Queue Cure '26

Thought Process Sheet

Wooble Hackathon Submission

GitHub: github.com/liquid-server-2/queue-cure-26

---

1. The Problem

76% of India's 1.5 million clinics still run on paper token slips. Patients receive a numbered chit and wait 2-3 hours with zero visibility into when they will actually be called. Doctors have no dashboard. Receptionists manage everything from memory.

This creates three core pain points:

- Patients cannot leave and come back — they risk missing their turn
- Receptionists are overwhelmed managing the queue from memory
- There is no data on average wait times or queue length at any moment

---

2. The Solution

Queue Cure is a two-screen real-time queue management system. The receptionist adds patients and calls the next token. Patients view a live waiting room screen that shows exactly who is being seen, how many people are ahead, and how long the wait will be.

The key design decision was to use WebSockets (Socket.io) instead of polling. This means both screens update the moment 'Call Next' is clicked — no refresh, no delay, no stale data.

---

3. System Architecture

Tech Stack

- Backend: Node.js + Express + Socket.io
- Frontend: React + React Router + Socket.io Client
- State management: Single shared React Context with one socket connection

How the two screens stay in sync

There is one server with one source of truth — the queue array in memory. Every action (add patient, call next, remove patient, change consult time) emits an event to the server. The server updates its state and immediately broadcasts a queue_update event to every connected client. Both the receptionist and the waiting room screen listen for this event and re-render.

Socket Event Flow

Client emits	Payload	Server response

add_patient	{ name }	Adds to queue, broadcasts queue_update to all clients
call_next	(none)	Shifts first item, updates current token, broadcasts queue_update
remove_patient	{ token }	Filters queue, broadcasts queue_update to all clients
set_consult_time	{ minutes }	Updates consult time, recomputes all wait estimates, broadcasts

---

4. Wait Time Calculation

The estimated wait time is computed from real data on every state change. It is never hardcoded.

Estimated wait = position in queue × average consultation time

For example: if a patient is 3rd in queue and the average consultation time is 8 minutes, their estimated wait is 24 minutes. This recalculates every time someone is added, removed, or called — so the number is always accurate.

The receptionist can adjust the average consultation time at any time. When they do, all wait estimates across both screens update instantly for all connected clients.

---

5. Concurrency & Edge Cases

Double-click race condition on Call Next

If the receptionist clicks 'Call Next' twice very quickly, two events could arrive at the server before the first one finishes processing. This would skip a patient. To prevent this, the server uses a boolean lock called 'calling'. The first event sets it to true, processes the call, then releases the lock after 500ms. Any event that arrives while the lock is active is rejected with an error message.

New client joining mid-session

When a new browser tab connects, it needs the current state immediately — it cannot wait for the next event. On every 'connect' socket event, the server emits the full current state directly to that socket. This means the waiting room screen is always accurate even if someone opens it mid-session.

Empty queue

Calling 'Call Next' on an empty queue returns an error_msg event to the sender. The frontend also disables the button when the queue is empty, so this is a two-layer defence.

Patient leaving before being called

The receptionist can remove any patient from the queue using the Remove button. This emits a remove_patient event. The server filters the queue and broadcasts the updated state. All position numbers and wait estimates recompute automatically.

Input validation

All inputs are validated on the server side, not just the frontend. Patient names must be non-empty strings. Consultation time must be an integer between 1 and 60. The frontend is a convenience layer — the server is the source of truth.

---

6. What I Would Improve Next

- Persist queue to a database (MongoDB) so it survives server restarts
- Add patient check-in via QR code so patients can see their own position on their phone
- Track actual consultation times and use a rolling average instead of a fixed estimate
- Add a doctor screen separate from the receptionist with call history
- SMS notification when a patient is 2 tokens away
