Technical Specification: Subscription & Credit System

1. Overview

This document outlines the technical requirements for implementing a subscription-based credit system for the tender bidding platform. The system will allow users (bidders) to purchase plans that grant them a specific number of "credits." These credits are consumed each time a user downloads a tender proposal. The system must handle credit deduction, plan expiration, and automatic subscription renewals (autopay).You can choose any payment gateway by satisfying all requirements.

NOTE: if Payment gateway provides apis for the following service then you can use them but our MongoDB database should be updated well.

Tech Stack:

- **Backend:** Node.js, Express.js (With TS)
- **Database:** MongoDB
- **Frontend:** React.js(With TS)

2. Database Schema (MongoDB)

We will use two main collections: users and orders.

2.1. users Collection

This collection stores information about each user, including their current subscription status and credit balance.

Schema:

```
{
  "_id": ObjectId,
  "userType": {
    "type": "String",
    "enum": ["bidder", "owner", "admin"],
    "required": true
  },
  "email": {
    "type": "String",
    "required": true,
    "unique": true
  },
  "credit": {
    "type": "Number",
    "description": "Number of proposals the user can download.",
    "default": 0,
    "min": 0
  },
  "planType": {
    "type": "String",
    "description": "The user's current subscription plan.",
    "enum": ["none", "base", "enterprise"],
    "default": "none"
  },
  "currentOrderId": {
    "type": "ObjectId",
    "ref": "Order",
    "description": "Reference to the currently active subscription order.",
    "default": null
  },
  "autoPayEnabled": {
    "type": "Boolean",
    "description": "Flag to indicate if the user has enabled automatic renewal.",
    "default": false
  },
  "paymentGatewayCustomerId": {
    "type": "String",
    "description": "Customer ID from the payment gateway (e.g., Stripe, Razorpay) for
managing recurring payments.",
    "default": null
  },
  "createdAt": {
    "type": "Date",
    "default": "Date.now"
  }
}
```

2.2. orders Collection

This collection logs every subscription payment transaction.

Schema:

```
{
```

```
"_id": ObjectId,
"userId": {
  "type": "ObjectId",
  "ref": "User",
  "required": true
},
"dateOfPayment": {
  "type": "Date",
  "default": "Date.now"
},
"amount": {
  "type": "Number",
  "required": true
},
"planType": {
  "type": "String",
  "enum": [ "base", "enterprise"],
  "required": true
},
"paymentStatus": {
  "type": "String",
  "enum": ["pending", "successful", "failed"],
  "default": "pending"
},
"startDate": {
  "type": "Date",
  "description": "The date the subscription becomes active."
},
"endDate": {
  "type": "Date",
  "description": "The date the subscription expires."
},
"creditsPurchased": {
  "type": "Number",
  "description": "Number of credits provided by this order."
},
"isExpiredProcessed": {
  "type": "Boolean",
  "description": "Flag to indicate if the expiration logic has been run for this order.",
  "default": false
},
"paymentGatewayTransactionId": {
  "type": "String",
    "description": "Transaction ID from the payment gateway."
  }
}
```

3. API Endpoints

The following RESTful API endpoints need to be created.

3.1. POST /api/proposals/download

- **Description:** Allows an authenticated user to download a tender proposal. This is the primary endpoint for credit consumption.
- **Authentication:** Required (User Token).
- **Request Body:** { "proposalId": "..." }
- **Logic:**
  1. Verify the user is authenticated.
  2. Fetch the user's document from the users collection.
  3. Check if user.credit > 0.
  4. **If credits are available:**
     - Decrement the user's credit count by 1 (credit: credit - 1).
     - Save the updated user document.
     - Proceed to generate and stream the proposal file to the user.
     - Return 200 OK with the file.
  5. **If credits are zero:**
     - Do not proceed with the download.
     - Return 402 Payment Required with a JSON response: { "message": "Insufficient credits. Please purchase a plan to download proposals." }

3.2. POST /api/payments/create-order

- **Description:** Initiates the payment process when a user chooses a subscription plan.
- **Authentication:** Required (User Token).
- **Request Body:** { "planType": "premium", "duration": "monthly" }
- **Logic:**
  1. Based on the planType, determine the amount and creditsPurchased.
  2. Create a new document in the orders collection with paymentStatus: 'pending', userId, planType, and amount.
  3. Integrate with the payment gateway (e.g., Razorpay, Stripe) to create a payment order.
  4. Return the payment order details from the gateway to the client-side to open the payment modal.
  5. Example Response: 201 Created with { "orderId": "...", "gatewayOrderId": "...", "amount": "...", "currency": "..." }.

3.3. POST /api/payments/verify

- **Description:** A webhook endpoint for the payment gateway to send confirmation of payment status.
- **Authentication:** Webhook signature verification.
- **Request Body:** Varies by payment gateway, but will contain details to identify the order and confirm payment success.
- **Logic:**
    1. Verify the webhook signature to ensure the request is from the payment gateway.
    2. On successful payment:
        - Find the corresponding order in the orders collection.
        - Update paymentStatus to "successful".
        - Set the startDate (now) and endDate (e.g., 30 days from now).
        - Store the paymentGatewayTransactionId.
        - Find the associated user via userId.
        - Update the user document:
            - Add the creditsPurchased to the user's existing credit.
            - Set the user's planType.
            - Set the currentOrderId to this order's ID.
    3. Return 200 OK to the payment gateway.

3.4. POST /api/user/autopay

- **Description:** Allows a user to enable or disable automatic subscription renewal. **Also send reminder one day before payment.**
- **Authentication:** Required (User Token).
- **Request Body:** { "enable": true }
- **Logic:**
    1. Find the authenticated user.
    2. Update the autoPayEnabled field to true or false.
    3. If enabling, ensure a paymentGatewayCustomerId exists or is created for the user.
    4. Return 200 OK with { "message": "Autopay settings updated successfully." }

4. Scheduled Task (Cron Job)

A background task must run daily to handle subscription expirations. A library like node-cron can be used for this.

- **Schedule:** Once every 24 hours (e.g., at 00:01 / 12:01 AM).
- **Logic:**
    1. Query the orders collection for documents that meet the following criteria:
        - paymentStatus: "successful"
        - endDate: { $lte: new Date() } (end date is today or in the past)
        - isExpiredProcessed: false
    2. For each expired order found:

- Find the corresponding user using order.userId.
- **Important Check:** Only proceed if the user's currentOrderId matches the ID of the expired order. This prevents conflicts if the user has manually renewed their plan already.
- **If Autopay is ON (user.autoPayEnabled === true):**
    - Attempt to charge the user for a new subscription period via the payment gateway using their saved paymentGatewayCustomerId.
    - If payment is successful, create a new order and update the user's credit, planType, endDate, and currentOrderId.
    - If payment fails, send the user a "Payment Failed" notification email. Then proceed with the "Autopay is OFF" logic below.
- **If Autopay is OFF (user.autoPayEnabled === false):**
    - Update the user's document:
        - Set credit: 0.
        - Set planType: "none".
        - Set currentOrderId: null.
    - (Optional) Send a "Your plan has expired" notification email.
- Finally, update the processed order: isExpiredProcessed: true. This is crucial to prevent the same order from being processed again.