### **PRODUCT REVIEW AND RATING WEB FORM**

### **1. Introduction**

**Purpose of the Document:**This document is designed to provide a comprehensive solution for creating a product rating and review web application. It will detail the flow of data, the tech stack to be used, and the structure of the application, both on the front-end and back-end.

**Overview of the Application:**The application allows users to rate and review products through a web form, where data is stored in a PostgreSQL database and can be reviewed by admins through a dedicated dashboard.

**Key Requirements:**

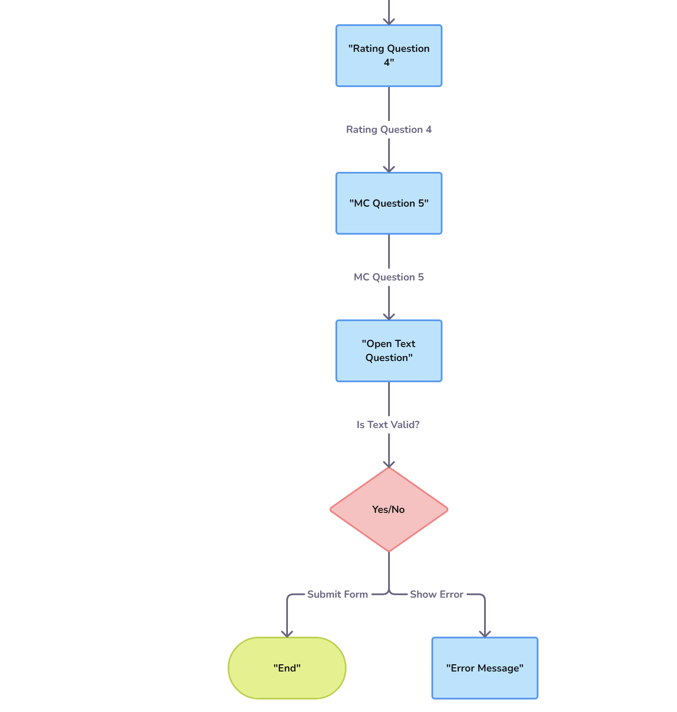
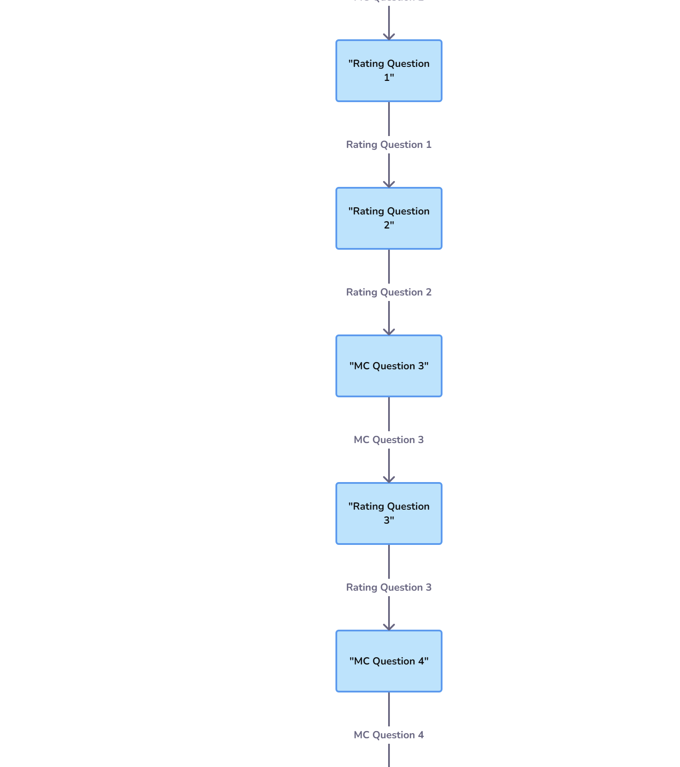
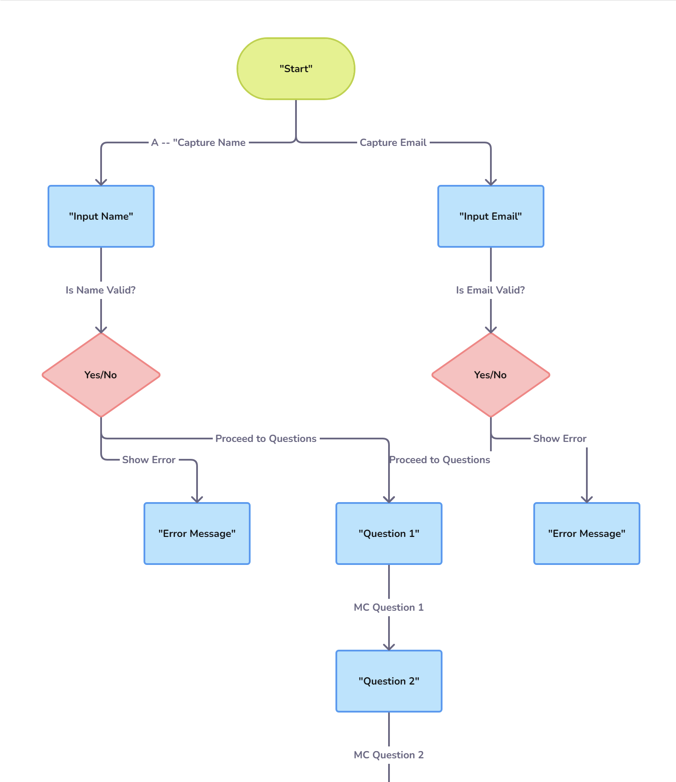
* 11 questions on the web form, including multiple-choice, rating scales, and an open text box.
* Two mandatory fields for capturing email and product name.
* Backend with a database for storing answers and an admin panel for viewing submissions.

### **2. Overall Flow Diagram**

**Description of the Flow:** The flow diagram will represent the entire process, starting from the user interacting with the web form, submitting the data, and the data being stored in the database. It will also cover the admin panel's interaction with the stored data.

**Diagram Representation:**

* **User Interaction:** Users access the form, fill in the mandatory fields, answer the questions, and submit the form.
* **Data Submission:** The data is sent to the back-end via API calls and stored in the PostgreSQL database.
* **Admin Interaction:** Admins log into the admin panel, access the dashboard, and review submissions.



### **3. Proposed Tech Stack**

**Front-End Technologies:**

* **ReactJS:** For building the interactive user interface of the web form.
* **Material-UI/Bootstrap:** For styling the form and ensuring a responsive design.
* **Axios:** For making API calls to the back-end.

**Back-End Technologies:**

* **PHP:** For handling server-side logic, form submission, and database interactions.
* **PostgreSQL:** As the database management system to store form data.

**Database Management:**

* **PostgreSQL:** For managing structured data with relations between different tables, ensuring data integrity.

**API Communication:**

* **RESTful APIs:** To handle the communication between the front-end and back-end, ensuring secure and efficient data transfer.

**Security Considerations:**

* **Validation and Sanitization:** Ensuring that the input from users is validated and sanitized before processing.
* **SSL/TLS:** For securing the data transmitted between the front-end and back-end.

### **4. Class Diagram**

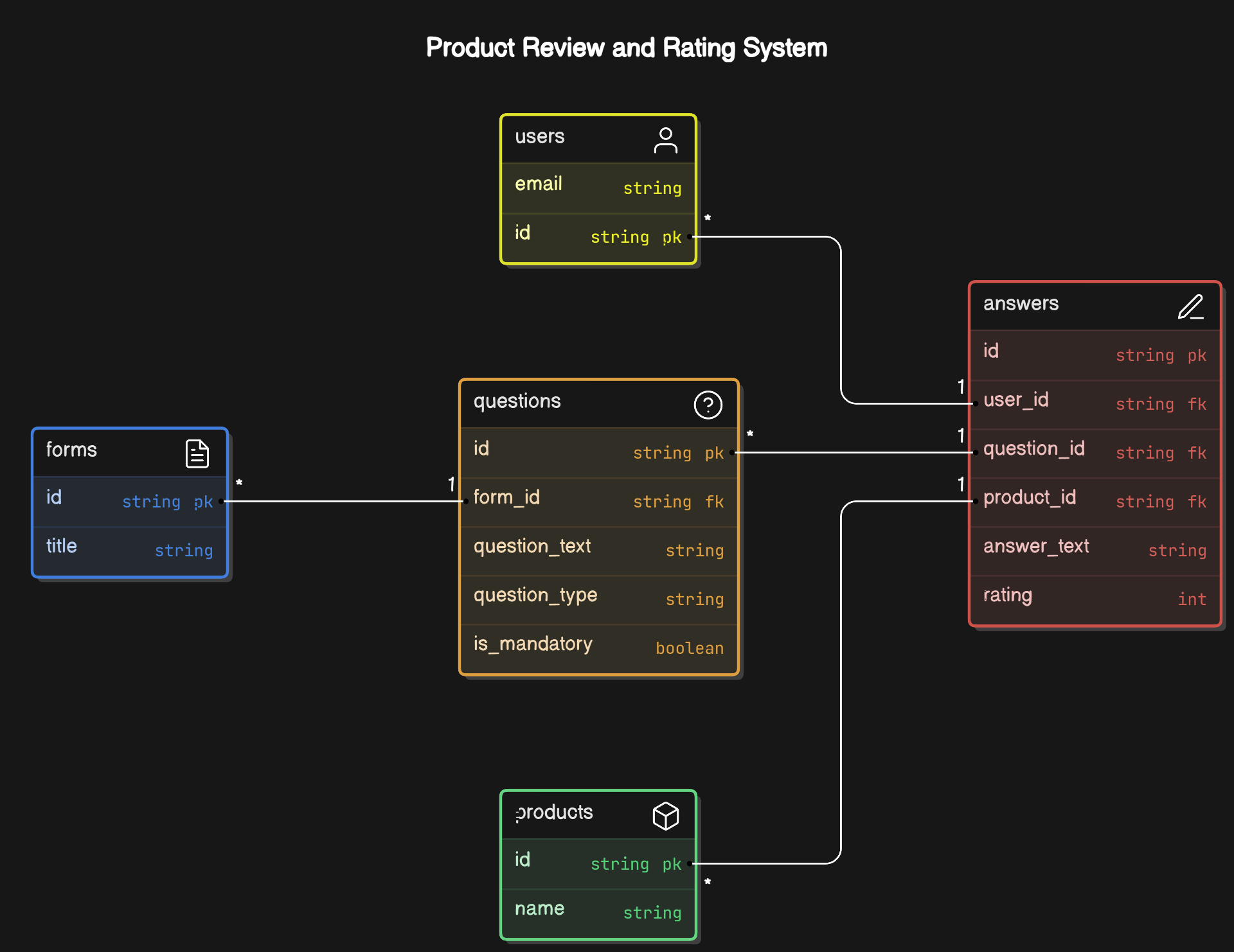
**Description of the Class Structure:** The class diagram will detail the different entities within the application, such as User, Admin, Form, Question, Answer, etc., and how they interact with each other.

**Class Diagram Representation:**

* **User:** Represents the end-user filling out the form.
* **Admin:** Represents the admin accessing the dashboard.
* **Form:** Represents the web form, containing questions and fields.
* **Question:** Represents each question in the form, including type and validation rules.
* **Answer:** Represents the answers submitted by the users.

**Explanation of Each Class and Its Relationships:**

* **User:** Has a relationship with Form where a user submits a form.
* **Admin:** Has access to review Answers stored in the database.
* **Form:** Contains multiple Questions, each of which can have a corresponding Answer.



### **5. Detailed Description of the Web Form**

**Structure of the Form:**

* **Two Mandatory Fields:** Email address and product name at the top.
* **Questions:**
  + 5 Multiple-choice questions.
  + 5 Rating scale questions.
  + 1 Open text question.

**Validation Rules:**

* **Mandatory Fields:** Email, product name, and 8 out of 11 questions.
* **Character Limits:** 2000 characters for the open text box.

**User Experience Considerations:**

* **Responsive Design:** Ensure the form is accessible on all devices.
* **Clear Instructions:** Guide the user on how to fill out the form.

### **6. Backend Functionality**

**Database Schema Design:**

* Tables: users, products, questions, answers.
* Relations: users table linked to answers, questions linked to answers.

**Data Handling and Storage:**

* **Submission Handling:** PHP scripts to validate and store data.
* **Data Integrity:** Using foreign keys and constraints in PostgreSQL.

**Admin Panel Features:**

* **View Submissions:** List all answers with options to filter by product or user.
* **Dashboard Analytics:** Average ratings, count of submissions, etc.

**Dashboard Analytics:**

* **Average Ratings:** Displayed for each product.
* **Counts:** Number of submissions per product, per rating score, etc.

**7. Implemented Plan**

App.js

import React from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import Form from './components/Form';

function App() {

return (

<Router>

<div className="App">

<Routes>

<Route path="/" element={<Form />} />

</Routes>

</div>

</Router>

);

}

export default App;

Form.js

import React, { useState } from 'react';

import MultipleChoice from './MultipleChoice';

import Rating from './Rating';

import TextBox from './TextBox';

import TextInput from './TextInput';

import './Form.css';

const Form = () => {

const [formData, setFormData] = useState({

email: '',

productName: '',

question1: '',

question2: '',

question3: '',

question4: '',

question5: '',

question6: null,

question7: null,

question8: null,

question9: null,

question10: null,

question11: '',

});

const [showSuccess, setShowSuccess] = useState(false);

const handleChange = (e) => {

setFormData({

...formData,

[e.target.name]: e.target.value,

});

setShowSuccess(false);

};

const handleSubmit = async (e) => {

e.preventDefault();

const errors = [];

if (!formData.email) errors.push('Email Address');

if (!formData.productName) errors.push('Product Name');

if (!formData.question1) errors.push('Question 1');

if (!formData.question2) errors.push('Question 2');

if (!formData.question3) errors.push('Question 3');

if (!formData.question4) errors.push('Question 4');

if (!formData.question5) errors.push('Question 5');

if (formData.question6 === '' || formData.question6 === null) errors.push('Question 6');

if (formData.question7 === '' || formData.question7 === null) errors.push('Question 7');

if (!formData.question11) errors.push('Question 11');

if (errors.length > 0) {

alert(`Please fill in the following mandatory fields: ${errors.join(', ')}`);

return;

}

const sanitizedFormData = { ...formData };

for (const key in sanitizedFormData) {

if (sanitizedFormData[key] === null) {

sanitizedFormData[key] = '';

}

}

try {

const response = await fetch('http://localhost/product\_reviews/submit.php', {

method: 'POST',

headers: {

'Content-Type': 'application/x-www-form-urlencoded',

},

body: new URLSearchParams(sanitizedFormData).toString(),

});

if (response.ok) {

setFormData({

email: '',

productName: '',

question1: '',

question2: '',

question3: '',

question4: '',

question5: '',

question6: null,

question7: null,

question8: null,

question9: null,

question10: null,

question11: '',

});

setShowSuccess(true);

} else {

console.error('Submission error:', await response.text());

setShowSuccess(false);

}

} catch (error) {

console.error('Submission error:', error);

setShowSuccess(false);

}

};

return (

<form onSubmit={handleSubmit}>

<h1>Product Review Form</h1>

{showSuccess && (

<p style={{ color: 'green', marginBottom: '20px', textAlign: 'center' }}>

Form submitted successfully!

</p>

)}

<label htmlFor="email">

Email Address <span className="required-asterisk">\*</span>

</label>

<TextInput

name="email"

value={formData.email}

onChange={handleChange}

required

/>

<label htmlFor="productName">

Product Name <span className="required-asterisk">\*</span>

</label>

<select

name="productName"

value={formData.productName}

onChange={handleChange}

required

>

<option value="">Select a Product</option>

<option value="Product A">Product A</option>

<option value="Product B">Product B</option>

<option value="Product C">Product C</option>

<option value="Product D">Product D</option>

</select>

<MultipleChoice

name="question1"

label="1. How would you rate the product quality?"

options={['Excellent', 'Good', 'Average', 'Poor']}

value={formData.question1}

onChange={handleChange}

required

/>

<MultipleChoice

name="question2"

label="2. How satisfied are you with the customer service?"

options={['Very Satisfied', 'Satisfied', 'Neutral', 'Dissatisfied']}

value={formData.question2}

onChange={handleChange}

required

/>

<MultipleChoice

name="question3"

label="3. Would you recommend this product to others?"

options={['Definitely', 'Maybe', 'Not Sure', 'No']}

value={formData.question3}

onChange={handleChange}

required

/>

<MultipleChoice

name="question4"

label="4. How do you rate the value for money?"

options={['Excellent', 'Good', 'Fair', 'Poor']}

value={formData.question4}

onChange={handleChange}

required

/>

<MultipleChoice

name="question5"

label="5. How likely are you to purchase this product again?"

options={['Very Likely', 'Likely', 'Unlikely', 'Never']}

value={formData.question5}

onChange={handleChange}

required

/>

<Rating

name="question6"

label="6. Rate the product's ease of use (1-10)"

value={formData.question6}

onChange={handleChange}

min={0}

max={10}

required

/>

<Rating

name="question7"

label="7. Rate the product's durability (1-10)"

value={formData.question7}

onChange={handleChange}

min={0}

max={10}

required

/>

<Rating

name="question8"

label="8. Rate the product's design (1-10)"

value={formData.question8}

onChange={handleChange}

min={0}

max={10}

/>

<Rating

name="question9"

label="9. Rate the product's performance (1-10)"

value={formData.question9}

onChange={handleChange}

min={0}

max={10}

/>

<Rating

name="question10"

label="10. Rate the product's overall satisfaction (1-10)"

value={formData.question10}

onChange={handleChange}

min={0}

max={10}

/>

<label htmlFor="question11">

11. Additional Feedback <span className="required-asterisk">\*</span>

</label>

<TextBox

name="question11"

value={formData.question11}

onChange={handleChange}

maxLength={2000}

required

/>

<button type="submit">Submit</button>

<p style={{ color: 'red', marginTop: '20px' }}>

\* Asterisk marked fields are mandatory.

</p>

</form>

);

};

export default Form;

admin\_dashboard.php

<?php

include 'db.php';

ini\_set('display\_errors', 1);

ini\_set('display\_startup\_errors', 1);

error\_reporting(E\_ALL);

try {

$query = 'SELECT \* FROM reviews';

$result = pg\_query($conn, $query);

if (!$result) {

throw new Exception('Database error: ' . pg\_last\_error());

}

$reviews = [];

while ($row = pg\_fetch\_assoc($result)) {

$reviews[] = $row;

}

function calculateAverage($reviews, $field) {

$total = array\_sum(array\_column($reviews, $field));

$count = count($reviews);

return $count > 0 ? $total / $count : 0;

}

$averageRatings = [

'question6' => calculateAverage($reviews, 'question6'),

'question7' => calculateAverage($reviews, 'question7'),

'question8' => calculateAverage($reviews, 'question8'),

'question9' => calculateAverage($reviews, 'question9'),

'question10' => calculateAverage($reviews, 'question10')

];

pg\_close($conn);

} catch (Exception $e) {

echo 'Error: ' . $e->getMessage();

exit;

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Admin Dashboard</title>

<style>

body {

background-color: #f0f4f7;

font-family: Arial, sans-serif;

}

.dashboard-container {

width: 90%;

margin: 0 auto;

padding: 20px;

}

table {

width: 100%;

border-collapse: collapse;

overflow-x: auto;

}

th, td {

border: 1px solid #dddddd;

text-align: left;

padding: 8px;

}

th {

background-color: #4CAF50;

color: white;

}

tr:nth-child(even) {

background-color: #f2f2f2;

}

.chart-container {

width: 50%;

margin: 20px auto;

}

</style>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

</head>

<body>

<div class="dashboard-container">

<h1>Admin Dashboard</h1>

<h2>All Submissions</h2>

<div style="overflow-x:auto;">

<table>

<thead>

<tr>

<th>ID</th>

<th>Email</th>

<th>Product Name</th>

<th>Question 1</th>

<th>Question 2</th>

<th>Question 3</th>

<th>Question 4</th>

<th>Question 5</th>

<th>Question 6</th>

<th>Question 7</th>

<th>Question 8</th>

<th>Question 9</th>

<th>Question 10</th>

<th>Question 11</th>

</tr>

</thead>

<tbody>

<?php if (!empty($reviews)) : ?>

<?php foreach ($reviews as $review) : ?>

<tr>

<td><?= htmlspecialchars($review['id']); ?></td>

<td><?= htmlspecialchars($review['email']); ?></td>

<td><?= htmlspecialchars($review['product\_name']); ?></td>

<td><?= htmlspecialchars($review['question1']); ?></td>

<td><?= htmlspecialchars($review['question2']); ?></td>

<td><?= htmlspecialchars($review['question3']); ?></td>

<td><?= htmlspecialchars($review['question4']); ?></td>

<td><?= htmlspecialchars($review['question5']); ?></td>

<td><?= htmlspecialchars($review['question6']); ?></td>

<td><?= htmlspecialchars($review['question7']); ?></td>

<td><?= htmlspecialchars($review['question8'] ?? 'N/A'); ?></td>

<td><?= htmlspecialchars($review['question9'] ?? 'N/A'); ?></td>

<td><?= htmlspecialchars($review['question10'] ?? 'N/A'); ?></td>

<td><?= htmlspecialchars($review['question11']); ?></td>

</tr>

<?php endforeach; ?>

<?php else : ?>

<tr>

<td colspan="14">No reviews available</td>

</tr>

<?php endif; ?>

</tbody>

</table>

</div>

<h2>Average Ratings Visualization</h2>

<div class="chart-container">

<canvas id="averageRatingsChart"></canvas>

</div>

</div>

<script>

const averageRatings = <?= json\_encode($averageRatings); ?>;

const ctx = document.getElementById('averageRatingsChart').getContext('2d');

new Chart(ctx, {

type: 'bar',

data: {

labels: ['Ease of Use', 'Durability', 'Design', 'Performance', 'Overall Satisfaction'],

datasets: [{

label: 'Average Ratings',

data: [

averageRatings.question6.toFixed(2),

averageRatings.question7.toFixed(2),

averageRatings.question8.toFixed(2),

averageRatings.question9.toFixed(2),

averageRatings.question10.toFixed(2)

],

backgroundColor: [

'#4CAF50',

'#FFC107',

'#2196F3',

'#FF5722',

'#9C27B0'

],

borderColor: [

'#388E3C',

'#FFA000',

'#1976D2',

'#D84315',

'#7B1FA2'

],

borderWidth: 1

}]

},

options: {

scales: {

y: {

beginAtZero: true

}

}

}

});

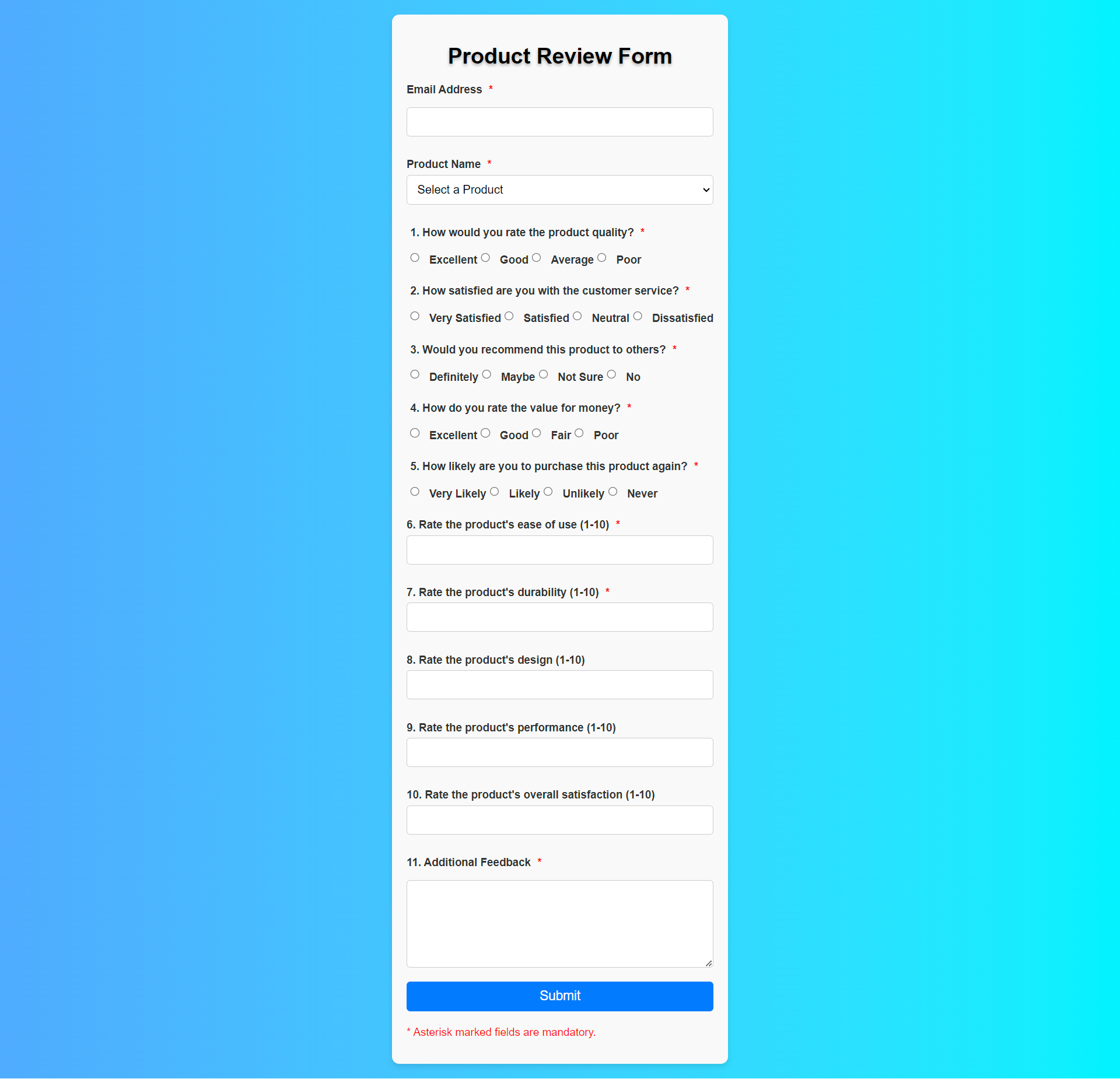
</script>

</body>

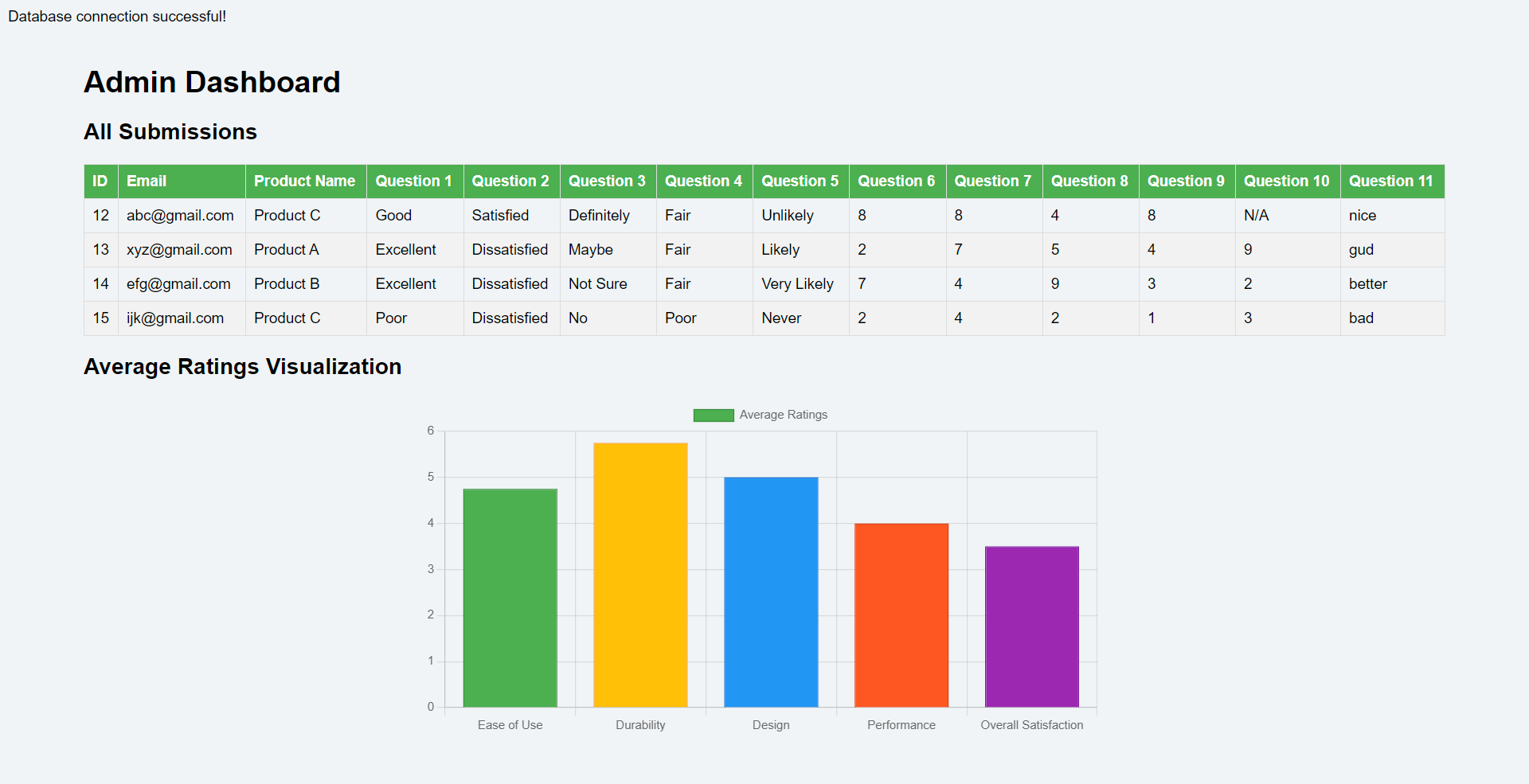
</html>

**8. Screenshots**

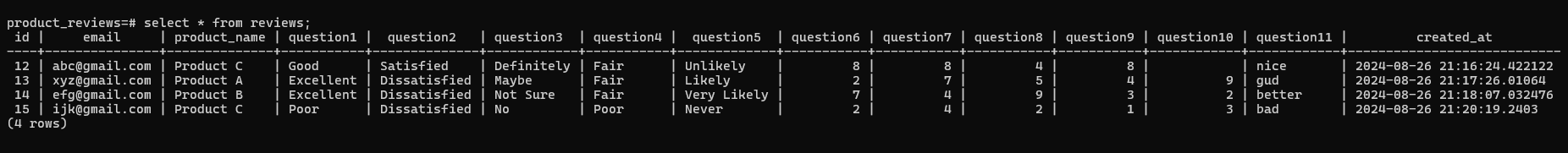
Form Page

****

Admin Dashboard

****

Database

****

### **9. Conclusion**

**Summary of the Solution:** The proposed solution outlines a comprehensive approach to building a product rating and review application using ReactJS for the front-end, PHP for the back-end, and PostgreSQL for data storage.

**Future Enhancements:**

* **Additional Features:** Implementing user authentication, exporting data, and more detailed analytics.
* **Scalability:** Optimizing the system for handling large volumes of data and concurrent users.