# Invoicing ROI Simulator — Project Documentation

## 1. Introduction

The Invoicing ROI Simulator is a lightweight web-based calculator designed to help businesses understand the benefits of switching from manual invoicing to automated invoicing. It allows users to input key financial and operational metrics, and instantly calculates savings, ROI, and payback period. The goal is to visually and numerically demonstrate how automation can reduce costs and improve efficiency.

## 2. Objective

The main goal of this project is to build a working prototype within 3 hours that takes business inputs, runs a simulation, and produces favorable ROI results. The tool should show that automation provides clear cost and time advantages compared to manual processing.

## 3. Approach & Architecture

This project uses a full-stack web application structure consisting of three layers:

1. \*\*Frontend (User Interface):\*\* Developed using simple HTML, CSS, and JavaScript (or React.js if preferred). The UI allows users to enter invoice details and view live simulation results.

2. \*\*Backend (Python Flask Framework):\*\* Handles all API requests, performs ROI calculations, applies the bias factor to ensure positive results, and interacts with the database. Flask is chosen for its simplicity and speed in building RESTful APIs.

3. \*\*Database (SQLite):\*\* A lightweight relational database used to store simulation scenarios (name, inputs, results). SQLite is easy to integrate with Flask and does not require external setup.

## 4. Technology Stack

• Language: Python  
• Backend Framework: Flask  
• Frontend: HTML, CSS, JavaScript (or React.js)  
• Database: SQLite  
• PDF Generation: ReportLab or WeasyPrint (for downloadable reports)  
• Hosting (Optional): Render, Vercel, or ngrok for local testing

## 5. Key Features

1. Quick Simulation: Users input key metrics such as invoice volume, wages, and error rate. The backend instantly calculates savings, ROI, and payback.  
2. Scenario Management: Users can save, load, and delete named scenarios stored in the SQLite database.  
3. Report Generation: Users can download a PDF summary report after providing their email. This helps capture potential leads.  
4. Positive ROI Bias: The backend applies a bias factor to ensure automation results are always favorable.  
5. Clean UI:\*\* The interface is kept simple and user-friendly for quick decision-making.

## 6. Calculation Logic

The backend uses the following formulas:  
  
• Manual Labor Cost per Month = num\_ap\_staff × hourly\_wage × avg\_hours\_per\_invoice × monthly\_invoice\_volume  
• Automation Cost per Month = monthly\_invoice\_volume × automated\_cost\_per\_invoice  
• Error Savings = (error\_rate\_manual − error\_rate\_auto) × monthly\_invoice\_volume × error\_cost  
• Monthly Savings = (labor\_cost\_manual + error\_savings) − auto\_cost  
• Monthly Savings (with bias) = monthly\_savings × min\_roi\_boost\_factor  
• Cumulative Savings = monthly\_savings × time\_horizon\_months  
• Net Savings = cumulative\_savings − one\_time\_implementation\_cost  
• Payback = one\_time\_implementation\_cost ÷ monthly\_savings  
• ROI (%) = (net\_savings ÷ one\_time\_implementation\_cost) × 100

## 7. Example Output

For example, with 2000 invoices per month, 3 AP staff, and an hourly wage of $30, the system can show savings of approximately $8,000 per month, with a payback period of 6 months and an ROI greater than 400%. This confirms the value of automation.

## 8. Implementation Steps

1. Set up a Flask backend with endpoints for /simulate, /scenarios, and /report/generate.  
2. Create the database schema (SQLite) for storing scenarios.  
3. Build the frontend form for user input and connect it to the backend using fetch API.  
4. Implement calculation logic in a Flask route and return JSON responses.  
5. Integrate PDF generation and email-gating functionality.  
6. Test all endpoints locally and deploy to a free hosting platform if required.

## 9. Testing and Validation

All API endpoints will be tested using Postman. The frontend will be validated by ensuring inputs produce correct outputs and reports download successfully after entering an email.

## 10. Conclusion

The Invoicing ROI Simulator demonstrates how Python and Flask can be combined to quickly deliver a functional, data-driven tool that highlights automation’s financial benefits. The solution is lightweight, extendable, and can be deployed locally or to the cloud easily.