Step 0: Preliminaries

1. Make sure your **server is running**:

node server.js

or

npm run dev

2. Make sure **MongoDB is connected** and accessible via your `.env` config.
3. In **Postman**, create a **new collection** called `Marvel Auction Test` to organize requests.

Step 1: Create Users (Registration)

We need **10 users**:

- 5 Sellers
- 5 Bidders

1.1 POST /api/auth/register

URL: `http://localhost:5000/api/auth/register`
Method: POST
Body (form-data):

Example **Seller**:

Key	Value	Type

username	ironman_seller	text
email	ironman@seller.com	text
password	stark123	text
role	seller	text
bankStatement	[upload a small PDF/image]	file

Example **Bidder**:

Key	Value	Type
username	spiderman_bidder	text
email	spiderman@bidder.com	text
password	webslinger	text
role	bidder	text

Repeat with **different Marvel usernames** until you have 5 sellers + 5 bidders.

Response:

```
{
  "message": "User registered successfully",
  "user": { "id": "...", "username": "...", "email": "...", "role": "seller" }
}
```

Notes:

- Upload fake bank statement files for sellers.
- Copy the **user IDs** from the response; you'll need them for items and bids.

Step 2: Login Users

2.1 POST /api/auth/login

URL: `http://localhost:5000/api/auth/login`
Method: POST
Body (raw JSON):

```
{
  "email": "ironman@seller.com",
  "password": "stark123"
}
```

Response:

```
{
  "token": "JWT_TOKEN_HERE",
  "user": { "id": "...", "username": "ironman_seller", "role": "seller" }
}
```

Save **JWT tokens** for all users; each request that needs auth must have header:

Authorization: Bearer JWT_TOKEN_HERE

Step 3: Create Items (Sellers)

3.1 POST /api/items

URL: `http://localhost:5000/api/items`
Method: POST
Headers: `Authorization: Bearer <seller_token>`
Body (raw JSON):

Example Marvel items for `ironman_seller`:

```
{
  "title": "Iron Man Suit Mark L",
  "description": "Advanced Iron Man armor with repulsor technology",
```

```
  "startingPrice": 5000,
  "category": "armor"
}
```

Repeat for **5 sellers × 2 items each**. Some ideas:

Seller	Items
ironman_seller	Iron Man Suit Mark L, Arc Reactor
captain_seller	Captain America Shield, Vibranium Helmet
thor_seller	Mjolnir, Stormbreaker
hulk_seller	Hulk Gauntlets, Gamma Reactor
blackwidow_seller	Widow's Bite, Stealth Suit

Response:

```
{
  "id": "...",
  "title": "...",
  "sellerId": "...",
  "startingPrice": ...
}
```

Step 4: Create Auctions (Sellers)

4.1 POST /api/auctions

URL: `http://localhost:5000/api/auctions`
Method: POST
Headers: `Authorization: Bearer <seller_token>`
Body (raw JSON):

```
{
  "itemId": "<item_id_from_previous_step>",
  "startTime": "2025-08-22T14:00:00Z",
  "endTime": "2025-08-23T14:00:00Z"
}
```

Notes:

- Set `startTime` in the **near future** or past to test `ongoing`.
- Repeat for all items.

Response:

```
{
  "id": "...",
  "itemId": "...",
  "status": "scheduled",
  "startTime": "...",
  "endTime": "..."
}
```

Step 5: Get Items & Auctions

5.1 GET /api/items

GET http://localhost:5000/api/items
Authorization: Bearer <any_user_token>

- Returns all items, filtered optionally by `category` or `maxPrice`.

5.2 GET /api/auctions

GET http://localhost:5000/api/auctions
Authorization: Bearer <any_user_token>

- Returns all scheduled and ongoing auctions, populated with item info.

5.3 GET /api/items/:id

- Returns single item with auction status and current top price.

Step 6: Place Bids (Bidders)

6.1 POST /api/bids/place

POST http://localhost:5000/api/bids/place
Authorization: Bearer <bidder_token>
Content-Type: application/json

Body:

```
{
  "itemId": "<auction_item_id>",
  "amount": 6000
}
```

Rules:

- Must be **higher than current top price**.
- Triggers proxy bid logic if exists.

Response:

```
{
  "status": "accepted",
  "bidId": "...",
  "amount": 6000,
  "highest": true
}
```

> Repeat for multiple bidders and items, testing **manual vs proxy bids**, edge cases (lower bids, same bids, auction completed).

Step 7: Get Current Highest Bid

7.1 GET /api/bids/highest/:itemId

GET http://localhost:5000/api/bids/highest/<itemId>
Authorization: Bearer <any_bidder_token>

- Returns the highest bid and bidder info.

Step 8: Complete Auction (Manual Step for Testing)

- Update auction status to `completed`:

PATCH http://localhost:5000/api/auctions/<auctionId>
Authorization: Bearer <seller_token>
Content-Type: application/json

Body:

```
{
  "status": "completed",
  "finalPrice": 7000,
  "winnerId": "<bidder_id>"
}
```

- Now the auction is done, winner can pay.

Step 9: Payment (Winner)

9.1 POST /api/payments/initiate

POST http://localhost:5000/api/payments/initiate
Authorization: Bearer <winner_token>
Content-Type: application/json

Body:

```
{
  "auctionId": "<auction_id>"
}
```

- Chapa will return a **payment link / status**.
- Simulate callback to test:

GET http://localhost:5000/api/payments/chapa/callback?status=success&auctionId=<auctionId>

Step 10: Fetch Payment History

10.1 GET /api/payments/history

GET http://localhost:5000/api/payments/history
Authorization: Bearer <winner_token>

- Returns all payments made by the logged-in user.

Step 11: Test Users / Admin Actions

- Fetch all users (admin only):

GET /api/users
Authorization: Bearer <admin_token>

- Update / delete users: `PUT /api/users/:id`, `DELETE /api/users/:id`
- Update / delete items: `PATCH /api/items/:id`, `DELETE /api/items/:id`

Step 12: Test Edge Cases

1. Place bid **lower than current highest** → should fail.
2. Update item after auction started → should fail.
3. Delete auction → should succeed only if no bids or admin.
4. Unauthorized access → `401` or `403`.

✅ Summary

By following these **12 steps**, you will:

- Fully populate DB with **10 users, 10 sellers' items, auctions**.
- Test **all item/auction routes**.
- Test **bidding logic**, including proxy bids.
- Test **payments** and callbacks.
- Validate **user permissions** and **error handling**.

If you want, I can **generate a ready-to-use Postman collection** with all **10 Marvel users, items, auctions, and bid requests pre-filled**, so you just import it and run everything in order.

Do you want me to do that next?