Technical Design Document Example

Goal:

Develop a functional web-based application for reporting cyber security issues, complete with a form for submissions, tracking stages, and admin capabilities.

Technologies/Libraries:

- **Frontend**: HTML, CSS & Javascript
- **Email**: Nodemailer
- **UI**: Material-UI or Tailwind

Workflow:

1. **Frontend**:
    - Design and implement the form for report submissions.
    - Develop interfaces for displaying report details and tracking stages.
2. **Backend**:
    - Set up the server to handle form submissions and stage updates.
    - Implement email functionality to send report data to the admin user email address.
3. **Database**:
    - Design schemas for reports, stages, and user roles.
    - Implement database operations for creating, reading, updating, and deleting reports.

Participants:

- **Team Lead**: Oversees the project and ensures timely progress.
- **Frontend Developer**: Designs and implements the user interface.
- **Backend Developer**: Sets up the server and handles API endpoints.
- **Database Administrator**: Manages the database schema and operations.
- **UI/UX Designer**: Focuses on the design and user experience.
- **Quality Assurance**: Tests the application for bugs and ensures it meets requirements.

Checkpoints:

1. **1 Hour**: Initial brainstorming and finalizing the idea.
2. **3 Hours**: Design document completion and approval.
3. **6 Hours**: Basic form and initial backend setup.
4. **12 Hours**: Frontend and backend integration, database schema designed.
5. **15 Hours**: Report submission and tracking functionality working.
6. **20 Hours**: Admin features implemented, and initial testing begins.
7. **22 Hours**: Final testing, bug fixing, and preparing the pitch.

Flow:

Main Components:

1. **Report Submission**:
    - Path: POST /api/reports
    - Body:

```
{
  "title": "Issue Title",
  "description": "Detailed description of the issue",
```

```
  "image": "imageURL",
  "type": "Type of report",
  "subcategory": "Subcategory"
}
```

- **Logic**:
  - Handle form submissions
  - Validate inputs
  - Store report data in the database
  - Send a notification email

Example:

```
{
  "title": "Phishing Email",
  "description": "Received a suspicious email...",
  "image": "http://example.com/image.png",
  "type": "Phishing",
  "subcategory": "Email"
}
```

Output:

```
{
  "status": "success",
  "message": "Report submitted successfully."
}
```

2. **Report Tracking**:
   - Path: GET /api/reports/
   - Logic: Retrieve and display a specific report's details and current stage.
   - Example:
     - **Input**: /api/reports/12345

Output:

```
{
 "title": "Phishing Email",
```

```
  "description": "Received a suspicious email...",
 "image": "http://example.com/image.png",
  "type": "Phishing",
  "subcategory": "Email",
  "status": "In Progress",
  "stages": ["Submitted", "Reviewed", "Investigating", "Resolved"]
}
```

3. **Admin Management**:
    - Path: PUT /api/reports/
    - Body:

```
{
  "status": "New Status",
  "stage": "New Stage"
}
```

- Logic: Allow admin users to update the status and stage of reports, apply filters, and view all reports.

  Example:

Input:

```
{
  "status": "Investigating",
  "stage": "Stage 3"
}
```

Output:

```
{
  "status": "success",
  "message": "Report updated successfully."
}
```

4. Database:

- **Schema**:

  Reports:

```
{
  "_id": "ObjectId",
  "title": "string",
 "description": "string",
  "image": "string",
```

```
  "type": "string",
  "subcategory": "string",
  "status": "string",
  "stages": ["string"]
}
```

Example:

```
{
  "_id": "5f8f8c44b54764421b7156d9",
  "title": "Phishing Email",
  "description": "Received a suspicious email...",
  "image": "http://example.com/image.png",
  "type": "Phishing",
  "subcategory": "Email",
  "status": "In Progress",
  "stages": ["Submitted", "Reviewed", "Investigating", "Resolved"]
}
```

Error Handling:

- **Form Validation Errors**: Return a 400 status code detailing the validation failure.
- **Database Errors**: Retry operations up to 3 times before returning a 500 status code with an error message.
- **Unauthorized Access**: Return a 403 status code for unauthorized access attempts.

Logging:

- **Logging**: Return structured logging.
  Logs are timestamped and include request/response details.

Deployment Instructions:

- **Setup**: Ensure all environment variables are configured. Deploy the application using Docker.
- **Rollback**: Instructions for rolling back to the previous stable version in case of issues.

Security Measures:

- **IP Whitelisting**: Verify requests against a whitelist of IP addresses.
- **CORS Policy**: Restrict API access to specified origins.
- **Environment Variables**: Secure sensitive information using environment variables.