Product Description Document

Collaborators: Shoval Benjer 319037404, Adir Amar 209017755, Alon Berkovich 208432625

1. Introduction

Purpose

This project aims to develop an AI-powered complaint management system to automate the processing of customer complaints for online delivery services. By leveraging advanced Natural Language Processing (NLP) and decision-making algorithms, the system will extract key details, apply business-specific compensation rules, and resolve issues efficiently with minimal human intervention.

Scope

The system will automate complaints submitted via app and email, streamlining information extraction, rule-based decision-making, and resolution processes. Manual complaint handling and chatbot-based interactions are explicitly out of scope.

Target Audience

- Delivery service providers (e.g., Wolt, 10bis) aiming to:
- Reduce complaint resolution times.
- Enhance customer satisfaction through streamlined automation.

Key Performance Indicators (KPIs):

- Reduce average resolution time by **40%**.
- Achieve a 90% first-time resolution rate.
- Increase customer satisfaction ratings by 20% within six months of deployment.

2. Functional Requirements

2.1 User Authentication and Authorization

- Implement a secure mechanism for user login and role-based permissions.
- Allow differentiation between users (e.g., customers, support agents) to control access to system features.

2.2 Complaint Submission

Provide a user-friendly interface for submitting complaints, supporting text, file uploads, and additional inputs like categories.

Ensure submitted data is processed and stored securely in the system.

2.3 Complaint Classification and Prioritization

Use a system to classify complaints into predefined categories and prioritize them based on urgency or severity indicators (e.g., sentiment analysis or predefined rules).

2.4 **Complaint Resolution Recommendation**

Develop an engine to recommend resolutions based on the type of complaint and organizational policies.

Allow manual override for exceptional cases.

2.5 **Complaint Tracking**

Provide a dashboard for users to track complaint status, with filters for categories, priorities, and timeframes.

2.6 **Notifications**

Enable automated notifications to inform users about updates on their complaints.

Notifications can be delivered via multiple channels (e.g., app, email).

2.7 **Proactive Issue Identification**

Implement analytics to detect recurring complaint patterns and suggest system-level improvements.

2.8 **Data Extraction and Analysis**

Use text processing techniques to extract relevant details (e.g., order numbers, customer names) from complaints to streamline resolution.

2.9 **Performance Metrics for Customer Satisfaction**

2.9.1 **Average Resolution Time**: Measure the time taken to resolve complaints.

2.9.2 **Customer Feedback Scores**: Collect and aggregate customer feedback to evaluate satisfaction levels post-resolution.

2.9.3 **Complaint Closure Rate**: Calculate the percentage of complaints resolved within specified timeframes.

2.9.4 **Recurring Complaint Trends**: Analyze patterns in recurring complaints to identify systemic issues.

2.9.5 **Sentiment Improvement**: Track shifts in sentiment from initial complaint submission to resolution, leveraging NLP techniques for sentiment analysis.

2.10 **Batch Complaint Processing**

Enable bulk upload and processing of complaints for businesses dealing with high complaint volumes, using standardized formats such as CSV or Excel.

2.11 **Role-Specific Dashboards**

Provide tailored dashboards for different roles (e.g., customers, support agents, administrators) to ensure relevant information and features are accessible.

2.12 **Real-Time Collaboration**

Allow support agents or teams to collaborate on complex or escalated complaints in real-time, ensuring quicker resolution.

2.13 **Compensation Tracking**

Implement a module to track compensations provided to customers, linking them to complaint details for financial accountability.

2.14 **Sentiment Trends and Analytics**

Provide visualization tools to monitor sentiment trends over time, helping businesses understand customer sentiment and act proactively.

2.15 **Data Export and API Access**

Enable users to export complaint data in formats like CSV or Excel for offline analysis.

Provide APIs for external systems to access and interact with the complaint management data.

3. Architectural Requirements

3.1 System Architecture

- Design the system using modular components, such as a complaint submission module, a resolution engine, and a notification system.
- Use a pattern that supports flexibility, scalability, and independent deployment of components.

3.2 Scalability

- Ensure the system can handle increased loads by enabling additional computing resources or distributing workloads across multiple servers.

3.3 Performance

- Optimize data processing to ensure fast response times for complaint handling and system interactions.

3.4 Reliability and Availability

- Implement mechanisms to recover from system failures, such as backups and redundant systems.
- Ensure minimal downtime through failover strategies and high availability setups.

3.5 Security

- Protect user data with encryption and secure communication protocols.
- Implement access controls to ensure users only access authorized features.

3.6 Data Architecture

- Store structured complaint data and related metadata in databases designed for performance and scalability.
- Ensure data consistency, integrity, and easy retrieval across components.

3.7 Integration

- Design interfaces for seamless integration with external systems (e.g., CRM, messaging platforms).

- Use standardized protocols for communication between components and external tools.

3.8 Monitoring and Logging

- Implement tools for tracking system health, performance metrics, and logging activities to aid in debugging and analysis.

4. Technical Requirements

4.1 Programming Languages and Frameworks

- Use widely supported programming languages and frameworks that align with team expertise and project goals.
- Ensure the chosen technologies allow for modular and scalable development.

4.2 Database Technology

- Implement databases suitable for storing both structured and unstructured data, with support for efficient querying and scalability.
- Maintain backups to ensure data durability and disaster recovery.

4.3 Frontend Technologies

- Build a responsive user interface that works across devices and browsers.
- Provide an intuitive and consistent experience for users.

4.4 Backend Technologies

- Use server-side frameworks to handle business logic, data processing, and API communication.
- Ensure APIs are well-documented and follow standard practices for reliability and maintainability.

4.5 Cloud Services

- Utilize cloud-based hosting and storage solutions to scale resources based on demand.
- Leverage tools for monitoring, automation, and cost-effective deployment.

4.6 Development Tools

- Use version control systems for collaborative development.
- Automate build, testing, and deployment processes for faster iterations.

4.7 Security Tools

- Regularly scan the system for vulnerabilities and ensure compliance with relevant security standards.

- Use encryption and secure access mechanisms to protect data.

4.8 Testing and Quality Assurance

- Perform unit, integration, and system testing to ensure components work as expected.
- Test the system under various conditions to identify performance bottlenecks and potential issues.

5. User Personas

- **Customer: Needs quick and accurate resolutions to complaints.**
- **Business Manager: Requires efficient complaint handling to improve customer satisfaction and reduce operational costs.**
- **Customer Support Agent: Aims to resolve complex complaints escalated by the system.**

Design Principles

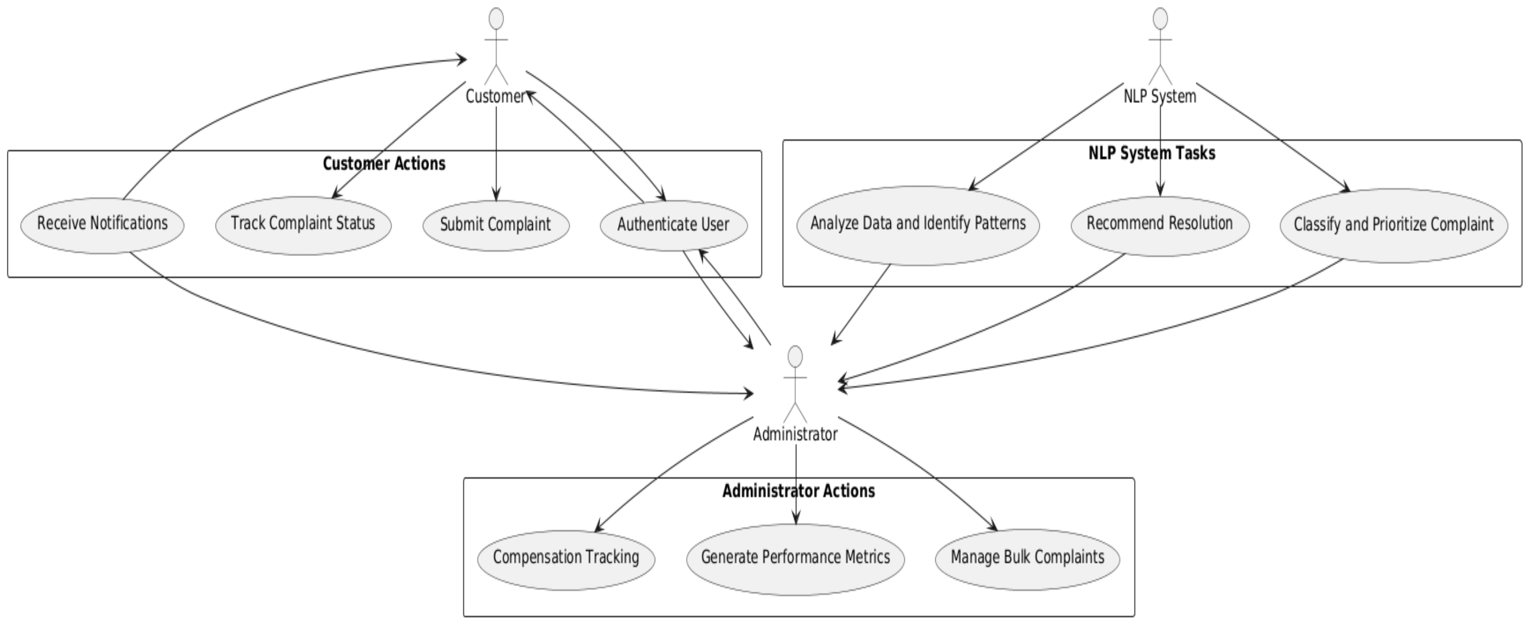
User-Centric: Provide an intuitive and responsive experience for users.

Scalability: Ensure the system can handle increasing complaint volumes seamlessly.

Efficiency: Minimize processing times to enhance customer satisfaction.

Proactivity: Identify and address recurring issues before escalation.

6. Use Case Diagram



7. Inputs and Outputs

Inputs:

- **User-Submitted Complaints**:
    - Free-text complaints.
    - Optional attachments (e.g., images, receipts).
    - Selected complaint categories (e.g., delivery issue, damaged item).
- **Predefined Business Rules**:
    - Organizational policies for resolution recommendations.
    - Severity and prioritization criteria.
- **NLP Models**:
    - Models for text classification, sentiment analysis, and entity recognition.
- **Batch Complaints**:
    - Bulk upload files in standardized formats (e.g., CSV, Excel).

Outputs:

- **Tailored Responses**:
    - Recommended resolution options delivered to users (e.g., refund, replacement).
    - Sent via app, email, or other notification channels.
- **Complaint Status Updates**:
    - Notifications to users about progress and resolution of their complaints.
- **Escalation Notifications**:
    - Alerts for unresolved or high-priority complaints sent to administrators.
- **Analytical Insights**:
    - Reports on recurring complaint patterns, resolution trends, and customer sentiment shifts.
- **Performance Metrics**:
    - Average resolution times, closure rates, and customer feedback scores.
- **Compensation Tracking Logs**:

- Detailed logs of compensations issued for complaints, linked to complaint data.

8. Non-Functional Requirements

- Performance: Process 90% of complaints within 2 seconds.
- Scalability: Handle up to 10,000 complaints daily.
- Security: Ensure GDPR-compliant data handling and storage.
- Supports hundreds of concurrent requests without degradation.

9. Frameworks

Backend Framework: Flask

Criteria	Flask	Django	Node.js
Modularity	Lightweight and highly modular	Full-stack, monolithic	Modular but less control than Flask
Ease of Integration	Simple RESTful API development	Built-in features for APIs	Requires additional libraries
Scalability	Effective for high workload	Scales well but heavier	Non-blocking, excellent for I/O
Developer Control	High customization	Limited due to built-in features	Moderate, depends on ecosystem

Final Choice: Flask

- Flask is lightweight and modular, making it ideal for RESTful APIs in the system's modular architecture.

Database: PostgreSQL

Criteria	PostgreSQL	MongoDB	MySQL
Structured Data Handling	Excellent for structured data	Lacks ACID compliance for transactions	Good, but less flexibility
Scalability	Supports horizontal scaling	Designed for scaling, good for unstructured data	Moderate scaling abilities
GDPR Compliance	Built-in tools for data encryption and auditing	Limited compared to PostgreSQL	Comparable to PostgreSQL
Advanced Querying	Strong support for complex queries	Weak in relational operations	Limited compared to PostgreSQL

Final Choice: PostgreSQL

- PostgreSQL supports structured data, GDPR compliance, and advanced analytics, essential for complaint processing and insights.

Cloud Platforms: AWS vs. Google Cloud

Criteria	AWS	Google Cloud	Azure (excluded)
NLP Tools	SageMaker for model training and deployment	Vertex AI for model training	Comparable, costlier
Scalability	Serverless architecture with Lambda	Similar serverless offerings	Excellent but expensive
GDPR Compliance	Strong compliance tools	Comparable compliance features	Comparable but costlier
Cost Efficiency	Pay-as-you-go; ~$0.20 per GB/month storage	Pay-as-you-go; ~$0.26 per GB/month storage	Generally higher costs

Final Choice: AWS

- AWS provides cost-effective, scalable services with strong NLP tools and GDPR compliance. **Alternate Option: Google Cloud**
- Google Cloud is viable with excellent AI tools but slightly higher storage and function execution costs.

Version Control and Collaboration: GitHub

Criteria	GitHub	GitLab	Bitbucket
Familiarity	✓ Team expertise	Less familiarity	Less familiarity
CI/CD Support	Built-in GitHub Actions	Comparable CI/CD pipelines	Comparable but less widely used
Centralization	Single platform for code, issues, and roadmap	Similar features	Similar features

Final Choice: GitHub

- GitHub is the team's familiar tool and supports seamless CI/CD workflows via GitHub Actions.
-

Final Technology Choices Summary

- **Backend Framework**: Flask – Lightweight, modular, and efficient for the project's RESTful API needs.
- **Database**: PostgreSQL – Ideal for structured data, GDPR compliance, and advanced querying.
- **Cloud Platform**: AWS (preferred), Google Cloud (alternative) – Cost-effective and scalable services with excellent AI tools.
- **Version Control and Collaboration**: GitHub – Familiar and integrates CI/CD seamlessly.

10. Regulatory Compliance and Transparency

10.1 Algorithmic Transparency:

- Conduct regular audits of AI algorithms to identify and mitigate biases.
- Implement explainable AI models to enhance user trust.
- Provide users with clear disclosures about how AI decisions are made.

10.2 Compliance with AI Regulations:

- Align with state-level laws such as the California AI Safety Act and Colorado's AI regulations.
- Develop a compliance checklist covering GDPR, HIPAA, and other relevant standards.
- Perform periodic assessments to ensure adherence to all applicable regulations.

10.3 Ethical AI Practices:

- Design algorithms that prioritize equity and fairness in customer complaint handling.
- Incorporate feedback loops to refine and improve decision-making processes.

10.4 Data Encryption:

- Use AES-256 encryption for data at rest.
- Implement TLS 1.3 for secure data transmission.
- Establish a centralized key management system to safeguard encryption keys.

10.4 Access Control:

- Deploy role-based access control (RBAC) to limit access to sensitive data.
- Enable multi-factor authentication (MFA) for all users accessing the platform.

10.4 Vulnerability Management:

- Conduct regular vulnerability scans using tools like Nessus and OpenVAS.
- Perform penetration testing with tools such as Metasploit to simulate real-world attack scenarios.

10.4 Data Handling:

- Ensure all data storage and processing align with GDPR and HIPAA standards.

- Develop automated data anonymization processes to minimize exposure of sensitive information.

10.5 Documentation Standards:

- Create comprehensive system documentation, including API workflows, architecture diagrams, and operational guidelines.
- Regularly update documentation to reflect system enhancements and regulatory changes.

10.5 Regular Updates:

- Schedule quarterly updates to integrate security patches and feature improvements.
- Monitor AI system performance to preemptively identify areas requiring optimization.

10.5 Scalability Planning:

- Design the platform to handle up to 10,000 complaints daily without degradation.
- Utilize cloud-native solutions like AWS Lambda for cost-effective scalability.

11. UI/UX Design

Wireframe Concept
A three-column layout with:

1. Complaint overview.
2. Suggested resolutions and actions.
3. Escalation or override options.

Design Principles

- Intuitive interface with clear navigation.
- Mobile and desktop responsiveness.
- Accessibility features for visually impaired users.

****Check Appendix (UI/UX Sketch) for more information.**

12. Testing and QA Plan

Testing Phases:

- Unit Testing: Ensure NLP accuracy (e.g., 95% extraction accuracy on key details).
- Integration Testing: Validate email and WhatsApp complaint intake.
- User Acceptance Testing (UAT): Conduct with pilot companies to gather feedback.

QA Measures:

- Automated regression tests post-deployment.
- Regular updates to improve NLP model accuracy and compliance.

13. Project Timeline and Milestones

Platform Architecture and Design

- Timeline: November 10, 2024 – December 22, 2024
- Objective: Define the technical foundation of the platform.

Develop Core Automated System

- Timeline: December 23, 2024 – March 15, 2025
- Objective: Build the core AI-driven system for automated interactions.

Client Customization Interface

- Timeline: February 16, 2025 – April 15, 2025
- Objective: Provide tools for clients to customize their platform.

User Interface (UI) Design

- Timeline: March 16, 2025 – May 10, 2025
- Objective: Create user-friendly interfaces for clients and end-users.

Integrations and Testing

- Timeline: April 1, 2025 – June 10, 2025
- Objective: Integrate platform with external tools and perform rigorous testing.

Deploy and Monitor

- Timeline: June 1, 2025 – June 30, 2025
- Objective: Launch the platform and ensure stable operations.

14. Risks and Assumptions

Risks:

- Incorrect data extraction may delay resolutions.
- Resistance to change as users may prefer manual workflows.

Mitigation Strategies:

- Regular training of NLP models with diverse datasets.
- Stakeholder workshops to demonstrate system value.

15. Appendices

1. Literature Review and Competitor Analysis.
2. Project Repository -
3. Project Roadmap
4. UI/UX Sketch