Tasks

Task 1: Basic API with GET and POST Routes

- **Description:** Create a simple Express.js server that handles GET and POST requests.
- **Functionality:**
  - **GET `/hello`:** Respond with a simple "Hello, world!" message.
  - **GET `/greet/:name`:** Respond with "Hello, \[name]!" where `name` is taken from the URL parameter.
  - **POST `/echo`:** Receive data in the request body (JSON format) and respond with the same data as a JSON response.
- **Requirements:**
  - Use `express` to set up the server.
  - Use `app.listen()` to start the server.
  - Use `app.get()` for the GET routes.
  - Use `app.post()` for the POST route.
  - Use `req.params` to retrieve the name from the `/greet/:name` route.
  - Use `app.use(express.json())` to parse JSON request bodies.
  - Use `req.body` to access data from the POST request.
  - Test the routes with Postman:
    - Send a GET request to `/hello` and verify the response.
    - Send a GET request to `/greet/John` and verify the response.
    - Send a POST request to `/echo` with a JSON body and verify the response.
- **Hints:** Make sure to install express using `npm install express` and then create a `.js` file for all your code.

Task 2: API with Query Parameters and Wildcard Route

- **Description:** Create an Express.js server that utilizes query parameters and a wildcard route.
- **Functionality:**
    - **GET `/products`:**
        - Accept query parameters `category` and `price`.
        - If both parameters are present, respond with "Products in \[category] with price less than \[price]".
        - If only `category` is present, respond with "Products in \[category]".
        - If only `price` is present, respond with "Products with price less than \[price]".
        - If neither is present, respond with "All products."
    - **Wildcard Route:** For any route that is not defined, respond with "404 - Route not found".
- **Requirements:**
    - Set up the server with `express` and `app.listen()`.
    - Use `app.get()` for the `/products` route.
    - Use `req.query` to access query parameters.
    - Use a wildcard route with `app.use('*', ...)` to handle undefined routes.
    - Use `res.status(404)` to send the 404 status code in the wildcard route.
    - Test with Postman:
        - Test `/products` with different query parameters (category, price, both, none).
        - Test with a non-existing route to see the 404 message.

- **Hints:** Make sure to understand the order of routes in express and set the wild card route at the very end to avoid capturing all the routes.

Task 3: User Management API (POST and GET with req.body)

- **Description:** Build an Express.js server for simple user management.
- **Functionality:**
  - **POST `/users`:**
    - Accept user data (name, email, age) in the request body as JSON.
    - Respond with "User created successfully" and the user data.
  - **GET `/users`:**
    - Respond with a dummy array of users.
- **Requirements:**
  - Set up the server with `express` and `app.listen()`.
  - Use `app.post()` for the POST `/users` route.
  - Use `app.get()` for the GET `/users` route.
  - Use `app.use(express.json())` to parse JSON request bodies.
  - Use `req.body` to access the user data.
  - Use Postman to:
    - Send a POST request with user data in JSON format to `/users`.
    - Check for a response with "User created successfully" along with the user data.
    - Send a GET request to `/users` and verify that you see the dummy array of users.
- **Hints:** Make sure to check what type of response you are sending through the use of `res.send()` and also make sure to send a response for every

request.