Lab Asset Management REST API Documentation

Base URL

http://localhost:5000

All routes except /auth are protected and require a JWT token in the Authorization header:

Authorization: Bearer <token>

Middlewares

1. **CORS Middleware**
   - Allows requests from frontend: http://localhost:3000
   - Allows methods: GET, POST, PUT, DELETE, OPTIONS
   - Allows headers: Content-Type, Authorization
2. **Rate Limit Middleware**
   - Global limit: 100 requests per 15 minutes per IP
   - Response when limit exceeded:

     ```
     {
       "status": 429,
       "error": "Too many requests from this IP, please try again later."
     }
     ```

3. **Auth Middleware**
   - Protects /assets and /reservations routes
   - Checks for Authorization: Bearer <token> header
   - Response if token is missing or invalid:

     ```
     { "error": "No token provided" }

     { "error": "Token missing" }

     { "error": "Invalid or expired token" }
     ```

4. **Error Handling Middleware**
   - Catches all server errors and sends:

   ```
   {
     "success": false,
     "message": "<error message>"
   }
   ```

1. Authentication API

1.1 Signup

- **URL:** /auth/signup
- **Method:** POST
- **Request Body:**

```
{
  "username": "JohnDoe",
  "email": "john@example.com",
  "password": "password123"
}
```

- **Response (Success – 200 Created):**

```
{
  "message": "User created successfully",
  "user": {
    "id": 1,
    "username": "JohnDoe",
    "email": "john@example.com"
  }
}
```

- **Error Responses:**

```
{ "message": "User already exists" }
```

1.2 Signin

- **URL:** /auth/signin
- **Method:** POST
- **Request Body:**

```
{
  "email": "john@example.com",
  "password": "password123"
}
```

- **Response (Success – 200 OK):**

```
{
  "message": "Login successful",
  "token": "<jwt-token>"
}
```

- **Error Responses:**

```
{"message": "User not found"}
{"message": "Invalid credentials"}
```

- **Token Validity:** 1 hour

2. Asset API (Protected)

2.1 Get All Assets

- **URL:** /assets/
- **Method:** GET
- **Response (Success – 200 OK):**

```
[
   { "id": 2, "name": "VM-42", "ip": "192.168.1.20", "description": "Ubuntu test2 machine" },
   { "id": 11, "name": "VM-1", "ip": "192.168.0.10", "description": "" },
   { "id": 17, "name": "VM-6", "ip": "192.168.1.23", "description": "Windows Machine" },
   { "id": 19, "name": "VM-8", "ip": "192.168.1.26", "description": "Windows Machine" }
]
```

2.2 Add Asset

- **URL:** /assets/add
- **Method:** POST
- **Request Body:**

```
{ "name": "VM-10", "ip": "192.168.1.30", "description": "New test machine" }
```

- **Response (Success – 200 OK):**

```
{ "id": 20, "name": "VM-10", "ip": "192.168.1.30", "description": "New test machine" }
```

- **Error Responses:**

```
{ "error": "Name and IP are required" }
{ "error": "Invalid IP address" }
```

2.3 Update Asset

- **URL:** /assets/update/:id
- **Method:** POST
- **Request Body:**

```
{"id":19,"name":"VM-8","ip":"192.168.1.28","description":"Windows Machine"}
```

- **Response (Success – 200 OK):**

```
{ "id": 20, "name": "VM-10 Updated", "ip": "192.168.1.30", "description": "Updated
description" }
```

2.4 Delete Asset

- **URL:** /assets/delete/:id
- **Method:** POST
- **Response (Success – 200 OK):**

```
{ "message": "Asset deleted" }
```

3. Reservation API (Protected)

3.1 Get All Reservations

- **URL:** /reservations/
- **Method:** GET
- **Response (Success – 200 OK):**

```
[
  {
    "id": 6,
    "user_name": "Mynewuss",
    "start_time": "2025-08-29T00:43",
    "end_time": "2025-08-30T00:43",
    "note": "",
    "asset_id": 11,
    "Asset": { "id": 11, "name": "VM-1", "ip": "192.168.0.10", "description": "" }
  }
]
```

3.2 Add Reservation

- **URL:** /reservations/add
- **Method:** POST
- **Request Body:**

```
{ "asset_id": 11, "user_name": "NewUser", "start_time": "2025-08-31T09:00", "end_time":
"2025-08-31T11:00", "note": "Testing reservation" }
```

- **Response (Success – 200 OK):**

```
{ "id": 7, "asset_id": 11, "user_name": "NewUser", "start_time": "2025-08-31T09:00",
"end_time": "2025-08-31T11:00", "note": "Testing reservation" }
```

- **Error Responses:**

```
{ "error": "All required fields must be filled" }
{ "error": "Start time must be before end time" }
{ "error": "Asset already reserved during this time" }
```

3.3 Update Reservation

- **URL:** /reservations/update/:id
- **Method:** POST
- **Request Body:**

```
{

    "asset_id": 11,

    "user_name": "Mynewussupdate",

    "start_time": "2025-08-30T03:43",

    "end_time": "2025-08-31T00:43",

    "note": "Updated time"

}
```

- **Response (Success – 200 OK):**

```
{
  "id": 7,
  "asset_id": 11,
  "user_name": "NewUser",
  "start_time": "2025-08-31T10:00",
  "end_time": "2025-08-31T12:00",
  "note": "Testing reservation",
  "Asset": { "id": 11, "name": "VM-1", "ip": "192.168.0.10", "description": "" }
}
```

3.4 Delete Reservation

- **URL:** /reservations/delete/:id
- **Method:** POST
- **Response (Success – 200 OK):**

```
{ "message": "Reservation cancelled" }
```

4. Database Models (SQLite)

4.1 User

- Table: users
- Fields: id (PK), username, email, password
- username & email unique

4.2 Asset

- Table: assets

- Fields: id (PK), name, ip, description
- ip unique

4.3 Reservation

- Table: reservations
- Fields: id (PK), user_name, start_time, end_time, note, asset_id (FK)
- Relationships:
    - Reservation.belongsTo(Asset)
    - Asset.hasMany(Reservation)