Task: User Strategy Optimization Query Endpoint

Context

You are building a feature for **Journalyst**, a trading journal and strategy refinement tool. Traders log their trades into MongoDB with details like strategy, risk level, and performance notes. You need to create an **API endpoint in Express/Node.js** that analyzes a user's trades and generates actionable **optimization suggestions**.

Requirements

1. **MongoDB Schema**

Assume trades are stored in a collection with fields like:

```
{
  userId: ObjectId,
  strategyId: String,
  tradeDate: Date,
  riskLevel: String, // "low", "medium", "high"
  outcome: Number,   // profit/loss in dollars
  win: Boolean,      // true if profitable
  performanceNotes: String
}
```

2. Endpoint Specification

- **Route:** `GET /api/strategies/optimize/:userId`
- **Functionality:**
  - Retrieve all trades belonging to the specified user.
  - Determine each strategy's win rate for the last 30 days.
  - Identify strategies with win rates below 50% and flag them as underperforming.

    - Analyze trades grouped by risk level to calculate their average outcomes.
    - Assess the overall relationship between risk level and trade outcomes by computing a simple correlation measure.

3. Suggestions Logic

The endpoint should return JSON with:

- A list of **underperforming strategies**.
- Suggestions, such as:
    - *"Refine entry criteria for strategy X"* if underperforming.
    - *"Increase position size for low-risk trades"* if average outcome for low-risk is strongly positive.
    - *"Reduce exposure to high-risk trades"* if correlation between higher risk and negative outcome is detected.

4. Constraints

- No external statistical libraries (implement correlation manually).
- Use **MongoDB aggregation pipeline** for grouping and filtering.
- Keep the code **readable, modular, and testable**.

Deliverables

1. Express.js route handler code (`strategies.js`).
2. Example response JSON for a test user.
3. Brief notes on how you tested edge cases (e.g., no trades, all wins, all losses).

Time Allocation (25 minutes)

- **5 mins:** Design the aggregation and logic.
- **15 mins:** Implement code (focus on complex `$group` and `$project`).
- **5 mins:** Edge-case testing.

Note: The solution should be implemented in **TypeScript** (not plain JavaScript). You may leverage **LLM models** to assist with building the solution.