C) Bonus Boosters (500 Points)

These optional, extra-tough challenges are for those ready to push the limits—and earn serious points.

- **Q1 (250 points):** Design a RESTful API for order placement that handles high concurrency without errors, detailing endpoints, request/response formats, and concurrency controls.
- **Q2 (250 points):** Enhance your API to support extreme scalability, processing numerous simultaneous orders without data loss, and explain your strategies.

Refer to Software Demostration to see the RESTful api working

Q1)

Part 1: List of API Methods in the Backend

Below are the API methods (endpoints) defined in your index.js file, extracted by identifying all app.<method> routes:

1. **POST /register**
   - Handles user registration by sending an OTP to the provided email.
2. **POST /verify-otp**
   - Verifies the OTP and completes user registration by saving to the database.
3. **POST /login**
   - Authenticates users and creates a session.
4. **GET /dashboard**
   - Serves the dashboard page for authenticated users (admin or regular).
5. **GET /logout**
   - Destroys the user session and logs them out.
6. **GET /orders**
   - Serves the orders view (dashboard.html) for admins.
7. **GET /api/orders**
   - Retrieves all orders (limited to the last 24 hours, max 20) with associated items and table info.
8. **POST /api/mark-served**
   - Updates an order’s status to 'served' and sets the completion time.
9. **POST /add-item**
   - Adds a new item to the Item table and updates items.txt.
10. **POST /add-restaurant-table**
    - Adds a new table to the Restaurant_Table table.

11. **POST /add-order-package**
    - Adds a new order package to the OrderPackage table.
12. **POST /add-cart-item**
    - Adds an item to the Cart table for a specific order package.
13. **POST /add-payment**
    - Adds a payment record to the Payment table.
14. **GET /api/stats**
    - Retrieves statistics like average fulfillment time and total sales for the last 24 hours.
15. **GET /api/last-hour-orders**
    - Fetches orders from the last hour, including items and table info.
16. **GET /api/paymentreceipt/:package_id**
    - Retrieves a payment receipt for a specific order, including items and grand total.
17. **GET /api/reports/advanced-sales**
    - Provides an advanced sales report with top-selling items, peak order times, table utilization, and items for discount.

Q2)

Concurrent Access:

When multiple clients call /add-order-package simultaneously, PostgreSQL ensures that INSERT operations on OrderPackage are serialized, preventing duplicate package_id conflicts (assuming a unique constraint exists)

Concurrent Query Execution:

- The /api/reports/advanced-sales endpoint executes multiple queries concurrently using Promise.all, reducing response time:

Connection Pooling:

- The pg.Pool configuration ensures that database connections are reused efficiently, supporting a high number of simultaneous requests without exhausting resources.

Error Recovery:

```
catch (err) {
  console.error('Error inserting order package:', err.stack);
  res.status(500).send('Error inserting order package');
}
```

Database-Level Controls:

- PostgreSQL's ACID compliance ensures that simultaneous INSERT operations don't result in data corruption.

Integration with Hardware:

- The Arduino sketch (from your previous context) uses these endpoints (/add-order-package, /add-cart-item) to place orders via HTTP POST requests, ensuring seamless integration between the hardware interface and the backend.