This assignment requires you to extend the existing application by creating a new protected API route and writing a Supertest case to verify its security.

Question 1: Implement the New Route (Application Code)

You need to create an API endpoint accessible only to users with the **`user`** role (i.e., both standard users and admins should be able to access it, as they are both users of the system, but guests cannot).

Task A: Create the Route in `routes/authRoutes.js`

Add a new `GET` route to your **`routes/authRoutes.js`** file called `/protected/user-status`.

This route must perform the following checks:

1. **Authentication:** It must first pass the `verifyToken` middleware.
2. **Authorization:** It must check if the user's role is either **`admin`** OR **`user`**. If the role is missing or anything else, it should return a **403 Forbidden** error.
3. **Success:** If authorized, return a JSON response with a **200 OK** status and a message confirming access.

Task B: Define the Authorization Logic

the code snippet for this new route:

JavaScript

```
None


// Add this route to routes/authRoutes.js

router.get('/protected/user-status', verifyToken, (req, res) => {
   // 1. Check for 'admin' OR 'user' role

   // 2. If Authorization fails, send 403 Forbidden

   // 3. If Authorization succeeds, send 200 OK
});
```

Question 2: Create the Supertest Case (Testing)

Now, you must write the corresponding Supertest case in **test/auth.test.js** to ensure your new route works exactly as intended.

Task: Write the Test Suite Block

Write a new describe block containing **three** separate it tests that cover all possible scenarios for the /protected/user-status endpoint.

1. **Test 1 (Success):** Verify access using the **Standard User**'s token.
2. **Test 2 (Success):** Verify access using the **Admin User**'s token.
3. **Test 3 (Failure):** Verify denial of access when **no token** is provided.

JavaScript

```
None


// Add this new describe block to test/auth.test.js

describe('GET /api/auth/protected/user-status', () => {

  it('should allow access with a valid standard user token (200 OK)', async () => {
    // ...
  });

  it('should allow access with a valid admin token (200 OK)', async () => {
    // ...
  });

  it('should deny access with 401 when the token is missing', async () => {
    // ...
  });

});
```

Single-Agent ReAct + MRKL Bike-Share Pass Optimizer (Public Data)

Build a single agent that recommends whether a rider should buy a monthly bike-share membership or pay per ride/minute, using a ReAct loop for control and MRKL tools for capability. The agent must read a period of public trip data and the system's official pricing/policy page , then output a cited recommendation with a transparent **Thought +Action + Observation + Final Answer** trace.

Choose ONE city (both data & policy are public)

- **New York City — Citi Bike**
    - Trip data (CSV, monthly): Citi Bike System Data. (Citi Bike NYC)
    - Pricing/policy: Citi Bike Pricing page. (Citi Bike NYC)

- **Chicago — Divvy**
    - Trip data: Divvy System Data / Divvy Trips Dashboard. (divvybikes.com)
    - Pricing/policy: Divvy Pricing page (note that prices change; cite page + date). (divvybikes.com)

- **San Francisco Bay Area — Bay Wheels**
    - Trip data: Bay Wheels System Data. (Lyft)
    - Pricing/policy: Bay Wheels pricing page on the same site (follow "Pricing" link from system page).

- **Washington, DC — Capital Bikeshare**
    - Trip data: Capital Bikeshare System Data. (capitalbikeshare.com)
    - Pricing/policy: Capital Bikeshare Pricing page (recent changes possible—include date). (capitalbikeshare.com)

Deliverables

- A small web UI that:
  - Accepts: a **month** CSV from the chosen city (user uploads one file) and a **URL** to the official pricing page.
  - Shows a **timeline** of agent thinking process or delegation of tasks to tools.
  - Shows a **cost comparison**: pay-per-ride/minute vs monthly membership, with any **overage/minute surcharges** and **included ride durations** applied per policy.
  - Includes **citations** to the pricing/policy dataset sections you used.

- A **single agent** using a **ReAct loop** + **MRKL tools** .
- Per-step logs: tool, args hash, latency, success/failure, stop reason.

Tools (MRKL) implement as pure functions with JSON schemas

1. csv_sql
   input: { "sql": string }
   output: { "rows": Array<object>, "row_count": number, "source": "uploaded.csv" }
   Behavior: run read-only SQL over the uploaded trips.

2. policy_retriever
   input: { "url": string, "query": string, "k"?: number }
   output: { "passages": [{ "text": string, "source": string, "score": number }] }
   Behavior: fetch the pricing page URL, extract text, and return **short, quotable** snippets (membership price, included ride minutes, per-minute ebike surcharges, unlock fees, overage rules).

3. calculator
   input: { "expression": string, "units"?: string } (whitelist ^[0-9+\\-*/().\\s]+$)
   output: { "value": number, "units"?: string }
   Behavior: safe arithmetic (no eval).

Common pattern: every tool must returns { "success": bool, "data"?: any, "error"?: string, "source"?: string, "ts"?: string }.

What the agent must produce

1. **Decision**: “Buy Monthly Membership” or “Pay Per Ride/Minute”.

2. **Justification** (3–6 sentences) with **citations** to pricing policy snippets (e.g., included minutes, ebike surcharge, unlock fees, membership price).

3. **Cost breakdown for the uploaded month**:

    - Total cost under **pay-per-use** (apply policy: included minutes per ride, minute-based surcharges, unlock fees; treat missing fields conservatively).

    - Monthly membership cost and any residual per-minute charges or unlock fees that still apply to members.

    - **Break-even** rides/time vs actual.

    - A small **table** by week: ride count, average duration, ebike usage share (if available in the city’s schema), spend.

4. **Assumptions & caveats** (e.g., if ebike/classic indicators are missing, assume classic; or infer from columns the dataset provides).

UI Requirements

- One file upload (CSV), one text field (pricing URL), and a **Run** button.

- A **timeline** panel showing Thought/Action/Observation/Final Answer.

- A **results** panel with the cost table, final decision, and citations (include the pricing page URL + capture date/time).

- A metrics strip: **steps**, **total time**, **stop reason**.

Acceptance Tests

Run your agent on **two different months or two station pairs** and include:

1. A case where **membership wins** (many, longer rides and/or frequent ebike use).
2. A case where **pay-per-use wins** (few/short rides).

For each:

- One paragraph explaining the decision with numbers.
- The **Stop Reason** and total steps.
- Start with NYC (Citi Bike) or Chicago (Divvy) if you want the largest, well-documented datasets. (Citi Bike NYC) (use a small subset)
- Policy pages can change—**save the text** you cite and include the date. (Citi Bike NYC)
- If a month's CSV is very large, filter early (station pair + commute windows) before computing costs. (my recommendation is to use a subset of the dataset, minimum 500 rows and keep your queries related to those rows)

References

- Citi Bike System Data / Pricing. (Citi Bike NYC)
- Divvy System Data / Pricing. (divvybikes.com)
- Bay Wheels System Data. (Lyft)
- Capital Bikeshare System Data / Pricing. (capitalbikeshare.com)