ResilientOps Web Application API Documentation

Version: 1.0
Base URL: `/api`
Description: API for managing services, risk analysis, authentication, audit logs, and alerts with JWT-based authentication and Swagger UI support.

1 Overview

The ResilientOps API provides endpoints for managing services, business impact analysis (BIA), risk scores, downtimes, integrations, audit logs, and alerts. It uses JWT for authentication, requiring a valid token for most endpoints. The API is organized into namespaces: `auth`, `services`, `risk`, `audit`, and `alerts`.

1.1 Authentication

- **JWT Token**: Required for all endpoints except `/api/auth/signup` and `/api/auth/login`. Include the token in the `Authorization` header as `Bearer <token>`.
- **Roles**: Endpoints are restricted by user roles (`Business Owner`, `Ops Analyst`, `Engineer`).
- **Token Expiry**: Tokens expire after 1 hour. Refresh by re-authenticating via `/api/auth/login`.

1.2 Error Handling

All endpoints return JSON responses with appropriate HTTP status codes. Common errors include:

- **400 Bad Request**: Invalid or missing request parameters.
- **401 Unauthorized**: Invalid or missing JWT token.
- **403 Forbidden**: Insufficient role permissions.
- **404 Not Found**: Resource (e.g., service, user) not found.
- **500 Internal Server Error**: Database or unexpected errors.

Error responses follow this format:

```
{
  "error": "Error message"
}
```

1.3 Common Headers

- `Content-Type: application/json`
- `Authorization: Bearer <JWT_TOKEN>` (for protected endpoints)

2 Namespaces

2.1 Auth Namespace (`/api/auth`)

Handles user authentication and registration.

2.1.1 Signup

- **Endpoint**: `POST /api/auth/signup`
- **Description**: Register a new user.
- **Request Body**:

```
{
  "username": "string",
  "password": "string",
  "role": "string" // Optional, defaults to "user"
}
```

- **Constraints**:
  - `username`: Required, unique.
  - `password`: Required, minimum 6 characters.
  - `role`: Optional, defaults to `user`.
- **Response**:
  - **201 Created**:

```
{
  "message": "User registered successfully"
}
```

  - **400 Bad Request**:

```
{
  "error": "Username is required | Password is required |
     Password must be at least 6 characters long | User
     already exists"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to register user due to database error |
     Internal server error"
}
```

- **Audit Log**: Logs action `User Signup` with `entity=User` and `entity_id=user.id`.

2.1.2 Login

- **Endpoint**: `POST /api/auth/login`
- **Description**: Authenticate a user and return a JWT token.
- **Request Body**:

```
{
  "username": "string",
  "password": "string"
}
```

- **Constraints**:
  - `username`: Required.
  - `password`: Required.
- **Response**:
  - **200 OK**:

```
{
  "access_token": "string"
}
```

  - **400 Bad Request**:

```
{
  "error": "Username and password are required"
}
```

  - **401 Unauthorized**:

```
{
  "error": "Invalid username or password"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to process login due to database error | Internal server error"
}
```

2.2 Services Namespace (`/api/services`)

Manages services, BIA, status, downtimes, integrations, and dependencies.

2.2.1 Create Service

- **Endpoint**: `POST /api/services`
- **Description**: Create a new service with optional BIA and dependencies.
- **Roles**: `Business Owner`
- **Request Body**:

```
{
  "name": "string",
  "description": "string",
  "criticality": "string",
  "impact": "string",
  "rto": integer,
  "rpo": integer,
```

```
  "dependencies": [integer],
  "signed_off": boolean
}
```

- **Constraints**:
  - `name`: Required.
  - `dependencies`: Array of valid service IDs.
- **Response**:
  - **201 Created**:

```
{
  "message": "Service created",
  "service_id": integer
}
```

  - **400 Bad Request**:

```
{
  "error": "Service name is required | Invalid dependency IDs:
     [ids]"
}
```

  - **404 Not Found**:

```
{
  "error": "User not found"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to create service due to database error |
     Internal server error"
}
```

- **Audit Log**: Logs action `Service Created` with `entity=Service` and `entity_id=service.id`.

2.2.2 Get All Services

- **Endpoint**: `GET /api/services`
- **Description**: Retrieve all services with their BIA and status.
- **Roles**: Any authenticated user.
- **Response**:
  - **200 OK**:

```
[
  {
    "id": integer,
    "name": "string",
    "description": "string",
    "created_by": "string",
    "bia": {
```

```
      "criticality": "string",
      "impact": "string",
      "rto": integer,
      "rpo": integer,
      "signed_off": boolean,
      "dependencies": [integer]
    },
    "status": "string",
    "last_updated": "string" // ISO 8601
  }
]
```

– **500 Internal Server Error**:

```
{
  "error": "Failed to retrieve services due to database error
     | Internal server error"
}
```

2.2.3 Update Service

- **Endpoint**: `PUT /api/services`
- **Description**: Update an existing service and its BIA.
- **Roles**: `Business Owner`
- **Request Body**:

```
{
  "id": integer,
  "name": "string",
  "description": "string",
  "criticality": "string",
  "impact": "string",
  "rto": integer,
  "rpo": integer,
  "dependencies": [integer],
  "signed_off": boolean
}
```

- **Constraints**:
  - `id`: Required.
  - `dependencies`: Array of valid service IDs.
- **Response**:
  - **200 OK**:

```
{
  "message": "Service updated successfully"
}
```

  - **400 Bad Request**:

```
{
  "error": "Service ID is required | Invalid dependency IDs:
     [ids]"
}
```

– **404 Not Found**:

```
{
  "error": "Service not found"
}
```

– **500 Internal Server Error**:

```
{
  "error": "Failed to update service due to database error |
     Internal server error"
}
```

- **Audit Log**: Logs action `Service Updated` with `entity=Service` and `entity_id=service.id`.

2.2.4 Delete Service

- **Endpoint**: `DELETE /api/services`
- **Description**: Delete a service and its related data (cascade).
- **Roles**: `Business Owner`
- **Request Body**:

```
{
  "id": integer
}
```

- **Constraints**:
  – `id`: Required.
- **Response**:
  – **200 OK**:

```
{
  "message": "Service deleted successfully"
}
```

– **400 Bad Request**:

```
{
  "error": "Service ID is required"
}
```

– **404 Not Found**:

```
{
  "error": "Service not found | User not found"
}
```

- **500 Internal Server Error**:

```
{
  "error": "Failed to delete service due to database error | Internal server error"
}
```

- **Audit Log**: Logs action `Service Deleted` with `entity=Service` and `entity_id=service.id`.

2.2.5 Update Service Status

- **Endpoint**: `POST/PUT /api/services/<int:service_id>/status`
- **Description**: Create or update the status of a service.
- **Roles**: `Business Owner`
- **Request Body**:

```
{
  "status": "string"
}
```

- **Constraints**:
  - `status`: Required.
- **Response**:
  - **200 OK**:

```
{
  "message": "Status updated successfully"
}
```

  - **400 Bad Request**:

```
{
  "error": "Status is required"
}
```

  - **404 Not Found**:

```
{
  "error": "Service not found"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to update status due to database error | Internal server error"
}
```

- **Audit Log**: Logs action `Status Updated` with `entity=Status` and `entity_id=service_id`.

2.2.6 Update BIA

- **Endpoint**: `PUT /api/services/<int:service_id>/bia`
- **Description**: Update the BIA for a service.
- **Roles**: `Business Owner`
- **Request Body**:

```
{
  "criticality": "string",
  "impact": "string",
  "rto": integer,
  "rpo": integer,
  "dependencies": [integer],
  "signed_off": boolean
}
```

- **Constraints**:
  - `dependencies`: Array of valid service IDs.
- **Response**:
  - **200 OK**:

```
{
  "message": "BIA updated successfully"
}
```

  - **400 Bad Request**:

```
{
  "error": "Invalid dependency IDs: [ids]"
}
```

  - **404 Not Found**:

```
{
  "error": "Service not found"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to update BIA due to database error |
     Internal server error"
}
```

- **Audit Log**: Logs action `BIA Updated` with `entity=BIA` and `entity_id=service_id`.

2.2.7 Delete BIA

- **Endpoint**: `DELETE /api/services/<int:service_id>/bia`
- **Description**: Delete the BIA for a service.
- **Roles**: `Business Owner`

- **Response**:
  - **200 OK**:

```
{
  "message": "BIA deleted successfully"
}
```

  - **404 Not Found**:

```
{
  "error": "Service not found | No BIA found for this service"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to delete BIA due to database error |
     Internal server error"
}
```

- **Audit Log**: Logs action `BIA Deleted` with `entity=BIA` and `entity_id=service_id`.

2.2.8 Log Downtime

- **Endpoint**: `POST /api/services/<int:service_id>/downtime`
- **Description**: Log a downtime event for a service.
- **Roles**: Any authenticated user.
- **Request Body**:

```
{
  "start_time": "string", // ISO 8601
  "end_time": "string", // Optional, ISO 8601
  "reason": "string" // Optional
}
```

- **Constraints**:
  - `start_time`: Required, ISO 8601 format.
  - `end_time`: Optional, must be after `start_time` if provided.
- **Response**:
  - **200 OK**:

```
{
  "message": "Downtime logged",
  "downtime": {
    "service_id": integer,
    "start_time": "string",
    "end_time": "string",
    "reason": "string"
  }
}
```

– **400 Bad Request**:

```
{
  "error": "Start time is required | Invalid date format. Use
     ISO 8601 (YYYY-MM-DDTHH:MM:SS) | End time cannot be
     before start time"
}
```

– **404 Not Found**:

```
{
  "error": "Service not found"
}
```

– **500 Internal Server Error**:

```
{
  "error": "Failed to log downtime due to database error |
     Internal server error"
}
```

- **Audit Log**: Logs action `Downtime Logged` with `entity=Downtime` and `entity_id=service_id`.

2.2.9 Get Downtime

- **Endpoint**: `GET /api/services/<int:service_id>/downtime`
- **Description**: Retrieve all downtime events for a service.
- **Roles**: Any authenticated user.
- **Response**:

– **200 OK**:

```
{
  "service_id": integer,
  "service_name": "string",
  "downtime_count": integer,
  "downtimes": [
    {
      "start_time": "string", // ISO 8601
      "end_time": "string", // ISO 8601 or null
      "reason": "string",
      "duration": "string",
      "total_minutes": integer
    }
  ]
}
```

– **404 Not Found**:

```
{
  "error": "Service not found"
}
```

– **500 Internal Server Error**:

```
{
  "error": "Failed to retrieve downtime due to database error
     | Internal server error"
}
```

2.2.10 Add Integration

- **Endpoint**: `POST /api/services/integrations`
- **Description**: Add an integration (e.g., Slack) for a service.
- **Roles**: `Engineer`
- **Request Body**:

```
{
  "service_id": integer,
  "type": "string",
  "config": {} // JSON object
}
```

- **Constraints**:
  - `service_id`: Required, valid service ID.
  - `type`: Required (e.g., `Slack`).
  - `config`: Required, valid JSON object.
- **Response**:
  - **201 Created**:

```
{
  "message": "Integration added successfully"
}
```

  - **400 Bad Request**:

```
{
  "error": "Service ID, type, and config are required"
}
```

  - **404 Not Found**:

```
{
  "error": "Service not found"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to add integration due to database error |
     Internal server error"
}
```

- **Slack Integration**: If `type=Slack`, sends a notification to the specified `webhook_url` in `config`.

2.2.11 Get Integrations

- **Endpoint**: `GET /api/services/integrations`
- **Description**: Retrieve all integrations.
- **Roles**: `Engineer`

- **Response**:
  - **200 OK**:

```
[
  {
    "id": integer,
    "service_id": integer,
    "type": "string",
    "config": {},
    "created_by": "string",
    "created_at": "string" // ISO 8601
  }
]
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to retrieve integrations due to database
     error | Internal server error"
}
```

2.2.12 Get Dependencies

- **Endpoint**: `GET /api/services/dependencies`
- **Description**: Retrieve all service dependencies.
- **Roles**: `Engineer`
- **Response**:
  - **200 OK**:

```
{
  "dependencies": [
    {
      "service_id": integer,
      "service_name": "string",
      "dependencies": [
        {
          "service_id": integer,
          "service_name": "string",
          "criticality": "string",
          "impact": "string",
          "rto": integer,
          "rpo": integer,
          "status": "string"
        }
      ]
    }
  ]
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to retrieve dependencies due to database
     error | Internal server error"
}
```

2.2.13 Get Service Health

- **Endpoint**: `GET /api/services/<int:service_id>/health`
- **Description**: Retrieve the health status of a service, including risk and uptime.
- **Roles**: Any authenticated user.
- **Response**:
  - **200 OK**:

```
{
  "service_id": integer,
  "name": "string",
  "status": "string",
  "last_updated": "string", // ISO 8601
  "bia": {
    "criticality": "string",
    "rto": integer,
    "rpo": integer
  },
  "downtime": {
    "start_time": "string", // ISO 8601
    "reason": "string"
  },
  "overall_health": "string", // Healthy, Degraded, Unhealthy
  "risk_score": integer,
  "is_critical": boolean,
  "reason": "string",
  "uptime_percentage": float
}
```

  - **404 Not Found**:

```
{
  "error": "Service not found"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to retrieve service health due to database
     error | Internal server error"
}
```

2.3 Risk Namespace (`/api/risk`)

Manages risk scores for services.

2.3.1 Get Risk Score

- **Endpoint**: `GET /api/risk/<int:service_id>`
- **Description**: Retrieve the latest risk score for a service.
- **Roles**: Any authenticated user.
- **Response**:

- **200 OK**:

```
{
  "service_id": integer,
  "risk_score": integer,
  "risk_level": "string", // Low, Medium, High
  "is_critical": boolean,
  "reason": "string",
  "source": "string", // automated, manual
  "created_by": "string",
  "created_at": "string" // ISO 8601
}
```

- **404 Not Found**:

```
{
  "error": "Service not found | No risk score available for
     this service"
}
```

- **500 Internal Server Error**:

```
{
  "error": "Failed to retrieve risk score due to database
     error | Internal server error"
}
```

2.3.2 Save Automated Risk Score

- **Endpoint**: `POST /api/risk/<int:service_id>/save`
- **Description**: Calculate and save an automated risk score for a service.
- **Roles**: `Ops Analyst`
- **Response**:
  - **200 OK**:

```
{
  "message": "Risk score saved",
  "service_id": integer,
  "risk_score": integer,
  "risk_level": "string",
  "is_critical": boolean,
  "reason": "string"
}
```

  - **404 Not Found**:

```
{
  "error": "Service not found"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to save risk score due to database error |
     Internal server error"
}
```

- **Audit Log**: Logs action `Automated Risk Score Saved` with `entity=Risk` and `entity_id=service_id`.

2.3.3 Add/Update Manual Risk Score

- **Endpoint**: `POST/PUT /api/risk/<int:service_id>/manual`
- **Description**: Add or update a manual risk score for a service.
- **Roles**: `Ops Analyst`
- **Request Body**:

```
{
  "risk_score": integer,
  "risk_level": "string", // Low, Medium, High
  "reason": "string", // Optional
  "is_critical": boolean // Optional
}
```

- **Constraints**:
  - `risk_score`: Required, integer between 0 and 100.
  - `risk_level`: Required, must be `Low`, `Medium`, or `High`.
- **Response**:
  - **200 OK**:

```
{
  "message": "Manual risk score added | Manual risk score
     updated"
}
```

  - **400 Bad Request**:

```
{
  "error": "Risk score and risk level are required | Risk
     score must be an integer between 0 and 100 | Risk level
     must be Low, Medium, or High"
}
```

  - **404 Not Found**:

```
{
  "error": "Service not found | No manual risk record found to
     update"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to add/update manual risk score due to
     database error | Internal server error"
}
```

- **Audit Log**:
  - `POST`: Logs action `Manual Risk Score Added` with `entity=Risk` and `entity_id=service_id`.
  - `PUT`: Logs action `Manual Risk Score Updated` with `entity=Risk` and `entity_id=service_id`.

2.4 Audit Namespace (`/api/audit`)

Manages audit logs.

2.4.1 Get Audit Logs

- **Endpoint**: `GET /api/audit`
- **Description**: Retrieve all audit logs, ordered by timestamp (descending).
- **Roles**: Any authenticated user.
- **Response**:
  - **200 OK**:

```
[
  {
    "id": integer,
    "action": "string",
    "entity": "string",
    "entity_id": integer,
    "timestamp": "string", // ISO 8601
    "user_id": integer
  }
]
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to retrieve audit logs due to database
     error | Internal server error"
}
```

2.5 Alerts Namespace (`/api/alerts`)

Manages alerts and SLA breaches.

2.5.1 Get Alerts

- **Endpoint**: `GET /api/alerts`
- **Description**: Retrieve all alerts, ordered by creation time (descending).
- **Roles**: Any authenticated user.
- **Response**:
  - **200 OK**:

```
[
  {
    "id": integer,
    "service_id": integer,
    "type": "string",
    "message": "string",
    "severity": "string",
    "created_at": "string", // ISO 8601
    "acknowledged": boolean
  }
]
```

- **500 Internal Server Error**:

```
{
  "error": "Failed to retrieve alerts due to database error | Internal server error"
}
```

2.5.2 Update Alert

- **Endpoint**: `PUT /api/alerts`
- **Description**: Update the acknowledgment status of an alert.
- **Roles**: `Ops Analyst`
- **Request Body**:

```
{
  "id": integer,
  "acknowledged": boolean
}
```

- **Constraints**:
  - `id`: Required.
- **Response**:
  - **200 OK**:

```
{
  "message": "Alert updated"
}
```

  - **400 Bad Request**:

```
{
  "error": "Alert ID is required"
}
```

  - **404 Not Found**:

```
{
  "error": "Alert not found"
}
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to update alert due to database error | Internal server error"
}
```

- **Audit Log**: Logs action `Alert Acknowledged` with `entity=Alert` and `entity_id=alert.service_id`.

2.5.3 Get SLA Breaches

- **Endpoint**: `GET /api/alerts/sla_breaches`
- **Description**: Retrieve all SLA breaches, ordered by creation time (descending).
- **Roles**: Any authenticated user.
- **Response**:
  - **200 OK**:

```
[
  {
    "id": integer,
    "service_id": integer,
    "type": "string", // RTO, RPO
    "downtime_minutes": integer,
    "threshold_minutes": integer,
    "start_time": "string", // ISO 8601
    "end_time": "string", // ISO 8601 or null
    "reason": "string",
    "created_at": "string" // ISO 8601
  }
]
```

  - **500 Internal Server Error**:

```
{
  "error": "Failed to retrieve SLA breaches due to database
     error | Internal server error"
}
```

3 Background Processes

3.1 Health Checks

- **Frequency**: Every 5 minutes (via APScheduler).
- **Description**: Runs `run_health_checks` to:
  - Update service statuses (`Up`, `Degraded`, `Down`, `Unknown`) based on `last_updated` timestamp.
  - Calculate risk scores using `calculate_risk_score`.
  - Generate alerts for status changes, high risk scores, or critical services.
  - Check for SLA breaches (RTO/RPO violations) and create corresponding alerts.
- **Alert Triggers**:
  - Status changes (e.g., `Down`, `Degraded`).
  - High risk scores or critical services.
  - RTO/RPO violations based on downtime duration.
- **Slack Notifications**: Sends alerts to Slack if a `Slack` integration is configured for the service.

3.2 Risk Score Calculation

- **Function**: `calculate_risk_score(service, bia, status, all_services)`
- **Logic**:
  - Base score starts at 0, capped at 100.
  - Adds points based on:
    * Service status (`Down`: +40).
    * Recent downtime (>120 minutes: +20).
    * BIA criticality (`High`: +15, `Medium`: +10).
    * BIA impact (`High`/`Severe`: +10).
    * RTO (<60 minutes: +10).
    * RPO (<60 minutes: +5).
    * Down dependencies (+20).
    * Integration count (>3: +10, >5: +5).
  - Determines `risk_level`:
    * >=80: `High`
    * >=50: `Medium`
    * <50: `Low`
  - Sets `is_critical=true` if:
    * BIA criticality is `High`.
    * BIA impact is `High` or `Severe`.
    * RTO < 30 minutes.
    * Score >= 80.
    * Service is `Down` with >120 minutes downtime.
    * Dependencies are down.
    * >5 integrations.
  - Returns:

```
{
  "risk_score": integer,
  "risk_level": "string",
  "is_critical": boolean,
  "reason": "string"
}
```

4 Database Schema

4.1 Tables

1. **User**:
   - `id`: Integer, Primary Key
   - `username`: String(80), Unique, Not Null
   - `password`: String(200), Not Null
   - `role`: String(20), Not Null, Default=`user`
2. **Service**:

- `id`: Integer, Primary Key
- `name`: String(100), Not Null
- `description`: Text
- `created_by`: String(100)

3. **BIA**:
   - `id`: Integer, Primary Key
   - `service_id`: Integer, Foreign Key (`service.id`), Not Null
   - `criticality`: String(20)
   - `impact`: String(50)
   - `rto`: Integer
   - `rpo`: Integer
   - `signed_off`: Boolean, Default=`false`
4. **Status**:
   - `id`: Integer, Primary Key
   - `service_id`: Integer, Foreign Key (`service.id`), Not Null
   - `status`: String(20)
   - `last_updated`: DateTime
5. **Downtime**:
   - `id`: Integer, Primary Key
   - `service_id`: Integer, Foreign Key (`service.id`), Not Null
   - `start_time`: DateTime, Not Null
   - `end_time`: DateTime
   - `reason`: String(255)
6. **Integration**:
   - `id`: Integer, Primary Key
   - `service_id`: Integer, Foreign Key (`service.id`), Not Null
   - `type`: String(50)
   - `config`: JSON
   - `created_by`: String(100)
   - `created_at`: DateTime
7. **Risk**:
   - `id`: Integer, Primary Key
   - `service_id`: Integer, Foreign Key (`service.id`), Not Null
   - `risk_score`: Integer, Not Null
   - `risk_level`: String(20), Not Null
   - `reason`: Text
   - `is_critical`: Boolean, Default=`false`
   - `source`: String(20), Default=`automated`

- `created_by`: String(100)
- `created_at`: DateTime

8. **AuditLog**:
   - `id`: Integer, Primary Key
   - `action`: String(100), Not Null
   - `entity`: String(50), Not Null
   - `entity_id`: Integer, Not Null
   - `timestamp`: DateTime, Not Null
   - `user_id`: Integer, Not Null
9. **Alert**:
   - `id`: Integer, Primary Key
   - `service_id`: Integer, Foreign Key (`service.id`), Not Null
   - `type`: String(50), Not Null
   - `message`: Text, Not Null
   - `severity`: String(20), Not Null
   - `created_at`: DateTime
   - `acknowledged`: Boolean, Default=`false`
10. **SLABreach**:
    - `id`: Integer, Primary Key
    - `service_id`: Integer, Foreign Key (`service.id`), Not Null
    - `type`: String(20), Not Null
    - `downtime_minutes`: Integer, Not Null
    - `threshold_minutes`: Integer, Not Null
    - `start_time`: DateTime, Not Null
    - `end_time`: DateTime
    - `reason`: Text
    - `created_at`: DateTime
11. **service_dependencies** (Association Table):
    - `service_id`: Integer, Foreign Key (`service.id`), Primary Key
    - `dependency_id`: Integer, Foreign Key (`service.id`), Primary Key

4.2 Relationships

- **Service**:
  - One-to-One: `bia`, `status`
  - One-to-Many: `downtimes`, `integrations`, `risks`, `alerts`, `sla_breaches`
  - Many-to-Many: `dependencies` (via `service_dependencies`)
- **BIA**:

  - Belongs to: `service`
  - Many-to-Many: `dependencies` (via `service_dependencies`)
- **Status**, **Downtime**, **Integration**, **Risk**, **Alert**, **SLABreach**:
  - Belongs to: `service`

5 Security Considerations

- **JWT Authentication**: All endpoints except signup/login require a valid JWT token.
- **Role-Based Access Control**:
  - `Business Owner`: Manage services, BIA, status.
  - `Ops Analyst`: Manage risk scores, acknowledge alerts.
  - `Engineer`: Manage integrations, view dependencies.
- **Input Validation**:
  - Validates `risk_score` (0-100), `risk_level` (`Low`, `Medium`, `High`), date formats (ISO 8601), and dependency IDs.
- **Database Security**:
  - Uses parameterized queries via SQLAlchemy to prevent SQL injection.
  - Passwords are hashed using `werkzeug.security`.
- **CORS**: Enabled for cross-origin requests, but should be restricted to trusted origins in production.
- **SSL**: Runs with `adhoc` SSL context in debug mode; use proper certificates in production.

6 Setup and Running

6.1 Prerequisites

- Python 3.8+
- MySQL 8.0+
- **Dependencies** (install via `pip`):

```
pip install flask flask-restx flask-sqlalchemy flask-jwt-extended
    flask-cors pymysql python-dotenv requests apscheduler werkzeug
```

6.2 Environment Variables

- `MYSQL_USER`: MySQL username (default: `root`).
- `MYSQL_PASSWORD`: MySQL password (default: `""`).
- `MYSQL_HOST`: MySQL host (default: `localhost`).
- `MYSQL_PORT`: MySQL port (default: `3306`).
- `JWT_SECRET_KEY`: Secret key for JWT (auto-generated if not set).
- `DATABASE_URL`: Auto-set to `mysql+pymysql://<user>:<password>@<host>:<port>/auth`.

6.3 Running the API

1. Set up MySQL and ensure the `auth` database is created.
2. Create a `.env` file with necessary variables.
3. Install dependencies.
4. Run the application:

```
python app.py
```

5. Access the API at https://localhost:5001/api and Swagger UI at https://localhost:5001/api/.

7 Example Requests

7.1 Signup

```
curl -X POST https://localhost:5001/api/auth/signup \
-H "Content-Type: application/json" \
-d '{"username": "testuser", "password": "password123", "role": "Ops
    Analyst"}'
```

7.2 Login

```
curl -X POST https://localhost:5001/api/auth/login \
-H "Content-Type: application/json" \
-d '{"username": "testuser", "password": "password123"}'
```

7.3 Create Service

```
curl -X POST https://localhost:5001/api/services \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <JWT_TOKEN>" \
-d '{"name": "Test Service", "description": "A test service",
    "criticality": "High", "impact": "Severe", "rto": 30, "rpo": 15,
    "signed_off": true, "dependencies": [1]}'
```

7.4 Update Manual Risk Score

```
curl -X PUT https://localhost:5001/api/risk/1/manual \
-H "Content-Type: application/json" \
-H "Authorization: Bearer <JWT_TOKEN>" \
-d '{"risk_score": 80, "risk_level": "High", "reason": "Manual
    override", "is_critical": true}'
```

7.5 Get Service Health

```
curl -X GET https://localhost:5001/api/services/1/health \
-H "Authorization: Bearer <JWT_TOKEN>"
```

8 Notes

- **Swagger UI**: Available at `/api/` for interactive testing.
- **Health Checks**: Run every 5 minutes, updating statuses and generating alerts.
- **Database**: Automatically creates tables on startup (`db.create_all()`).
- **Logging**: Uses Python `logging` with level `INFO`, logs errors to console.
- **Error Handling**: All endpoints include try-catch blocks with session rollback on database errors.
- **Slack Integration**: Supports sending notifications for integrations and alerts if configured.