1. Introduction

1.1 Purpose

The purpose of this System Requirements Specification (SRS) document is to define the requirements for the **Product Review System**. This system is designed to facilitate user-generated reviews of products and allow an administrative team to monitor, approve, and manage these reviews. The system will consist of two main components:

- A **user-facing web application** for submitting and viewing reviews.
- An **admin desktop application** for managing reviews and assigning credit points to users.

1.2 Scope

The **Product Review System** will provide an online platform where users can register, log in, and submit reviews for various products. The system will categorize products into "Free Review" and "Paid Review," allowing users to earn credit points for high-quality reviews in the paid category. Administrators will use a desktop application to review and approve user submissions, assign credit points, and manage user accounts. The system will securely store user data, including personal information, bank details, UPI IDs, and PAN card information.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: System Requirements Specification.
- **UI**: User Interface.
- **UX**: User Experience.
- **ER Diagram**: Entity-Relationship Diagram.
- **DFD**: Data Flow Diagram.
- **SQL**: Structured Query Language.
- **UPI**: Unified Payments Interface.
- **PAN**: Permanent Account Number.

1.4 References

- IEEE Standards for SRS Documentation.
- Web Application Security Guidelines (OWASP).
- Database Design and Development Guidelines.

1.5 Overview

This SRS document is organized into the following sections:

- **System Overview**: Provides a high-level description of the system, its context, and its key features.
- **Functional Requirements**: Details the system’s functionalities, including user interface requirements, authentication, review management, and the credit point system.
- **Non-Functional Requirements**: Outlines performance, security, usability, reliability, and other quality attributes of the system.
- **System Architecture**: Describes the architecture of the system, including database design and system interfaces.
- **System Models**: Includes use case diagrams, ER diagrams, DFDs, and other relevant models.
- **System Implementation**: Discusses the development tools, coding standards, and testing strategies.
- **Deployment**: Provides details on the deployment environment, installation instructions, and the deployment plan.
- **Maintenance and Support**: Covers the maintenance strategy and support plan for the system.
- **Appendices**: Includes additional diagrams, glossary, and references.

2. System Overview

2.1 Product Perspective

The **Product Review System** is a standalone system designed to manage user-generated product reviews. It consists of two main components:

- **User-Facing Web Application**: This component allows users to register, log in, view products, submit reviews, and manage their profile and credit points. It will be developed using HTML, CSS, JavaScript, and other web technologies.
- **Admin Desktop Application**: This component allows administrators to monitor, approve, and manage user reviews and assign credit points. It will be developed using .NET and C# as a Windows desktop application.

The system is intended to function independently without integration with external systems, except for potential third-party services for user authentication, payment processing, or analytics. The database will be hosted on a cloud server, accessible by both the web and desktop applications.

2.2 Product Features

The key features of the **Product Review System** include:

- **User Registration and Login**: Users can create accounts and log in to access the system. Authentication will be handled securely using hashed passwords and database validation.
- **Product Review Submission**: Users can view products and submit reviews, including ratings and text comments. Reviews will be categorized into "Free Review" and "Paid Review."
- **Credit Point System**: Users can earn credit points based on the quality of their reviews. Points will be displayed on the user's dashboard.
- **Admin Review Management**: Administrators can view and approve or reject user-submitted reviews. They can also assign credit points to users.
- **User Data Management**: The system will store and manage user data, including personal information, bank details, UPI IDs, and PAN card information, with appropriate security measures.

2.3 User Classes and Characteristics

The system will support two primary user classes:

- **Regular Users**: Individuals who register on the platform to submit and view product reviews. These users can:
    - Create an account and log in.
    - View products and submit reviews.
    - Earn and track credit points.
    - Manage their account information, including personal details and bank/UPI information.
- **Administrators**: Individuals responsible for managing the system and user-generated content. These users can:
    - Monitor user-submitted reviews.
    - Approve or reject reviews.
    - Assign credit points to users.
    - Manage user accounts and handle any administrative tasks.

2.4 Operating Environment

The **Product Review System** will operate in the following environments:

- **User-Facing Web Application**:

    - **Client Side**: Modern web browsers (e.g., Google Chrome, Mozilla Firefox, Microsoft Edge).
    - **Server Side**: Web server hosting the application, potentially on a cloud platform like AWS or Azure.
- **Admin Desktop Application**:
    - **Operating System**: Windows 10 or later.
    - **Development Framework**: .NET Framework or .NET Core, depending on the version requirements.
    - **Database**: Cloud-hosted SQL database accessible by the desktop application.

2.5 Design and Implementation Constraints

The system will need to adhere to the following constraints:

- **Security**: Sensitive user data (e.g., bank details, UPI IDs, PAN card information) must be securely stored and transmitted using encryption (e.g., HTTPS, AES encryption).
- **Scalability**: The system should be designed to handle a growing number of users and reviews, with a scalable database and server architecture.
- **Compatibility**: The web application must be compatible with modern web browsers, and the desktop application must be compatible with Windows 10 or later.
- **Performance**: The system should provide a smooth and responsive user experience, with minimal latency for data retrieval and processing.
- **Data Integrity**: The system must ensure the accuracy and consistency of data, especially in financial transactions related to the credit point system.

3. Functional Requirements

This section details the specific functionalities that the **Product Review System** must support. Each requirement will be described in terms of the system's expected behavior and interactions with users.

3.1 User Interface Requirements

3.1.1 User Registration

- **Description**: The system must provide a registration form where new users can create an account.
- **Inputs**: The user must provide a username, email address, password, and optional profile details (e.g., name, phone number).
- **Processing**: The system must validate the provided data for correctness and uniqueness (e.g., ensuring the email isn’t already in use).
- **Outputs**: Upon successful registration, the system will store the user’s information in the database and notify the user of the successful registration via email.

3.1.2 User Login

- **Description**: The system must provide a login form where registered users can enter their credentials to access their account.
- **Inputs**: The user must provide a username or email address and a password.
- **Processing**: The system will verify the credentials against the stored data in the database.
- **Outputs**: If the credentials are correct, the user is granted access to their account and redirected to the product review page. If incorrect, an error message is displayed.

3.1.3 Product Review Submission

- **Description**: Users can view a list of products and submit reviews, which include a rating and text comment.
- **Inputs**: The user selects a product, enters a rating (1-5 stars), and writes a text review.
- **Processing**: The system will save the review data to the database and mark it as pending for admin approval.
- **Outputs**: The review is stored in the database and flagged for admin review. The user is notified that their review is pending approval.

3.1.4 Credit Point System

- **Description**: Users earn credit points for high-quality reviews, which are reflected on their dashboard.
- **Inputs**: Admin assigns credit points based on the quality of the review.
- **Processing**: The system updates the user’s credit point balance and records the transaction.
- **Outputs**: The updated balance is displayed on the user’s dashboard.

3.1.5 User Dashboard

- **Description**: The dashboard provides users with an overview of their reviews, credit points, and account information.
- **Inputs**: The user logs in and navigates to the dashboard.
- **Processing**: The system retrieves and displays relevant data from the database.
- **Outputs**: The dashboard displays the user’s reviews, credit points, and personal details.

3.1.6 Admin Dashboard

- **Description**: The admin dashboard allows administrators to monitor, approve, or reject user reviews and assign credit points.
- **Inputs**: The admin logs in to the desktop application and accesses the review management interface.
- **Processing**: The system fetches pending reviews and allows the admin to take action (approve/reject) and assign credit points.
- **Outputs**: The review status is updated, and if approved, the review is published on the user-facing site. Credit points are added to the user's balance.

3.2 Authentication and Security Requirements

3.2.1 Password Encryption

- **Description**: All user passwords must be securely encrypted before being stored in the database.
- **Processing**: Passwords will be hashed using a secure algorithm (e.g., bcrypt).
- **Outputs**: Only the hashed version of the password is stored.

3.2.2 Session Management

- **Description**: The system must maintain secure sessions for logged-in users, ensuring that only authenticated users can access protected areas.
- **Processing**: Sessions are managed using secure cookies with proper timeout and expiration policies.
- **Outputs**: The user's session is maintained securely until they log out or the session expires.

3.2.3 Data Encryption

- **Description**: Sensitive user data, such as bank details, UPI IDs, and PAN card information, must be encrypted.
- **Processing**: Data is encrypted using AES or similar encryption algorithms before being stored or transmitted.
- **Outputs**: Encrypted data is stored in the database, and decrypted only when necessary.

3.3 Review Management

3.3.1 Review Approval Workflow

- **Description**: All user-submitted reviews must go through an approval process managed by the admin.
- **Inputs**: Reviews submitted by users.
- **Processing**: Reviews are marked as pending until an admin approves or rejects them.
- **Outputs**: Approved reviews are published on the user-facing site; rejected reviews are not displayed.

3.3.2 Review Categorization

- **Description**: Reviews are categorized as either "Free Review" or "Paid Review."
- **Inputs**: The user selects the review category during submission.
- **Processing**: The system categorizes and stores the review based on the user's selection.
- **Outputs**: Reviews are displayed in the appropriate category on the user-facing site.

3.4 Credit Point System

3.4.1 Point Assignment

- **Description**: Admins can assign credit points to users based on the quality of their reviews.
- **Inputs**: Admin assigns points through the desktop application.
- **Processing**: The system updates the user’s credit point balance.
- **Outputs**: The updated balance is displayed on the user’s dashboard.

3.4.2 Point Redemption

- **Description**: Users can redeem their credit points for rewards (optional feature for future development).
- **Inputs**: The user initiates a redemption request.
- **Processing**: The system processes the request and updates the point balance.
- **Outputs**: The reward is processed, and the point balance is adjusted.

4. Non-Functional Requirements

This section outlines the non-functional requirements that the **Product Review System** must meet. These requirements focus on the system's performance, usability, security, and other quality attributes.

4.1 Performance Requirements

4.1.1 Response Time

- **Requirement**: The system must respond to user actions within 2 seconds for most operations, including page loads, form submissions, and data retrieval.
- **Justification**: Ensuring a responsive user interface will enhance the user experience and prevent frustration due to delays.

4.1.2 Scalability

- **Requirement**: The system must be capable of scaling to support up to 10,000 concurrent users without significant degradation in performance.
- **Justification**: As the user base grows, the system should be able to handle increased traffic and data without compromising on performance.

4.1.3 Throughput

- **Requirement**: The system must handle at least 100 transactions per second during peak usage times.
- **Justification**: This ensures that the system can process a high volume of actions, such as reviews being submitted, without delays.

4.2 Security Requirements

4.2.1 Data Encryption

- **Requirement**: All sensitive data, including user credentials, bank details, UPI IDs, and PAN card information, must be encrypted both in transit and at rest.
- **Justification**: Protecting sensitive user data from unauthorized access is critical for maintaining user trust and complying with data protection regulations.

4.2.2 Authentication

- **Requirement**: The system must implement secure authentication mechanisms, including multi-factor authentication (MFA) for admin accounts.
- **Justification**: Enhanced authentication ensures that only authorized users can access sensitive areas of the system.

4.2.3 Data Privacy

- **Requirement**: The system must comply with data privacy regulations such as GDPR (if applicable) or similar local data protection laws.
- **Justification**: Compliance with data privacy laws is essential to avoid legal repercussions and to protect user rights.

4.2.4 Secure Session Management

- **Requirement**: The system must implement secure session management practices, including the use of HTTPS, secure cookies, and session timeouts.

- **Justification**: Secure session management prevents session hijacking and other attacks that could compromise user accounts.

4.3 Usability Requirements

4.3.1 User Interface Design

- **Requirement**: The user interface must be intuitive and easy to navigate, with a consistent design across all pages.
- **Justification**: A user-friendly interface ensures that users can easily interact with the system, reducing the likelihood of errors or frustration.

4.3.2 Accessibility

- **Requirement**: The system must be accessible to users with disabilities, following WCAG 2.1 guidelines to the AA level.
- **Justification**: Ensuring accessibility broadens the system's usability to include users with various disabilities.

4.3.3 User Feedback

- **Requirement**: The system must provide users with clear feedback for every action they take, including form submissions, errors, and successful operations.
- **Justification**: Clear feedback enhances the user experience by confirming actions and guiding users through the process.

4.4 Reliability Requirements

4.4.1 Availability

- **Requirement**: The system must be available 99.9% of the time, with planned downtime limited to non-peak hours.
- **Justification**: High availability ensures that users can access the system whenever they need to, increasing user satisfaction and trust.

4.4.2 Data Backup and Recovery

- **Requirement**: The system must perform daily backups of all critical data and have a recovery plan in place to restore data within 24 hours in the event of a failure.
- **Justification**: Regular backups and a solid recovery plan ensure that data loss is minimized in the event of a system failure.

4.4.3 Error Handling

- **Requirement**: The system must handle errors gracefully, providing users with informative error messages and logging errors for administrative review.
- **Justification**: Effective error handling prevents user frustration and allows admins to quickly diagnose and fix issues.

4.5 Maintainability Requirements

4.5.1 Modularity

- **Requirement**: The system's codebase must be modular, with distinct components for user management, review processing, and admin functionality.

- **Justification**: A modular design makes it easier to maintain, update, and expand the system as needed.

4.5.2 Documentation

- **Requirement**: The system must include comprehensive documentation, including a user manual, admin guide, and developer documentation.
- **Justification**: Detailed documentation ensures that users, admins, and developers can effectively interact with and maintain the system.

4.5.3 Code Quality

- **Requirement**: The system’s code must adhere to coding standards and best practices, including consistent naming conventions, proper commenting, and code reviews.
- **Justification**: High-quality code is easier to maintain, debug, and extend, reducing the risk of bugs and security vulnerabilities.

5. System Architecture

This section provides an overview of the system architecture, describing the components, their interactions, and the technologies used. The architecture is designed to meet the functional and non-functional requirements outlined in previous sections.

5.1 Architectural Overview

The **Product Review System** is composed of two primary components:

1. **User-Facing Web Application**
2. **Admin Desktop Application**

These components interact with a centralized database hosted on a cloud server, ensuring seamless data sharing and consistency across the system.

5.1.1 User-Facing Web Application

- **Frontend**: The frontend is developed using HTML, CSS, and JavaScript. It is responsible for rendering the user interface, managing user interactions, and sending/receiving data from the backend.
- **Backend**: The backend is implemented using a server-side technology (e.g., Node.js, ASP.NET) to handle business logic, database operations, and communication between the frontend and database.
- **Database**: A cloud-hosted SQL database (e.g., MySQL, PostgreSQL) stores user data, reviews, credit points, and other essential information. The backend communicates with the database via an ORM (Object-Relational Mapping) tool or raw SQL queries.

5.1.2 Admin Desktop Application

- **User Interface**: The desktop application, built using .NET and C#, provides administrators with tools to monitor and manage user reviews and credit points.
- **Data Access Layer**: The application connects to the cloud-hosted SQL database to fetch, update, and delete data as necessary. Data access is secured through appropriate authentication and encryption mechanisms.

5.1.3 Database

- **Centralized Cloud Database**: A relational database hosted on a cloud platform (e.g., AWS RDS, Azure SQL Database) serves as the central repository for all system data. It is accessible by both the web application and the desktop application.
- **Tables and Relationships**:
  - **Users**: Stores user credentials, personal information, bank details, UPI IDs, PAN card information, and credit points.
  - **Reviews**: Stores user-submitted reviews, including ratings, comments, and approval status.
  - **Admins**: Stores administrator credentials and access levels.
  - **Credit Points**: Tracks the credit points awarded to each user based on review quality.

5.2 System Components

5.2.1 Frontend

- **Technologies**: HTML, CSS, JavaScript, Bootstrap (or another UI framework)
- **Responsibilities**:
  - Rendering the user interface.
  - Capturing user inputs and events.

    - Communicating with the backend via RESTful APIs.

5.2.2 Backend

- **Technologies**: Node.js/Express or ASP.NET (for server-side logic)
- **Responsibilities**:
    - Handling user authentication and authorization.
    - Processing user requests and sending appropriate responses.
    - Managing business logic, including review submissions, credit point calculations, and dashboard updates.
    - Communicating with the database to perform CRUD (Create, Read, Update, Delete) operations.

5.2.3 Desktop Application

- **Technologies**: .NET Framework/.NET Core, C#
- **Responsibilities**:
    - Providing a user interface for administrators.
    - Connecting to the database to retrieve and manage review data.
    - Allowing admins to approve/reject reviews and assign credit points.
    - Ensuring secure communication with the database.

5.2.4 Database

- **Technologies**: MySQL/PostgreSQL (SQL Database)
- **Responsibilities**:
    - Storing all user, review, admin, and credit point data.
    - Ensuring data consistency and integrity.
    - Supporting complex queries for data retrieval and reporting.

5.3 System Interfaces

5.3.1 User-Facing Web Application Interface

- **User Interface (UI)**: The UI provides users with forms for registration, login, review submission, and dashboard access. It is designed to be intuitive and responsive, ensuring a seamless user experience.
- **API Interface**: The frontend communicates with the backend via RESTful APIs. These APIs handle requests such as user authentication, review submission, and data retrieval for dashboards.

5.3.2 Admin Desktop Application Interface

- **User Interface (UI)**: The desktop application UI allows admins to view pending reviews, approve/reject them, and assign credit points. The interface is designed to be straightforward and efficient for administrative tasks.
- **Data Access Interface**: The application connects directly to the cloud database using secure connections. It uses SQL queries or stored procedures to manage data.

5.3.3 Database Interface

- **Data Access Layer (DAL)**: The backend and desktop application communicate with the database via a Data Access Layer, which abstracts the complexity of SQL queries and ensures secure data transactions.

- **Security Measures**: The database interface must enforce security measures such as role-based access control, encrypted connections, and data validation to protect against SQL injection and other vulnerabilities.

6. System Models

In this section, we'll develop visual representations of the system architecture to clarify the relationships between various components and the flow of data. These models include the Entity-Relationship (ER) Diagram, Data Flow Diagram (DFD), and Use Case Diagram.

6.1 Entity-Relationship (ER) Diagram

The ER diagram visually represents the database structure, showing the entities (tables), their attributes, and the relationships between them.

6.1.1 Entities and Attributes

1. **User**
   - **user_id** (Primary Key)
   - username
   - password
   - email
   - full_name
   - bank_details
   - upi_id
   - pan_card
   - credit_points
2. **Review**
   - **review_id** (Primary Key)
   - user_id (Foreign Key)
   - product_id
   - rating
   - comment
   - status (pending, approved, rejected)
   - submission_date
3. **Admin**
   - **admin_id** (Primary Key)
   - username
   - password
   - role (e.g., Super Admin, Review Manager)
4. **Credit Points**
   - **credit_id** (Primary Key)
   - user_id (Foreign Key)
   - review_id (Foreign Key)
   - points_awarded
   - date_awarded

6.1.2 Relationships

1. **User - Review**: One-to-Many (One user can submit multiple reviews)
2. **Review - Admin**: Many-to-One (Multiple reviews can be approved/rejected by one admin)
3. **User - Credit Points**: One-to-Many (One user can have multiple credit points records)
4. **Review - Credit Points**: One-to-One (Each review can be associated with one credit point record)

6.2 Data Flow Diagram (DFD)

The DFD illustrates how data flows through the system, showing the interaction between users, the system components, and the database.

6.2.1 Level 0 DFD (Context Diagram)

- **External Entities**:
  - **User**: Interacts with the system to log in, submit reviews, and view credit points.
  - **Admin**: Manages user reviews and assigns credit points.
- **System**: Represents the overall Product Review System, interacting with the User and Admin.
- **Database**: Centralized repository for storing all user, review, and admin data.

6.3 Use Case Diagram

The Use Case Diagram illustrates the various interactions between the users (both regular users and admins) and the system.

6.3.1 Actors

- **User**
- **Admin**

6.3.2 Use Cases

1. **User Use Cases**
   - Register
   - Login
   - Submit Review
   - View Reviews
   - View Credit Points
2. **Admin Use Cases**
   - Login
   - View Pending Reviews
   - Approve/Reject Reviews
   - Assign Credit Points
   - Manage Users

6.3.3 Use Case Relationships

- **User - Register**: Allows a new user to create an account.
- **User - Login**: Authenticates the user and grants access to the system.
- **User - Submit Review**: Enables the user to submit a product review.
- **Admin - View Pending Reviews**: Admin can view all reviews awaiting approval.
- **Admin - Approve/Reject Reviews**: Admin decides the status of a user’s review.
- **Admin - Assign Credit Points**: Admin awards credit points to users based on review quality.

7. Detailed Design

In this section, we'll delve into the detailed design of the **Product Review System**. This will cover the implementation specifics, including database schema, class design, and interface details.

7.1 Database Schema Design

The database schema is a more detailed representation of the database structure, building on the ER diagram. This includes table definitions, primary and foreign keys, and any necessary indexes.

7.1.1 Table Definitions

1. **Users Table**
   - **user_id** (Primary Key, Auto Increment)
   - username (VARCHAR, UNIQUE)
   - password (VARCHAR)
   - email (VARCHAR, UNIQUE)
   - full_name (VARCHAR)
   - bank_details (TEXT)
   - upi_id (VARCHAR, UNIQUE)
   - pan_card (VARCHAR, UNIQUE)
   - credit_points (INT, DEFAULT 0)
   - created_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)
   - updated_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)
2. **Reviews Table**
   - **review_id** (Primary Key, Auto Increment)
   - user_id (Foreign Key references Users(user_id))
   - product_id (VARCHAR)
   - rating (INT)
   - comment (TEXT)
   - status (ENUM: 'pending', 'approved', 'rejected')
   - submission_date (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)
3. **Admins Table**
   - **admin_id** (Primary Key, Auto Increment)
   - username (VARCHAR, UNIQUE)
   - password (VARCHAR)
   - role (ENUM: 'Super Admin', 'Review Manager')
   - created_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)
   - updated_at (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP)
4. **Credit_Points Table**
   - **credit_id** (Primary Key, Auto Increment)
   - user_id (Foreign Key references Users(user_id))
   - review_id (Foreign Key references Reviews(review_id))
   - points_awarded (INT)
   - date_awarded (TIMESTAMP, DEFAULT CURRENT_TIMESTAMP)

7.1.2 Indexes and Keys

- **Primary Keys**: Each table has a primary key, ensuring unique identification of each record.
- **Foreign Keys**: Establish relationships between tables (e.g., linking reviews to users).
- **Indexes**: Create indexes on columns frequently used in WHERE clauses (e.g., username, email) to improve query performance.

7.2 Class Design (For Backend)

If using a language like C# or JavaScript for the backend, define classes that mirror the database structure and encapsulate business logic.

7.2.1 User Class

```
csharp
Copy code
public class User
{
    public int UserId { get; set; }
    public string Username { get; set; }
    public string Password { get; set; }
    public string Email { get; set; }
    public string FullName { get; set; }
    public string BankDetails { get; set; }
    public string UpiId { get; set; }
    public string PanCard { get; set; }
    public int CreditPoints { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime UpdatedAt { get; set; }
}
```

7.2.2 Review Class

```
csharp
Copy code
public class Review
{
    public int ReviewId { get; set; }
    public int UserId { get; set; }
    public string ProductId { get; set; }
    public int Rating { get; set; }
    public string Comment { get; set; }
    public string Status { get; set; }
    public DateTime SubmissionDate { get; set; }
}
```

7.2.3 Admin Class

```
csharp
Copy code
public class Admin
{
    public int AdminId { get; set; }
    public string Username { get; set; }
    public string Password { get; set; }
    public string Role { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime UpdatedAt { get; set; }
}
```

7.2.4 CreditPoints Class

```
csharp
Copy code
public class CreditPoints
{
    public int CreditId { get; set; }
    public int UserId { get; set; }
    public int ReviewId { get; set; }
    public int PointsAwarded { get; set; }
    public DateTime DateAwarded { get; set; }
}
```

7.3 Interface Design

The interface design outlines how users will interact with the system. This includes the design of web pages for the user-facing application and the desktop application interface for admins.

7.3.1 User-Facing Web Application

1. **Login Page**
   - **Fields**: Username, Password
   - **Buttons**: Login, Create New Account
   - **Features**: Password recovery, CAPTCHA for security
2. **Dashboard**
   - **Sections**: Overview, My Reviews, My Credit Points
   - **Features**: Graphical representation of credit points earned, List of reviews submitted with status
3. **Review Submission**
   - **Fields**: Product ID (auto-filled by selection), Rating (1-5), Comment
   - **Buttons**: Submit Review
4. **Paid Review Section**
   - **Sections**: Available Reviews, Earn Credit Points
   - **Features**: List of paid reviews with potential earnings, Button to claim review task

7.3.2 Admin Desktop Application

1. **Login Screen**
   - **Fields**: Username, Password
   - **Buttons**: Login
   - **Features**: Admin role-based access control
2. **Review Management**
   - **Sections**: Pending Reviews, Approved Reviews, Rejected Reviews
   - **Features**: View details of each review, Approve/Reject with comments
3. **User Management**
   - **Features**: View and edit user profiles, Manage user roles, Award Credit Points manually
4. **Dashboard**
   - **Sections**: System Overview, User Activity, Review Statistics
   - **Features**: Graphs and charts for review status, User engagement metrics

8. Implementation Plan

The Implementation Plan details the step-by-step approach to building the Product Review System, including timelines, resources, and the sequence of development tasks.

8.1 Development Phases

The project will be divided into several phases, each focusing on a specific aspect of the system.

8.1.1 Phase 1: Project Setup and Database Design

- **Tasks**:
  - Set up the development environment (IDE, version control, etc.).
  - Design and create the database schema.
  - Implement basic database operations (CRUD for users, reviews, and admin).
- **Timeline**: 1 week
- **Resources**: 1 Backend Developer, 1 Database Administrator

8.1.2 Phase 2: User Authentication and Dashboard

- **Tasks**:
  - Develop the user registration and login system.
  - Implement session management for user authentication.
  - Create the user dashboard, displaying reviews and credit points.
- **Timeline**: 2 weeks
- **Resources**: 1 Backend Developer, 1 Frontend Developer

8.1.3 Phase 3: Review Submission and Management

- **Tasks**:
  - Implement the review submission feature.
  - Develop the interface for users to submit and view their reviews.
  - Create the admin interface for managing and approving reviews.
- **Timeline**: 2 weeks
- **Resources**: 1 Backend Developer, 1 Frontend Developer

8.1.4 Phase 4: Credit Points System

- **Tasks**:
  - Implement the logic for awarding credit points based on approved reviews.
  - Integrate the credit points system with the user dashboard.
  - Enable admin management of credit points.
- **Timeline**: 1 week
- **Resources**: 1 Backend Developer

8.1.5 Phase 5: Admin Desktop Application

- **Tasks**:
  - Develop the .NET C# desktop application for admin management.
  - Implement features for managing reviews, users, and credit points.
  - Ensure secure admin login and role-based access control.
- **Timeline**: 3 weeks
- **Resources**: 1 C# Developer

8.1.6 Phase 6: Frontend Design and Final Integration

- **Tasks**:
  - Finalize the design of the web application’s frontend (HTML, CSS, JS).
  - Integrate the frontend with the backend services.
  - Test the complete system to ensure smooth interaction between components.
- **Timeline**: 2 weeks
- **Resources**: 1 Frontend Developer, 1 Backend Developer

8.2 Resource Allocation

- **Backend Developer(s)**: Responsible for server-side logic, database interactions, and API development.
- **Frontend Developer(s)**: Focus on creating the user-facing part of the application using HTML, CSS, and JavaScript.
- **C# Developer(s)**: Develop the admin desktop application using .NET and C#.
- **Database Administrator(s)**: Design, implement, and manage the database.
- **Project Manager**: Oversee the project’s progress, manage timelines, and ensure that all requirements are met.

8.3 Milestones and Deadlines

- **Milestone 1: Project Setup and Database Design** - [Week 1]
- **Milestone 2: User Authentication and Dashboard** - [Week 3]
- **Milestone 3: Review Submission and Management** - [Week 5]
- **Milestone 4: Credit Points System** - [Week 6]
- **Milestone 5: Admin Desktop Application** - [Week 9]
- **Milestone 6: Final Integration and Testing** - [Week 11]

8.4 Testing and Quality Assurance

Once the development phases are complete, a comprehensive testing strategy will be implemented:

8.4.1 Unit Testing

- **Scope**: Individual components and functions.
- **Responsibility**: Developers.

8.4.2 Integration Testing

- **Scope**: Interactions between integrated modules.
- **Responsibility**: QA Team.

8.4.3 User Acceptance Testing (UAT)

- **Scope**: Validate that the system meets user needs.
- **Responsibility**: End-users and QA Team.

8.4.4 Performance Testing

- **Scope**: Ensure the system can handle expected load and perform well under stress.
- **Responsibility**: QA Team.

8.4.5 Security Testing

- **Scope**: Identify vulnerabilities and ensure data protection.
- **Responsibility**: Security Specialist.

9. Testing Plan

The Testing Plan is a crucial part of the project, ensuring that the Product Review System functions correctly, meets user requirements, and is free from critical bugs or vulnerabilities. The plan will cover different types of testing to be performed throughout the development lifecycle.

9.1 Types of Testing

9.1.1 Unit Testing

- **Objective**: Validate that each individual unit of the system functions as intended.
- **Scope**: Methods and functions within classes (e.g., `User`, `Review`, `Admin`).
- **Tools**: NUnit (for .NET C#), Jest (for JavaScript).
- **Responsibility**: Developers.
- **Timeline**: Ongoing throughout development.

9.1.2 Integration Testing

- **Objective**: Ensure that different modules of the system work together as expected.
- **Scope**: Interaction between components such as the database, backend APIs, and frontend interface.
- **Tools**: Postman (for API testing), Selenium (for web UI testing).
- **Responsibility**: QA Team.
- **Timeline**: After the integration of major modules.

9.1.3 System Testing

- **Objective**: Validate the entire system as a whole, ensuring all components work together seamlessly.
- **Scope**: Full end-to-end testing, covering all functionalities (e.g., user login, review submission, admin approval).
- **Tools**: Manual testing and automated testing tools.
- **Responsibility**: QA Team.
- **Timeline**: After completing integration testing.

9.1.4 User Acceptance Testing (UAT)

- **Objective**: Ensure the system meets user requirements and expectations.
- **Scope**: Testing real-world scenarios as a user or admin.
- **Participants**: End-users, project stakeholders.
- **Responsibility**: QA Team and selected end-users.
- **Timeline**: After system testing.

9.1.5 Performance Testing

- **Objective**: Assess the system's performance under various conditions (load, stress, scalability).
- **Scope**: Testing system response time, throughput, and resource utilization.
- **Tools**: JMeter, LoadRunner.
- **Responsibility**: QA Team.
- **Timeline**: Before deployment.

9.1.6 Security Testing

- **Objective**: Identify and mitigate security vulnerabilities.
- **Scope**: Testing for SQL injection, cross-site scripting (XSS), data encryption, and secure authentication.
- **Tools**: OWASP ZAP, Burp Suite.
- **Responsibility**: Security Specialist.

- **Timeline**: Throughout the testing phase.

9.2 Test Cases and Scenarios

Each type of testing will include specific test cases to validate the corresponding functionality. Here are examples for key components:

9.2.1 User Authentication

- **Test Case**: Verify that the login system accepts valid credentials.
- **Expected Result**: User is authenticated and redirected to the dashboard.
- **Test Case**: Ensure the system rejects invalid credentials.
- **Expected Result**: User receives an error message and is prompted to retry.

9.2.2 Review Submission

- **Test Case**: Validate that a user can submit a review for a product.
- **Expected Result**: Review is saved in the database with a status of "pending."
- **Test Case**: Ensure the system displays the correct list of products for review.
- **Expected Result**: User sees products categorized into "Free Review" and "Paid Review."

9.2.3 Admin Review Management

- **Test Case**: Verify that an admin can approve or reject a user review.
- **Expected Result**: Review status is updated in the database, and the user is notified.
- **Test Case**: Ensure credit points are correctly awarded upon review approval.
- **Expected Result**: User's credit points are updated, and the change is reflected on the dashboard.

9.2.4 Security

- **Test Case**: Attempt SQL injection in the login form.
- **Expected Result**: System prevents injection and shows an error message.
- **Test Case**: Validate that user data, including bank details and PAN card information, is securely encrypted.
- **Expected Result**: Data is encrypted in transit and at rest.

9.3 Testing Schedule

- **Week 1-2**: Unit Testing for Phase 1 (Project Setup and Database Design)
- **Week 3-4**: Integration Testing for Phase 2 (User Authentication and Dashboard)
- **Week 5-6**: System Testing for Phase 3 (Review Submission and Management)
- **Week 7**: Performance and Security Testing for Phase 4 (Credit Points System)
- **Week 8-9**: UAT and System Testing for Phase 5 (Admin Desktop Application)
- **Week 10**: Final Testing and Bug Fixes for Phase 6 (Frontend Design and Final Integration)

10. Deployment Plan

The Deployment Plan outlines the process of releasing the Product Review System to a live environment where users and admins can access it. This plan covers pre-deployment activities, deployment steps, and post-deployment considerations to ensure a smooth launch.

10.1 Pre-Deployment Activities

10.1.1 Final Testing

- **Objective**: Ensure that all features work as intended and that the system is free from critical bugs.
- **Activities**:
  - Conduct a final round of testing (UAT, system testing, performance testing).
  - Address any issues found during the final testing phase.
- **Responsibility**: QA Team, Development Team.

10.1.2 Security Review

- **Objective**: Confirm that the system is secure and user data is protected.
- **Activities**:
  - Perform a comprehensive security audit.
  - Implement necessary security patches and fixes.
- **Responsibility**: Security Specialist.

10.1.3 Backup and Recovery Plan

- **Objective**: Prepare for potential data loss or system failure during deployment.
- **Activities**:
  - Set up regular backups of the database and critical system files.
  - Develop a recovery plan to restore the system in case of failure.
- **Responsibility**: Database Administrator, System Administrator.

10.1.4 Training and Documentation

- **Objective**: Ensure that users and admins know how to use the system effectively.
- **Activities**:
  - Create user manuals and admin guides.
  - Conduct training sessions for end-users and admins.
- **Responsibility**: Project Manager, Training Team.

10.2 Deployment Steps

10.2.1 Environment Setup

- **Objective**: Prepare the live environment for deployment.
- **Activities**:
  - Configure the production server (web server, database server, etc.).
  - Set up necessary software and tools (e.g., .NET framework, web hosting services).
  - Ensure that the live environment mirrors the development and testing environments.
- **Responsibility**: System Administrator.

10.2.2 Database Migration

- **Objective**: Populate the production database with necessary data.

- **Activities**:
    - Run database scripts to create tables, indexes, and relationships.
    - Migrate data from the development environment to the live environment.
- **Responsibility**: Database Administrator.

10.2.3 Application Deployment

- **Objective**: Deploy the web application and admin desktop application to the live environment.
- **Activities**:
    - Deploy the frontend and backend code to the web server.
    - Distribute the admin desktop application to the relevant stakeholders.
    - Configure application settings (e.g., API keys, environment variables).
- **Responsibility**: Development Team, System Administrator.

10.2.4 Testing in Production

- **Objective**: Verify that the system works as expected in the live environment.
- **Activities**:
    - Perform smoke testing to ensure the critical functionalities are operational.
    - Conduct a limited release to a small group of users (beta testing).
    - Monitor system performance and address any issues immediately.
- **Responsibility**: QA Team, Development Team.

10.2.5 Go-Live

- **Objective**: Make the system available to all users.
- **Activities**:
    - Announce the official launch to users and stakeholders.
    - Monitor the system closely for any issues during the initial period.
- **Responsibility**: Project Manager, System Administrator.

10.3 Post-Deployment Activities

10.3.1 Monitoring and Maintenance

- **Objective**: Ensure ongoing system performance and stability.
- **Activities**:
    - Set up monitoring tools to track system performance, usage, and security.
    - Implement a maintenance schedule for regular updates and patches.
- **Responsibility**: System Administrator, Development Team.

10.3.2 Support and Troubleshooting

- **Objective**: Provide assistance to users and resolve any post-launch issues.
- **Activities**:
    - Set up a support system (helpdesk, ticketing system).
    - Assign a team to handle troubleshooting and user queries.
- **Responsibility**: Support Team, Development Team.

10.3.3 Feedback and Continuous Improvement

- **Objective**: Gather user feedback to improve the system.
- **Activities**:
    - Create channels for users to provide feedback.

    - Analyze feedback and plan for future updates or feature enhancements.
- **Responsibility**: Project Manager, Development Team.

11. Maintenance and Support Plan

The Maintenance and Support Plan ensures the Product Review System remains functional, secure, and up-to-date after deployment. This plan outlines the strategies for regular maintenance, ongoing support, and future enhancements.

11.1 Maintenance Plan

11.1.1 Regular Updates

- **Objective**: Keep the system current with the latest features, bug fixes, and security patches.
- **Activities**:
    - **Software Updates**: Regularly update the system software, including the web application, admin desktop application, and any third-party libraries or frameworks.
    - **Security Patches**: Apply security patches as they are released to protect against vulnerabilities.
    - **Performance Tuning**: Monitor and optimize the system's performance, addressing any bottlenecks or inefficiencies.
- **Frequency**: Monthly for routine updates; as needed for critical patches.
- **Responsibility**: Development Team, System Administrator.

11.1.2 Backup and Recovery

- **Objective**: Ensure data integrity and availability in case of system failure or data loss.
- **Activities**:
    - **Automated Backups**: Set up automated backups of the database and critical files, storing them in a secure location.
    - **Recovery Testing**: Regularly test the backup and recovery process to ensure data can be restored quickly and accurately.
    - **Disaster Recovery Plan**: Maintain a disaster recovery plan that outlines steps to recover from major system failures.
- **Frequency**: Daily for backups; quarterly for recovery testing.
- **Responsibility**: Database Administrator, System Administrator.

11.1.3 System Monitoring

- **Objective**: Continuously monitor the system to detect and resolve issues proactively.
- **Activities**:
    - **Performance Monitoring**: Use monitoring tools to track system performance metrics such as CPU usage, memory utilization, and response times.
    - **Security Monitoring**: Implement security monitoring to detect unauthorized access attempts, data breaches, or other security incidents.
    - **Error Logging**: Set up error logging to capture and analyze system errors, enabling quick resolution of issues.
- **Frequency**: Continuous monitoring with real-time alerts.
- **Responsibility**: System Administrator, Security Specialist.

11.1.4 Preventive Maintenance

- **Objective**: Prevent system issues by performing regular preventive maintenance tasks.
- **Activities**:
    - **Database Maintenance**: Regularly optimize database indexes, remove obsolete data, and check for corruption.
    - **Server Maintenance**: Perform server updates, patching, and hardware checks.
    - **Code Review**: Periodically review and refactor the codebase to ensure it remains clean, efficient, and secure.

- **Frequency**: Monthly for database and server maintenance; quarterly for code reviews.
- **Responsibility**: Database Administrator, System Administrator, Development Team.

11.2 Support Plan

11.2.1 Helpdesk and Ticketing System

- **Objective**: Provide users with a structured way to report issues and request assistance.
- **Activities**:
  - **Ticketing System**: Implement a ticketing system where users can submit support requests.
  - **Helpdesk Support**: Set up a helpdesk to provide first-level support for common issues.
  - **Escalation Process**: Define an escalation process for more complex issues that require advanced troubleshooting.
- **Availability**: 24/7 for ticketing; business hours for helpdesk support.
- **Responsibility**: Support Team.

11.2.2 Troubleshooting and Issue Resolution

- **Objective**: Quickly resolve user-reported issues and minimize downtime.
- **Activities**:
  - **Issue Triage**: Prioritize issues based on severity and impact.
  - **Bug Fixing**: Assign bugs to the development team for resolution, with a focus on critical and high-priority issues.
  - **User Communication**: Keep users informed of issue status, expected resolution times, and any workarounds.
- **Response Time**: Within 24 hours for critical issues; 48-72 hours for non-critical issues.
- **Responsibility**: Support Team, Development Team.

11.2.3 User Feedback and Improvement

- **Objective**: Continuously improve the system based on user feedback and evolving needs.
- **Activities**:
  - **Feedback Collection**: Provide users with a feedback mechanism (e.g., surveys, feedback forms) to share their experiences and suggestions.
  - **Analysis and Planning**: Analyze feedback to identify areas for improvement, and plan future updates or new features.
  - **Release Management**: Manage the release of new features and improvements, ensuring they are thoroughly tested before deployment.
- **Frequency**: Quarterly feedback analysis; bi-annual release cycles.
- **Responsibility**: Project Manager, Development Team.