WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Garment360: An AI-Powered Garment Management System with Production Intelligence, What-If Simulation and Tamil Voice-Based Reporting

Mrs. S. Nandhini, Swetha A, Swedha S, Tamilarasi B, Tarun Pranav RS

Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamilnadu, India

DOI : https://doi.org/10.55248/gengpi.07.1026.35003

A B S T R A C T

Apparel manufacturers in South Asia still track cutting, stitching, quality-check and packing progress on paper bundle cards and spreadsheets, so bottlenecks and late deliveries are noticed only after deadlines are threatened. This paper presents Garment360, a role-based, web-native garment management system that connects factory managers and floor workers through one PostgreSQL-backed platform. The system combines (i) deterministic AI production-intelligence algorithms for delivery-delay risk prediction, inter-stage bottleneck detection, worker recommendation and order-urgency ranking; (ii) a profit-aware What-If simulator that models added workers and overtime before a plan is applied atomically; (iii) a Tamil voice-first reporting pipeline that converts spoken Tamil production phrases into verified database records; and (iv) a live AI Manager Assistant that answers natural-language questions from current order data. The backend uses Node.js, Express and PostgreSQL with JWT authentication, bcrypt hashing and fully parameterized SQL, while the vanilla HTML, CSS and JavaScript frontend targets low-specification shop-floor tablets. An automated suite of 88 assertions covering authentication, order lifecycle, intelligence algorithms, simulation and Tamil parsing passes completely. The work shows that explainable heuristics and native-language voice input can give small and medium garment units practical decision support without costly ERP infrastructure.

Keywords: Garment manufacturing; Production intelligence; Bottleneck detection; What-if simulation; Tamil speech recognition; Piece-rate payroll; PostgreSQL

1. Introduction

The apparel industry is labour-intensive and deadline-driven. Every order moves through the sequential stages of cutting, stitching, quality check (QC) and packing, and a queue or delay in one stage propagates to all downstream stages (Hopp & Spearman, 2008). In many small and medium garment units in South Asia, progress is still written on paper bundle cards or in disconnected spreadsheets, so managers learn of a problem only when a shipping date is already at risk.

Two further barriers make conventional software hard to adopt. First, many machine operators are native Tamil speakers with limited English digital literacy, whereas standard ERP screens demand manual English numeric entry, which causes transcription errors and late updates. Second, planning is reactive: supervisors decide on overtime or worker reassignment by intuition, and managers cannot test how an extra worker or an overtime shift would change delivery time and profit before committing money.

This paper presents **Garment360**, an AI-powered garment management system designed around these constraints. The main contributions are:

A role-isolated web platform with separate manager and worker portals backed by a relational PostgreSQL schema and atomic transactions.

Explainable, deterministic production-intelligence algorithms for delay-risk prediction, bottleneck detection, worker recommendation and order-urgency scoring.

A profit-aware What-If simulator that projects velocity, labour and overtime cost and gross margin, and applies an approved plan atomically.

A Tamil voice-first pipeline that parses spoken Tamil number words and defect terms into production records and plays Tamil instructions back to workers.

A live AI Manager Assistant grounded in current database records, with a deterministic fallback when no language-model key is configured.

The rest of the paper is organised as follows. Section 2 reviews related work, Section 3 states the problem, Section 4 describes the architecture, Section 5 gives the intelligence algorithms, Section 6 describes the voice pipeline, Section 7 summarises the implementation, Section 8 reports verification results and Section 9 concludes.

2. Related work

Production scheduling theory (Pinedo, 2016) and the Theory of Constraints (Goldratt & Cox, 1984) establish that the throughput of a serial line is governed by its slowest stage. Factory Physics (Hopp & Spearman, 2008) links work-in-process queues and flow balance to cycle time. Garment360 turns these ideas into two simple, explainable indicators, namely queue accumulation and flow-balance ratio between consecutive stages, instead of a full optimisation model that would need more historical data than small units usually have.

Commercial enterprise resource planning (ERP) and manufacturing execution systems offer broad functionality, but they are costly, English-centric and heavy to configure. Garment360 instead targets lightweight deployment: a vanilla-JavaScript frontend, a single Node.js process and an in-memory database fallback for offline evaluation.

Voice interfaces are now practical in the browser because the W3C Web Speech API exposes speech recognition and speech synthesis (W3C Web Speech API Community Group, n.d.). Availability of Tamil voices, however, depends on the operating system. Garment360 therefore adds a deterministic Tamil number-word and defect-term parser and a server-side text-to-speech proxy so that Tamil audio is still produced when no local voice exists. For natural-language manager queries, large-language-model APIs (OpenAI, n.d.) are used only as an optional layer on top of answers grounded in live database records.

3. Problem statement and objectives

Four operational problems motivate the work: (1) opaque production tracking across the four stages; (2) language and literacy barriers that make English-only data entry error-prone; (3) reactive planning without quantitative line-balancing or profit models; and (4) duplicate production records and disputed piece-rate payroll caused by manual logbooks.

The objectives are therefore to: (a) record order, stage and worker data in one consistent relational store; (b) predict delivery delay and expose bottlenecks early; (c) recommend suitable workers and rank orders by urgency; (d) let managers simulate capacity and overtime changes with their financial effect; (e) let workers report progress and receive instructions in Tamil; and (f) compute piece-rate earnings automatically and without duplicates.

4. System architecture

4.1 Overview

Garment360 follows a three-tier architecture (Fig. 1). The client layer contains a Manager Portal and a Worker Portal built with plain HTML5, CSS3 and JavaScript. The application layer is an Express.js REST API in which every request passes through CORS, JSON parsing, JWT verification and role guards. The data layer is PostgreSQL accessed through a connection pool; if the PostgreSQL server is unreachable, the system falls back automatically to an in-memory engine (pg-mem), which allows demonstrations without installation effort.

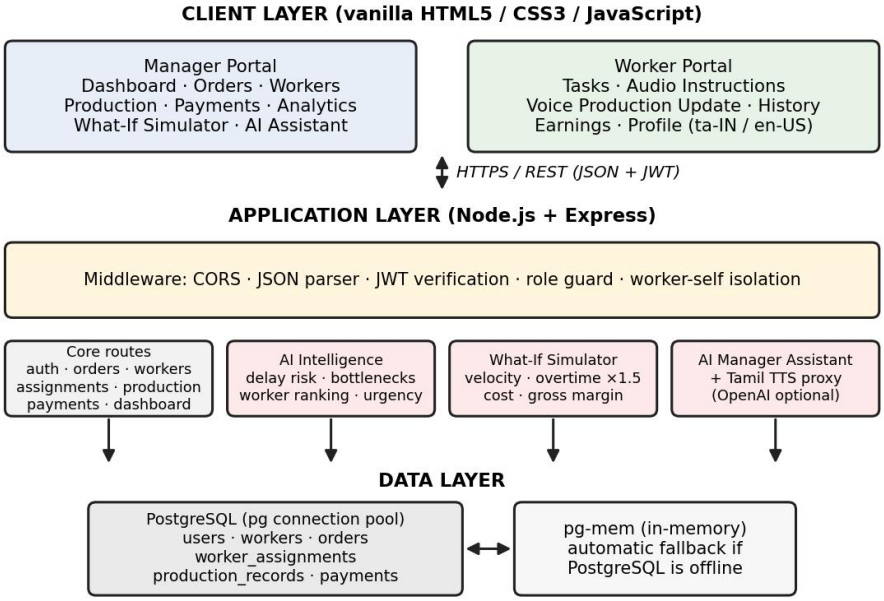


Fig. 1 - Three-tier architecture of Garment360.

4.2 Technology stack

Table 1 lists the technologies used. The frontend deliberately avoids a framework build pipeline so that pages load quickly on low-specification factory tablets.

Table 1 - Technology stack of Garment360.

Layer	Technology	Purpose
Client	HTML5, CSS3, JavaScript (ES2022)	Manager and worker portals; no framework build step
Speech	Web Speech API (SpeechRecognition, SpeechSynthesis)	Tamil (ta-IN) and English (en-US) voice input and audio output
Application	Node.js, Express 4	REST API, middleware chain, static delivery
Data access	node-postgres (pg), pg-mem	Pooled parameterized queries; in-memory fallback
Security	jsonwebtoken, bcryptjs, dotenv	Signed tokens, password hashing, secret isolation
AI assistant	OpenAI Node SDK (optional)	Natural-language manager chat; deterministic engine when no key
Testing	Native Node.js HTTP test suite	88 automated assertions (npm test)

4.3 Database design

The relational schema has six tables: *users*, *workers*, *orders*, *worker_assignments*, *production_records* and *payments*. Foreign keys with ON DELETE CASCADE or SET NULL rules, check constraints and fixed-precision decimal types for currency maintain integrity. Multi-table operations such as recording production and creating the related payment, or approving a simulated plan, run inside BEGIN/COMMIT/ROLLBACK transactions so that either all changes are stored or none.

4.4 Security

Passwords are hashed with bcrypt (cost factor 10) (Provos & Mazières, 1999). Sessions use stateless JSON Web Tokens (Jones, Bradley & Sakimura, 2015) signed with HMAC-SHA256 and valid for seven days; the payload carries only the user id, role, e-mail and worker id. Manager-only routes (simulation, intelligence and assistant) reject worker tokens with HTTP 403, and a worker-self guard stops a worker from reading or writing another worker's assignments. All SQL statements use numbered parameters, which removes SQL-injection risk, and secrets are kept in an environment file excluded from version control.

5. AI production intelligence and simulation

The intelligence layer is deliberately deterministic and explainable: every score can be traced to a visible rule, which is important for supervisors who must justify decisions to customers and workers. Let T be the order target quantity and C_s the completed quantity at stage s.

5.1 Order progress and status

Order progress averages the capped completion of the four stages:

$$Progress = \min\left(100, \left\lfloor \Sigma_s \min(T, C_s) / (4T) \times 100 \right\rfloor\right) \quad (1)$$

An order becomes *Completed* only when packing reaches the target ($C_{Pack} \geq T$); it moves from *Pending* to *In Progress* as soon as any stage records output.

5.2 Delay-risk prediction

Daily throughput velocity V is estimated from history when at least two active days exist; otherwise a baseline of fifteen units per assigned worker, or a factory default of 20 units, is used:

$$V = \max\left(5, \left\lfloor \Sigma C / (D_a \times 4) \right\rfloor\right), \quad D_a \geq 2 \quad (2)$$

Here ΣC is the total completed quantity over all shifts and D_a the number of unique active days. The days needed and days remaining are

$$D_{need} = \lceil (T - C_{Pack}) / V \rceil \quad (3)$$

$$D_{rem} = \lceil (\text{Delivery date} - \text{Today}) / 86\,400\,000 \text{ ms} \rceil \quad (4)$$

and the risk class follows the rules in Table 2.

Table 2 - Delay-risk classification rules.

Condition	Risk class
Remaining units = 0	Completed
$D_{rem} < 0$ (overdue)	High
$D_{need} > D_{rem}$ (projected delay)	High
$D_{need} \geq D_{rem} - 2$ (buffer of two days or less)	Medium
Otherwise	Low

5.3 Bottleneck detection

For each consecutive stage pair (S_i, S_{i+1}) the system computes the queue accumulation and flow-balance ratio:

$$Q_i = C_i - C_{i+1}, \quad R_i = C_{i+1} / \max(1, C_i) \quad (5)$$

A stage is flagged as a bottleneck when $Q_i \geq 30$ units or $R_i < 0.60$. Severity is *High* when the queue exceeds 50 units or the backlog exceeds three production days, and *Medium* otherwise. Each flag is returned with evidence and a corrective action.

5.4 Worker recommendation

Candidate workers for a bottlenecked stage are ranked by a composite score made of skill match, free capacity and historical productivity (Table 3):

$$Score = S_{skill} + S_{cap} + S_{prod} \quad (6)$$

Table 3 - Worker recommendation score components.

Component	Rule	Points
Skill match	Department equals target stage	45
	Skills contain stage keyword	35
	Otherwise	10
Free capacity	0 / 1 / 2 / ≥3 active tasks	35 / 25 / 15 / 5
Productivity	min(20, ⌊ productivity rate / 5 ⌋)	0 – 20

5.5 Order urgency

Orders are ranked by an urgency score built from base risk (High 40, Medium 25, Low 10), deadline pressure and order value:

$$U = B_{risk} + S_{dl} + S_{rev} \quad (7)$$

$$S_{dl} = \min(40, \max(0, (14 - D_{rem}) \times 3)), \quad S_{rev} = \min(20, \lfloor Value / 10\,000 \rfloor) \quad (8)$$

A score of 65 or more maps to *High* priority, 40 or more to *Medium*, and anything lower to *Low*.

5.6 What-If simulation and financial model

The simulator lets a manager change the number of workers (ΔW) and the daily overtime hours (OT) and compares the result with the baseline. Each added worker contributes fifteen units per day and overtime is assumed to be 85 % as productive as regular time:

$$V_{prop} = V_{base} + \Delta W \times 15 + (OT / 8) \times V_{base} \times 0.85 \quad (9)$$

The projected duration is $D_{prop} = \lceil$ remaining units / $V_{prop} \rceil$. Overtime is paid at 1.5 times the hourly rate. With n assigned workers, hourly rate h and daily overtime OT, the total cost and gross margin (GM) are

$$TC = M + n \cdot D_{prop} \cdot 8h + n \cdot D_{prop} \cdot OT \cdot 1.5h + O \quad (10)$$

$$GM\% = (Revenue - TC) / Revenue \times 100 \quad (11)$$

where TC is total production cost, M is material cost and O is overhead. The simulator reports duration, labour and overtime cost, gross profit and gross margin, and an approved scenario is written to the database atomically.

5.7 Piece-rate payroll

Earnings are computed automatically as quantity completed multiplied by the rate per piece, and monthly productivity is the ratio of completed to target quantity, capped at 120 %. A duplicate-submission guard prevents the same record from being saved twice when a button is pressed repeatedly.

6. Tamil voice-first reporting

The Worker Portal supports two audio directions (Fig. 2). For *spoken reporting*, the browser captures speech with the ta-IN locale (with English fallbacks). The microphone permission is first warmed up, then speech recognition runs continuously with a live waveform, and the transcript is parsed automatically after a period of silence or when the worker presses Stop and Parse. A deterministic parser extracts the completed quantity and the defective (rejected) quantity from digits or Tamil number words and recognised defect terms. The worker sees an editable draft and confirms by voice or by tapping, and only then is the record saved through the authenticated production API.

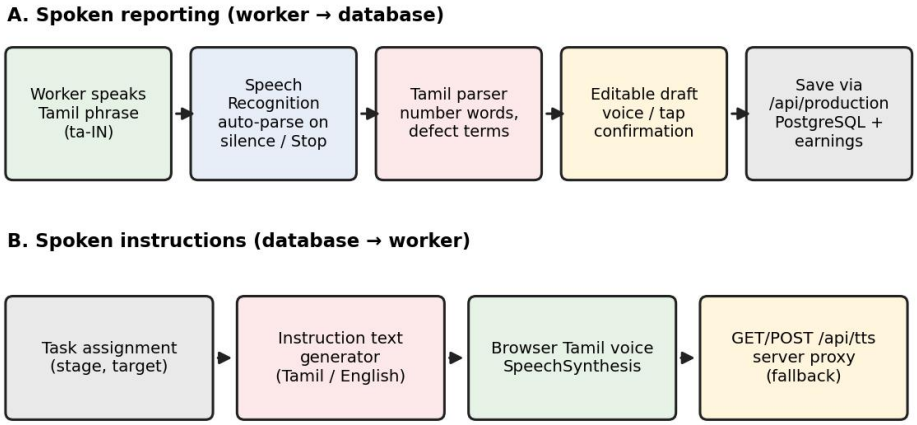


Fig. 2 - Voice pipelines: (A) spoken reporting and (B) spoken instructions.

For *spoken instructions*, task details are converted into short operating instructions in Tamil or English. They are played with a local Tamil voice when available; otherwise the client requests Tamil audio from the server text-to-speech proxy (/api/tts) so that playback never silently falls back to English. Because parsing is rule-based and runs inside the application, it needs no external cloud service and has predictable behaviour on the shop floor.

7. Implementation

7.1 Manager portal

The Manager Portal comprises a dashboard with live statistics, order management with stage progress, a worker registry, assignment of workers to stages with target quantities, production tracking, payments, analytics charts, the What-If simulator and the AI Manager Assistant.

7.2 Worker portal

The Worker Portal provides a personal dashboard, assigned tasks, audio instructions, a voice and manual production-update page, work history, earnings and a profile. The interface uses high-contrast status colours and large tap targets suited to industrial tablets.

7.3 AI Manager Assistant

The assistant answers questions such as “Tell me about ORD-1042” by loading the live order, its assignments, stage progress, costs and risk from PostgreSQL and composing a reply. When an OpenAI key is configured it is used for fluent natural language; without a key the built-in deterministic engine answers from the same data.

8. Results and discussion

8.1 Functional verification

Verification used an automated Node.js test suite with 88 assertions (Table 4). All 88 assertions passed in the final regression run, covering authentication and authorisation, the order and production lifecycle, every intelligence algorithm in Section 5, simulation and atomic plan approval, and the Tamil voice utilities.

Table 4 - Automated test coverage.

Phase	Tests	Scope
1	1–12	PostgreSQL connection, password hashing, JWT issue and verify, role protection, table constraints
2	13–28	Order creation, worker assignments, stage transitions, production logs, duplicate rejection
3	29–41	Intelligence summary, delay risk, bottleneck queues, worker ranking, order urgency
4	42–57	Simulation baseline, overtime modelling (1.5×), profit formulas, atomic plan approval
5	58–88	Locale enforcement, Tamil and English instruction text, Tamil transcript parsing, voice confirmation, strict Tamil voice selection, production and payment persistence

8.2 Tamil transcript parsing

Table 5 shows representative spoken phrases used in the test suite and the values extracted by the parser. Number words and digits are both handled, and defect terms are mapped to rejected units.

Table 5 - Tamil transcript parsing examples.

Spoken transcript (Tamil)	Completed	Rejected
ஐம்பது முடிச்சிட்டேன், இரண்டு ரிஜெக்ட்	50	2
50 சட்டைகள் முடிந்தது, 2 பழுதானது	50	2
இன்று 75 முடித்தேன், 3 நிராகரிக்கப்பட்டது	75	3
முப்பது பீஸ் முடிச்சேன்	30	0
எழுபத்தைந்து துண்டுகள் முடித்தேன்	75	0

8.3 Discussion and limitations

The results show that the system behaves as specified: rule-based risk, bottleneck and cost outputs are reproducible, and Tamil phrases are converted into correct quantities. Because every score is a visible rule, managers can verify why an order is flagged, which supports trust in the recommendations.

Several limitations remain. The “AI” is heuristic rather than learned, and the baseline velocity assumptions (for example fifteen units per worker per day) must be calibrated for each factory. Tamil recognition quality depends on the browser and device. The reported evidence is functional verification on seeded data; a field trial with measured delay reduction, data-entry time and error rate has not yet been carried out. Only the four-stage garment workflow of a single factory is modelled.

9. Conclusion and future scope

This paper presented Garment360, a web-based garment management system that combines relational production tracking, explainable production intelligence, profit-aware What-If simulation, Tamil voice-first reporting and a live AI Manager Assistant. The implementation, built with Node.js, Express and PostgreSQL, passed all 88 automated tests.

Future work includes learning velocity and delay models from accumulated production history, a pilot deployment in a working garment unit with measured business metrics, browser-independent speech recognition, offline-capable progressive web app support, multi-factory management and integration with messaging and barcode or IoT devices.

Acknowledgements

The authors thank Mrs.S.Nandhini AP/CSE and Sri Shakthi Institute of Engineering and Technology for their guidance and support during this project.

References

Goldratt, E. M., & Cox, J. (1984). *The goal: A process of ongoing improvement*. Great Barrington, MA: North River Press.

Hopp, W. J., & Spearman, M. L. (2008). *Factory physics* (3rd ed.). Boston: McGraw-Hill/Irwin.

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)* (RFC 7519). Internet Engineering Task Force.

OpenAI. (n.d.). *OpenAI API documentation*. https://platform.openai.com/docs

Pinedo, M. L. (2016). *Scheduling: Theory, algorithms, and systems* (5th ed.). Cham: Springer.

PostgreSQL Global Development Group. (n.d.). *PostgreSQL documentation*. https://www.postgresql.org/docs/

Provos, N., & Mazières, D. (1999). A future-adaptable password scheme. In *Proceedings of the USENIX Annual Technical Conference* (pp. 81–91).

W3C Web Speech API Community Group. (n.d.). *Web Speech API specification*. https://webaudio.github.io/web-speech-api/