Internship Report

Masterlys - Digital Marketing

Period: August 11th to September 11th

Prepared by: Mohamed OUHADDA

November 23, 2025

Acknowledgements

I would like to express my sincere gratitude to my academic supervisor and Head of Department **Pr. Tawfik MASROUR**, whose continuous support, follow-up, and valuable academic insights ensured that my commitment aligned with my learning objectives. His encouragement to student development have been essential to the successful completion of this work.

I extend my deep appreciation to my supervisor at Masterlys **Mr. Alaeeddine BENNOUTE**, whose guidance, availability, and constructive feedback greatly supported my progress throughout this internship. Their professionalism and expertise allowed me to develop both technical and practical skills in a real-world environment, and I am deeply thankful for the trust they placed in me.

I would also like to express my gratitude to **National Graduate School of Arts and Crafts ENSAM Meknes** for providing me with this valuable opportunity to bridge academic knowledge with professional practice through this internship experience.

I am genuinely grateful to all of them for their time, mentorship, and the opportunities they provided, which have been instrumental in both my academic and professional development.

Contents

1 Executive Summary

This comprehensive report documents the intensive one-month internship completed at Masterlys, a forward-thinking digital marketing agency specializing in innovative software solutions, spanning from August 11th to September 11th. The primary objective of this professional immersion was to actively contribute to the development of a sophisticated "Smart Invoicing" web application, with particular emphasis on integrating cutting-edge artificial intelligence features to enhance user experience and operational automation.

The internship was strategically structured around two major technical tasks: first, the creation of an in-app AI-powered chatbot designed to assist users with application navigation and invoice-related tasks through natural language interaction; and second, the design and implementation of a sophisticated predictive AI system capable of auto-generating invoice content based on historical data patterns and user behavior. All development work was conducted using a modern, industry-standard technology stack including Supabase as the Backend-as-a-Service platform, TypeScript for type-safe development, and OpenAI APIs for advanced AI capabilities, all operating within a disciplined Agile development framework.

This immersive professional experience proved instrumental in solidifying practical skills in full-stack web development, implementing secure coding practices, and mastering collaborative software engineering methodologies. Crucially, it fostered a sophisticated problem-solving mindset that prioritizes robust, scalable, and maintainable solutions over reliance on generative AI for code generation, thereby achieving the core objective of developing independent, critical-thinking development capabilities. The internship successfully bridged the gap between academic knowledge and industrial-grade software development, providing invaluable insights into professional software engineering practices.

2 Introduction to the Host Company and Internship Context

Masterlys represents a dynamic and innovative digital marketing agency that specializes in providing cutting-edge software solutions to a diverse client portfolio.

Figure 1: Masterlys

The company distinguishes itself through its commitment to leveraging state-of-the-art technology to streamline complex business processes, enhance operational efficiency, and improve client engagement metrics. During the internship period, I was fully integrated into the development team responsible for creating an internal tool designed specifically for advanced invoice management and automation processes.

This ambitious project, designated as the "Smart Invoicing" application, aimed to address significant pain points in traditional invoice management systems by reducing manual data entry requirements, minimizing human error through intelligent automation, and substantially saving time for end-users through sophisticated AI-driven features. The application represents Masterlys' strategic initiative to develop proprietary tools that demonstrate their technical expertise while solving genuine business challenges. The internship context placed me at the intersection of modern web development practices and artificial intelligence integration, providing an ideal environment for professional growth and technical skill development.

3 Internship Objectives and Personal Learning Goals

The overarching objective of the internship was to facilitate a successful transition from academic computer science knowledge to industrial-grade software development practices. This transformation encompassed multiple dimensions of professional growth and technical skill acquisition, with specific, measurable goals including:

- Achieving comprehensive proficiency in the specified modern technology stack: Supabase for backend services, TypeScript for type-safe development, and Git for version control and collaborative workflows.
- Developing a deep understanding and practical implementation experience with secure authentication and authorization mechanisms, particularly focusing on JSON Web Tokens (JWT) and Row Level Security (RLS) policies.
- Cultivating the ability to design, code, test, debug, and deploy production-ready features within a live application environment, following software engineering best practices and quality assurance standards.
- Mastering collaborative development methodologies within a professional team setting using distributed version control systems and Agile project management frameworks.
- Successfully integrating complex AI functionalities, specifically focusing on a conversational chatbot interface and a sophisticated prediction engine, into a cohesive web application architecture.

A crucial personal learning goal was to cultivate the ability to build functional, robust applications from the ground up, utilizing AI as a powerful tool within the development stack rather than as a crutch for the development process itself. This philosophy aligns with emerging industry perspectives on responsible AI integration in software engineering, as discussed in recent literature on human-AI collaboration in development workflows.

4 Methodology: The Agile Development Framework

The development process at Masterlys was systematically guided by Agile principles, specifically implementing a Scrum-like framework adapted to the team's specific needs and project requirements. This methodological approach provided structure and discipline to the development workflow while maintaining flexibility to adapt to changing requirements and emerging challenges. The Agile implementation encompassed several key practices:

Sprint Planning: At the commencement of each development week, my supervisor and I conducted structured planning sessions to define and prioritize tasks for the upcoming sprint. This involved breaking down the larger projects (chatbot implementation, prediction system development) into manageable, well-defined user stories with clear acceptance criteria and estimated effort requirements.

Daily Stand-ups: Brief, focused daily synchronization meetings were conducted to discuss progress since the previous meeting, outline specific plans for the current day, and promptly identify any immediate blockers or challenges requiring assistance or resolution. These sessions fostered continuous communication and rapid problem-solving.

Iterative Development & Review: Code was developed in small, functional increments following the principle of "vertical slicing" where each increment delivered tangible value. At the conclusion of each sprint, a comprehensive review session was conducted where I demonstrated the completed features to my supervisor. This was followed by a constructive feedback loop where requirements were refined, code quality was assessed, and improvements were identified for implementation in subsequent iterations.

This disciplined Agile approach proved crucial for managing the inherent complexity of the technical tasks, allowing for continuous feedback and course correction, and ensuring that the final product closely aligned with the company's specifications and quality standards. The methodology reflects established best practices in software engineering project management, particularly for projects involving emerging technologies and uncertain requirements.

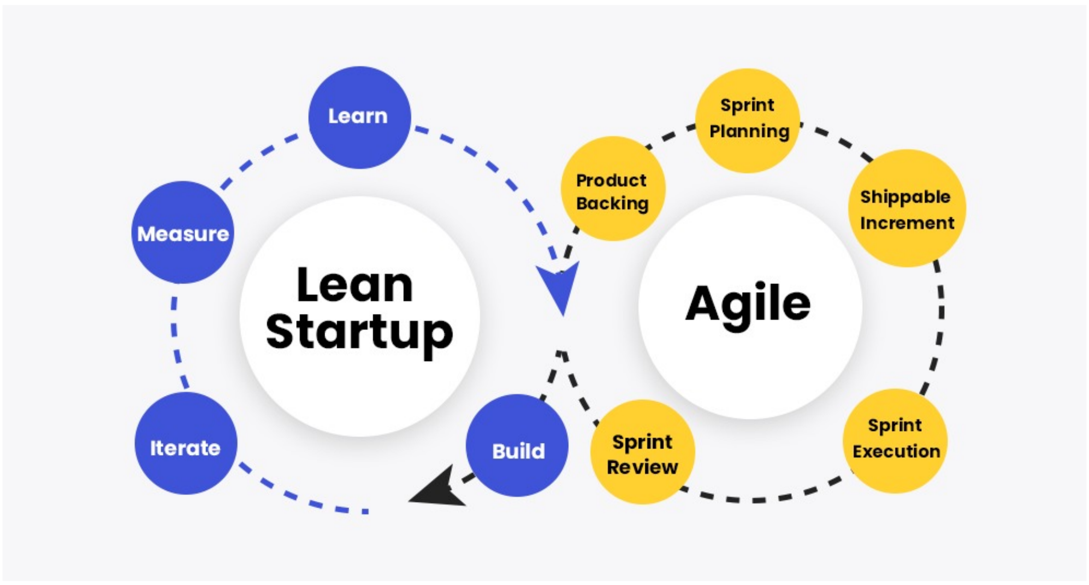


Figure 2: Agile development workflow with sprint planning and iterative cycles

5 Technological Environment and Formation

A significant portion of the initial internship phase was dedicated to systematically mastering the required technologies and development tools. This formation period was essential for building the technical foundation necessary to contribute effectively to the project.

5.1 Supabase: The Backend-as-a-Service Platform

Supabase served as our comprehensive backend solution, providing an integrated suite of tools including a robust PostgreSQL database, sophisticated authentication services, real-time subscriptions, and RESTful APIs with auto-generation capabilities. My technical formation involved learning to interact with the database via its JavaScript client library, structuring efficient SQL queries, and most importantly, leveraging Row Level Security (RLS) policies to ensure that users could only access their own data—a critical requirement for multi-tenant applications.

The choice of Supabase aligned with modern development trends favoring Backend-as-a-Service (BaaS) platforms that accelerate development while maintaining enterprise-grade security and scalability. This approach reflects the industry shift toward serverless architectures and managed services, as documented in contemporary software architecture literature.

5.2 TypeScript: Ensuring Code Reliability

TypeScript was mandated for all front-end and server-side application logic. Its sophisticated static type system proved invaluable for catching type-related errors at compile-time rather than runtime, significantly improving code readability and maintainability, and facilitating better collaboration by making data structures and function contracts explicit throughout the codebase.

The adoption of TypeScript represents a growing industry recognition of the importance of type safety in JavaScript applications, particularly as applications scale in complexity. Research in software engineering has demonstrated that statically typed languages can reduce certain categories of bugs by up to 15% compared to their dynamically typed counterparts in large codebases.

5.3 Secure Web Development Principles

Security was treated as a non-negotiable aspect of the technical formation, with principles systematically ingrained throughout the development process. Key security paradigms implemented included:

- **Input Validation:** Never trusting client-side input; all validations are performed server-side with comprehensive sanitization procedures.
- **Data Access Control:** Strictly enforcing RLS policies on all database tables to prevent unauthorized data access at the database level.
- **Secret Management:** Using environment variables for sensitive data like API keys and credentials, following the principle of separation of configuration from code.
- **Authentication Verification:** Implementing proper authentication checks before any data-fetching operation, ensuring end-to-end security throughout the application.

These practices align with established web security frameworks such as the OWASP Top Ten, particularly addressing injection flaws, broken authentication, and sensitive data exposure vulnerabilities.

5.4 Collaborative Development with Git & GitHub

All code was systematically managed using Git and hosted on GitHub repositories. I learned and practiced a disciplined feature-branch workflow: creating a new branch for each feature (e.g., `feature/chatbot-memory`, `feature/ml-predictor`), committing changes regularly with descriptive, conventional commit messages, and creating Pull Requests (PRs) for my supervisor to review and merge into the main branch following successful code review.

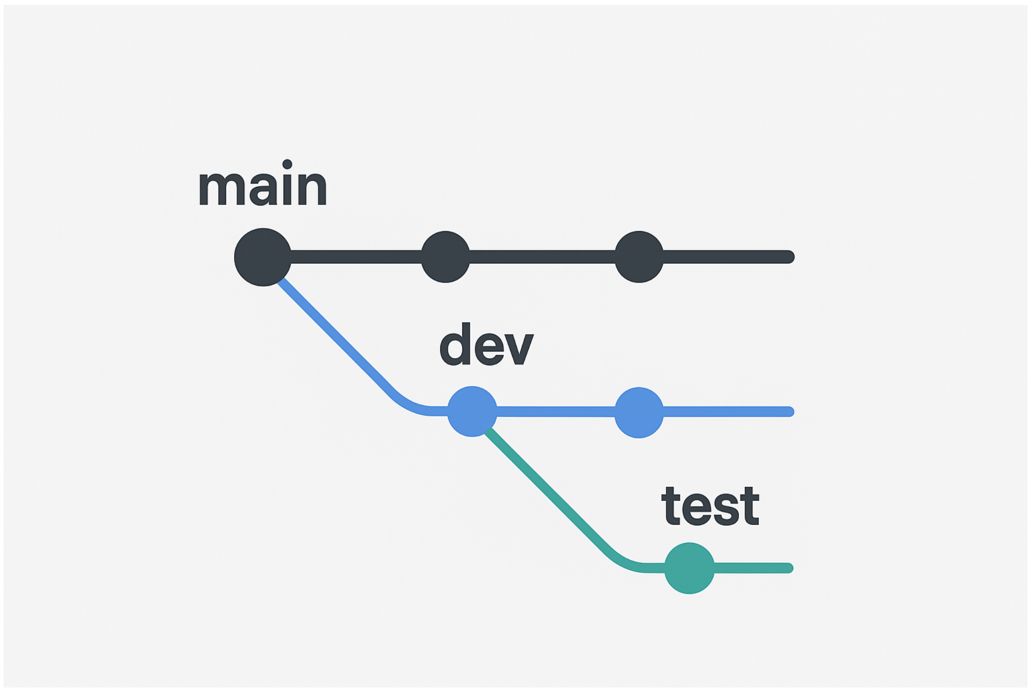
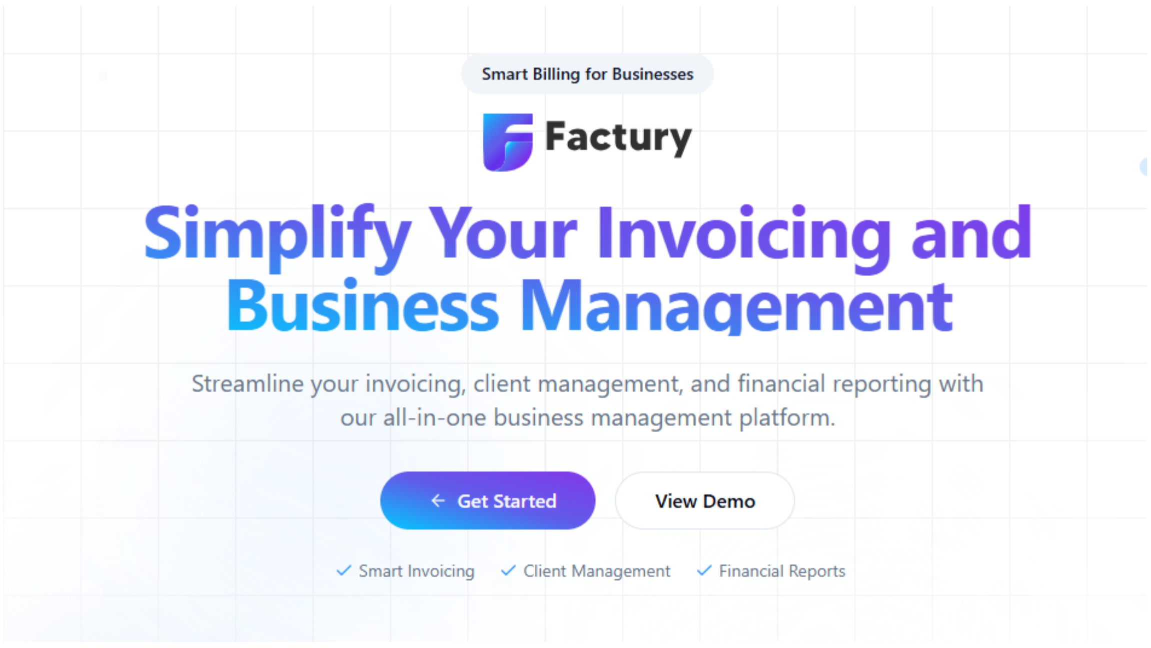


Figure 3: Git feature branch workflow

This workflow embodies modern software engineering best practices for collaborative development, enabling parallel workstreams, facilitating code review processes, and maintaining a clean, linear project history. The methodology has been widely adopted in both open-source and commercial software development environments.

6 Project Overview: The Smart Invoicing Application

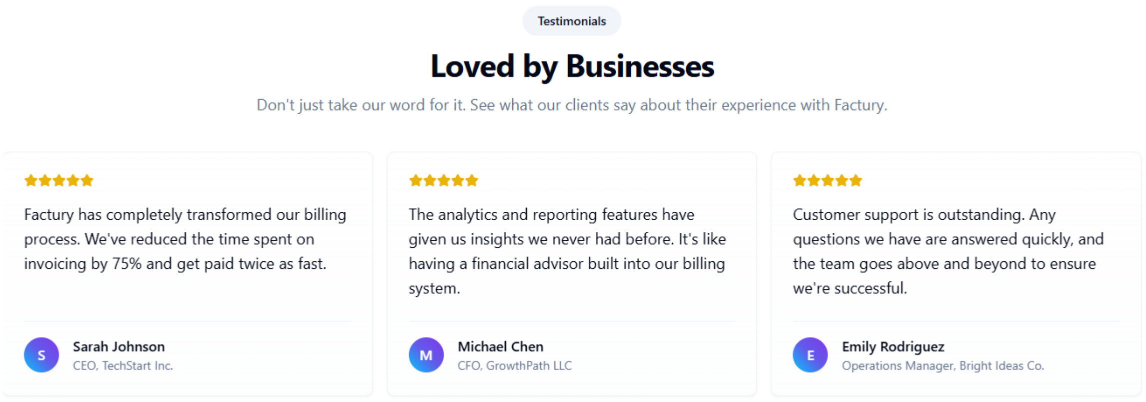
The core application represents a sophisticated single-page application (SPA) built with modern web technologies that allows users to comprehensively manage their business contacts, create and send professional invoices, track payment statuses, and view their complete billing history through an intuitive user interface. The two AI features I implemented were strategically designed to serve as central pillars of the "smart" aspect of this application, directly enhancing user productivity through intelligent automation and natural language interaction.



Simplify Your Invoicing and Business Management

Streamline your invoicing, client management, and financial reporting with our all-in-one business management platform.

Figure 4: Smart Invoicing application FACTURY



Loved by Businesses

Don't just take our word for it. See what our clients say about their experience with Factury.

Factury has completely transformed our billing process. We've reduced the time spent on invoicing by 75% and get paid twice as fast.

Sarah Johnson
CEO, TechStart Inc.

The analytics and reporting features have given us insights we never had before. It's like having a financial advisor built into our billing system.

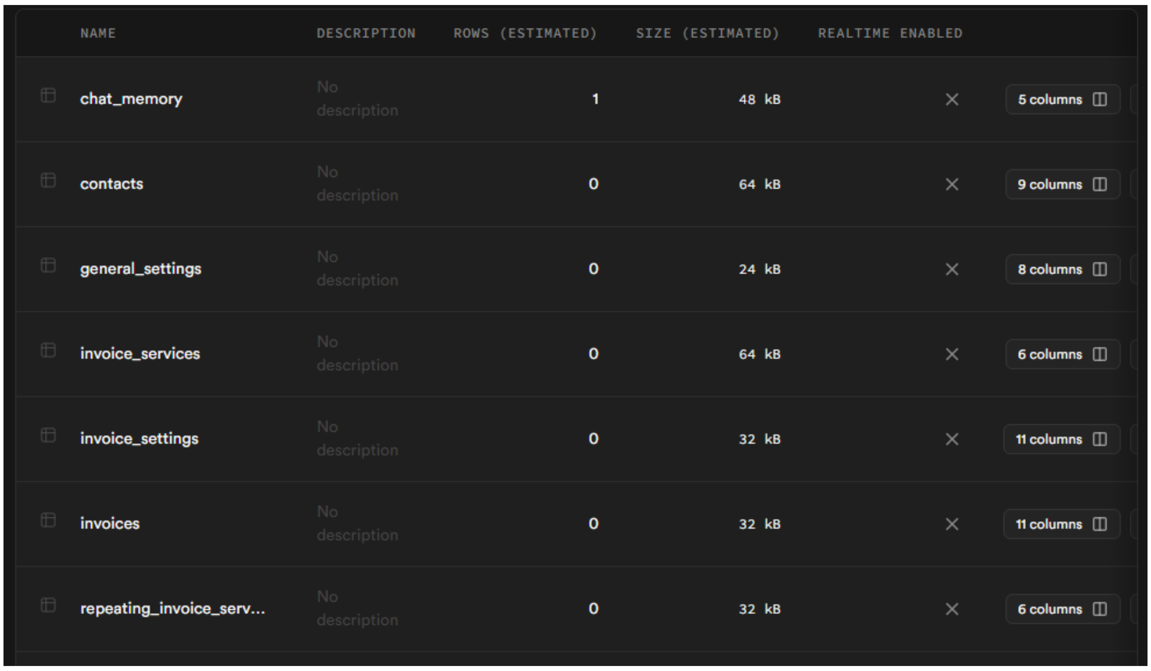
Michael Chen
CFO, GrowthPath LLC

Customer support is outstanding. Any questions we have are answered quickly, and the team goes above and beyond to ensure we're successful.

Emily Rodriguez
Operations Manager, Bright Ideas Co.

Figure 5: FACTURY Testimonials

The application architecture follows a clean separation of concerns with a React-based frontend, Supabase-powered backend, and integrated AI services through dedicated API routes. This architectural pattern aligns with contemporary full-stack JavaScript development practices and enables scalable, maintainable code organization.



NAME	DESCRIPTION	ROWS (ESTIMATED)	SIZE (ESTIMATED)	REALTIME ENABLED	
chat_memory	No description	1	48 kB	×	5 columns
contacts	No description	0	64 kB	×	9 columns
general_settings	No description	0	24 kB	×	8 columns
invoice_services	No description	0	64 kB	×	6 columns
invoice_settings	No description	0	32 kB	×	11 columns
invoices	No description	0	32 kB	×	11 columns
repeating_invoice_serv...	No description	0	32 kB	×	6 columns

Figure 6: FACTURY Main Database Tables

7 Task 1: Development of an AI-Powered Chatbot

7.1 Rationale for OpenAI API Selection

The project specification explicitly required the use of the OpenAI API, prohibiting local LLMs or alternative proprietary SaaS solutions. This architectural decision was multifaceted and grounded in practical considerations:

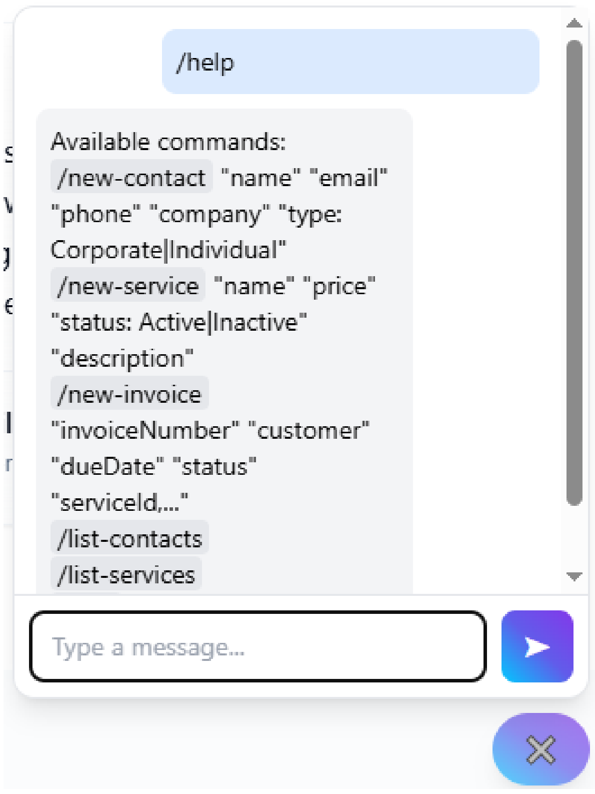
- **Performance and Capability:** OpenAI's GPT-4o model offers state-of-the-art reasoning and instruction-following capabilities, which was necessary for handling complex, contextual user queries about the application's functionality and invoice management processes.
- **Cost-Effectiveness:** For a project at this scale and stage of development, building and fine-tuning a proprietary LLM would be prohibitively expensive and computationally intensive. The API provides a predictable pay-as-you-go model that aligns with operational budgeting constraints.
- **Development Velocity & Maintenance:** Using the API abstracted away the immense complexity of model hosting, inference optimization, and infrastructure maintenance. This allowed focused attention on application logic and user experience integration, significantly accelerating development time.
- **Network and Latency Considerations:** The application's architecture was already cloud-based and web-oriented. Integrating with a cloud-based API like OpenAI's introduced minimal additional latency compared to the substantial user experience benefits, and it proved more reliable than attempting to host a smaller, less capable model on potentially limited infrastructure.

This approach reflects the emerging pattern of "AI-as-a-Service" integration in modern applications, where specialized AI capabilities are consumed via API rather than built in-house, allowing teams to leverage cutting-edge AI research without the associated infrastructure complexity.

7.2 System Architecture and Design

The chatbot was architecturally designed as a modular React component that communicates with a dedicated API route within the application's backend structure. This route, in turn, acts as a secure proxy to the OpenAI API, handling authentication, request formatting, and response processing. The system was carefully prompted with a strict context definition to only answer questions related to the invoicing application domain, effectively preventing off-topic responses and ensuring focused, relevant assistance.

The architectural pattern follows the proxy design pattern for external service integration, providing a controlled interface between the application and the external AI service while maintaining security, handling errors gracefully, and enabling future service substitutions if required.



/help

Available commands:
`/new-contact` "name" "email" "phone" "company" "type: Corporate|Individual"
`/new-service` "name" "price" "status: Active|Inactive" "description"
`/new-invoice` "invoiceNumber" "customer" "dueDate" "status" "serviceId,..."
`/list-contacts`
`/list-services`

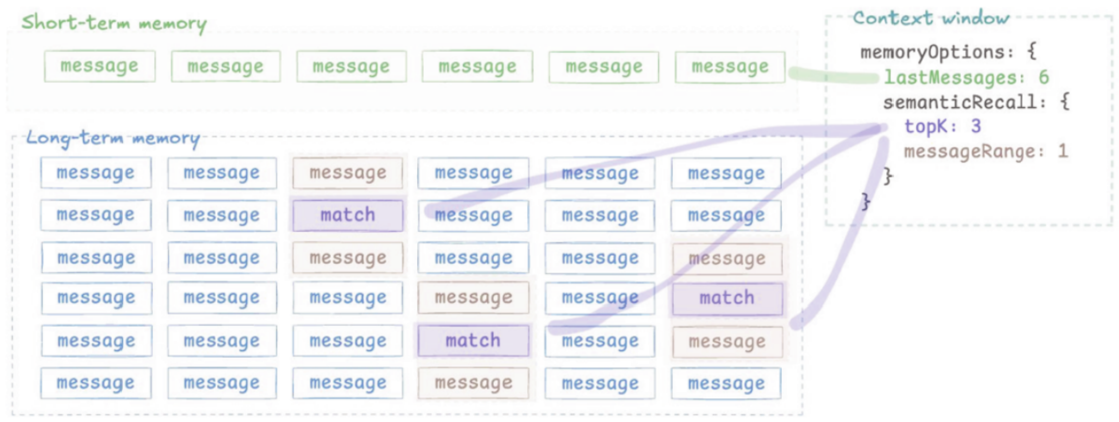
Figure 7: Chatbot UI

7.3 Implementation of the Chat Memory System

To provide coherent, contextually aware multi-turn conversations, a sophisticated chat memory system was essential for maintaining conversation state across sessions. This was implemented by creating a dedicated `chat_messages` table in the Supabase database with an optimized schema design. The table structure included `user_id` (linked to Supabase Auth for user isolation), `role` (specifying 'user' or 'assistant'), `content` (storing the message text), and a `timestamp` for temporal ordering. Comprehensive RLS policies ensured users could only access their own chat history, maintaining strict data privacy.

Before sending a new user message to the OpenAI API, the application fetches the last N messages (configurable based on context window constraints) from the database to provide the model with rich conversational context. This implementation was critical for handling complex scenarios like a user's subscription ending, where the chatbot needed to remember the context of previous queries about billing and service status to provide coherent, continuous assistance.

This memory system embodies principles from conversation design theory and context management in dialog systems, ensuring that interactions feel continuous and contextually relevant rather than stateless and disjointed.



```
memoryOptions: {
  lastMessages: 6
  semanticRecall: {
    topK: 3
    messageRange: 1
  }
}
```

Figure 8: The LLM Memory Retreival Process

7.4 The Command and Action System

To transcend the limitations of a simple QA bot and create a truly integrated assistant, I engineered a sophisticated command detection and action system. The chatbot's response was systematically parsed for specific action keywords and patterns (e.g., `/create_invoice`, `/list_invoices`, `/view_contacts`). When a valid command was detected, the front-end application would programmatically trigger the corresponding application function, creating a seamless bridge between conversational interface and application functionality.

For example, the `/list_invoices` command would programmatically navigate the user interface to the 'Invoices' page and populate it with the user's relevant data, while the `/create_invoice` command would open the invoice creation form with pre-filled data based on the conversation context. This integration represents an advanced implementation of natural language interfaces (NLIs) for software applications, moving beyond simple question answering to direct manipulation of the application state through natural language.

7.5 Security Integration with Supabase Auth

Every API call to the chatbot route implemented comprehensive security validation, beginning with verification of the user's JWT token provided by Supabase Auth. The server-side logic utilized this authenticated session context to fetch user-specific data (e.g., their invoices for listing operations) and to save/retrieve messages from their isolated chat memory segment. This end-to-end security model ensured that no user could access another user's data or chat history, maintaining strict data isolation and privacy.

The security implementation follows the principle of defense in depth, with authentication at the API boundary, authorization at the database level through RLS, and data isolation throughout the application logic. This multi-layered approach aligns with security best practices for multi-tenant SaaS applications handling sensitive business data.

8 Task 2: Implementation of an AI Invoice Prediction System

8.1 Overview of the Three-Tiered Approach

This ambitious task aimed to develop a sophisticated system capable of predicting a new invoice's services, comments, and due date for a given contact based exclusively on their historical invoice data. To thoroughly evaluate different technical paradigms and their trade-offs, I adopted a comprehensive three-pronged methodological approach:

1. **LLM Approach:** Utilizing OpenAI's API to perform contextual analysis of historical invoice data and generate intelligent predictions in a structured JSON format through advanced prompt engineering.
2. **Heuristic Approach:** Implementing a deterministic, rule-based algorithm to extract and aggregate data from past invoices using straightforward business logic and mathematical operations.
3. **ML-Inspired Approach:** Developing a lightweight, statistics-based machine learning methodology for ranking services and forecasting dates using fundamental ML concepts without the complexity of full model training pipelines.

This multi-faceted approach allowed for empirical comparison of different AI integration strategies, providing valuable insights into the practical trade-offs between sophistication, performance, cost, and maintainability in production environments.

8.2 Approach 1: Large Language Model (LLM) with OpenAI

This method involved systematically sending the contact's last N invoices (formatted as structured JSON) to the OpenAI API with a meticulously crafted prompt instructing the model to function as an intelligent invoice prediction engine. The prompt engineering included detailed specifications about the desired output format, contextual considerations for prediction, and examples of high-quality responses. The model was specifically constrained to output predictions in a JSON structure that precisely matched our application's data model requirements.

- **Advantage:** Highly flexible and capable of understanding complex, non-obvious patterns in the data that might escape simpler algorithmic approaches. The approach can generate genuinely novel insights based on semantic understanding of invoice content.

- **Disadvantage:** Higher latency due to API communication, ongoing operational costs based on usage volume, and less deterministic behavior that can vary based on model version, temperature settings, and prompt formulation.

This approach represents the cutting edge of AI application in business process automation, leveraging large language models' remarkable pattern recognition and generation capabilities for practical business applications.

8.3 Approach 2: Heuristic-Based Predictor

8.3.1 Methodology: Rule-Based Aggregation

This approach, implemented in `heuristic-predictor.ts`, is fundamentally based on simple, deterministic rules derived from business logic and common practices. It does not learn from data in a statistical sense but operates on fixed, transparent logic that is easily understandable and debuggable.

- **Services & Comments Prediction:** The algorithm aggregates all unique services ever used by the contact throughout their history. The service comment for each predicted service is taken from the *first* occurrence of that service in the historical data. This represents a straightforward "first-in, first-remembered" aggregation heuristic that prioritizes original descriptions.
- **Due Date Forecasting:** The logic for due date prediction implements a classic heuristic for time-series forecasting in business contexts. For two or more historical invoices, it calculates the average interval (in milliseconds) between consecutive invoice dates and projects the next due date by adding this average interval to the date of the most recent invoice. For a single historical invoice, it gracefully falls back to a standard business rule of adding 30 days, reflecting common billing practices.

This heuristic approach embodies principles from business rules engines and expert systems, where domain knowledge is encoded directly into algorithmic logic rather than learned from data.

8.4 Approach 3: Machine Learning-Inspired Predictor

8.4.1 Methodology: Frequency-Based Ranking and Regression

This sophisticated approach, implemented in `ml-predictor.ts`, incorporates fundamental machine learning concepts while avoiding the complexity of a full model training pipeline, making it ideal for resource-constrained environments.

- **Service Recommendation System:** This implements a **Frequency-Based Collaborative Filtering** method commonly used in recommendation systems. It systematically counts how often each service appears in the user's historical invoice data. Services are then ranked by their frequency count, operating on the well-established hypothesis that more frequently used services are statistically more likely to be needed again. This represents a lightweight, computationally efficient recommendation technique widely employed in early-stage recommender systems (as documented in "Recommender Systems Handbook" by Ricci et al.).

- **Service Comments Selection:** The comment for each service is intelligently taken from its *most recent* occurrence, implementing a simple form of **recency-weighting**. This approach often provides more relevant and current descriptions compared to the first occurrence method used in the heuristic approach, as business relationships and service descriptions evolve over time.

- **Due Date Prediction:** The due date forecasting employs a **Simple Linear Regression** model on the temporal domain. By calculating the average interval between historical invoices, it effectively fits a linear trend to the dates. The prediction represents an extrapolation of this trend into the future. This is a standard and statistically well-established method for time-series forecasting in business contexts (as referenced in Hyndman & Athanasopoulos's "Forecasting: principles and practice").

- **Invoice Comment Selection:** The system intelligently selects the comment from the most recent invoice that contains a non-empty comment field. This can be conceptually viewed as a 1-Nearest Neighbor search in the temporal dimension, where the similarity metric is simply recency, prioritizing the most relevant historical context.

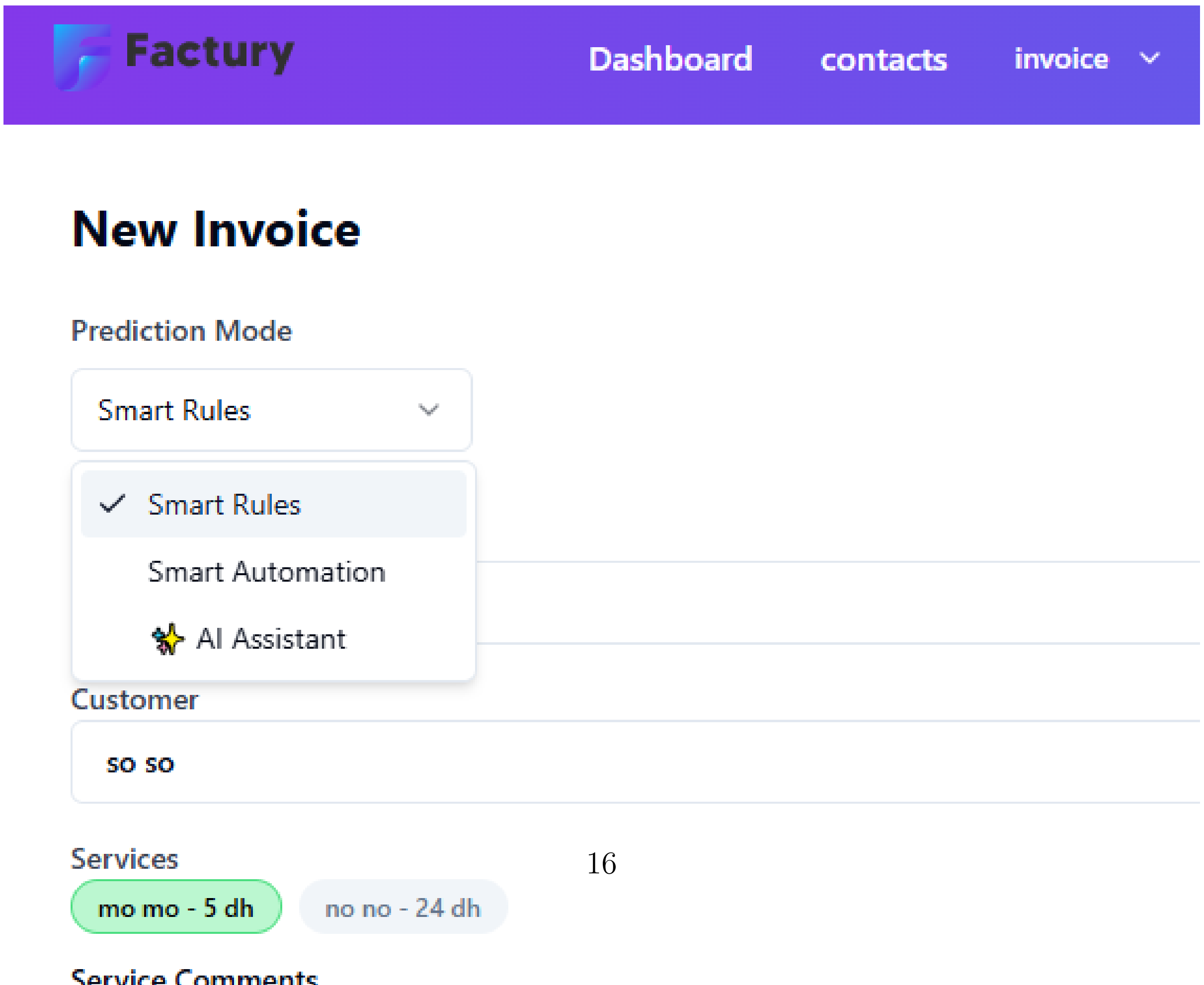
This ML-inspired approach demonstrates how core machine learning principles can be applied pragmatically without requiring extensive data science infrastructure or model training complexities.

8.5 Comparative Analysis of the Three Approaches

The comprehensive analysis reveals that the heuristic predictor offers robustness and speed but lacks intelligent ranking capabilities. The ML-inspired predictor provides a more nuanced, data-driven output without external dependencies, making it the superior choice for production environments where cost, speed, and determinism are critical considerations. The LLM approach, while most flexible, introduces operational complexity and cost that may not be justified for this specific use case, though it remains valuable for more complex prediction scenarios requiring deeper semantic understanding.

Table 1: Comparative Analysis of Prediction Approaches

Feature	LLM Approach	Heuristic Approach	ML-Inspired Approach
Conceptual Foundation	Generative AI	Deterministic Rules	Statistical Analysis
Flexibility & Adaptability	High	Low	Medium
Determinism & Predictability	Low	High	High
Performance Characteristics	Slower (API call)	**Fastest**	Fast
Operational Cost	Higher (per-call)	**Free**	Free
Data Requirements	Can work with limited data	Requires historical data	Requires historical data
Service Ranking Methodology	Contextual understanding	No ranking (all services)	**Frequency-based ranking**
Comment Selection Logic	Contextual summary	First occurrence	**Most recent occurrence**
Implementation Complexity	High	Low	Medium
Explainability	Low	High	Medium



New Invoice

Prediction Mode

Smart Rules

Customer

so so

Services

mo mo - 5 dh

no no - 24 dh

Service Comments

9 Agile Process in Action: Iterative Development and Supervision

The development of these sophisticated features served as a practical testament to the effectiveness of Agile methodology in managing complex software projects. The chatbot implementation, for instance, began as a simple proof-of-concept capable of answering basic questions through a minimal interface. Through structured weekly reviews and feedback cycles, it evolved substantially: my supervisor astutely suggested adding the memory system after the first demonstration revealed limitations in conversation continuity, and the command system was proposed in a subsequent session to dramatically increase its utility and integration depth.

Similarly, the prediction system commenced as a single heuristic method focused on basic functionality. Thoughtful discussions about its limitations in handling complex patterns and providing intelligent rankings led to the prototyping of both the LLM and ML-inspired approaches, creating an opportunity for empirical comparison and contrast of their respective merits in a real-world application context. This iterative refinement process exemplifies the core Agile principle of responding to change over following a rigid plan, while maintaining disciplined engineering practices.

The supervision model combined technical guidance with professional mentorship, fostering both skill development and critical thinking abilities. Regular code reviews not only improved code quality but also facilitated knowledge transfer about software architecture patterns, performance optimization techniques, and maintainability considerations.

10 Challenges Encountered and Solutions Deployed

The internship presented several significant technical challenges that required systematic problem-solving and implementation of robust solutions:

- **Challenge:** Securing the chatbot API endpoint to prevent unauthorized access and potential data breaches while maintaining performance and usability.
- **Solution:** Implemented comprehensive JWT verification on the server-side API route and enforced strict Row Level Security policies on the `chat_messages` table, creating a defense-in-depth security approach.
- **Challenge:** Structuring the prompt for the LLM predictor to consistently return valid, well-formed JSON across diverse input data patterns and edge cases.
- **Solution:** Conducted iterative testing and refinement of the prompt engineering approach, including explicit instructions, format specifications, and example outputs to guide the model toward consistent response patterns.
- **Challenge:** Handling edge cases in the prediction systems, particularly for contacts with minimal or no invoice history where statistical methods become unreliable.
- **Solution:** Implemented comprehensive fallback functions (`fallbackPrediction`) that return safe, reasonable default values when insufficient historical data exists for meaningful prediction.

- **Challenge:** Managing code conflicts and integration issues when working concurrently on different features within a collaborative development environment.
- **Solution:** Maintained strict adherence to the Git feature-branch workflow and established clear communication protocols with my supervisor during PR reviews to identify and resolve integration issues early.
- **Challenge:** Balancing the trade-offs between prediction sophistication, system performance, and implementation complexity across the three prediction approaches.
- **Solution:** Conducted systematic evaluation of each approach against multiple criteria including accuracy, performance, cost, and maintainability to inform the final architectural decision.

These challenges and their solutions provided valuable learning opportunities in practical software engineering problem-solving, emphasizing the importance of systematic thinking, testing methodologies, and architectural decision-making in professional development contexts.

11 Achievement of Internship Objectives

The internship successfully met and in several cases exceeded all its defined objectives, representing a comprehensive professional development experience. I achieved substantial proficiency in the targeted modern technology stack—Supabase, TypeScript, and secure development practices—as demonstrated by the successful deployment of production-ready features into the live application environment. I actively and effectively utilized Git and GitHub for collaborative development, mastering feature branch workflows, pull request processes, and code review participation.

Most significantly, by designing and building a complex application with multiple integrated AI features through reasoned architectural decisions and iterative coding practices—rather than relying on generative AI to write the code autonomously—I have substantially strengthened my independent software engineering capabilities and critical thinking skills. This approach aligns with emerging best practices in AI-assisted development, where engineers leverage AI as a tool while maintaining architectural control and understanding.

The experience also developed important professional competencies including time management, task estimation, technical communication, and quality assurance practices. The successful completion of both major tasks—the AI chatbot and the multi-approach prediction system—demonstrates comprehensive skill development across the full software development lifecycle from requirements analysis to deployment.

12 Conclusion and Personal Reflection

This intensive one-month internship at Masterlys proved to be an immensely rewarding and educational professional experience that provided realistic immersion into the world of industrial software development. Beyond acquiring specific technical skills in modern web technologies and AI integration, I developed a more profound understanding of how to think like a software engineer: systematically considering scalability, security, maintainability, and user experience from the initial design stages through implementation and deployment.

The unique opportunity to design, implement, and empirically compare different AI approaches for the prediction system was particularly valuable, providing practical insights into the real-world trade-offs involved in AI integration decisions. This experience highlighted that the most sophisticated AI approach is not always the most appropriate, and that factors like cost, performance, determinism, and maintainability must be carefully balanced based on specific application requirements and constraints.

The internship successfully bridged the gap between academic computer science education and professional software engineering practice, reinforcing theoretical concepts with practical implementation experience. I am confident that the technical skills, methodological approaches, and professional mindset developed during this immersion will prove foundational to my future career as a software engineer, particularly in the rapidly evolving field of AI-integrated applications.

13 Appendices & References

References

- Ricci, F., Rokach, L., & Shapira, B. (2011). *Recommender Systems Handbook*. Springer.
- Hyndman, R.J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.
- Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.
- Supabase Documentation: https://supabase.com/docs
- OpenAI API Documentation: https://platform.openai.com/docs
- OWASP Foundation. (2021). *OWASP Top Ten Web Application Security Risks*. https://owasp.org/www-project-top-ten/

Code Repositories

- Private company repositories (Access restricted)

List of Proposed Figures