

Section A: Quick Fixes

Q1: Essential features for customer satisfaction and efficient order processing

Instant Order Relay

- Orders placed via tablet devices are transmitted to the kitchen system without delay
- Utilizes WebSockets for real-time order push notification
- Eliminates delay between customer submission and kitchen receipt

Real-Time Status Feedback

- Order status updates displayed on customer devices:
 - "Order Confirmed"
 - "Cooking"
 - "Ready to Serve"
- Reduces customer anxiety about order status
- Minimizes unnecessary calls for wait staff

Order Customization Options

- Intuitive interface for customers to modify dishes (e.g., "no onions," "extra cheese")
- Comprehensive customization capabilities directly from the table device
- Reduces need for service staff intervention, supporting the self-service concept

Q2: Design Principles for an Intuitive Interface

Contextual State-Driven UI

- OLED display presents distinct visual states throughout the ordering process:

- Menu Browsing
- Item Selection
- Quantity Confirmation
- Order Pending
- Order Complete
- Intuitive error correction allows users to easily decrease quantity or remove items before checkout
- Reduces stress for users who make mistakes during the ordering process

Button Debounce and Double Confirmation

- Software debouncing prevents accidental multiple presses
- Critical actions (such as final order submission) require deliberate interaction:
 - Long press or hold to confirm
- Particularly beneficial for children and elderly users

Q3: Security Vulnerabilities and Solutions

Device Spoofing

- **Vulnerability:** Attackers could use laptops to impersonate smart pads and submit fraudulent orders
- **Solution:** Unique ESP32 certificates with Mutual TLS Authentication to verify legitimate device identity
- Each device possesses cryptographically secure credentials that cannot be easily replicated

Man-in-the-Middle Attack

- **Vulnerability:** Wi-Fi packet interception allowing malicious modification of order content
- **Solution:** End-to-end AES encryption at the device level
- Data is encrypted before leaving the ESP32 and decrypted only in server memory, protecting the entire transmission path

Order Denial Attack

- **Vulnerability:** Flooding the cloud server with garbage data to cause denial of service
- **Solution:** API Gateway Rate Limiting (maximum 5 requests/second per device ID)
- Suspicious activity triggers temporary device bans and administrator alerts
- Prevents resource exhaustion while maintaining service for legitimate customers

Q4: System Responsiveness During Peak Hours

Edge Node Local Processing

- Raspberry Pi cluster installed on-premises functions as a local edge computing node
- Smart devices communicate with the Edge Node over local Wi-Fi
- Orders are aggregated and batched before cloud transmission
- Dramatically reduces internet bandwidth requirements and dependency

Scalable Asynchronous Cloud Architecture

- Backend built on non-blocking, asynchronous frameworks (Node.js, Go)
- Automatic scaling of server instances based on load
- Queue-driven microservices process orders efficiently without blocking
- Ensures responsiveness even during peak hours

Q5: Integrate existing inventory system with our new ordering system

Middleware API Gateway for Inventory Synchronization

- Dedicated middleware microservice connects the smart ordering system to the legacy inventory
- Smart pads never communicate directly with inventory system
- Gateway handles inventory updates (e.g., ingredient deduction) upon order confirmation
- Allows both systems to operate in parallel without interference
- Preserves integrity of the original inventory system

Section B: Tech Tricks

Q1: Database schema to track users, orders, menu items, tables, payments

In the [q1.sql](#) file, we define the database schema for Bistro 92. We are using **MySQL** for our database. The schema includes the following tables:

- **tables**: Stores information about tables in the restaurant.
- **users**: Stores user information, including their role (admin or customer).
- **menu_items**: Stores information about menu items, including their price and availability.

- **orders**: Stores information about orders placed by users, including the table number and order status.
- **order_items**: Stores information about the items included in each order.
- **payments**: Stores information about payments made for orders.

Q2: SQL query to retrieve data

In the [q2.sql](#) file, we provide the SQL query to retrieve the following information:

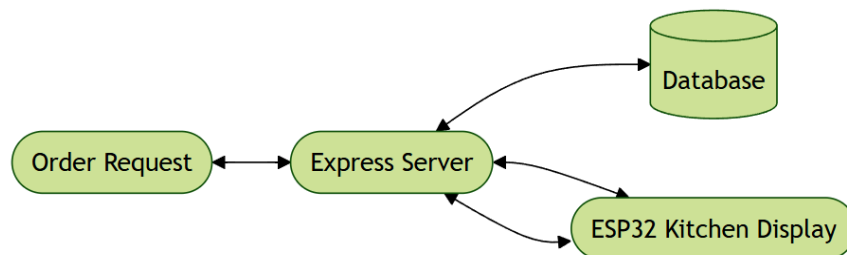
- The table number associated with each order.
- The order ID and the time the order was placed.
- Only orders placed within the last one hour are considered.
- A JSON array of items ordered, where each item includes:
 - The name of the menu item.
 - The quantity ordered.

Q3: Notify kitchen staff in real-time for new orders

In the [q3](#) directory, we implemented a real-time notification system using **webSockets** for notifying kitchen staff about new orders. Our tech stack includes:

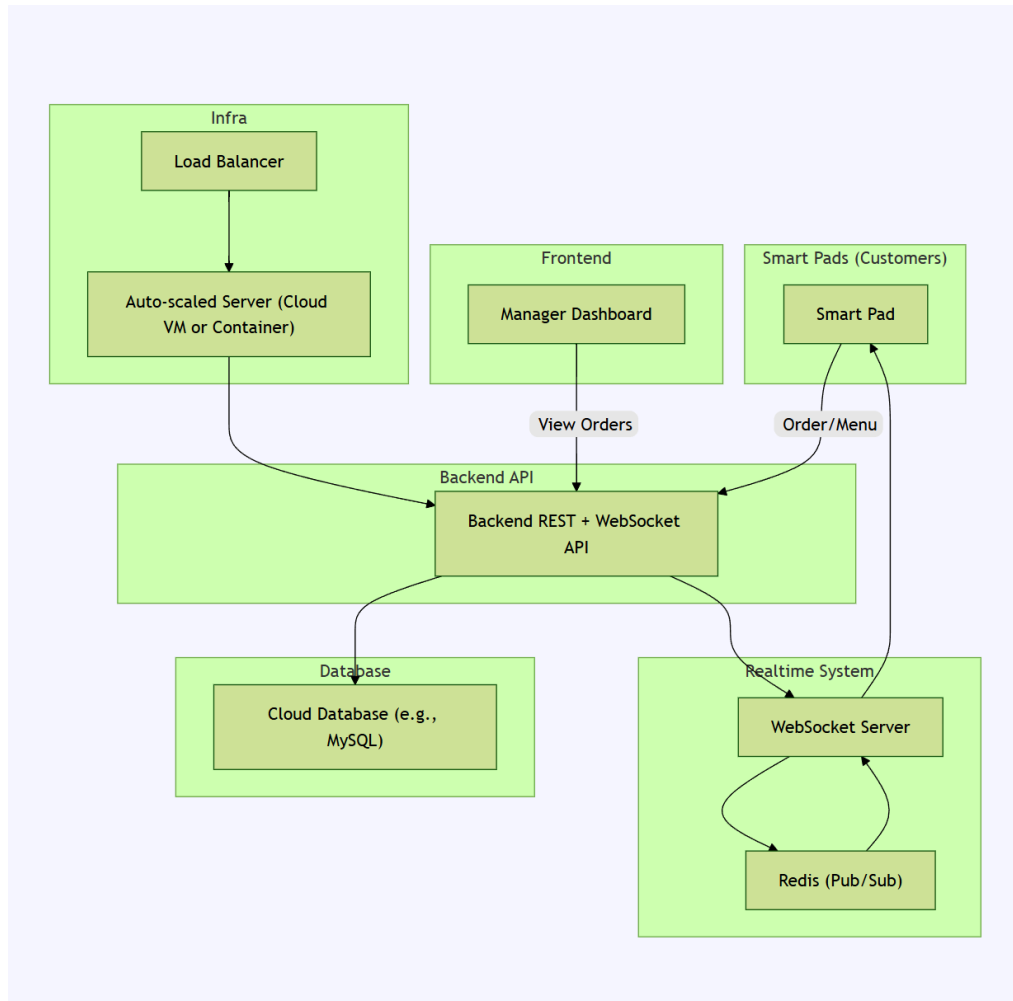
- **Express.js** as the server framework.
- **TypeScript** for type safety.
- **ws** as the node js websocket library.
- **MySQL** for the database.

A simple diagram of the architecture is provided in the [q3_diagram](#) file:



Q4: Cloud-based system architecture

The cloud-based system architecture diagram is provided in the [q4_cloud](#) file:



System Overview

- **Smart Pads (used by customers)** and the **Manager Dashboard** both will communicate with the **Backend**.
- The **Backend API** handles orders, menu, viewing orders and updates etc. requests to the Database.
- For real-time updates (like live order status), the API talks to a **WebSocket Server** running in the same port as the backend.
- The **WebSocket Server** uses **Redis Pub/Sub** to quickly broadcast messages to all connected devices.

- **Smart Pads** receive real-time messages through the WebSocket connection.
- All server traffic is managed by a **Load Balancer** and runs on auto-scaled cloud servers.
- **Backend API** will be running inside a Dockerized container.

Q5: Real-time admin/manager dashboard

We have designed a real-time admin/manager dashboard using [Next.js](#). The source code can be found in the [q5](#) directory. The prototype dashboard has been hosted on [Vercel](#), which can be accessed at [Bistro 92 Dashboard](#).

Section C: Bonus Boosters

Q2: Enhancing the API for extreme scalability

Reliable Communication Protocols

Instead of basic HTTP requests (which can fail), we can use MQTT (Message Queue Telemetry Transport). MQTT is lightweight, fast, reliable — perfect for ESP32.

If the server is handling many orders, MQTT brokers will queue and deliver messages properly without overwhelming ESP32s.

Authorization Tokens hardcoded in ESP32s

Integrating Public Key Private key (RSA) Encryption to send order requests. The public key is sent over request headers. The server will hold the private key. This will prevent unauthorized users from sending false or fake orders.

Database Transaction Management

ACID Transactions can help creating and updating by preventing data loss such as double booking and miscalculations.

Asynchronous & Non-blocking Code

Node js is non blocking and event driven, and can handle many requests at once.

Queue System for Orders

RabbitMQ, Kafka or Redis Pub/Sub can queue orders when traffic is too high. This can help handling burst load smoothly.

Scaling: Horizontal over Vertical

Load balancers such as [Nginx](#) can be used for scalability.

Caching for Read-Heavy Data

Data can be cached in device, so that frequent queries are not needed. The menu doesn't change very frequently for a restaurant.

Implement rate limiting

Using `express-rate-limit` to protect APIs.

Over-the-Air (OTA) Updates

A feature to update firmware remotely without pulling down devices.

Replace Esp32 with Raspberry Pi 4/5

- Esp32 can process around 30–50 messages per second (optimized code with C++, around 100 bytes each) which may not be enough for large scale.
- Full Linux, multi-core CPU, GBs of RAM, can run Node.js, Python, etc.
- This will enable complex UI/UX (touch, animations, fancy order sorting)
- Multiple parallel MQTT/WebSocket subscriptions

It is highly unlikely for a Restaurant management system to reach a heavy load of 10k – 100k users/sec. But if it ever happens it can still be prevented following these strategies.

Section D: Big Idea

BistroAI: AI-Based Dynamic Wait Time and Table Prediction

BistroAI is an intelligent system that leverages artificial intelligence to solve critical operational challenges in restaurant management, enhancing customer experience while optimizing business efficiency.

Current Challenges in Bistro92

1. **Order Wait-Time Anxiety**
 - Customers lack clarity on when their food will arrive
 - Results in frustration, negative reviews, and decreased loyalty
2. **Ingredient Waste and Stock Mismanagement**
 - Manual inventory ordering based on guesswork rather than data
 - Leads to food spoilage, stockouts, and revenue loss
3. **Slow Table Turnover**
 - No system to predict table availability
 - Results in wasted seating capacity and longer wait times
4. **Limited Operational Visibility**
 - No monitoring of staff or device efficiency
 - Operational bottlenecks remain undetected

Proposed Solutions

1. AI-Driven Predictive Wait Times

- **Solution:** Real-time wait time predictions with smart alerts
- **Benefit:** Reduces customer anxiety and builds trust
- **Technical Implementation:**
 - **Data Inputs:** Menu items, kitchen load, cooking times, staff availability, time patterns

- **Model:** Regression models (XGBoost → LSTM/RNN) XGBoost can predict using fixed features like "orders last month", "orders two months ago", etc. LSTM/RNN looks at the whole sequence of past sales and learns patterns over time more automatically.
- **Workflow:** Order placed → AI prediction → Customer countdown timer → Automated compensation for delays
- **Impact:** Improved customer satisfaction and patience during peak hours

2. Inventory Forecasting

- **Solution:** Real-time inventory predictions and management
- **Benefit:** Reduces overstock and stockouts, improves margins
- **Technical Implementation:**
 - **Data Inputs:** Live orders, historical patterns, seasonal data, consumption rates
 - **Model:** Prophet → DeepAR forecasting
 - **Workflow:** 6-24 hour consumption forecasts → Restocking recommendations → Supplier API integration
 - **Impact:** 30-50% reduction in food waste

3. Table Turnover Prediction

- **Solution:** AI-powered table availability forecasting
- **Benefit:** Optimized seating and improved guest flow
- **Technical Implementation:**
 - **Data Inputs:** Seating duration, order status, eating patterns, service times
 - **Model:** Survival Analysis or LSTM forecasting
 - **Workflow:** Table completion prediction → Front-desk availability dashboard
 - **Impact:** Reduced wait times and improved throughput


4. Service Efficiency Monitoring

- **Solution:** AI models to detect service anomalies
- **Benefit:** Proactive identification of operational bottlenecks
- **Technical Implementation:**
 - **Data Inputs:** Device logs, staff service times, order fulfillment metrics
 - **Model:** Anomaly Detection (Isolation Forests, SVMs → Graph Neural Networks)
 - **Workflow:** Real-time performance monitoring → Automated alerts for issues
 - **Impact:** Preventative service failure management

Technology Stack

Component	Technologies
Data Ingestion	Kafka
Database	MySQL/TimeScale/PostgresQL, Redis
AI Stack	TensorFlow / PyTorch, XGBoost
Smart Pad Frontend	Flutter/React
Dashboard	Next.js + Node.js/GO
Hardware	Raspberry Pi 5/Jetson Nano Xavier

Links

- Simulation Video Link -  bristo-92-simulation.mp4
- Github Repository Link - [Buns on Fire](#) [Mierobolution Github](#)
- Dashboard prototype - [Dashboard Prototype](#)