**Project: "Auto Plate Bidding API"**

**Description**

Students will create a REST API for a platform where users can bid on rare auto license plates. Each plate is unique, and users compete by placing bids. The highest bid wins the plate when bidding ends. The project emphasizes basic DRF concepts like authentication, serializers, and permissions, with a simple bidding mechanism.

**Requirements**

**General Requirements**

* Build a RESTful API using Django Rest Framework.
* Use token-based authentication (DRF’s TokenAuthentication) to secure endpoints.
* Implement CRUD operations for auto plates and bids.
* Include basic filtering and sorting for plate listings.
* Enforce permissions: users can only bid or edit their own bids, and only admins can manage plates.
* Use serializers with validation to ensure data integrity.
* Write minimal unit tests for core endpoints.

**Technical Stack**

* **Backend**: Django, Django Rest Framework
* **Database**: SQLite (for simplicity)
* **Authentication**: DRF TokenAuthentication
* **Filtering**: Basic query parameters

**Functional Requirements**

1. **User Management**:
   * Use Django’s built-in User model.
   * Users must authenticate to bid or manage their bids.
   * Admins can create and manage auto plates.
2. **Auto Plate Management**:
   * Plates have a unique number, description, and bidding deadline.
   * Only admins can create, update, or delete plates.
   * Plates can be listed publicly with current highest bid info.
3. **Bid Management**:
   * Authenticated users can place bids on plates.
   * Each bid is tied to a user and a plate.
   * Users can only update/delete their own bids before the deadline.
   * The highest bid wins when the deadline passes (manual check for simplicity).

**Models**

1. **User** (Built-in Django Model):
   * Fields: id, username, email, password, is\_staff (for admin status).
2. **AutoPlate**:
   * Fields:
     + id (PK)
     + plate\_number (CharField, max\_length=10, unique=True)
     + description (TextField)
     + deadline (DateTimeField)
     + created\_by (ForeignKey to User, related\_name='plates\_created', limit to staff)
     + is\_active (BooleanField, default=True, indicates if bidding is open)
   * Constraints:
     + plate\_number must be unique.
     + deadline must be in the future when created.
3. **Bid**:
   * Fields:
     + id (PK)
     + amount (DecimalField, max\_digits=10, decimal\_places=2)
     + user (ForeignKey to User, related\_name='bids')
     + plate (ForeignKey to AutoPlate, related\_name='bids')
     + created\_at (DateTimeField, auto\_now\_add=True)
   * Constraints:
     + amount must be positive.
     + Unique together: user and plate (one bid per user per plate).

**API Endpoints**

**Authentication Endpoint**

1. **POST /login/**
   * Description: Obtain an authentication token.
   * Request Body: { "username": "string", "password": "string" }
   * Response: { "token": "string" }
   * Permissions: Open to all.

**Auto Plate Endpoints**

1. **GET /plates/**
   * Description: List all active plates with current highest bid.
   * Query Params: ?ordering=deadline, ?plate\_number\_\_contains
   * Response: List of { "id": int, "plate\_number": string, "description": string, "deadline": datetime, "highest\_bid": decimal/null }
   * Permissions: Open to all (read-only).
2. **POST /plates/**
   * Description: Create a new plate.
   * Request Body: { "plate\_number": "string", "description": "string", "deadline": "datetime" }
   * Response: Created plate details.
   * Permissions: Admin only.
3. **GET /plates/{id}/**
   * Description: Retrieve plate details with all bids.
   * Response: Plate details including { "bids": [{"amount": decimal, "user": int, "created\_at": datetime}] }
   * Permissions: Open to all (read-only).
4. **PUT /plates/{id}/**
   * Description: Update plate details.
   * Request Body: Same as POST.
   * Permissions: Admin only.
5. **DELETE /plates/{id}/**
   * Description: Delete a plate.
   * Permissions: Admin only.

**Bid Endpoints**

1. **GET /bids/**
   * Description: List all bids by the authenticated user.
   * Response: List of { "id": int, "amount": decimal, "plate": int, "created\_at": datetime }
   * Permissions: Authenticated users (own bids only).
2. **POST /bids/**
   * Description: Place a bid on a plate.
   * Request Body: { "amount": decimal, "plate": int }
   * Response: Created bid details.
   * Permissions: Authenticated users.
3. **GET /bids/{id}/**
   * Description: Retrieve a specific bid.
   * Response: Bid details.
   * Permissions: Bid owner only.
4. **PUT /bids/{id}/**
   * Description: Update a bid (e.g., increase amount).
   * Request Body: { "amount": decimal }
   * Permissions: Bid owner only, before deadline.
5. **DELETE /bids/{id}/**
   * Description: Delete a bid.
   * Permissions: Bid owner only, before deadline.

**Edge Cases to Handle**

1. **Auto Plate Management**:
   * **Duplicate Plate Number**: Attempt to create a plate with an existing plate\_number → Return 400 Bad Request with "Plate number already exists".
   * **Past Deadline**: Create a plate with a deadline in the past → Return 400 with "Deadline must be in the future".
   * **Non-Admin Access**: Non-admin tries to create/update/delete a plate → Return 403 Forbidden.
   * **Delete Plate with Bids**: Allow deletion only if no bids exist, or cascade delete bids → Return 400 with "Cannot delete plate with active bids" if restricted.
2. **Bid Management**:
   * **Bid on Inactive Plate**: Bid on a plate where is\_active=False or deadline passed → Return 400 with "Bidding is closed".
   * **Lower Bid**: Bid amount less than the current highest bid → Return 400 with "Bid must exceed current highest bid".
   * **Multiple Bids**: User tries to bid again on the same plate → Return 400 with "You already have a bid on this plate".
   * **Update After Deadline**: Attempt to update/delete a bid after the plate’s deadline → Return 403 with "Bidding period has ended".
   * **Negative Amount**: Bid with amount <= 0 → Return 400 with "Bid amount must be positive".
3. **Authentication**:
   * **No Token**: Access protected endpoint without a token → Return 401 Unauthorized.
   * **Invalid Token**: Use an incorrect token → Return 401 with "Invalid token".
   * **Non-Owner Access**: Try to view/update/delete someone else’s bid → Return 403 Forbidden.
4. **General**:
   * **Non-Existent Resource**: Request GET /plates/999/ or GET /bids/999/ → Return 404 Not Found.
   * **Invalid Data**: Send a string for amount or invalid plate ID → Return 400 with validation error.
   * **Concurrent Bids**: Two users bid at the same time → Ensure the highest bid is correctly updated (e.g., use database transactions).