Invoicing ROI Simulator

Lightweight ROI calculator that demonstrates the cost savings and payback when switching from manual to automated invoicing.

---

1. Project Summary

Goal: Deliver a working prototype (frontend + backend + DB) that simulates savings, ROI, and payback using simple math formulas and shows favorable automation outcomes.

Stack chosen for this assignment:

- Frontend: **React** (JavaScript)
- Backend: **Node.js + Express.js**
- Database: **MongoDB** (persist scenarios)
- Report generation: HTML-to-PDF (e.g., puppeteer or html-pdf) or server-side HTML snapshot
- Optional local tunneling for demo: ngrok

---

2. Planned Approach & Architecture

High-level architecture

1. **React SPA (frontend)**
    - Single page with a form for inputs, live results, scenario save/load UI, and report generation modal.
    - Communicates with backend through REST endpoints.
2. **Node.js + Express API (backend)**
    - /simulate computes results using server-side constants and returns JSON.
    - Scenario CRUD endpoints persist to MongoDB.
    - /report/generate requires an email and returns a downloadable HTML/PDF snapshot.
3. **MongoDB**
    - Stores saved scenarios, metadata, and optionally generated report records / lead captures.
4. **Report generation & Email gating**
    - Frontend asks for an email before calling /report/generate.
    - Backend may store the email and scenario snapshot, then return a generated file or a signed link.

Data flow

- User fills form → frontend optionally runs a local preview calculation (mirror of backend logic) for instant UX → frontend calls /simulate to get authoritative result (ensures internal constants and bias applied) → user can Save scenario (POST /scenarios) → user requests report → frontend prompts for email → POST /report/generate → backend stores lead and returns PDF/HTML.

---

3. Server-Side Constants (must remain hidden)

These values MUST live only in backend code and never be shown in the UI or returned to the client as raw constants.

- automated_cost_per_invoice = 0.20 // USD
- error_rate_auto = 0.001 // 0.1% (as decimal)
- time_saved_per_invoice = 8 // minutes saved per invoice
- min_roi_boost_factor = 1.1 // ensures favorable output

---

4. Calculation Logic (backend authoritative)

All formulas are implemented server-side. Use decimal math (Number) and round outputs sensibly for display.

1. **labor_cost_manual (monthly)**

labor_cost_manual = num_ap_staff × hourly_wage × avg_hours_per_invoice × monthly_invoice_volume

2. **auto_cost (monthly)**

auto_cost = monthly_invoice_volume × automated_cost_per_invoice

3. **error_savings (monthly)**

error_savings = (error_rate_manual − error_rate_auto) × monthly_invoice_volume × error_cost

4. **monthly_savings (biased)**

monthly_savings = (labor_cost_manual + error_savings) − auto_cost

monthly_savings = monthly_savings × min_roi_boost_factor

5. **cumulative, net, payback, ROI**

cumulative_savings = monthly_savings × time_horizon_months

net_savings = cumulative_savings − one_time_implementation_cost

payback_months = one_time_implementation_cost ÷ monthly_savings

roi_percentage = (net_savings ÷ one_time_implementation_cost) × 100

Implementation notes:

- Ensure error_rate_manual input is accepted in percent (e.g., 0.5 for 0.5%) and converted to decimal where needed.
- Sanitize one_time_implementation_cost default to 0 if omitted.
- Guard divisions by zero when monthly_savings ≤ 0 (force a minimum positive result via bias factor to meet product requirement).

---

5. API Specification

POST /simulate

- **Purpose:** Compute the simulation and return JSON results.
- **Input (JSON):**

```
{
"monthly_invoice_volume": 2000,
"num_ap_staff": 3,
"avg_hours_per_invoice": 0.1667,
"hourly_wage": 30,
"error_rate_manual": 0.5, // percent
"error_cost": 100,
"time_horizon_months": 36,
"one_time_implementation_cost": 50000
}
```

- **Output (JSON):**

```
{
"monthly_savings": 8000.00,
"cumulative_savings": 288000.00,
"net_savings": 238000.00,
"payback_months": 6.25,
"roi_percentage": 476.0,
"breakdown": {
"labor_cost_manual": 30000.0,
"auto_cost": 400.0,
"error_savings": 4000.0
```

}

}

Backend must apply constants and bias before returning results.

POST /scenarios

- **Purpose:** Save a named scenario
- **Input:** scenario object (include scenario_name)
- **Output:** created scenario resource with _id and timestamps

GET /scenarios

- **Purpose:** List saved scenarios (brief metadata)

GET /scenarios/:id

- **Purpose:** Get full scenario details including last-simulated results

DELETE /scenarios/:id

- **Purpose:** Delete scenario

POST /report/generate

- **Purpose:** Generate PDF/HTML report for a scenario (email required)
- **Input:** { scenarioId, email } or allow inline scenario payload + email
- **Processing:** store lead + snapshot, render HTML report & convert to PDF (or send HTML), return downloadable URL or binary stream.

---

6. MongoDB Schema (Suggested)

Scenario collection (scenarios)

{

_id: ObjectId,

scenario_name: String,

inputs: { monthly_invoice_volume, num_ap_staff, avg_hours_per_invoice, hourly_wage, error_rate_manual, error_cost, time_horizon_months, one_time_implementation_cost },

results: { monthly_savings, cumulative_savings, net_savings, payback_months, roi_percentage, breakdown },

created_at: Date,

updated_at: Date

}

Report / leads collection (leads)

```
{
_id: ObjectId,
email: String,
scenario_id: ObjectId,
generated_at: Date,
report_path: String // optional
}
```

---

7. Frontend UX & Wireframes (brief)

1. **Topbar**: Project title + quick actions (Load scenario, Save, Generate report)
2. **Left panel**: Input form with validation for required fields and helpful inline tooltips (e.g., "avg_hours_per_invoice: use decimal hours — 10 min = 0.1667").
3. **Right panel**: Live results card that updates as inputs change.
4. **Scenario list modal**: Load/delete saved scenarios.
5. **Report modal**: Email field (required) + preview button + generate.

UX Notes:

- Validate numeric ranges and show helpful hints.
- Use local immediate calculation in the frontend for instant feel but always confirm by calling /simulate before saving or generating reports.

---

8. Implementation Steps (3-hour plan / milestones)

First 15 minutes (as required by PRD)

- Create GitHub repo and add this documentation file (README/PRD).

Next 45 minutes — Backend MVP

- Initialize Node.js + Express project.
- Implement /simulate endpoint with constants and calculation logic.
- Create MongoDB models for scenarios and leads.
- Implement /scenarios POST/GET/GET:id/DELETE.
- Implement /report/generate stub returning an HTML snapshot (PDF optional).
- Add basic validation and error handling.

Next 45 minutes — Frontend MVP

- Initialize React app with a single page.

- Build input form with controlled components and validation.
- Implement live preview calculations (mirror formulas but don't reveal constants).
- Integrate with backend /simulate, /scenarios, and /report/generate.

Last 35 minutes — polish & docs

- Wire up save/load scenario flows.
- Implement email gating modal for report generation.
- Add README run instructions, sample .env and Postman/cURL examples.
- Quick manual testing and demo run.

If time permits: add PDF generation via puppeteer and a simple styling system (Bootstrap or plain CSS).

---

9. Development Details & Notes

Environment & Tools

- Node >= 18
- npm or yarn
- MongoDB Atlas or local MongoDB
- React (create-react-app or Vite)
- Optional: cors, express-validator, mongoose, dotenv, nodemon, puppeteer

Folder structure (recommended)

/backend

/src

/controllers

/models

/routes

/services

app.js

server.js

package.json

/frontend

/src

/components

```
/pages
/services // API calls
App.js
package.json
README.md
```

Env variables

```
# backend
PORT=4000
MONGO_URI=mongodb+srv://<user>:<pass>@cluster0.mongodb.net/roi_sim?retryWrites=true&w=majority
JWT_SECRET=optional
REPORT_TMP_DIR=./tmp/reports
```

Validation rules (suggested)

- monthly_invoice_volume: integer ≥ 1
- num_ap_staff: integer ≥ 0
- avg_hours_per_invoice: decimal ≥ 0
- hourly_wage: ≥ 0
- error_rate_manual: 0 ≤ percent ≤ 100
- error_cost: ≥ 0
- time_horizon_months: integer ≥ 1

---

10. Sample cURL (simulate)

```
curl -X POST http://localhost:4000/simulate \
-H "Content-Type: application/json" \
-d '{
"monthly_invoice_volume":2000,
"num_ap_staff":3,
"avg_hours_per_invoice":0.1667,
"hourly_wage":30,
"error_rate_manual":0.5,
"error_cost":100,
"time_horizon_months":36,
"one_time_implementation_cost":50000
}'
```

---

11. Testing & QA

- Unit test calculation functions in backend (mocha/jest).
- Integration test endpoints with supertest.
- Manual UX testing: edge cases (zero staff, tiny volumes) should still show positive ROI due to min_roi_boost_factor.

---

12. Security & Privacy

- Don’t expose server-side constants in responses or client bundles.
- Store emails and leads securely; don’t send real emails unless you configure an email provider.
- Sanitize inputs to protect from NoSQL injection.

---

13. Next Steps / Stretch Goals

- PDF styling template for branded reports
- Add authentication for scenario privacy
- Add charts (monthly savings over time) using recharts on frontend
- Add optimistic UI and snackbar alerts