**Project: "Auto Plate Bidding API" (FastAPI Version)**

**Description**

This project involves creating a RESTful API using FastAPI for a platform where users can bid on rare auto license plates. Each plate is unique, and users compete by placing bids, with the highest bid winning when the bidding deadline passes. The project focuses on leveraging FastAPI’s features like asynchronous programming, Pydantic validation, and dependency injection, while implementing a simple bidding mechanism.

**Requirements**

**General Requirements**

* Build a RESTful API using FastAPI.
* Use JWT-based authentication to secure endpoints.
* Implement CRUD operations for auto plates and bids.
* Include basic filtering and sorting for plate listings using query parameters.
* Enforce permissions: users can only bid or edit their own bids, and only admins can manage plates.
* Use Pydantic models for request/response validation to ensure data integrity.
* Write minimal unit tests for core endpoints using pytest and httpx.

**Technical Stack**

* **Backend**: FastAPI
* **Database**: SQLite (for simplicity) with SQLAlchemy (async support)
* **Authentication**: JWT (JSON Web Tokens)
* **Filtering**: Query parameters handled natively by FastAPI
* **Validation**: Pydantic models

**Functional Requirements**

* **User Management**:
  + Use a custom user model with fields for authentication.
  + Users must authenticate to bid or manage their bids.
  + Admins can create and manage auto plates.
* **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 information.
* **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**

**User**

* **Fields**:
  + id: Primary key
  + username: Unique string
  + email: Unique string
  + hashed\_password: String (hashed)
  + is\_staff: Boolean (indicates admin status)
* **Notes**: Used for authentication and authorization.

**AutoPlate**

* **Fields**:
  + id: Primary key
  + plate\_number: String (max length 10, unique)
  + description: Text
  + deadline: DateTime
  + created\_by: Foreign key to User (admin who created it)
  + is\_active: Boolean (default True, indicates if bidding is open)
* **Constraints**:
  + plate\_number must be unique.
  + deadline must be in the future when created.

**Bid**

* **Fields**:
  + id: Primary key
  + amount: Decimal (max digits 10, 2 decimal places)
  + user: Foreign key to User
  + plate: Foreign key to AutoPlate
  + created\_at: DateTime (auto-set on creation)
* **Constraints**:
  + amount must be positive.
  + Unique together: user and plate (one bid per user per plate).

**API Endpoints**

**Authentication Endpoint**

* **POST /login/**
  + **Description**: Obtain a JWT token for authentication.
  + **Request Body**: { "username": "string", "password": "string" }
  + **Response**: { "access\_token": "string", "token\_type": "bearer" }
  + **Permissions**: Open to all.

**Auto Plate Endpoints**

* **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).
* **POST /plates/**
  + **Description**: Create a new plate.
  + **Request Body**: { "plate\_number": "string", "description": "string", "deadline": "datetime" }
  + **Response**: Created plate details.
  + **Permissions**: Admin only.
* **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).
* **PUT /plates/{id}/**
  + **Description**: Update plate details.
  + **Request Body**: Same as POST.
  + **Permissions**: Admin only.
* **DELETE /plates/{id}/**
  + **Description**: Delete a plate.
  + **Permissions**: Admin only.

**Bid Endpoints**

* **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).
* **POST /bids/**
  + **Description**: Place a bid on a plate.
  + **Request Body**: { "amount": decimal, "plate": int }
  + **Response**: Created bid details.
  + **Permissions**: Authenticated users.
* **GET /bids/{id}/**
  + **Description**: Retrieve a specific bid.
  + **Response**: Bid details.
  + **Permissions**: Bid owner only.
* **PUT /bids/{id}/**
  + **Description**: Update a bid (e.g., increase amount).
  + **Request Body**: { "amount": decimal }
  + **Permissions**: Bid owner only, before deadline.
* **DELETE /bids/{id}/**
  + **Description**: Delete a bid.
  + **Permissions**: Bid owner only, before deadline.

**Edge Cases to Handle**

**Auto Plate Management**

* **Duplicate Plate Number**: Return 400 Bad Request with "Plate number already exists".
* **Past Deadline**: Return 400 with "Deadline must be in the future".
* **Non-Admin Access**: Return 403 Forbidden for non-admin attempts to create/update/delete plates.
* **Delete Plate with Bids**: Return 400 with "Cannot delete plate with active bids" if restricted, or allow cascade deletion.

**Bid Management**

* **Bid on Inactive Plate**: Return 400 with "Bidding is closed" if is\_active=False or deadline passed.
* **Lower Bid**: Return 400 with "Bid must exceed current highest bid".
* **Multiple Bids**: Return 400 with "You already have a bid on this plate".
* **Update After Deadline**: Return 403 with "Bidding period has ended".
* **Negative Amount**: Return 400 with "Bid amount must be positive".

**Authentication**

* **No Token**: Return 401 Unauthorized for protected endpoints.
* **Invalid Token**: Return 401 with "Invalid token".
* **Non-Owner Access**: Return 403 Forbidden for attempts to view/update/delete someone else’s bid.

**General**

* **Non-Existent Resource**: Return 404 Not Found for invalid plate or bid IDs.
* **Invalid Data**: Return 400 with validation error for incorrect data types or values.
* **Concurrent Bids**: Ensure database-level consistency (e.g., using transactions) to handle simultaneous bids correctly.

**Implementation Notes**

* **Authentication**: Use JWT tokens managed via dependencies, validated on protected endpoints.
* **Database**: Use SQLAlchemy with async support (databases library) for non-blocking operations.
* **Permissions**: Implement reusable dependency functions (e.g., get\_current\_user, is\_admin, is\_bid\_owner) to enforce access control.
* **Validation**: Leverage Pydantic models to validate incoming data and format outgoing responses.
* **Filtering**: Handle query parameters directly in endpoint logic or use a library like fastapi-filter for advanced filtering.
* **Testing**: Use pytest with httpx for async testing of endpoints, mocking database interactions where needed.