Express.js Event Ticket Server — Test Description

Goal

Build an **Express.js server** that allows users to register, create events, buy tickets, and view purchase summaries.
All data must be stored using **JSON files** with Node.js file I/O.

Data Storage (JSON Files)

1. Users Database — `users.json`

Stores all registered users. Init to hold just [ ]

Each user includes:

- `username`
- `password`

---

2. Events Database — `events.json`

Stores all created events. Init to hold just [ ]

Each event includes:

- `eventName`
- `ticketsAvailable`
- `createdBy` (value is a username)

---

3. Receipts Database — `receipts.json`

Stores all ticket purchase receipts. Init to hold just [ ]

Each receipt includes:

- `username`
- `eventName`
- `ticketsBought`

Authentication Rule

A user **must be registered** before performing any action in the system.

For each POST/PUT endpoint you must receive username and password in the request and validate them before doing any operations.

API Endpoints

1. Register User

Purpose

Create a new user so they can use the system.

Method + URL

POST `/user/register`

Request Body

```
{
  "username": "string",
  "password": "string"
}
```

Expected Response Structure

```
{
  "message": "User registered successfully"
}
```

General Steps

- Load `users.json`
- Check that the username does not already exist
- Save the new user to the file - username + password. Make sure to comply with the way the user is structured, as described above.
- Return a success message

---

2. Create Event

Purpose

Allow a registered user to create a new event.

Method + URL

POST `/creator/events`

Request Body

```
{
  "eventName": "string",
  "ticketsForSale": number,
  "username": "string",
  “password”:”string”
}
```

Expected Response Structure

```
{
  "message": "Event created successfully"
}
```

General Steps

- Verify the user exists + correct password
- Load `events.json`
- Create a new event object
- Save the event to the file. Make sure to comply with the way the event is structured, as described above. Don’t overwrite - append.
- Return a success message

3. Buy Tickets

Purpose

Allow a user to buy tickets for an event.

Method + URL

POST `/users/tickets/buy`

Request Body

```
{
  "username": "string",
  “password”: “string”,
  "eventName": "string",
  "quantity": number
}
```

Expected Response Structure

```
{  "message": "Tickets purchased successfully"}
```

General Steps

- Verify the user exists + correct password
- Load `events.json`
- Find the event by name (case-insensitive)the receipt is structured, as described above.
    - Save it to `receipts.json`
    - Update remaining tickets in `events.json`
    - Check if enough tickets are available
    - If not enough tickets, return an error
    - If enough tickets:
        - Create a receipt object. Make sure to comply with the way

- Return a success message

4. User Purchase Summary

Purpose

Return a summary of all ticket purchases made by a specific user.

Method + URL

GET `/users/:username/summary`

Path Parameters

- `username` (string)

Expected Response Structure

```
{
  "totalTicketsBought": number,
  "events": ["event name", "event name"],
  "averageTicketsPerEvent": number
}
```

General Steps

- Load `receipts.json`
- Filter receipts by the given username
- Calculate:
  - Total number of tickets bought
  - Unique event names
  - Average tickets per event
- If no receipts exist, return zero values
- Return the summary object

deploy your server to render and add the url to your readme!

Bonus Features (Optional)

1. User Roles

- One role can create events - “user”
- Another role can only buy tickets - “admin”
- When a user is created - you need to add the type. “user” is the default.

2. Ticket Ownership Transfer

- Allow users to transfer purchased tickets to another user

3. Ticket Refunds

- Allow users to return tickets after purchase
- Increase available tickets accordingly
- Track the change in receipts or a separate log

Grading Table (100 points)

Setup & File-Based Storage	Express runs + uses `users.json`, `events.json`, `receipts.json` with real file I/O	10
Register Endpoint	Works as described, saves user, prevents duplicate username	20
Create Event Endpoint	Requires registered user, saves event with required fields	20
Buy Tickets Endpoint	Case-insensitive event search, validates availability, updates files correctly, creates receipt	30
User Summary Endpoint	Correct totals, unique events list, correct average, handles “no receipts”	20
Total		**100**

Optional (Bonus) Policy

If you include bonuses in grading:

- Bonus points **must not compensate for missing core requirements** unless you explicitly allow it.
- Bonus should be capped (example: up to +10) and not exceed 100 total.