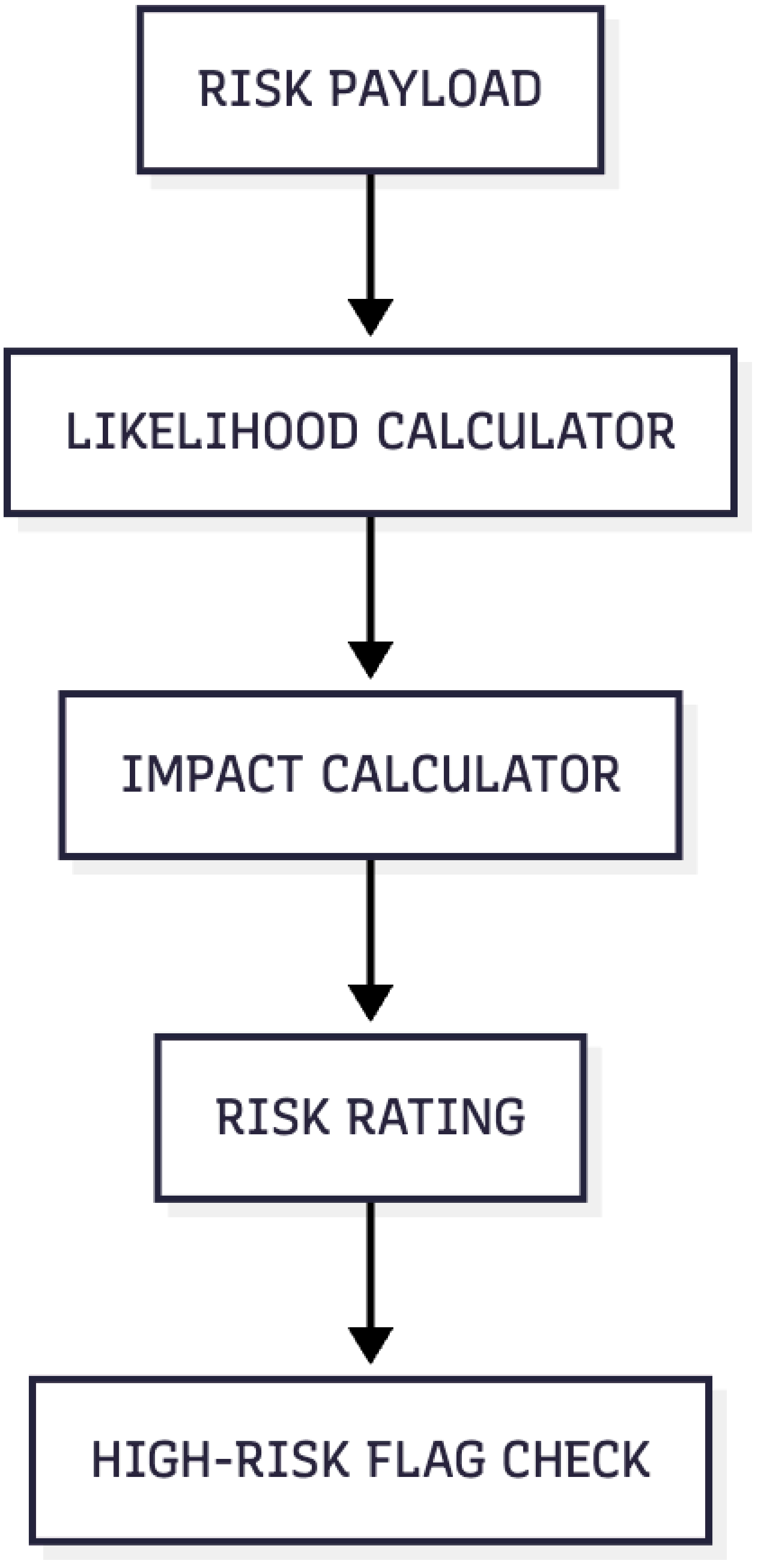
3.1 BIA Engine (Low-Level)

- Validates BIA numeric inputs
- Computes individual impact scores
- Computes weighted criticality
- Suggests RTO/MTPD
- Stores data in Supabase
- Triggers heatmap recalculation


INPUT BIA PAYLOAD
IMPACT SCORE PROCESSOR (Financial, Operational, Regulatory, Reputational)
CRITICALITY CALCULATOR
RTO/MTPD SUGGESTION
SAVE TO SUPABASE

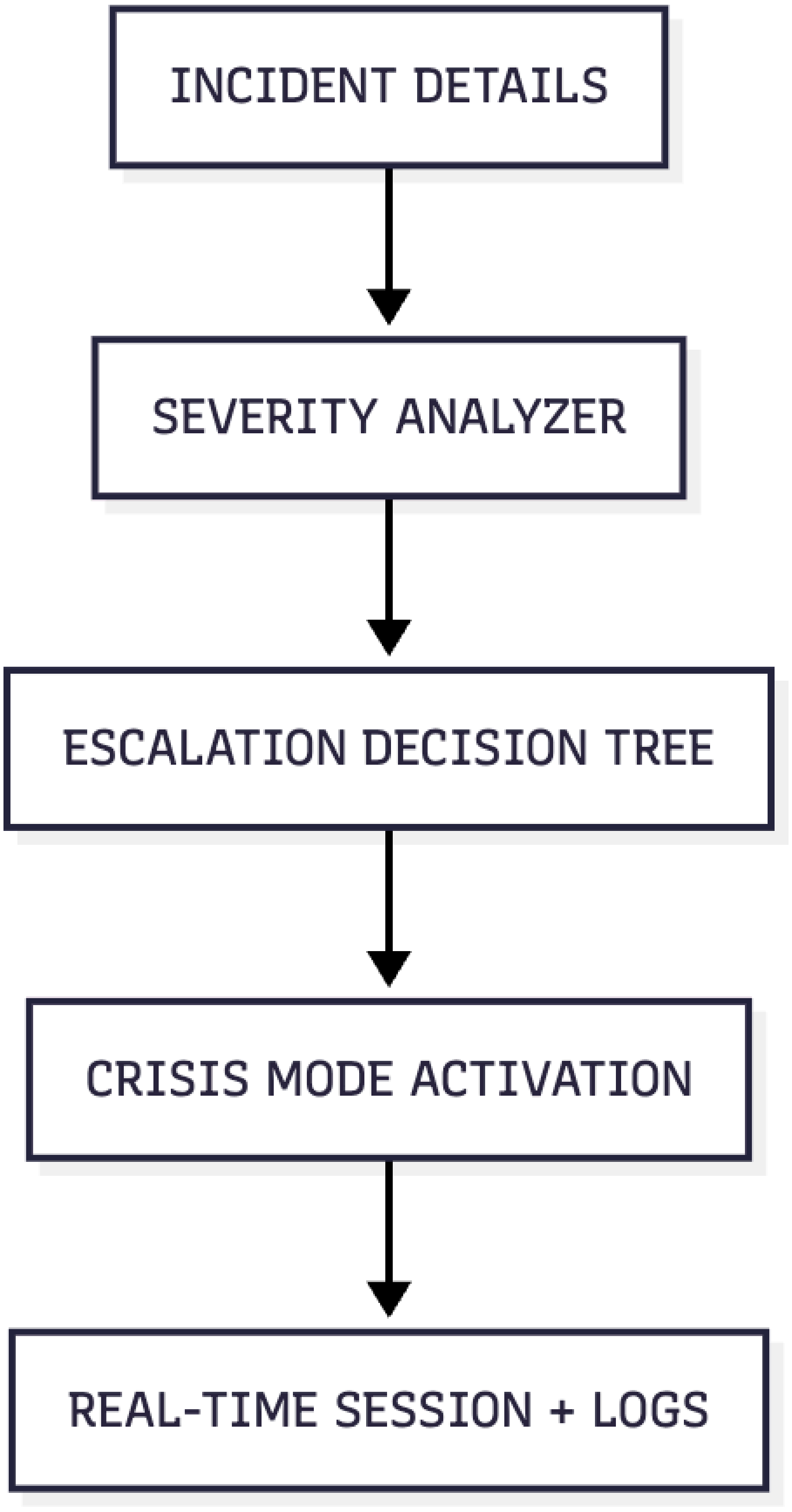
3.2 Risk Assessment Engine (Low-Level)

- Maps risk to process
- Calculates likelihood
- Calculates impact
- Computes risk rating
- Flags high-risk items
- Rebuilds risk heatmap


RISK PAYLOAD
LIKELIHOOD CALCULATOR
IMPACT CALCULATOR
RISK RATING
HIGH-RISK FLAG CHECK

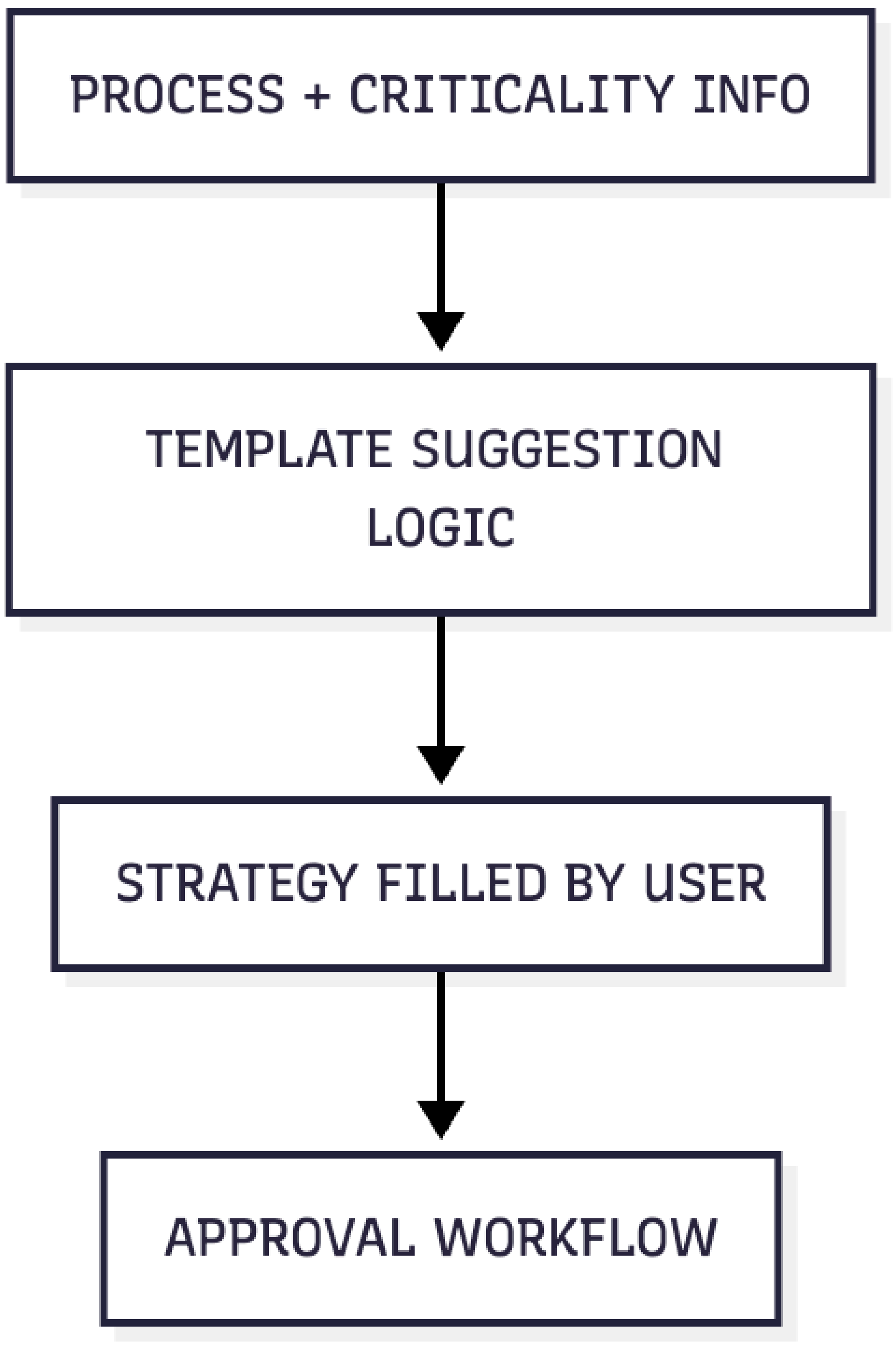
3.3 Crisis Management Engine (Low-Level)

- Assigns severity
- Checks escalation rules
- Activates crisis mode
- Logs every action
- Creates real-time crisis room
- Stores event logs in MongoDB


INCIDENT DETAILS
SEVERITY ANALYZER
ESCALATION DECISION TREE
CRISIS MODE ACTIVATION
REAL-TIME SESSION + LOGS

3.4 Recovery Strategy Engine (Low-Level)

- Collects dependencies
- Recommends strategy templates
- Validates timeline/resource structure
- Final approval workflow
- Saves strategy to Supabase



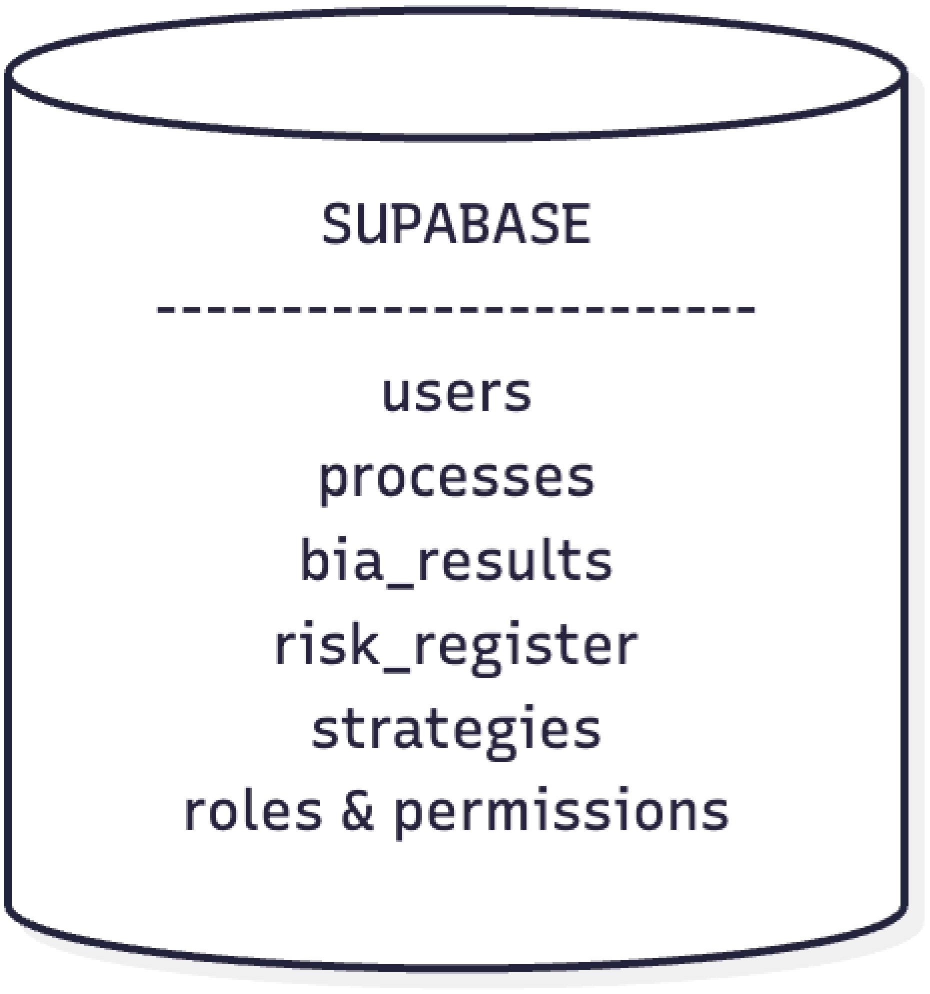
4. Data Sinks (Low-Level Breakdown)

Two main persistence layers:

4.1 Supabase (Structured Data)

Stores:

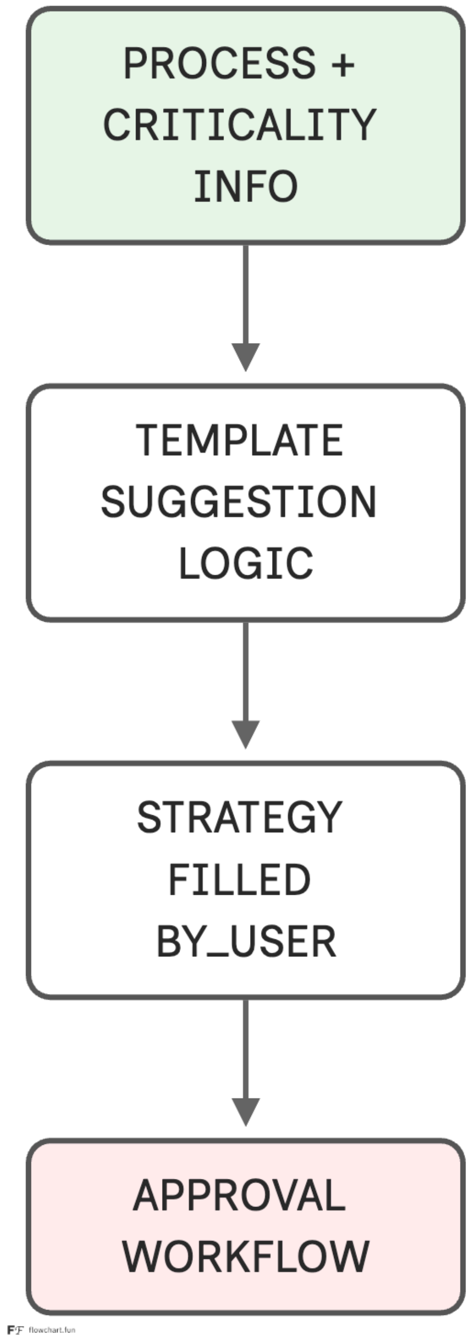
- Users
- Processes
- BIA reports
- Risk registers
- Recovery plans
- Metadata



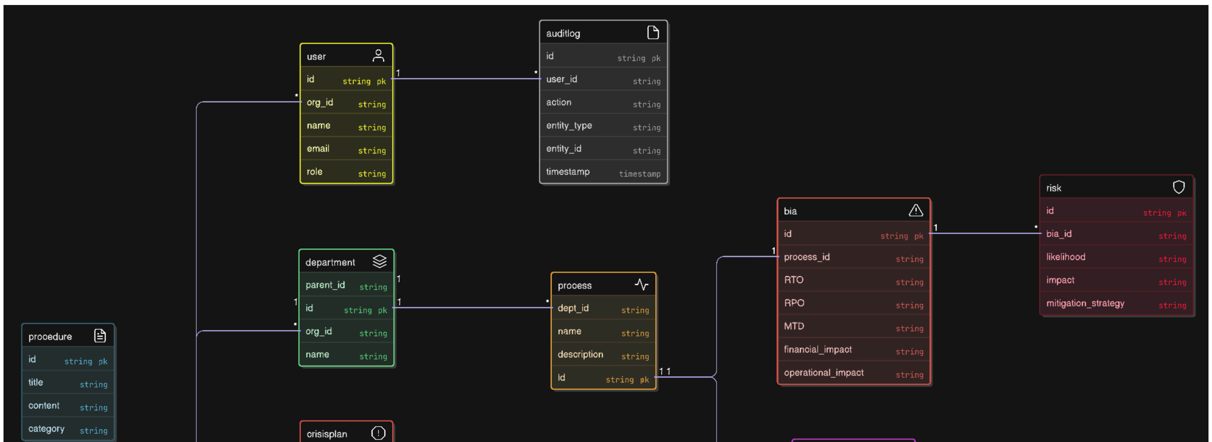
4.2 MongoDB (Unstructured Data)

Stores:

- Crisis logs
- Audit trails
- Chat messages
- Activity streams



Entity Relationship Diagram :



Entity Relationship Diagram (ERD)

The **Entity Relationship Diagram (ERD)** serves as the structural blueprint of the BCM (Business Continuity Management) system's database architecture. It provides a visual representation of how data is organized, how various entities interact with each other, and the logical relationships governing how information flows across modules such as BIA, Risk Assessment, Crisis Management, and Recovery Strategy.

A well-designed ERD ensures that the system maintains data consistency, avoids redundancy, enforces integrity constraints, and remains scalable as new features or data domains are added. This diagram is foundational for developers, architects, and analysts as it dictates how all BCM modules interact through shared or interconnected data tables.

---

1. Purpose of the ERD in the BCM System

At a foundational level, the ERD answers the question: **"How is information structured and stored within the BCM platform?"**
The diagram clarifies:

- What data entities exist (e.g., Processes, Procedures, Risks, Departments).
- What attributes belong to each entity.
- How entities relate to one another (One-to-Many, Many-to-Many).
- Which tables act as core vs. lookup/reference tables.
- How BCM data flows between modules.

Because BCM is deeply interconnected, having an ERD prevents data silos and ensures modules such as BIA, RTO/MTPD, and Strategies reference consistent datasets.

---

2. Key Entities Within the BCM ERD

a. Procedures (Standard Operating Procedures Table)

This entity contains all formalized SOPs defining how an organization responds to incidents, executes tasks, and manages critical functions.
Typical attributes:

- Procedure ID
- Procedure Title
- Description
- Associated Process
- Owner / Department
- Last Updated Timestamp

Procedures often link to **Processes**, because SOPs dictate what must be done if a process is disrupted.

---

b. RTO / MTPD Tables (Time-Critical Metrics Storage)

RTO (Recovery Time Objective) and MTPD (Maximum Tolerable Period of Disruption) are essential BCM metrics.
These tables store time sensitivity values for each process or sub-process.

Attributes often include:

- Metric ID
- Process ID (Foreign Key)
- RTO (in hours/minutes)
- MTPD (in hours/days)
- Criticality Rating
- Additional Notes

These metrics directly influence risk scoring, recovery strategies, dependency prioritization, and resource planning.

---

c. Region / Type (Lookup Reference Tables)

These are **standardized classification tables** used across multiple modules.
Examples include:

- **Region:** APAC, EMEA, NA, India, etc.
- **Process Type:** IT process, HR process, Financial process, Customer-facing process.
- **Risk Type:** Operational, Cyber, Environmental, Third-party, etc.

Lookup tables ensure data consistency and prevent typos or mismatched naming conventions when users select categories. They are linked via foreign keys to Processes, Departments, Risks, and other entities.

---

3. Organizational Hierarchy Entities

A BCM system usually models the organization structurally:

- **Organization → Departments → Processes → Sub-processes**

a. Organization

Represents the top-level entity.
Attributes:

- Org ID
- Organization Name

- Industry
- Address
- Admin Users

b. Department

A single organization may have dozens of departments, making this a **One-to-Many** relationship.
Attributes include:

- Department ID
- Department Name
- Department Head
- Region (FK)

c. Process

Each department performs many processes—again a **One-to-Many** relationship.
Processes are central to the BCM system because BCM analysis revolves around their criticality, dependencies, and vulnerabilities.

Attributes:

- Process ID
- Process Name
- Criticality Score
- Dependency Information

- RTO/MTPD (FK)
- Type (FK)

Processes also form parent–child relationships through sub-processes or activities.

---

4. Risk and Assessment Entities

a. Risk Entity

Each process can have multiple risks, forming a **One Process → Many Risks** relationship.
A risk may include:

- Risk ID
- Risk Description
- Likelihood Score
- Impact Score
- Existing Controls
- Residual Risk Rating
- Risk Type (FK)

b. Control / Mitigation Entity

Controls are often linked to Risks.
One Risk may have multiple Controls.
Example attributes:

- Control ID
- Control Description
- Control Owner
- Effectiveness Rating

---

5. Crisis & Recovery Strategy Entities

a. Crisis Event Entity

Holds incidents or crisis declarations.
Linked to:

- Processes impacted
- Departments
- Logs
- Communications

b. Recovery Strategy Entity

Each Process may have multiple recovery strategies depending on scenario severity.
Attributes:

- Strategy ID
- Process ID
- Recovery Method

- RTO Alignment
- Resource Requirements
- Alternate Site Information

---

6. Core Relationships Visualized in the ERD

The ERD models several important relationships:

✔ One Organization → Many Departments

Ensures organizational-level grouping.

✔ One Department → Many Processes

Reflects operational structure.

✔ One Process → Many Risks

Aligns with BCM risk methodology.

✔ One Process → One or Many SOPs (Procedures)

Depending on complexity.

✔ One Process → One RTO/MTPD Record

Each process has a defined criticality metric.

✔ Lookup Tables → Referenced by Many Entities

Ensures standardization for:

- Region
- Type

- Risk Category

✔ Many-to-Many Cases (Handled via Junction Tables)

Examples:

- Processes ↔ Dependencies
- Risks ↔ Controls
- Crisis Events ↔ Impacted Processes

These junction tables maintain referential integrity and prevent duplication.

---

7. Why the ERD Matters

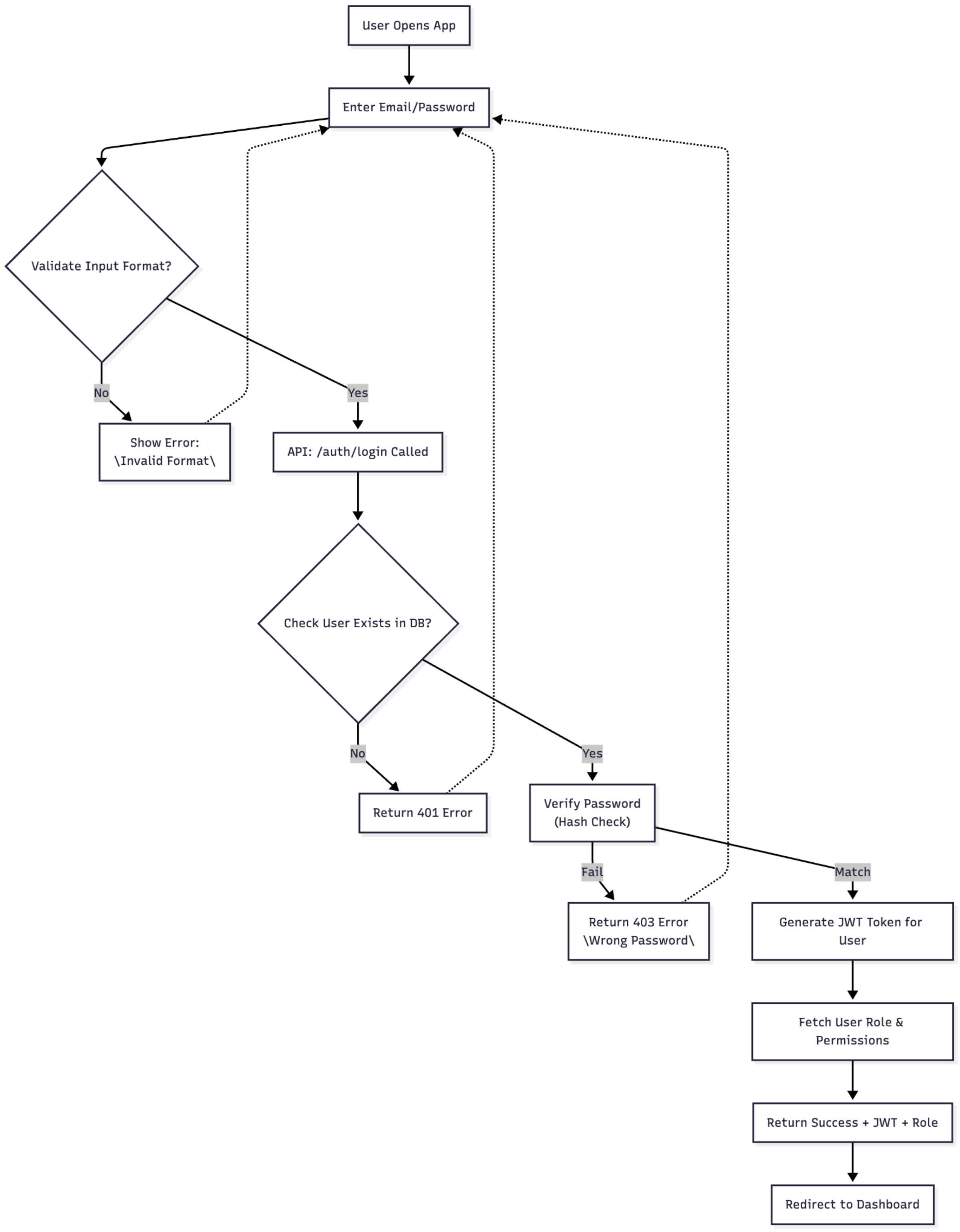
A strong ERD supports the BCM system by:

- Maintaining clean, normalized data
- Preventing redundancy through relational design
- Creating strong linkages between BCM modules
- Enabling scalable analytics and reporting
- Supporting fast disaster-recovery queries (e.g., “Show all critical processes in Region X with RTO < 4 hours”)

LOW-LEVEL FLOWCHARTS & EXPLA

NATIONS

1 LOW-LEVEL FLOWCHART: User Login & Authentication Workflow


User Opens App
Enter Email/Password
Validate Input Format?
No
Yes
Show Error: \Invalid Format\
API: /auth/login Called
Check User Exists in DB?
No
Yes
Return 401 Error
Verify Password (Hash Check)
Fail
Match
Return 403 Error \Wrong Password\
Generate JWT Token for User
Fetch User Role & Permissions
Return Success + JWT + Role
Redirect to Dashboard

This low-level flowchart outlines exactly how the authentication module works. When the user launches the BCM application, the frontend gathers the email and password. The first validation happens on the frontend to ensure the format is correct (reducing unnecessary API calls). Once validated, a login API request is triggered.

The backend then checks whether the user exists. If not, a 401 “Unauthorized” response is returned. If the user exists, the system verifies the hashed password. If the password fails, a 403 error is shown indicating incorrect credentials.

For successful logins, the system generates a **JWT token**, fetches the user’s RBAC permissions, attaches them to the response, and returns everything to the frontend. With a valid token, the user is redirected to the dashboard where additional modules become available based on role (Admin, BCM Manager, Department Head, Process Owner, etc.).

This flow ensures secure, standardized, and scalable authentication supporting future features such as MFA, SSO, or federated identity integration.

LOW-LEVEL FLOWCHART: BIA (Business Impact Analysis) Form Submission Workflow

This flowchart shows how the BIA module handles the submission of critical process data. Users begin by entering the details of their process: name, description, dependencies, consequences of failure, recovery requirements, and other operational metrics.

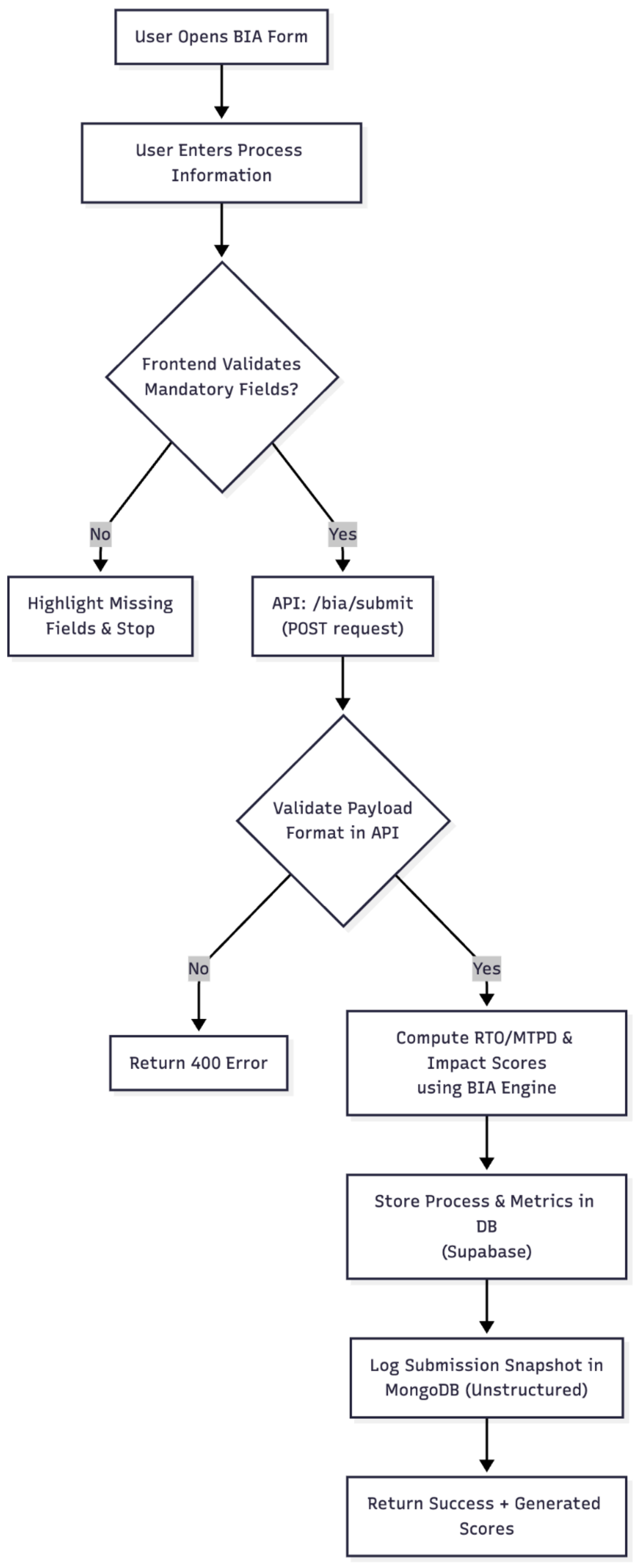
The frontend performs real-time checks for mandatory fields to prevent incomplete submissions. Only when validation passes does the frontend issue a POST request to the /bia/submit endpoint.

The backend performs deeper validation here—checking field types, allowed values, dependency structures, and whether this process already exists in the system. Once validated, the **BIA Engine** calculates:

- RTO (Recovery Time Objective)
- MTPD (Maximum Tolerable Period of Disruption)
- Impact severity
- Overall criticality

After calculations, all structured BIA records go into Supabase (Postgres tables). A snapshot of the entire submission, including raw JSON data, is saved into MongoDB to support audit trails, version history, and forensic analysis.

Finally, computed results are sent back to the user interface, allowing real-time visualization of impact scores and heatmaps.


User Opens BIA Form
User Enters Process Information
Frontend Validates Mandatory Fields?
No
Yes
Highlight Missing Fields & Stop
API: /bia/submit (POST request)
Validate Payload Format in API
No
Yes
Return 400 Error
Compute RTO/MTPD & Impact Scores using BIA Engine
Store Process & Metrics in DB (Supabase)
Log Submission Snapshot in MongoDB (Unstructured)
Return Success + Generated Scores

LOW-LEVEL FLOWCHART: Risk Assessment Processing Workflow

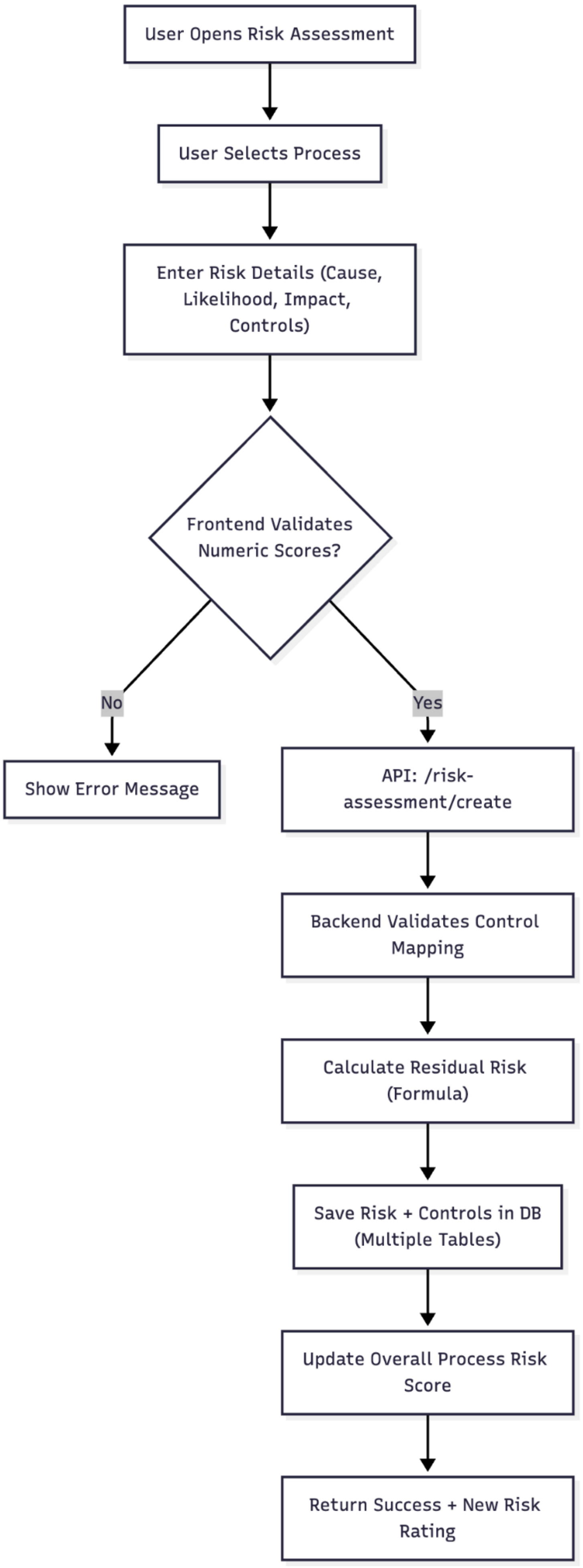
This low-level workflow describes how risks associated with a process are captured, scored, and saved. For each identified risk, the user inputs its description, threat type, impact rating, likelihood rating, and the effectiveness of current controls.

The frontend verifies numeric fields (1–5 scales), ensuring invalid values don't reach the backend. Once validated, the backend maps controls to risk IDs, performs deeper integrity checks, and uses the Risk Engine to compute:

- Inherent Risk
- Control Effectiveness
- Residual Risk Score
- Overall Risk Category (Low/Medium/High/Extreme)

Next, the system updates Process-level risk summaries by recalculating average or weighted risk scores. All structured risk records go into Supabase, and JSON logs go into MongoDB.

This modular workflow ensures risk information remains auditable, complete, and analytically useful.


User Opens Risk Assessment
User Selects Process
Enter Risk Details (Cause, Likelihood, Impact, Controls)
Frontend Validates Numeric Scores?
No
Yes
Show Error Message
API: /risk-assessment/create
Backend Validates Control Mapping
Calculate Residual Risk (Formula)
Save Risk + Controls in DB (Multiple Tables)
Update Overall Process Risk Score
Return Success + New Risk Rating

LOW-LEVEL FLOWCHART: Recovery Strategy Recommendation Workflow

This workflow runs when a department or process owner is determining the best recovery strategy to adopt. The system begins by fetching two critical datasets:

1. The process's **RTO/MTPD** values
2. Its **dependency chain** (applications, suppliers, departments)

The Recovery Strategy Engine uses these inputs to filter which strategy types are feasible. For example:

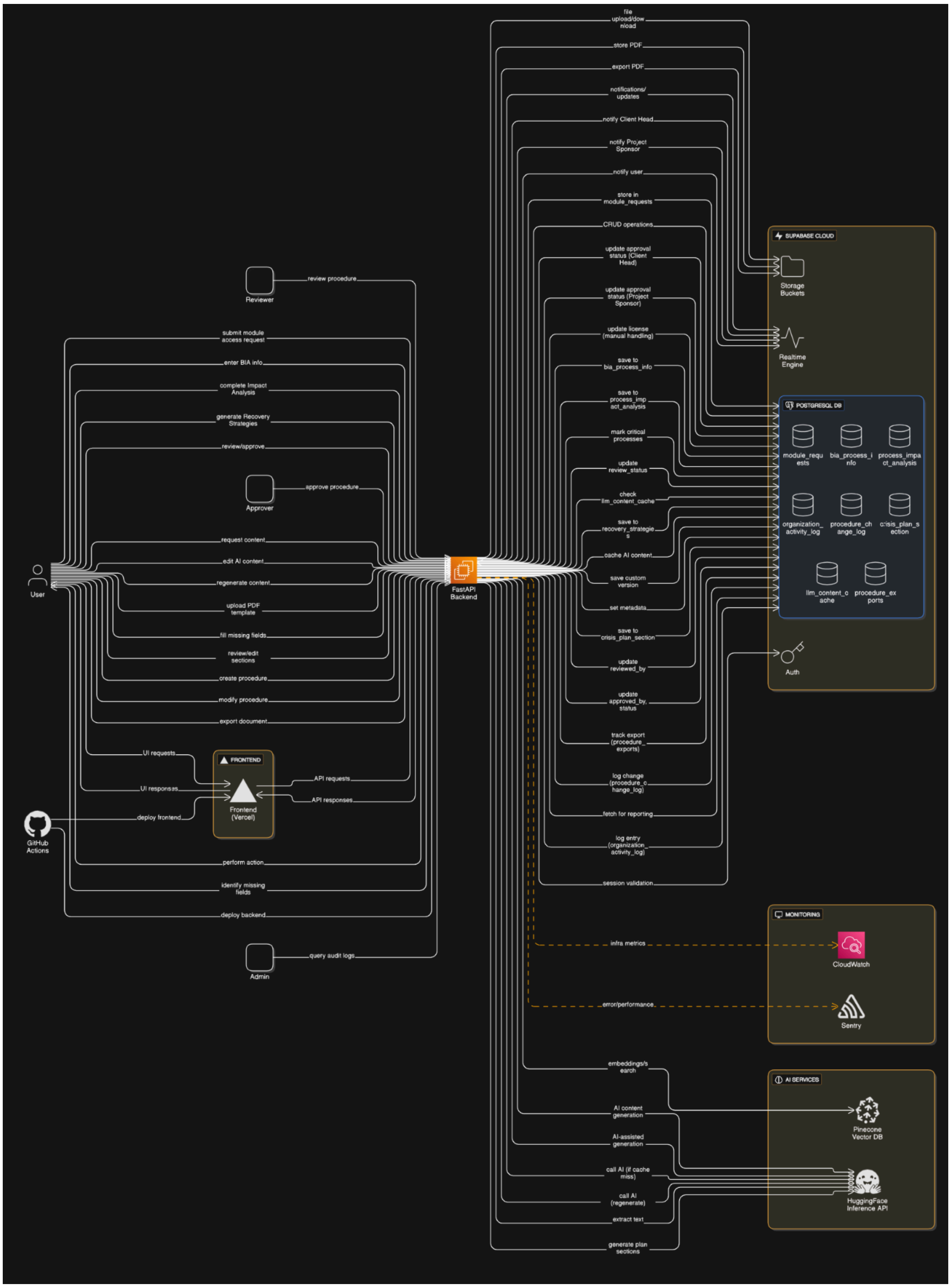
- If RTO < 1 hour → Only Hot Site or Failover
- If RTO < 4 hours → Warm Site, Replicated DB, Cloud DR
- If RTO > 24 hours → Cold Site or Manual Workarounds

The engine then generates multiple options and ranks them based on:

- RTO compliance
- Cost
- Resource availability
- Staff readiness
- Complexity
- External dependencies

Once the user selects a recommended strategy, the system saves it to Supabase and generates a checklist of resources, roles, and restoration steps. This gives BCM teams clarity on what to do during an actual crisis event.

Cloud deployment architecture diagram :



High-Level Infrastructure Overview

What it is

The Infrastructure Topology provides a holistic view of the BCM system, detailing where each component resides, how data flows between them, and how external services integrate with the system. This view is crucial for understanding the deployment, scaling, and reliability of the system.

Key Components (Expanded)

1. User / Client

The user interacts with the system via a standard web browser. The client layer handles all input operations, such as logging in, filling out BIA forms, submitting risk assessments, and viewing dashboards. All communication from the client occurs over secure HTTPS channels to ensure data integrity and confidentiality.

2. Frontend (React + Vite)

The frontend is typically hosted on a Content Delivery Network (CDN) or static hosting environment to ensure fast global access. The frontend responsibilities include:

- Rendering the user interface with forms, dashboards, and interactive components.
- Performing client-side validation and preprocessing of user inputs.
- Packaging API requests with authentication tokens, metadata, and structured payloads.
- Receiving and displaying processed data from backend services, such as BIA results, risk summaries, or crisis alerts.

The frontend is optimized for performance, leveraging Vite for build-time optimizations, code splitting, and caching.

3. FastAPI Backend Service

The FastAPI backend serves as the central orchestrator of business logic. It handles authentication, routing, data validation, and coordination between databases and external services. Key backend responsibilities include:

- Receiving requests from the frontend and verifying authentication and permissions.
- Performing CRUD operations on structured data stored in Supabase.
- Routing unstructured event logs and audit trails to MongoDB.
- Initiating AI service calls for risk analysis, crisis management recommendations, and automated report generation.
- Aggregating data for dashboards and exporting reports in various formats.

The backend can be deployed on cloud services using Docker containers, serverless environments, or managed hosting platforms to enable scalability and fault tolerance.

4. Supabase Cloud

Supabase provides a managed PostgreSQL database along with additional services:

- **Structured data storage**: Users, processes, BIA tables, risk registers, recovery strategies, and departmental information.
- **Realtime engine**: Supports live dashboard updates.
- **Edge functions**: Optional serverless compute for fast data processing and lightweight backend tasks.
- **Storage**: Handles attachments, reports, and generated PDFs.

Supabase ensures secure access through row-level security (RLS) and integrates seamlessly with the backend API.

Deep-Dive Infrastructure & External Services

5. MongoDB Atlas

MongoDB is used alongside Supabase to handle unstructured data and flexible document storage. Typical use cases include:

- Crisis event logs and timelines.
- Audit trails of user actions and system events.
- AI-generated summaries and embeddings.
- Chat messages and real-time collaboration data.

MongoDB's schema-less architecture allows for storing heterogeneous data without extensive migrations, making it ideal for dynamic and evolving system requirements.

6. AI Services (Groq / OpenAI)

The backend integrates with external AI services to enhance intelligence and automation capabilities:

- **Groq APIs**: Low-latency inference for quick computations and real-time recommendations.
- **OpenAI GPT models**: Used for complex reasoning, summarization, and generating natural-language insights.
- **Embeddings & Similarity Search**: Helps in linking related processes, risks, and strategies efficiently.

These AI services operate over secure HTTPS connections, and all data exchanged is limited to what is necessary for inference tasks to maintain privacy and compliance.

7. Internal Data Flows & Communication

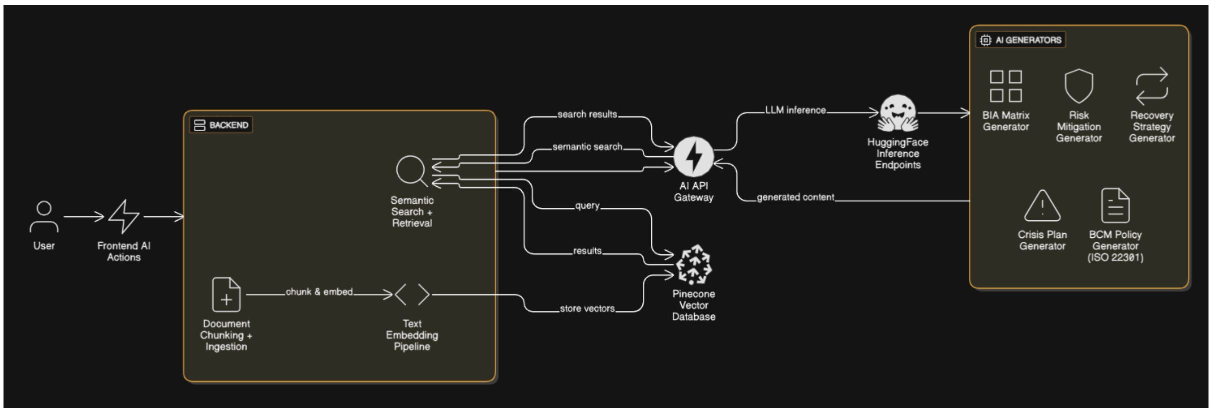
Data moves between components in a highly structured manner:

- **Frontend → Backend**: User interactions, form submissions, dashboard requests.
- **Backend → Databases**: FastAPI routes structured requests to Supabase and unstructured logs to MongoDB.
- **Backend → AI Services**: Intelligent computations, embeddings, and text summarizations.
- **Realtime Updates**: Dashboards leverage Supabase Realtime to push changes to clients instantly.

8. Security & Reliability Considerations

- All communication uses **TLS/HTTPS** to protect data in transit.
- Authentication tokens (JWT/Refresh tokens) ensure secure sessions.
- Databases are managed with replication, backup, and monitoring to ensure high availability.
- External AI calls are rate-limited and monitored for latency to maintain performance.

AI Integration Architecture diagram :



Generative AI (RAG) Overview
What it is

The Generative AI (RAG – Retrieval Augmented Generation) component of the BCM system enables advanced automation, intelligent insights, and semantic reasoning over organizational data. It allows users to generate business continuity content dynamically using AI models by combining retrieved knowledge from structured and unstructured data sources with generative reasoning capabilities.

Unlike standard LLM workflows, RAG enhances accuracy by retrieving relevant information from pre-indexed documents, rather than relying solely on the LLM's internal knowledge. This ensures that all AI-generated outputs remain grounded in organizational context, internal documentation, and regulatory requirements.

Key Components (Expanded)
1. Ingestion Pipeline

The ingestion stage is the entry point for documents and data into the RAG system. It consists of the following steps:

Document Upload: Users upload relevant files such as SOPs, BIA reports, risk assessments, and recovery strategies.

Chunking: Large documents are divided into smaller, semantically meaningful chunks. This prevents context loss during retrieval and improves LLM efficiency.

Embedding Generation: Each chunk is converted into a numerical vector using pre-trained embedding models. This vector represents the semantic meaning of the content, allowing the AI system to perform similarity searches during query processing.

The ingestion pipeline ensures that all organizational knowledge is pre-processed and ready for rapid retrieval during inference.

2. Vector Database (Pinecone)

Once embeddings are created, they are stored in a vector database such as Pinecone. The vector database serves as a highly optimized index for semantic search.

Key responsibilities include:

Storing embeddings: Each chunk's vector is indexed for quick lookup.

Semantic similarity search: Allows AI queries to retrieve the most relevant documents or text chunks.

Scalability: Handles large volumes of embeddings efficiently, ensuring low-latency retrieval even with thousands of documents.

Versioning & updates: Newly uploaded documents are automatically embedded and added to the index, ensuring that the RAG system remains up-to-date.

By using a vector database, the RAG system ensures that LLMs always have access to relevant organizational context before generating any content.

3. AI API Gateway

The AI API Gateway acts as a traffic manager between the backend and the generative models. Its key functions include:

Request routing: Directs queries to the appropriate LLM based on the type of request (e.g., BIA analysis, risk mitigation recommendations, or strategy generation).

Rate limiting & monitoring: Ensures that LLM requests do not exceed quotas and tracks response times for performance analysis.

Authentication & security: Validates that only authorized services or users can access the generative models.

Load balancing: Distributes requests across multiple model endpoints for scalability and reliability.

The API gateway abstracts model-specific complexities, allowing the backend to interface with multiple LLM providers seamlessly.

Detailed RAG Workflow and Generators
4. Inference / LLM Processing

After relevant document chunks are retrieved from the vector database, the inference stage generates the final outputs:

Query Formation: Retrieved chunks are combined with the user's query to form a structured prompt for the LLM.

Model Selection: The system selects an appropriate model for generation, either HuggingFace models for specialized tasks or Groq for low-latency inference.

Content Generation: The LLM produces outputs based on retrieved knowledge, ensuring that generated content aligns with internal documents, policies, and best practices.

Inference leverages the retrieval augmented mechanism to ensure that responses are both factually accurate and contextually relevant.

5. Generators / Specialized Agents

The RAG system supports multiple specialized generators, each tailored for specific BCM workflows:

BIA Matrix Generator

Generates structured Business Impact Analysis summaries based on uploaded documents and historical BIA data.

Provides weighted scores, impact charts, and criticality assessments.

Risk Mitigation Generator

Produces actionable recommendations for identified risks.

Cross-references organizational policies and prior incident logs to ensure feasibility and compliance.

Recovery Strategy Generator

Creates detailed recovery plans, including timelines, resource allocation, and fallback procedures.

Integrates previous BIA and risk mitigation outputs to maintain consistency across plans.

Each generator operates as an intelligent agent, automatically retrieving relevant context, formatting outputs, and ensuring alignment with organizational standards.

6. Benefits of the RAG Workflow

The combination of retrieval and generation provides several advantages:

Accuracy: Retrieval ensures that LLM outputs are grounded in organizational data.

Efficiency: Pre-indexed embeddings reduce the time required to locate relevant information.

Consistency: Generated outputs maintain alignment with existing policies, reports, and standards.

Scalability: Multiple AI endpoints can be added to handle increasing loads without modifying the core workflow.

Automation: Specialized generators reduce manual effort in creating BIA matrices, risk mitigation plans, and recovery strategies.

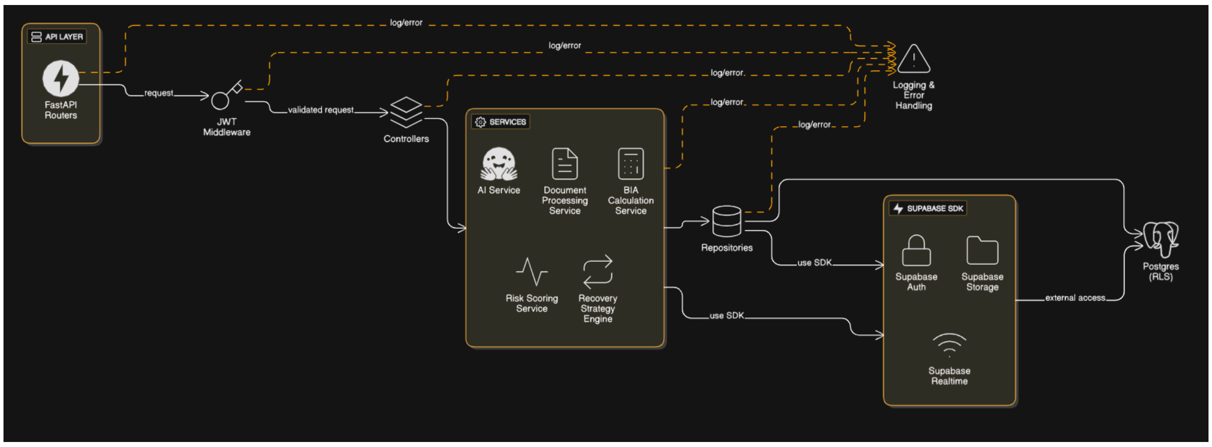
7. Integration with the BCM System

Frontend: Users submit requests for BIA, risk analysis, or recovery strategies.

Backend (FastAPI): Orchestrates retrieval from Pinecone, calls AI APIs, and returns structured results to the frontend.

Databases: Supabase stores structured results, while MongoDB logs AI outputs for auditing and traceability.

Real-time Dashboards: Generated content can immediately feed into dashboards or reports for BCM officers.



React Application Internal Structure (Overview)

What it is

The internal structure of the React application defines how the frontend is organized, how components interact, and how state and data flow through the system. This structure ensures maintainability, scalability, and reusability across the application while allowing seamless communication with the backend.

By organizing the application into well-defined layers, developers can separate concerns, reduce redundancy, and streamline development workflows.

Key Layers (Expanded)

1. Routing Layer

The Routing Layer controls **navigation and URL management** within the application.

- Uses **React Router DOM** to define routes for different pages and features.
- Handles **nested routes**, route guards, and redirects based on user authentication and roles.
- Integrates with context providers to enforce access control.
- Supports dynamic route parameters for pages such as process details, BIA forms, or risk dashboards.

This layer ensures users can move seamlessly between pages while maintaining application state and security constraints.

---

2. Context Providers

Context Providers are used to **manage global state** that is shared across multiple components. Common contexts include:

- **Auth Context**: Manages login state, JWT tokens, user roles, and session expiry.
- **Theme Context**: Handles application theming (dark/light mode, custom styles).
- **Organization Context**: Stores organization-specific data such as department lists, process hierarchies, and default settings.

Context providers eliminate the need for prop drilling, allowing deeply nested components to access shared data directly.

---

3. Pages / Views

Pages are the **core views of the application**, representing the actual screens users interact with.

Examples include:

- **Dashboard Page**: Displays summaries, charts, and critical metrics.
- **BIA Page**: Allows users to fill out Business Impact Analysis forms and view results.
- **Recovery Strategy Page**: Provides forms and visualizations for planning recovery strategies.

Pages are typically composed of **reusable components** and are connected to context providers and API services for dynamic data rendering.

React Application Internal Structure (Components & Utilities)

4. Reusable Components

Reusable Components are **shared UI elements** that appear in multiple pages. Examples:

- **Forms**: Input fields, select dropdowns, date pickers, and validation wrappers.
- **Tables**: Data grids, sortable/filterable tables for processes, risks, or recovery items.
- **Charts**: Bar charts, pie charts, and line graphs for dashboards or risk metrics.
- **Buttons & Modals**: Standardized buttons, dialog modals, and alert components.

By abstracting common UI elements, developers ensure **consistency**, **reusability**, and easier maintenance across the application.

---

5. Utility Layer

The Utility Layer provides **helper functions** used across the application:

- **Validators**: Input validators for email, numbers, required fields, and custom rules.
- **Formatters**: Functions for date formatting, currency, percentage, or custom labels.
- **Transformers**: Map raw API responses to frontend-friendly structures.

- **Helpers**: General-purpose functions such as deep copying, array manipulation, or pagination.

This layer centralizes logic that does not belong in the UI, keeping components clean and focused on rendering.

---

6. API Service

The API Service layer handles all **communication with the backend** using libraries such as Axios or Fetch.

Key responsibilities:

- **Request Wrappers**: Standardized methods for GET, POST, PUT, DELETE requests.
- **Error Handling**: Centralized error catching and response parsing.
- **Token Injection**: Automatically adds authentication headers to requests.
- **Response Transformation**: Converts raw backend data into structures suitable for frontend consumption.
- **Retry & Timeout Logic**: Ensures reliable communication with the backend, handling intermittent network issues.

By isolating API calls in a dedicated service layer, the application achieves **modularity**, making it easier to swap endpoints, mock services, or extend functionality in the future.

---

7. Summary

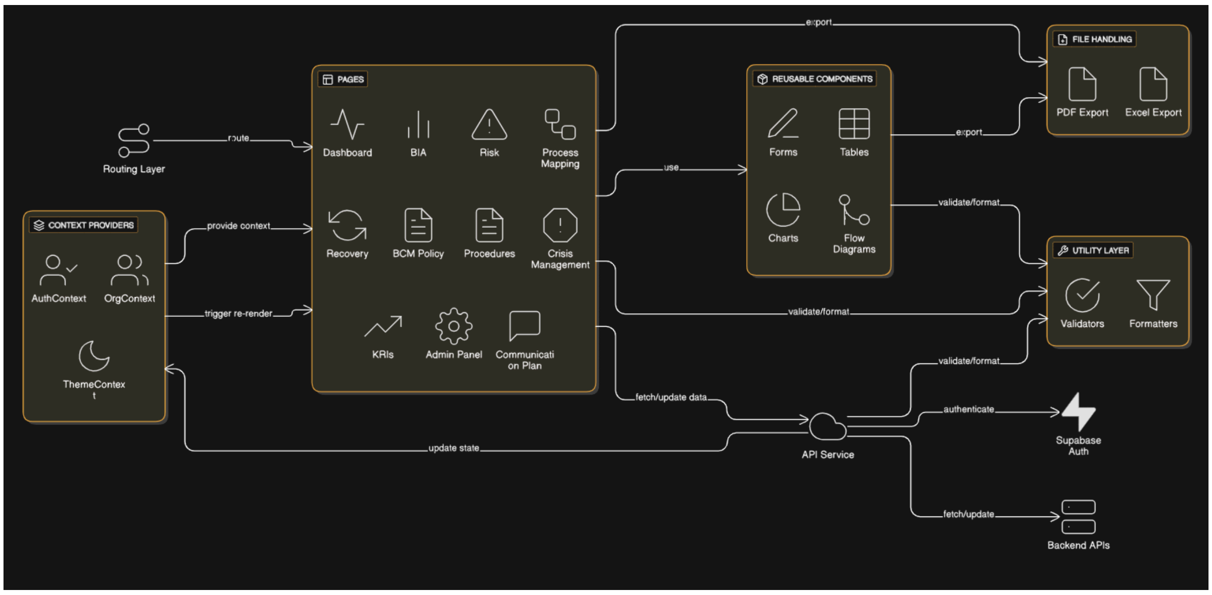
The React application's internal structure is carefully organized into:

1. **Routing Layer** – navigation and access control

2. **Context Providers** – global state management
3. **Pages** – main views for user interaction
4. **Reusable Components** – standardized UI elements
5. **Utility Layer** – helper functions for validations and transformations
6. **API Service** – backend communication layer

This modular design ensures **scalability**, **maintainability**, and **consistency**, enabling the BCM frontend to efficiently support complex workflows like BIA, risk management, and recovery strategy generation.

Backend architecture diagram :



FastAPI Application Internal Structure (Overview)

What it is

The FastAPI application forms the backend of the BCM system, providing RESTful APIs for all frontend operations, orchestrating business logic, and managing data persistence. Its internal structure is modular and layered to ensure maintainability, scalability, and testability.

Each layer has a distinct responsibility, ensuring separation of concerns. This design allows developers to extend functionality, add new services, or modify business rules without affecting other parts of the application.

Key Layers (Expanded)

1. API Layer (Routes / Endpoints)

The API layer exposes **RESTful endpoints** to the frontend and other clients. Responsibilities include:

- **Route Definitions**: Mapping URLs to controllers for specific operations, such as `/login`, `/bia/submit`, or `/risk/create`.
- **Request Parsing**: Validates and parses incoming requests into structured data using **Pydantic models**.
- **Response Formatting**: Ensures responses conform to consistent JSON schemas with proper HTTP status codes.
- **Swagger / OpenAPI Integration**: Automatically documents endpoints for developer reference and testing.

The API layer acts as the **first interface** for all external interactions, providing a clean contract between clients and backend services.

---

2. Middleware Layer

Middleware functions intercept requests **before they reach the controller** and responses before they are sent to the client. Responsibilities include:

- **JWT Authentication**: Validates tokens, extracts user information, and enforces role-based access control.
- **Request Validation**: Ensures headers, query parameters, and request bodies meet expected formats and constraints.
- **Logging & Monitoring**: Records request metadata, performance metrics, and any anomalies for auditing and debugging.
- **Error Handling**: Catches exceptions raised in downstream layers and converts them into structured HTTP responses.

Middleware ensures that the application is **secure, consistent, and resilient**.

---

3. Controllers Layer

Controllers orchestrate the **flow of requests** from the API layer to the business logic services. Responsibilities include:

- Accepting validated request data from the API layer.
- Calling the appropriate service based on the operation type.
- Handling service responses, formatting them for the API layer, and returning structured results.
- Performing additional checks, such as permission validation or concurrency handling.

Controllers act as **coordinators**, decoupling the API layer from the internal service logic, which simplifies testing and maintenance.

FastAPI Application Internal Structure (Services & Repositories)

4. Services Layer

The Services layer contains the **core business logic** of the application. Each service is responsible for a specific domain or workflow. Examples include:

- **Risk Scoring Service**: Calculates risk scores based on process inputs, historical incidents, and configurable formulas.
- **Document Processing Service**: Handles uploaded documents, including parsing, validation, and embedding generation for AI integration.
- **BIA Analysis Service**: Computes criticality matrices, weighted impact scores, and timelines for recovery.
- **Strategy Generation Service**: Integrates risk and BIA outputs to propose actionable recovery strategies.

Responsibilities of the Services layer:

- Encapsulate all domain-specific logic separate from controllers.
- Handle orchestration between multiple repositories if required.
- Return structured outputs for controllers to send to the frontend.

This layer ensures that **business rules are centralized**, maintainable, and reusable across multiple endpoints.

---

5. Repositories Layer (Data Access Layer)

Repositories abstract all interactions with databases and external storage. Using the Supabase SDK, they provide:

- **CRUD Operations**: Create, Read, Update, Delete operations for structured entities like users, BIA records, risk entries, and strategies.
- **Query Abstractions**: Encapsulates complex queries, joins, and filters.
- **Transactions**: Manages transactional consistency for multi-step operations.
- **Data Validation**: Ensures data integrity before persisting.

By isolating database access, the Repositories layer allows **flexibility to swap or upgrade the storage backend** without impacting higher layers.

---

6. Summary of Layered Architecture

The FastAPI application is structured into the following layers:

1. **API Layer** – defines endpoints and handles request/response parsing.
2. **Middleware** – manages authentication, validation, logging, and error handling.
3. **Controllers** – orchestrates requests and responses between API and services.
4. **Services** – encapsulates business logic for BIA, risk, recovery, and AI processes.
5. **Repositories** – handles database interactions via Supabase SDK.

This architecture ensures **modularity, scalability, and maintainability**, making the backend capable of supporting complex workflows like risk management, BIA evaluation, and recovery strategy generation.