People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
M'hamed Bougara - Boumerdès University



Faculty of Sciences
Computer Science department

Domain : Computer Mathematics
Sector : Computer Science
Specialty : Computer Systems
N ° of the specialty authorization decree: Decree No. 872 of 07/26/2016

Final graduation project report in view of obtaining the academic license diploma

Topic

Intelligent IT ticket management platform with AI-based automatic classification

Study case : Optimum Telecom Algeria Djezzy

Carried out by :

AYACHE Melissa
BENDOU Maysa
BOUALOUACHE Anfal
NAIT SAIDI Lydia

Defended on: June 4, 2026 before the jury composed of:

Dr. KEDJAR Saàdia	: Examiner
Dr. HAMADOUCHE Samiya	: Supervisor
Mr. LAMARA Mohamed Messaoud	: External Supervisor

Academic Year: 2025/2026

Acknowledgements

First and foremost, we are grateful to God for his blessings, for granting us the strength, the will, and the perseverance to carry out this project under good conditions and to see it through to its completion.

We would like to express our sincere gratitude to Dr. HAMADOUCHE Samiya for being our supervisor, for her invaluable guidance, precious advice, patience, and continuous support throughout this project, as well as for her availability whenever we needed assistance and guidance.

Our sincere appreciation goes to Dr. KEDJAR Saàdia for agreeing to examine our final graduation project, and we are honored to have her as a member of the jury and deeply appreciate her time and consideration.

We are also grateful to all the teachers of the Computer Science department for their dedication and support throughout our academic journey.

Special recognition goes to the host company for welcoming us during this internship, and especially to Mr. LAMARA Mohamed Messaoud, our external supervisor, for his guidance, availability, valuable advice, and the knowledge he shared with us, which gave us the opportunity to gain meaningful professional experience.

We owe our deepest thanks to our parents for their encouragement, moral and financial support, and patience throughout this project and our academic journey.

Finally, our warmest thoughts go to all those who contributed, near or far, our friends and our loved ones, to the completion of this work.

Abstract

This project was carried out as part of our final graduation project to obtain the license degree in Computer Science. It consists of the design and development of an intelligent IT ticket management platform for Optimum Telecom Algeria (Djezzy). We started by studying the existing system, understanding its limitations, and identifying the expectations of the proposed solution. We then proceeded with the modeling of the solution using UML diagrams, followed by the development of an AI-based automatic classification module for ticket categorization, and finally the implementation of the platform.

Résumé

Ce projet a été réalisé dans le cadre de notre projet de fin d'études pour l'obtention du diplôme de licence en informatique. Il consiste en la conception et le développement d'une plateforme intelligente de gestion des tickets IT pour Optimum Telecom Algeria (Djezzy). Nous avons commencé par étudier le système existant, comprendre ses limites et identifier les attentes de la solution proposée. Nous avons ensuite procédé à la modélisation de la solution à l'aide de diagrammes UML, suivi du développement d'un module de classification automatique basé sur l'intelligence artificielle, et enfin l'implémentation de la plateforme.

ملخص

هذا المشروع أُنجز في إطار مشروع التخرج للحصول على شهادة الليسانس في علوم الإعلام الآلي. يتمحور حول تصميم وتطوير منصة ذكية لإدارة تذاكر تقنية المعلومات لشركة Optimum Telecom Algeria (Djezzy). بدأنا بدراسة النظام الحالي وفهم حدوده وتحديد متطلبات الحل المقترح، ثم انتقلنا إلى نمذجة الحل باستخدام مخططات UML، يليه تطوير وحدة التصنيف التلقائي المبنية على الذكاء الاصطناعي، وأخيرًا تنفيذ المنصة.

Table of contents

Table of contents . . . . . **i**
List of figures . . . . . **iii**
List of tables . . . . . **v**
List of abbreviations . . . . . **vi**
General introduction . . . . . **vii**
1 Study of the existing system **1**
1.1 Introduction 1
1.2 Presentation of Djezzy 1
1.3 Presentation of IT Department 1
1.4 Existing system study 2
1.4.1 Technical environment: Service Manager 9 (SM9) 2
1.4.2 Ticket management process 2
1.5 Project presentation 2
1.5.1 Problem statement 2
1.5.2 Proposed solution 3
1.6 Conclusion 3
2 Application modeling **4**
2.1 Introduction 4
2.2 Requirements expression 4
2.2.1 Use case diagram definition 4
2.2.2 Identification of actors and use cases 4
2.2.3 Use case diagram presentation 5
2.3 Requirements analysis 6
2.3.1 Sequence diagram and textual description definitions 6
2.3.2 Proposed sequence diagrams and textual descriptions 6
2.3.3 Activity diagram definition 13
2.3.4 Activity diagram of the ticket resolution process 14
2.4 Design 14
2.4.1 Class diagram definition 14
2.4.2 Data dictionary 15
2.4.3 Management rules 16
2.4.4 Class diagram presentation 17
2.4.5 From class diagram to relational model 17
2.5 Conclusion 18
3 Automatic ticket classification module **19**
3.1 Introduction 19
3.2 General process 19
3.3 Data collection 19
3.4 Data preprocessing 20
3.5 Models training 21
3.5.1 Data splitting 21
3.5.2 Models 21
3.5.3 Machine learning model 22
3.5.4 Deep learning models 23
3.6 Evaluation and metrics 24

3.7 Best model selection . . . 26
3.8 Translation model . . . 26
3.9 Conclusion . . . 26
4 Implementation . . . 27
4.1 Introduction . . . 27
4.2 Development environment . . . 27
4.2.1 Hardware environment . . . 27
4.2.2 Software environment . . . 27
4.3 Deployment diagram . . . 30
4.4 Application interfaces . . . 30
4.4.1 Ticket creation interface for employee . . . 30
4.4.2 Service tickets interface for technician . . . 32
4.4.3 Ticket assignment for manager . . . 32
4.4.4 Ticket details interface for technician . . . 33
4.4.5 Ticket details interface for employee . . . 34
4.4.6 Saving the solution and closing the ticket . . . 34
4.4.7 Alternative ticket handling actions for technician . . . 35
4.4.8 Performance tracking for manager . . . 36
4.4.9 User management for admin . . . 38
4.5 Conclusion . . . 39
General conclusion . . . viii
Bibliography . . . ix
Webography . . . x

List of figures

1.1 IT department organizational structure . . . 1
1.2 Current ticket management process . . . 2
1.3 Intelligent IT ticketing platform . . . 3

2.1 Use case diagram . . . 5
2.2 Sequence diagram for the Authenticate use case . . . 6
2.3 Sequence diagram for the Create ticket use case . . . 8
2.4 Sequence diagram for the Launch AI module . . . 10
2.5 Sequence diagram for the Assign ticket to technician use case . . . 12
2.6 Activity diagram for ticket resolution process . . . 14
2.7 Class diagram . . . 17

3.1 General process of the ticket classification module . . . 19
3.2 Categories distribution before and after preprocessing . . . 21
3.3 Example of the dataset after preprocessing . . . 21
3.4 SVM pipeline . . . 22
3.5 Deep learning pipeline (BiLSTM, BERT Base, DistilBERT) . . . 23
3.6 Model performance comparison on the test set . . . 25

4.1 JavaScript . . . 27
4.2 Python . . . 27
4.3 HTML . . . 28
4.4 CSS . . . 28
4.5 SQL . . . 28
4.6 React.js . . . 28
4.7 Tailwind CSS . . . 28
4.8 Node.js . . . 28
4.9 Express.js . . . 28
4.10 Prisma . . . 29
4.11 FastAPI . . . 29
4.12 HuggingFace Transformers . . . 29
4.13 PyTorch . . . 29
4.14 PostgreSQL . . . 29
4.15 Visual Studio Code . . . 29
4.16 Git / GitHub . . . 30
4.17 Google Colab . . . 30
4.18 Deployment diagram . . . 30
4.19 Ticket creation interface . . . 31
4.20 Ticket submission success form . . . 31
4.21 Service tickets interface . . . 32
4.22 Ticket assignment interface . . . 32
4.23 Technician selection list . . . 33
4.24 Technician communication and ticket action interface . . . 33
4.25 Employee ticket details interface . . . 34
4.26 Technician close tab after employee confirmation . . . 34
4.27 Closed ticket with final solution . . . 35
4.28 Take charge interface . . . 35
4.29 Ticket redirection form . . . 36
4.30 Ticket closing action . . . 36
4.31 Manual ticket closing form . . . 36
4.32 Performance tracking summary cards . . . 36
4.33 Performance tracking team efficiency . . . 37

4.34 Performance tracking ticket distribution 37
4.35 Performance tracking trends and priority distribution 38
4.36 User management interface 38
4.37 Add new user 39
4.38 View and update user 39

List of tables

2.1 Actors definitions and use cases 4
2.2 Textual description for the Authenticate use case 7
2.3 Textual description for the Create ticket use case 9
2.4 Textual description for the Launch AI module use case 11
2.5 Textual description for the Assign ticket to technician use case 13
2.6 Data dictionary 16

3.1 Dataset structure 20
3.2 Hyperparameters of the SVM (LinearSVC) model 22
3.3 Hyperparameters of the BiLSTM model 23
3.4 Hyperparameters of the BERT Base model 24
3.5 Differences between BERT Base and DistilBERT 24
3.6 Classification metrics for evaluation 25
3.7 Overall comparison of model performance on the test set 25

4.1 Hardware environment 27

List of abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory
DNS	Domain Name System
FK	Foreign Key
GPU	Graphics Processing Unit
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IT	Information Technology
ITSM	IT Service Management
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
ORM	Object-Relational Mapping
OTA	Optimum Telecom Algeria
PDF	Portable Document Format
PK	Primary Key
REST	Representational State Transfer
SLA	Service Level Agreement
SQL	Structured Query Language
SVM	Support Vector Machine
TF-IDF	Term Frequency–Inverse Document Frequency
UML	Unified Modeling Language
VPN	Virtual Private Network

General introduction

In any enterprise, Information Technology Service Management (ITSM) practices are essential to ensure system availability, support employees, resolve technical issues, enhance security, and maintain operational efficiency. As organizations grow, the number of information technology incidents and user requests increases significantly.

To manage this, companies use IT ticketing systems that centralize requests and track issues. The Service Desk is responsible for receiving, analyzing, and routing tickets to the appropriate teams. With increasing workload, manual processing becomes slow and difficult to scale.

This is the case at Optimum Telecom Algeria (Djezzy), where ticket handling is mainly manual. The Service Desk manages the routing process, leading to delays in assignment, increased workload, and reduced service quality. Resolution times are also slowed down by IT technicians' reliance on external tools during the resolution process within the current ticketing system.

As part of our graduation project, the objective is to address the limitations of the current system. Our solution is an intelligent IT ticket management platform incorporating an AI-based automatic classification module. This application automates ticket categorization and routing to the appropriate IT services, enabling the management of IT tickets between employees and IT teams. It reduces resolution time, improves communication between users and IT teams, and ensures full traceability within a unified platform.

Our work is organized into four chapters:

- The first chapter presents the company of our internship, its IT department, the study of the existing system, its limitations, and our proposed solution along with its objectives.
- The second chapter presents the modeling of our platform through the expression and analysis of requirements, as well as the design of our application using UML diagrams.
- The third chapter describes the automatic ticket classification module, with a general process, steps for realization, and a detailed explanation of each step.
- The fourth chapter presents the implementation of the platform, including the hardware and software environment, the deployment diagram, and the presentation of the main interfaces.

Chapter 1
Study of the existing system

1.1 Introduction

In this chapter, we present Optimum Telecom Algeria Djezzy and its IT department, where our internship took place. We then describe the existing system, its problems, and our proposed solution.

1.2 Presentation of Djezzy

Djezzy, founded in July 2001, is a leading mobile telecommunications operator in Algeria, with more than 15 million subscribers as of September 2023. The company offers a wide range of services including prepaid, postpaid, Data, value-added services, and SUT. Since July 2022, Djezzy has become a Public Economic Enterprise and is now 96.57% owned by the FNI-National Investment Fund. The company aims to strengthen connectivity, support the digitalization of Algeria, and expand its commercial network across the country [13].

1.3 Presentation of IT Department

Djezzy's Information Technology (IT) department serves as the company's strategic core, developing and supporting information systems, providing user assistance, and ensuring information security. Its main challenge is balancing technological potential with the innovation needs of OTA. The department also provides shared IT and audiovisual resources while fostering an environment that links business opportunities with recent research and academic developments [1].

The figure 1.1 presents the IT department organizational structure:

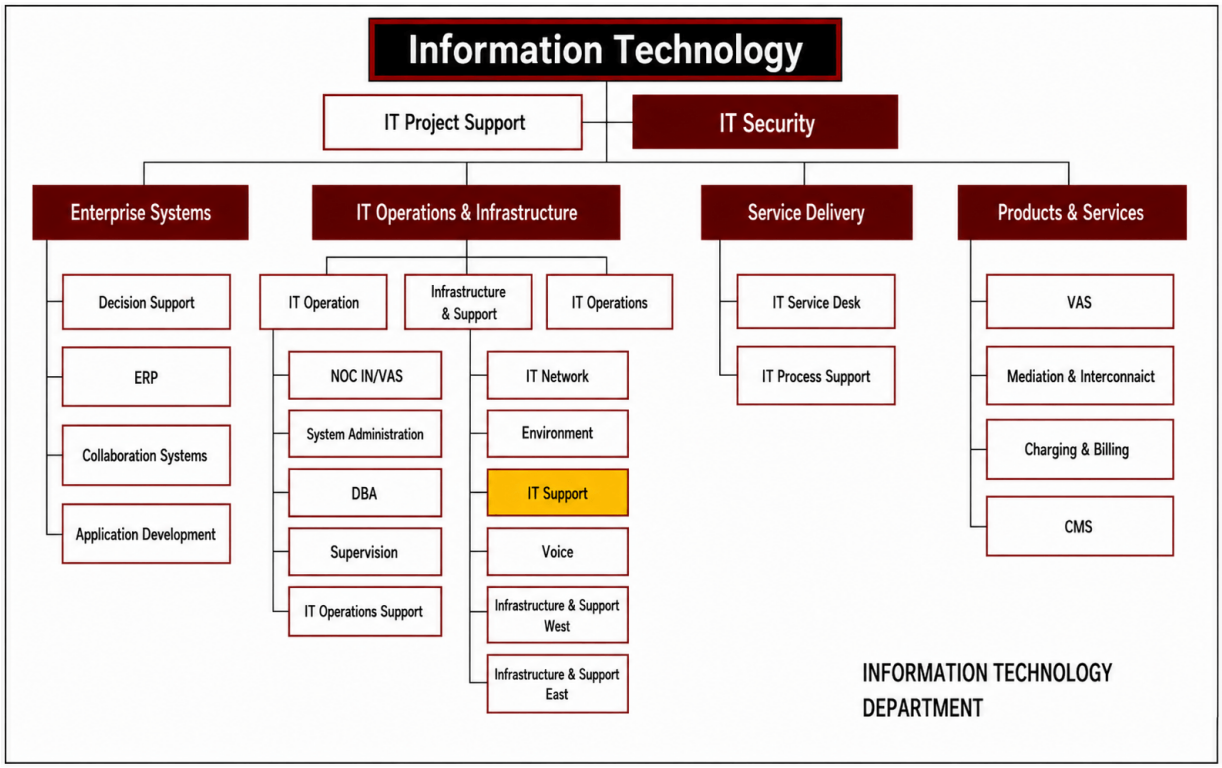


Figure 1.1: IT department organizational structure

1.4 Existing system study

1.4.1 Technical environment: Service Manager 9 (SM9)

The IT service management system used at the IT department of Djezzy is Service Manager 9 (SM9). This platform centralizes the management of incidents and user requests, which are referred to as tickets, ensuring ticket tracking, intervention traceability, and controlled workflow management across IT processes. SM9 includes several modules such as incident management, request management, change management, and a knowledge base.

1.4.2 Ticket management process

In the current platform, ticket processing follows a mainly manual workflow:

1. Employees create tickets via SM9 by providing details such as issue description, attachments, impact level, and urgency.
2. The Service Desk reviews and analyzes the ticket, corrects the provided information, classifies it by category, determines the expected resolution time, known as the Service Level Agreement (SLA), and assigns it to the appropriate IT team.
3. Once assigned, the technician processes the ticket and may contact employees through external tools (email, WhatsApp, or phone calls) if additional information is required. The ticket is resolved, and the resolution or any relevant solution steps or guidance are communicated to the employee when applicable.
4. The employee confirms the resolution. The ticket is closed by the technician and the resolution is documented.

The figure 1.2 presents the current ticket management process:

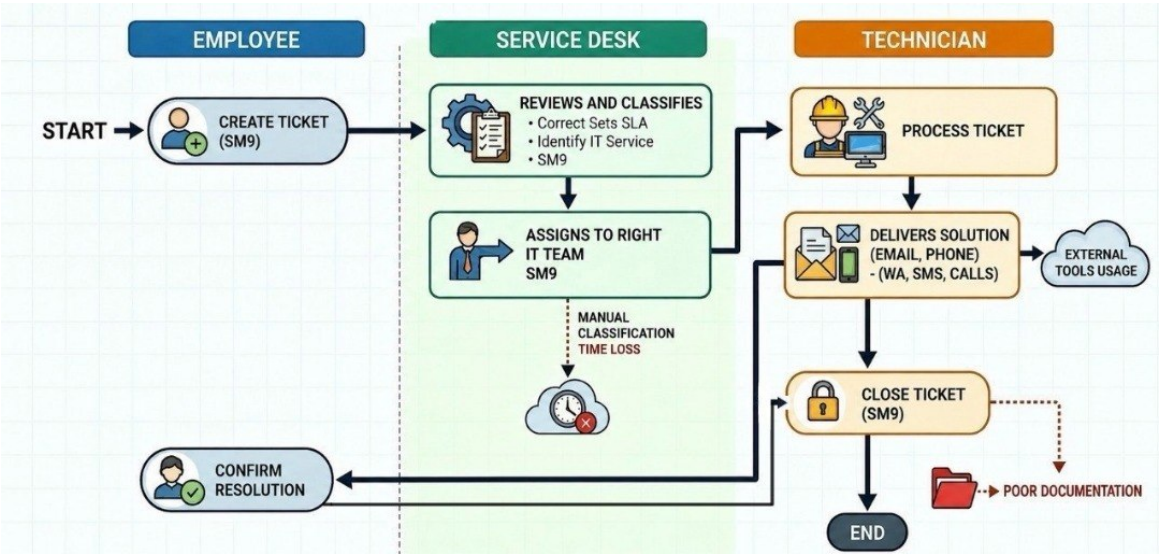


Figure 1.2: Current ticket management process

1.5 Project presentation

1.5.1 Problem statement

The current ticket management platform presents several limitations:

- Manual ticket analysis and assignment by the Service Desk lead to time loss, slower processing, and delays in identifying the appropriate service and assigning tickets to technicians. The platform lacks intelligent automation for ticket classification and service identification.
- Employee information retrieval through Outlook, as well as communication and interaction with

employees, is not integrated into the platform. This requires the use of external tools such as phone calls, WhatsApp, and SMS.
- Although SM9 allows documentation, its text-based interface limits the documentation of resolution steps, such as adding attachments when closing tickets.

1.5.2 Proposed solution

IT ticketing systems represent one of the tools used to implement ITSM processes in practice. Due to the limitations of the existing platform and in order to improve the quality of IT services, our internship project aims to develop a new IT ticketing platform that incorporates automation based on artificial intelligence for ticket classification and service identification, as well as improved ticket management. This platform is designed to manage employees' requests and incidents in a more efficient way. It enables automatic categorization and routing of tickets to the appropriate IT teams, reducing the manual work performed by the Service Desk.

The proposed solution has the following objectives:

- Automate ticket classification by category and automatically determine the appropriate service for routing based on the identified category within the platform, reducing assignment time and improving resolution speed and employee satisfaction.
- Improve IT services quality and response time.
- Optimize workflow through automatic ticket routing and integrated communication between employees and technicians, reducing reliance on external tools such as WhatsApp, email, and phone calls, and improving system centralization and information availability for efficient interventions.
- Provide reliable traceability through a complete history of actions and interventions, enhance documentation and decision-making statistics for continuous improvement.

The figure 1.3 presents the general schema of our solution:

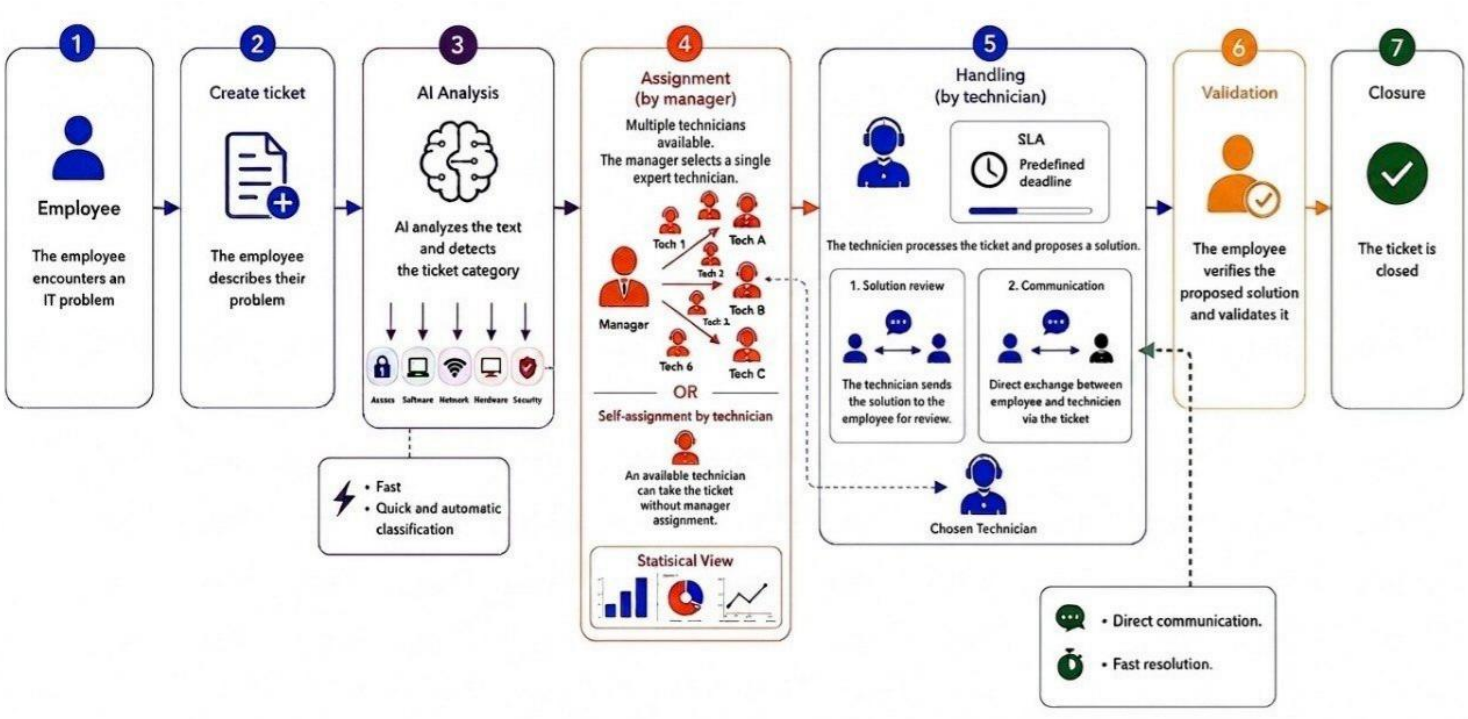


Figure 1.3: Intelligent IT ticketing platform

1.6 Conclusion

In this chapter, we presented a general presentation of the company and the existing system that led to our project and solution. The next chapter focuses on the modeling of the proposed solution.

Chapter 2
Application modeling

2.1 Introduction

Application modeling represents a core phase of our project. It begins with the expression of needs and requirements analysis, then continues with the design phase, and finally concludes with the conversion of the class diagram into a relational model. We used the Unified Modeling Language (UML), through use case, sequence, activity, and class diagrams, as a standardized and widely accepted approach for modeling application requirements and design while improving communication between users and developers.

2.2 Requirements expression

2.2.1 Use case diagram definition

The use case diagram allows the collection, analysis, and organization of requirements. It identifies the main functionalities of a system while separating it from the external environment.

A use case represents a specific way of using the system, delivering a complete service from start to finish initiated by an actor, while an actor models any external entity that interacts with the system, whether a human user, software, or another system [2].

2.2.2 Identification of actors and use cases

After studying the existing system, we identified five actors and their use cases.

Table 2.1 presents each actor with its definition and associated use cases:

Actor	**Definition and use cases**
Employee	A user with limited access rights can interact with the platform and: • Manage own tickets. • Consult profile.
Technician	A user with limited access rights can interact with the platform and: • Handle a ticket. • Consult IT service tickets. • Consult profile.
Manager	A user with limited access rights can interact with the platform and: • Consult IT service tickets. • Assign ticket to technician. • View IT service statistics. • Consult profile.
Director	A user with limited access rights can interact with the platform and: • View IT department statistics. • Consult profile.
Admin	A user with management and database control rights can: • Modify service level agreement duration. • Manage users. • Consult profile.

Table 2.1: Actors definitions and use cases

2.2.3 Use case diagram presentation

The figure 2.1 presents the use case diagram of our platform:

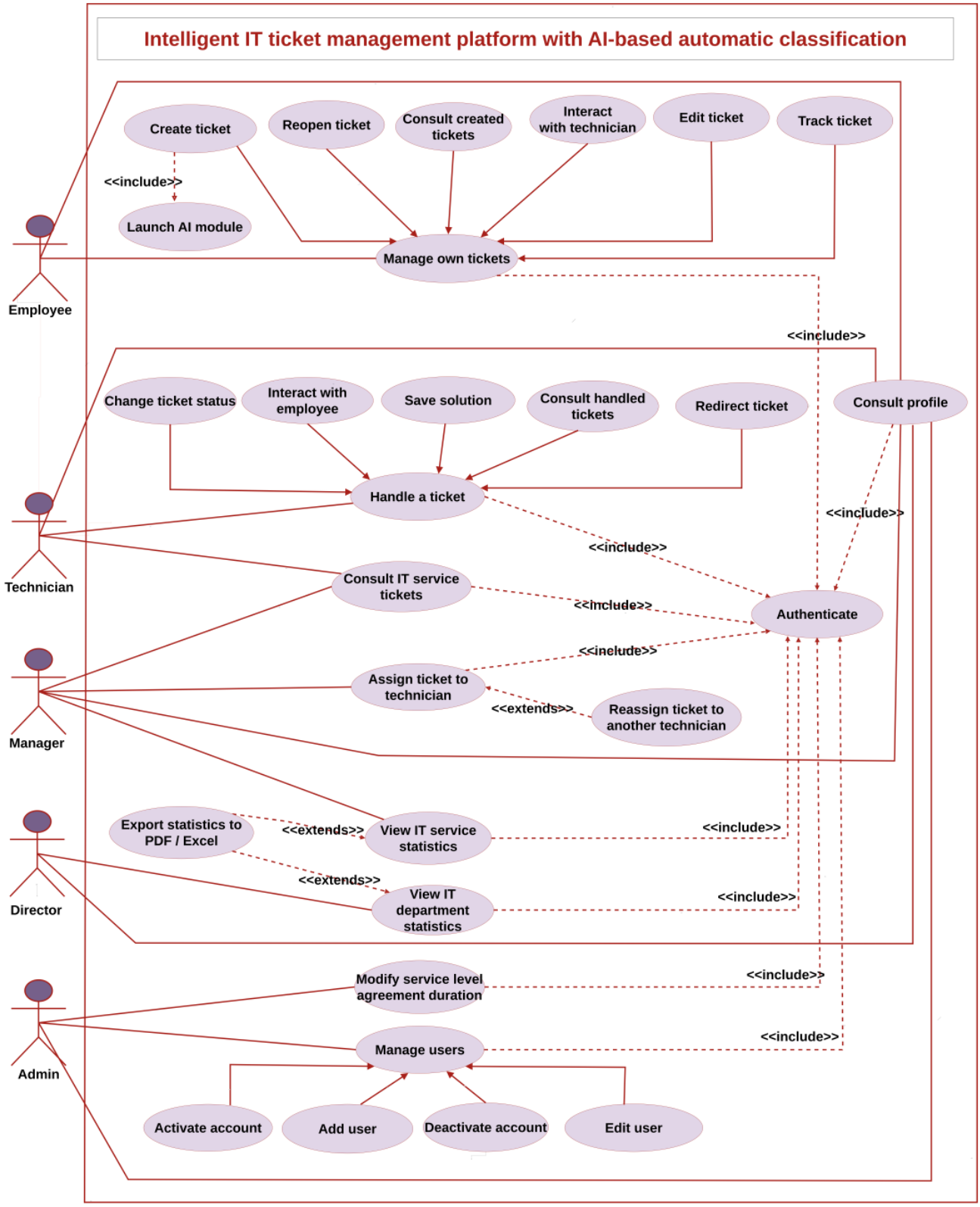


Figure 2.1: Use case diagram

2.3 Requirements analysis

2.3.1 Sequence diagram and textual description definitions

A sequence diagram shows the interactions between system elements from a temporal perspective, representing the chronological sequencing of messages exchanged between lifelines [2]. It is used to describe specific scenarios of a use case, with the main actor on the left, the system in the middle, and any secondary actors on the right [3].

The textual description of a use case is an essential complement to the diagram, as it helps communicate easily with users and agree on the business terminology employed [3].

2.3.2 Proposed sequence diagrams and textual descriptions

This section presents the sequence diagrams and textual descriptions of the use cases we selected:

- **Sequence diagram for the Authenticate use case**

  Figure 2.2 presents the sequence diagram for the Authenticate use case for all platform users:

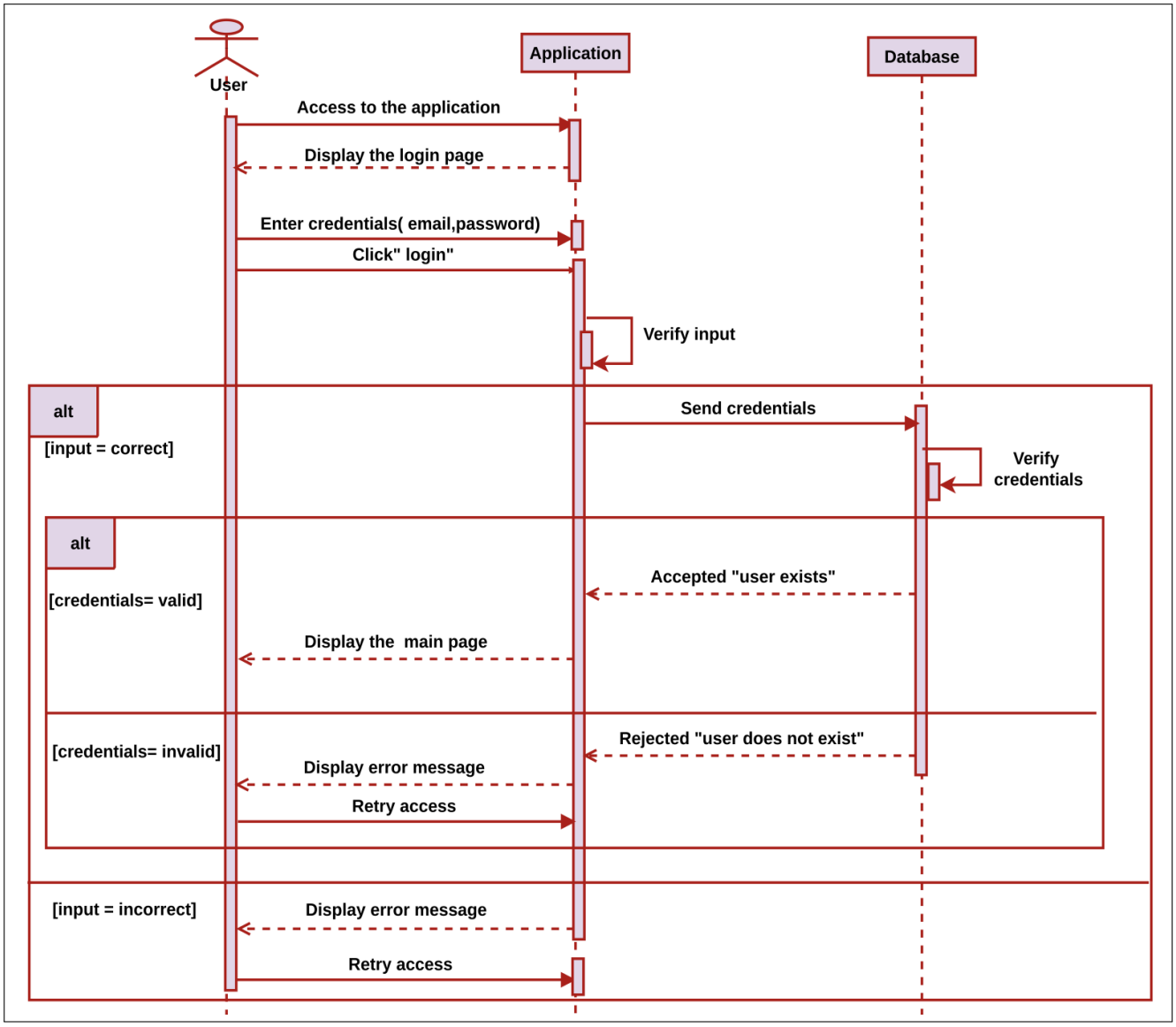


Figure 2.2: Sequence diagram for the Authenticate use case

Table 2.2 presents the textual description corresponding to the sequence diagram of the Authenticate use case:

Use case name	Authenticate
Goal	To allow any user (Employee, Technician, Manager, Director, Admin) to securely log into the system using their credentials.
Actors	All users (Employee / Technician / Manager / Director / Admin).
Pre-conditions	• The application is accessible. • Network connection is available.
Nominal flow (Main success scenario)	
1. The user accesses the application. 2. The application displays the login page. 3. The user enters their credentials (email, password). 4. The user clicks “Login”. 5. The application verifies the input format. 6. The application sends the credentials to the database. 7. The database verifies the credentials. 8. The database returns Accepted “user exists”. 9. The application displays the main page.	
Alternative flows	
Incorrect Input Format	At Step 5: if the input format is incorrect, the application displays an error message and asks the user to retry access without querying the database.
Invalid Credentials	At Step 7: if the credentials are invalid, the database returns Rejected “user does not exist”. The application displays an error message, and the user retries access.
Post-condition	
Final State	The user is authenticated and redirected to the main page according to their role.

Table 2.2: Textual description for the Authenticate use case

- **Sequence diagram for the Create ticket use case**

  Figure 2.3 presents the Create ticket sequence diagram performed by the employee:

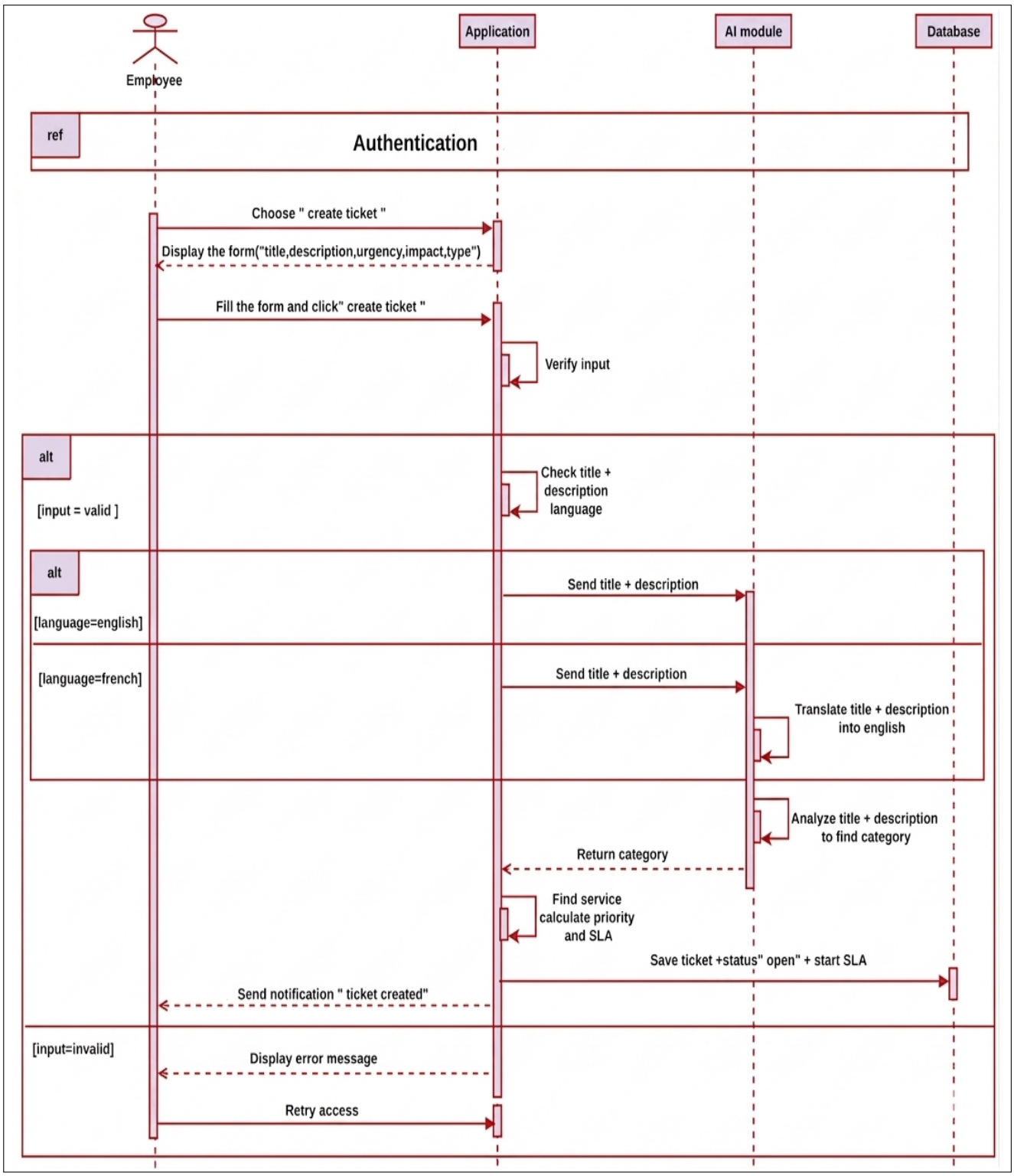


Figure 2.3: Sequence diagram for the Create ticket use case

Table 2.3 presents the textual description corresponding to the sequence diagram of the Create ticket use case:

Use case name	Create ticket
Goal	To allow an employee to submit a request which is categorized by AI to determine the appropriate service.
Actors	Employee.
Pre-conditions	• Employee is authenticated. • Network connection is available.
Nominal flow (Main success scenario)	
1. The employee chooses “create ticket” from the menu. 2. The application displays the form (title, description, urgency, impact, type). 3. The employee fills the form and clicks “create ticket”. 4. The application verifies the input format. 5. The application checks the title and the description language. 6. The application sends the title and the description to the AI module. 7. The AI module analyzes the title and the description to find the category. 8. The AI module returns the category to the Application. 9. The application identifies the service matching that category and calculates the Priority and SLA. 10. The application saves the ticket with status “open” in the database and starts the SLA timer. 11. The application sends a notification “ticket created” to the employee.	
Alternative flows	
Input Error	At Step 4: If data is invalid, the application displays an error message and the employee retries.
Language translation	At Step 5: if the title and description language is French, the AI module first translates them into English.
Post-condition	
Final State	The ticket is categorized, assigned to a service, and stored with its calculated SLA and priority.

Table 2.3: Textual description for the Create ticket use case

- **Sequence diagram for the Launch AI module use case**
  Figure 2.4 presents the sequence diagram for the Launch AI module use case:

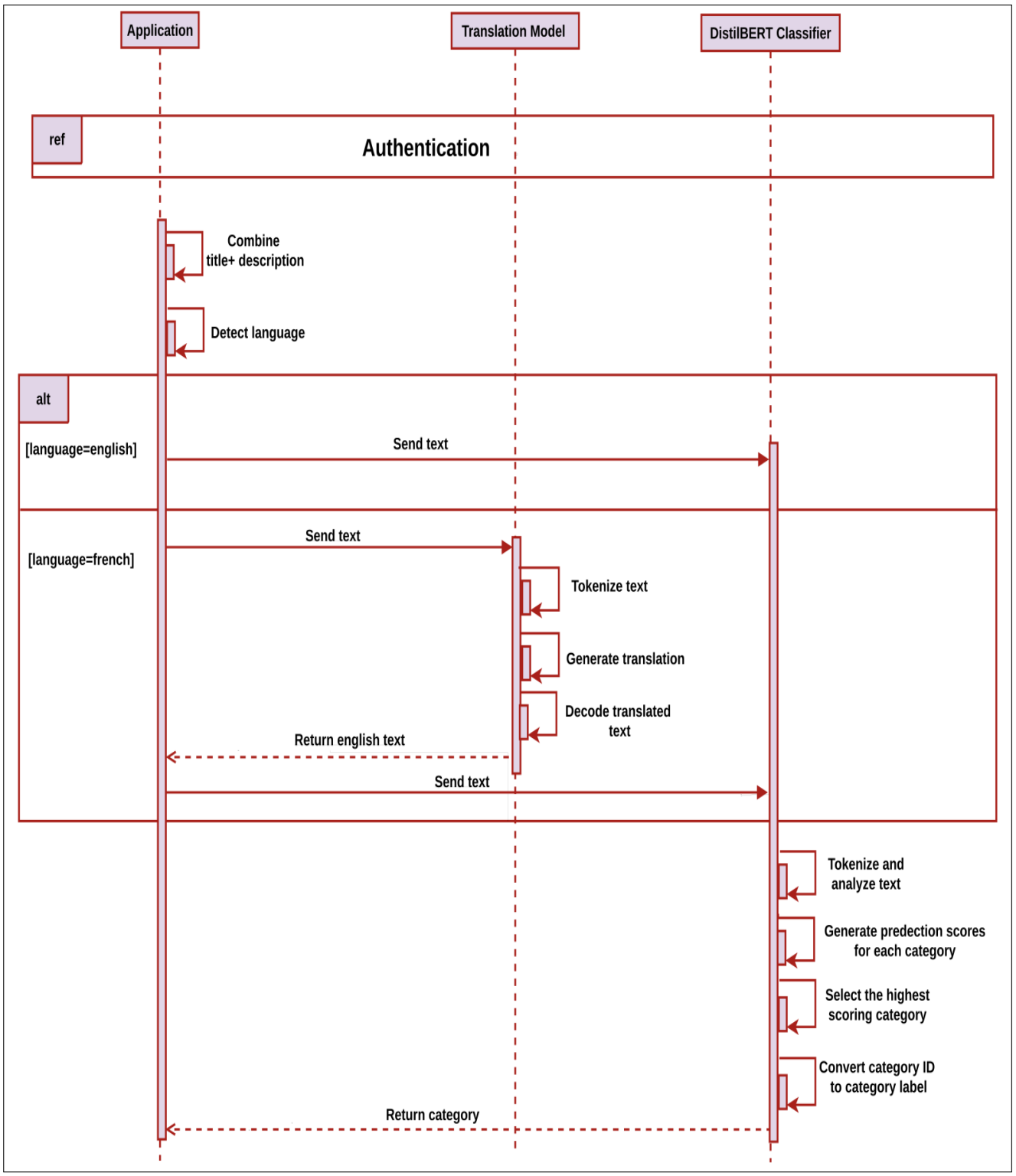


Figure 2.4: Sequence diagram for the Launch AI module

Table 2.4 presents the textual description corresponding to the sequence diagram of the Launch AI module use case:

Use case name	Launch AI module
Goal	To automatically detect the language and classify a ticket using DistilBERT Classifier, with translation only if the ticket is in French.
Actors	Application, Translation Model, DistilBERT Classifier.
Pre-conditions	• The user is authenticated. • A ticket (title + description) has been submitted.
Nominal flow (Main success scenario)	
1. The application combines the ticket title and description into a single text. 2. The application detects the language of the text. 3. The application sends the text to the DistilBERT Classifier. 4. The DistilBERT Classifier tokenizes and analyzes the text. 5. The DistilBERT Classifier generates prediction scores for each category. 6. The DistilBERT Classifier selects the highest scoring category. 7. The DistilBERT Classifier converts the category ID to a category label. 8. The DistilBERT Classifier returns the category to the application.	
Alternative flows	
Language = French	At Step 2: If the detected language is French, the application sends the text to the Translation Model. The Translation Model tokenizes the input text, generates the English translation, and decodes the generated tokens into readable text. The Translation Model then returns the translated English text to the application. The application sends the translated text to the DistilBERT Classifier.
Post-condition	
Final State	The ticket is classified and a category is returned to the application to identify the appropriate service.

Table 2.4: Textual description for the Launch AI module use case

- **Sequence diagram for the Assign ticket to technician use case**

  Figure 2.5 presents the sequence diagram for the ticket assignment use case:

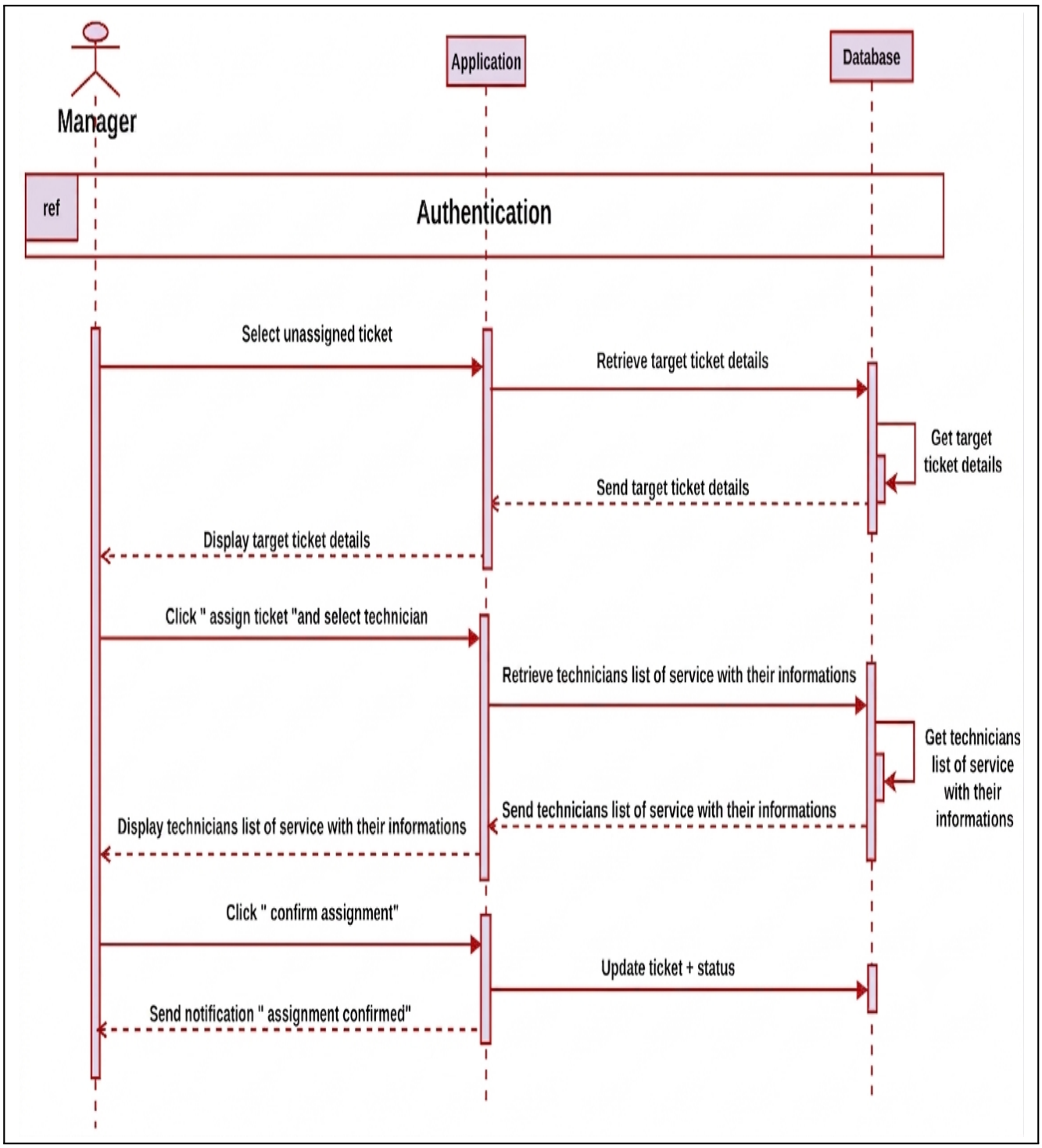


Figure 2.5: Sequence diagram for the Assign ticket to technician use case

Table 2.5 presents the textual description corresponding to the Assign ticket to technician sequence diagram:

Use case name	Assign ticket to technician
Goal	To allow a Manager to assign an unassigned ticket to a specific technician within the appropriate service.
Actors	Manager.
Pre-conditions	• Manager is authenticated. • At least one unassigned ticket exists in the system.
Nominal flow (Main success scenario)	
1. The manager selects an unassigned ticket. **2.** The application retrieves the target ticket details from the database. **3.** The database sends the target ticket details to the application. **4.** The application displays the target ticket details to the manager. **5.** The manager clicks “assign ticket” and selects a technician. **6.** The application retrieves the service’s technicians list with their information from the database. **7.** The database sends the technicians list to the application with their information. **8.** The application displays the technicians list with their information to the manager. **9.** The manager clicks “confirm assignment”. **10.** The application updates the ticket status in the database. **11.** The application sends an “assignment confirmed” notification to the manager.	
Alternative flows	
—	
Post-condition	
Final State	The ticket is assigned to the selected technician, its status is updated in the database, and the manager is notified.

Table 2.5: Textual description for the Assign ticket to technician use case

2.3.3 Activity diagram definition

An activity diagram is used to specify sequential processes. It is a directed graph of actions and transitions, where transitions are triggered at the end of actions, and steps can be executed in parallel or in sequence. It is particularly suited for describing the detailed flow of a use case [2, 3].

2.3.4 Activity diagram of the ticket resolution process

Figure 2.6 presents the activity diagram for the ticket resolution process involving the Employee, Application, AI Module, Technician, and Manager:

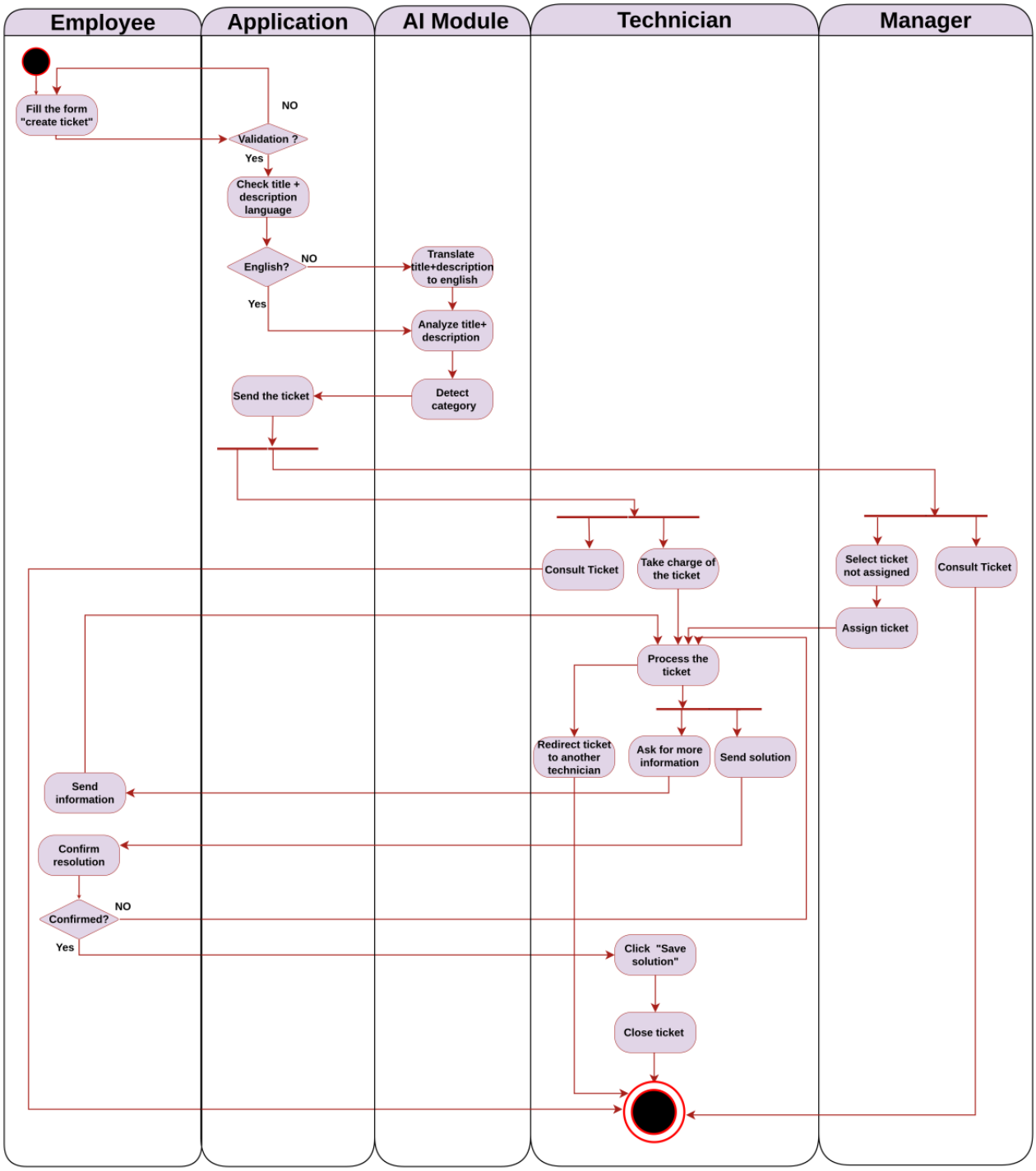


Figure 2.6: Activity diagram for ticket resolution process

2.4 Design

2.4.1 Class diagram definition

The class diagram is considered the most important diagram in object-oriented modeling. It describes the static structure of a system by presenting its classes, their attributes, their operations, and the

relationships between them. A class is a description of a set of objects sharing the same semantics, attributes, methods, and relationships [2].

2.4.2 Data dictionary

Table 2.6 presents the data dictionary of our application:

Class	**Attributes**				**Methods**
	Name	**Description**	**Type**	**Size**	
User	id	User identifier	int	–	login() logout() viewProfile() activateAccount() deactivateAccount() createUser() updateUser()
	name	First name	varchar	255	
	surname	Last name	varchar	255	
	email	Email address	varchar	255	
	password	Hashed password	varchar	255	
	role	User role	enum	–	
	phone	Phone number	varchar	20	
	department	Department name	varchar	255	
	job_title	Job title	varchar	255	
	office	Office location	varchar	255	
	block_number	Building block	varchar	50	
	is_active	Active status	boolean	–	
	created_at	Creation timestamp	timestamp	–	
Employee	id_emp	Employee ID	int	11	
Manager	id_manager	Manager ID	int	11	
	service	Supervised Service	enum	20	
Director	id_Director	Director ID	int	11	
Admin	id_admin	Admin ID	int	11	
Technician	id_tech	Technician ID	int	11	
	service	Assigned Service	enum	20	
Ticket	id	Unique Ticket ID	int	11	createTicket() updateTicket() changeStatus() getDetails() assignTicket() reassignTicket() redirectTicket() calculateDeadline() viewTickets() editSla()
	type	Problem or Request	enum	20	
	status	Current Status	enum	–	
	impact	Impact Level	enum	10	
	urgency	Urgency Level	enum	10	
	priority	Priority Level	enum	10	
	category	Ticket Category	enum	20	
	createdAt	Creation date	date	–	
	updateAt	Last update	date	–	
	resolvedAt	Resolution date	date	–	

Class	Attributes				Method
	Name	**Description**	**Type**	**Size**	
	closedAt	Closing date	date	–	
	reopeningCount	Count of reopens	int	5	requestConfirmation()
	sla	SLA level	enum	10	reopenTicket()
	service	Service type	enum	20	consultStatistics()
	solution	Solution text	text	–	exportStatistics()
	confirmation_request	Confirm flag	boolean	1	filterTicketsByCategory()
	sla_date_time	SLA deadline	timestamp	–	filterTicketsByPriority()
					filterTicketsByTechnician()
					filterTicketsByStatus()
					filterTicketsByService()
Message	id	Message ID	int	11	addComment()
	message	Message content	text	–	deleteComment()
	created_at	Creation timestamp	timestamp	–	sendSolution()
	message_type	Message type	enum	20	saveSolution()
					confirmSolution()
Ticket attachment	id	Attachment ID	int	11	addAttachment()
	file_name	File name	varchar	255	uploadFile()
	file_path	Storage path	text	–	deleteFile()
	uploaded_at	Upload date	date	–	
Ticket history	action	Action performed	varchar	100	
	reason	Description	text	–	viewHistory()
	created_at	Creation timestamp	timestamp	–	getAssignHistory()
	notification	Notification	varchar	255	sendNotification()

Table 2.6: Data dictionary

2.4.3 Management rules

The application follows the rules:

- A user can be an Admin, Employee, Technician, Manager, or Director.
- The Admin manages user accounts and updates the SLA duration.
- An employee can create multiple tickets (problem or service request), but a ticket belongs to exactly one employee.
- A Manager can assign multiple tickets to technicians, but a ticket can be assigned by one manager.
- A Technician can manage multiple tickets within their service, but a ticket can be redirected to another technician.
- Every action performed on a ticket is recorded in the ticket history.
- A ticket can have zero or more attachments, but each attachment belongs to exactly one ticket.
- A ticket can have multiple messages (comments or solutions), but each message belongs to one ticket.

2.4.4 Class diagram presentation

The figure 2.7 represents the class diagram of our application:

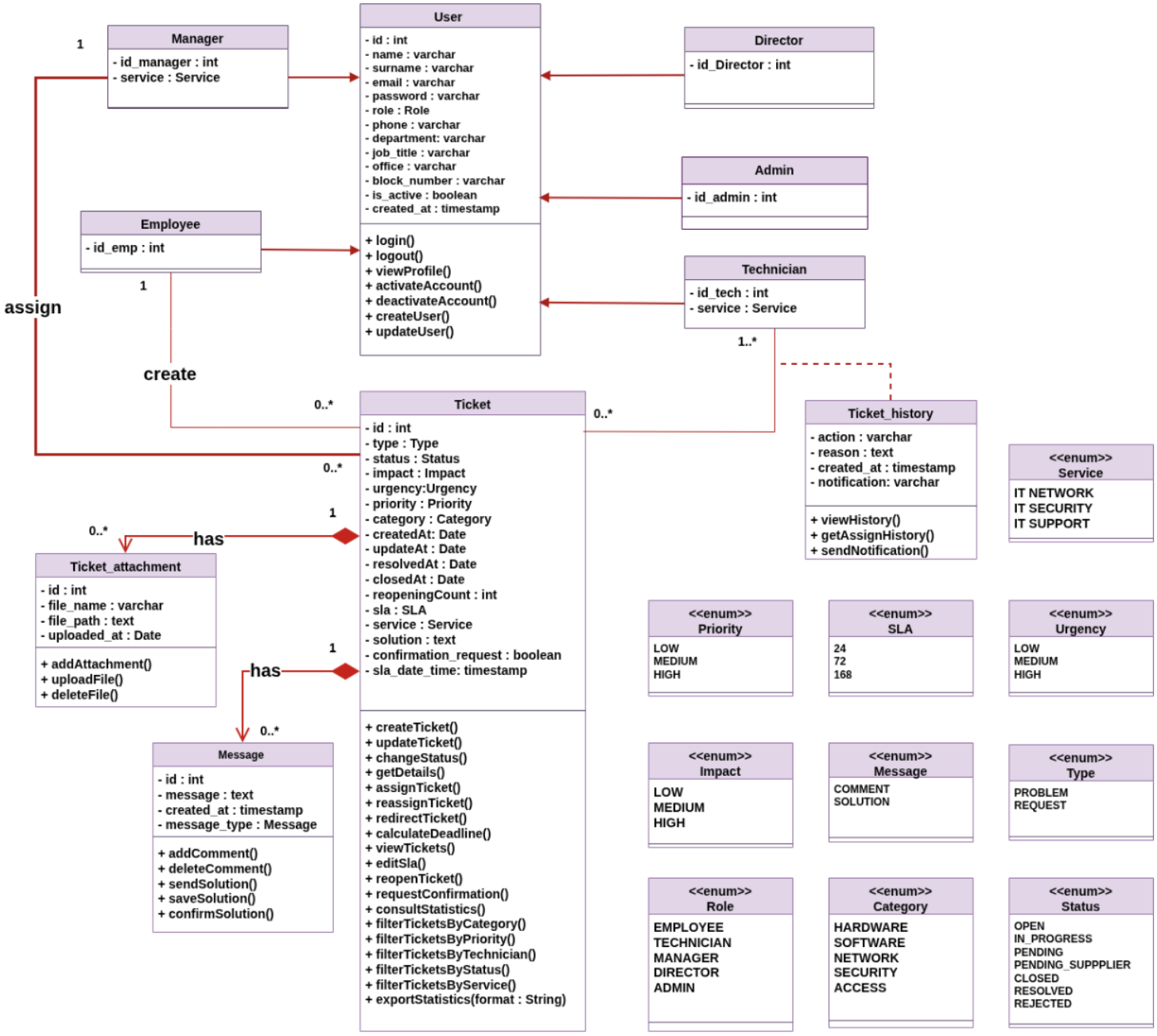


Figure 2.7: Class diagram

2.4.5 From class diagram to relational model

To convert the class diagram into a relational model, we apply the following conversion rules. In this model, a **Primary Key (PK)** is underlined, while a **Foreign Key (FK)** is preceded by #:

- **Rule 1:** Each class becomes a table. Class attributes become table attributes, and the class identifier becomes the table's primary key.
- **Rule 2:** Each one-to-one (1–1) association is represented by including the primary key of one

relation as a foreign key in the other relation.
- **Rule 3:** Each one-to-many (1–N) association is represented by including the primary key of one relation as a foreign key in the relation whose maximum multiplicity is *.
- **Rule 4:** Each many-to-many (M–N) association is represented by creating a new relation whose primary key is the combination of the primary keys of the participating relations. The attributes of the association class are added to this new relation when necessary.
- **Rule 5:**
  - The relational model does not directly support inheritance relationships, since it only includes the concepts of relations and keys.
  - Three solutions can be used to transform an inheritance relationship:
    1. Represent only the parent class using a single table.
    2. Represent only the child classes, each with its own table.
    3. Represent inheritance using a reference between the parent class and the child class.

By applying the conversion rules presented in the previous section to the class diagram presented in figure 2.7, we obtain the relational model, which contains the following tables:

- **User** (id, name, surname, email, password, role, phone, department, job_title, office, block_number, is_active, created_at).
- **Manager** (id_manager, service, **#id_user**).
- **Director** (id_Director, **#id_user**).
- **Admin** (id_admin, **#id_user**).
- **Employee** (id_emp, **#id_user**).
- **Technician** (id_tech, service, **#id_user**).
- **Ticket** (id, type, status, impact, urgency, priority, category, createdAt, updateAt, resolvedAt, closedAt, reopeningCount, sla, service, confirmation_request, sla_date_time, solution, **#id_emp**, **#id_manager**).
- **Ticket_History** (**#ticket_id**, **#id_tech**, **created_at**, action, reason, notification).
- **Ticket_Attachment** (id, file_name, file_path, uploaded_at, **#ticket_id**).
- **Message** (id, message, created_at, message_type, **#ticket_id**).

2.5 Conclusion

This chapter presented the modeling of our project, in which the initial requirements were transformed into a structured representation of the platform. Starting from the expression of needs, through requirements analysis, and concluding with the conceptual design using UML modeling, we obtained the relational model from the class diagram. In the next chapter, we present the automatic ticket classification module integrated into our platform.

Chapter 3
Automatic ticket classification module

3.1 Introduction

In this chapter, we present the AI module on which our platform relies for the classification of IT ticket categories. We describe the steps taken to develop the model, the models explored before evaluation, and the selection of the best-performing model for integration into the IT ticketing platform.

3.2 General process

To build the ticket classification module, we followed a structured approach consisting of the steps:

1. **Data collection**
2. **Data preprocessing**
3. **Models training**
4. **Models evaluation**
5. **Best model selection**
6. **Translation model**

The figure 3.1 presents the general process of the ticket classification module:

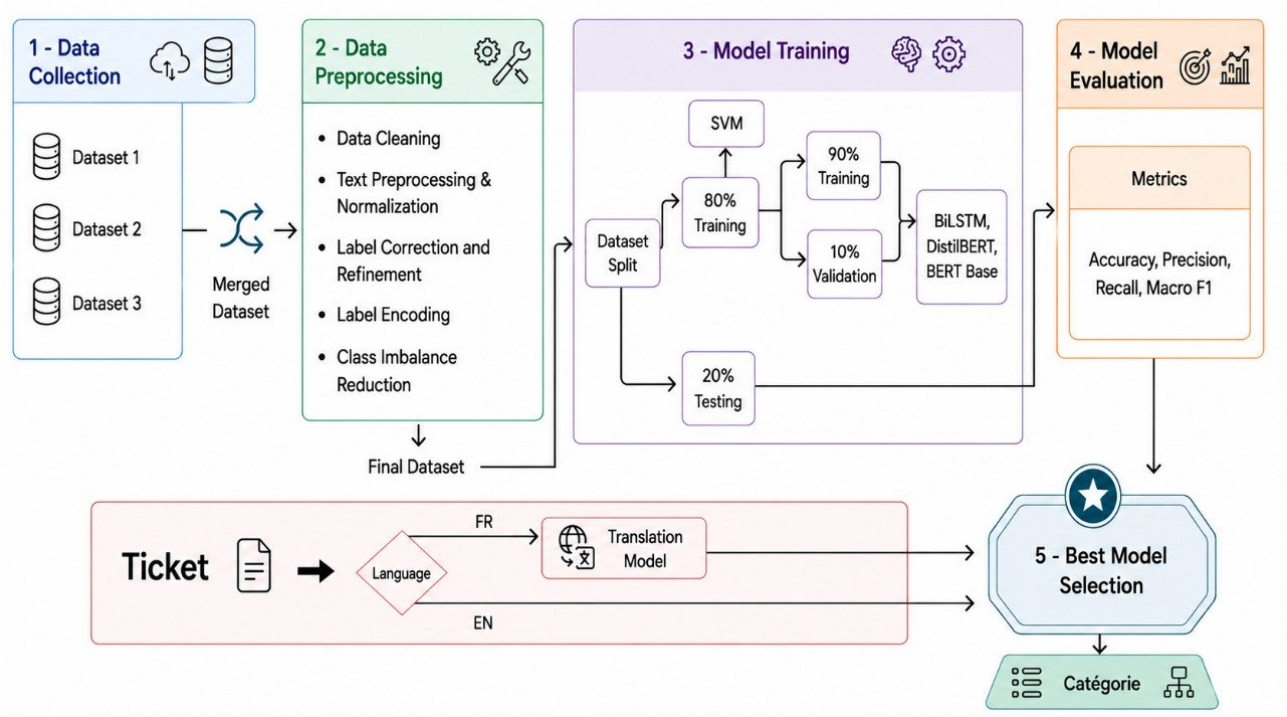


Figure 3.1: General process of the ticket classification module

3.3 Data collection

The first step of the IT ticket classification process involved collecting and preparing the data used to train and evaluate the models. We identified the target categories and ticket types for the classification task. The selected categories were inspired by common ITSM environments, open-source datasets, and IT tickets from *Djezzy*, focusing on the most frequent categories in ITSM ticketing systems.

The final selected categories and their ticket type descriptions are presented as follows:

- **Access:** authentication, authorization, and account access issues, including login failures, password resets, and permission issues.
- **Software:** errors, crashes, and installation requests, including app freezes and update failures.
- **Hardware:** physical device failures and hardware-related requests.
- **Network:** connectivity, Wi-Fi, and VPN issues, including firewall blocking, IP conflicts, and DNS errors.
- **Security:** security incidents and suspicious activities, including phishing, ransomware, data leakage, and stolen devices.

During the exploration phase, we analyzed several open-source datasets with different structures, distributions, and quality levels. Some contained repetitive patterns, limited diversity, or categories not aligned with our objectives. Therefore, we selected only the tickets relevant to the predefined categories.

To construct the final dataset, we combined tickets from multiple sources organized into three datasets:

- **Dataset 1:** selected tickets collected from Kaggle datasets [14, 15, 16].
- **Dataset 2:** selected tickets collected from HuggingFace datasets [17, 18, 19].
- **Dataset 3:** tickets manually created and generated using Large Language Models and other explored resources in order to improve dataset quality, diversity, and category representation.

All collected and generated tickets were written in English and included different writing styles and formats, ranging from short requests to detailed issue descriptions, in order to increase data diversity. The three datasets were then merged into a unified dataset containing 64,778 tickets. Each row represents a ticket, while the columns represent ticket attributes. The final dataset follows a supervised learning structure with two main attributes: `text` and `category`.
The table 3.1 presents the Dataset structure:

Column	Description
description	Ticket description containing the user request or issue
category	Target label representing the ticket category

Table 3.1: Dataset structure

3.4 Data preprocessing

Preprocessing is a crucial step in model training. Before training the models, we applied several steps to clean, normalize, and refine the merged dataset, improving data quality, reducing noise, correcting inconsistencies, and preparing the data for training:

- **Data cleaning:** removal of duplicate and near-duplicate records, null values, and corrupted header rows introduced during data loading. Highly similar ticket patterns were also reduced to limit redundancy in the dataset.
- **Text preprocessing and normalization:** the textual content of tickets was cleaned through normalization operations, including correcting encoding issues (such as corrupted apostrophes), fixing malformed characters, and normalizing spacing and punctuation. Irrelevant content, such as greetings and non-informative text, was removed to retain only meaningful ticket information.
- **Label correction and refinement:** category names were standardized, and wrong or inconsistent labels were corrected. Some cases of category leakage were detected and fixed, and tickets were reassigned to the correct classes.
- **Label encoding:** since machine learning and deep learning models require numerical inputs, the five category labels were encoded into integers (0–4) using a Label Encoder. This encoding was applied across all four models before the dataset was split into training and testing subsets.
- **Class imbalance reduction:** the merged dataset had class imbalance because some categories, such as software and hardware, were more frequent than others. This imbalance could cause

the model to favor majority classes. To address this, we applied undersampling to the majority classes, reducing the imbalance across categories. As a result, the dataset size was reduced from 64,778 rows to 16,012 rows.

The figure 3.2 presents Categories distribution before and after preprocessing:

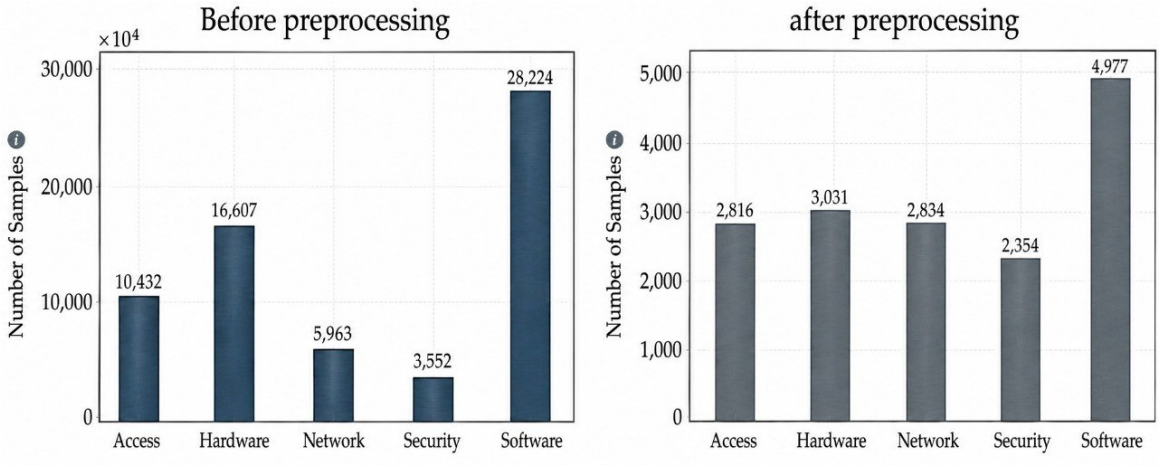


Figure 3.2: Categories distribution before and after preprocessing

The figure 3.3 presents an Example of the dataset after preprocessing:

Description	Category
I'm a new employee and my IT account was not set up properly. I can log into Windows but I can't access Outlook, shared drives or any company applications.	access
I'm having some issues with this one app, it keeps crashing when I run a script, can you guys take a look and figure out whats going on.	software
The RGB backlight on my keyboard no longer activates when I log in. The keyboard itself functions but has no lighting	hardware
The connection is extremely unstable. I keep seeing proxy server misconfiguration every time I open a browser.	network
I think I have an issue with Suspicious RAT detected. It started right after I opened a suspicious attachment. Please investigate.	security

Figure 3.3: Example of the dataset after preprocessing

3.5 Models training

3.5.1 Data splitting

After label encoding, the dataset was split using a stratified approach to preserve class distribution, with 80% for training and 20% for testing. For deep learning models, the training set was further divided into 90% training and 10% validation for model evaluation.

3.5.2 Models

To classify IT tickets automatically, we explored both machine learning and deep learning approaches, considering that machine learning methods have been widely applied to NLP tasks while deep learning models have demonstrated superior performance in recent years [4], and based on an exploration of commonly used models for text classification tasks [20]. The following section presents the model pipelines, along with their architectures and hyperparameters.

3.5.3 Machine learning model

The machine learning model we selected for classification is the Support Vector Machine (SVM), a supervised learning algorithm introduced by Cortes and Vapnik that separates classes by finding the optimal hyperplane with the maximum margin between data points. It is particularly well suited for text classification because it handles high-dimensional and sparse data effectively [5], as it has demonstrated strong performance on text categorization problems [6].

The SVM classification pipeline, after data preprocessing and splitting, consists of the following steps:

1. **Tokenization**: breaks raw ticket text into tokens (words and bigrams) before vectorization.
2. **TF-IDF Vectorization**: converts tokenized text into numerical vectors using up to 20,000 unigrams and bigrams, with sublinear TF scaling and a minimum document frequency of two.
3. **Cross-validation**: applies 5-fold stratified cross-validation on the training set to verify model stability before committing to full training.
4. **Training**: trains a LinearSVC with squared hinge loss and balanced class weights to handle class imbalance.
5. **Hyperparameter tuning**: tests regularization values $C \in \{0.01, 0.05, 0.1, 0.5, 1.0\}$ to address detected overfitting, selecting the value that maximizes F1-macro while minimizing the train/test accuracy gap.
6. **Evaluation**: assesses the final model on the test set using accuracy, precision, recall, and macro F1.

The figure 3.4 presents the SVM pipeline:

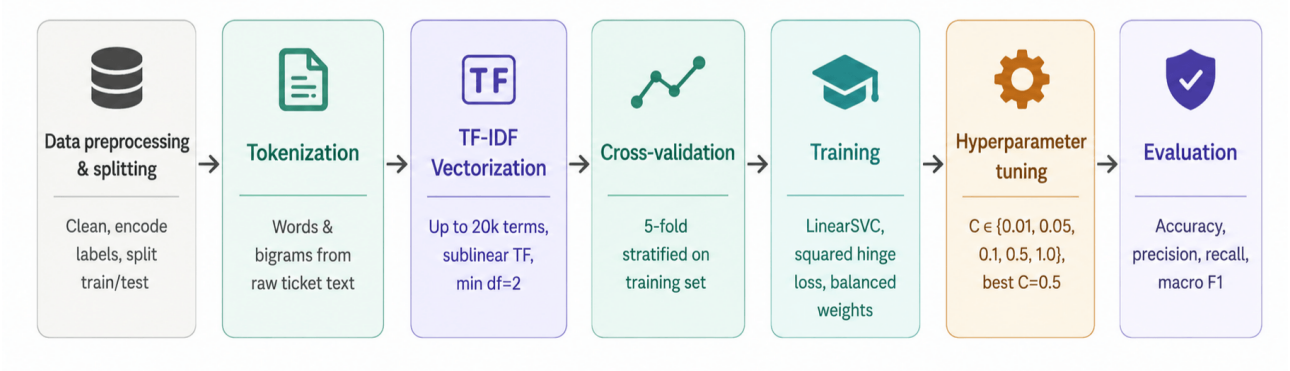


Figure 3.4: SVM pipeline

The table 3.2 presents hyperparameters of the SVM model:

Component	Hyperparameter	Value	Description
TF-IDF Vectorizer	max_features	20,000	Maximum number of terms to consider
	ngram_range	(1, 2)	Range of n-grams extracted
	sublinear_tf	True	Apply logarithmic scaling to term frequency
	min_df	2	Minimum document frequency for a term
LinearSVC	C	0.5	Regularization parameter (selected via grid search)
	class_weight	balanced	Handles class imbalance
	max_iter	2,000	Maximum number of iterations
	loss	Squared Hinge Loss	Loss function used for optimization
Cross-Validation	method	Stratified K-Fold	Cross-validation strategy
	n_splits	5	Number of folds

Table 3.2: Hyperparameters of the SVM (LinearSVC) model

3.5.4 Deep learning models

The deep learning models we selected are BiLSTM, BERT Base, and DistilBERT, chosen for their proven effectiveness in NLP text classification tasks [4][7][8], and their pipeline, after data preprocessing and splitting, consists of the following steps:

1. **Tokenization**: converts raw ticket text into numerical tokens.
2. **Padding**: ensures all sequences reach the same fixed length so they can be processed in batches.
3. **Training**: the model is trained on the training set with class weights applied to reduce bias toward majority classes.
4. **Early stopping**: monitors validation loss and halts training when no improvement is observed for a defined number of epochs, saving the best model weights.
5. **Evaluation**: the model is evaluated on the test set using accuracy, precision, recall, and macro F1.

the figure 3.5 presents the deep learning pipeline:



Figure 3.5: Deep learning pipeline (BiLSTM, BERT Base, DistilBERT)

- **BiLSTM** (Bidirectional Long Short-Term Memory)
  BiLSTM is a recurrent neural network that processes text in both forward and backward directions, capturing contextual information from both sides of each word, making it more effective than a standard LSTM for text classification [9, 10].
  the table 3.3 presents hyperparameters used for the BiLSTM model:

Component	Hyperparameter	Value	Description
Tokenizer	num_words	10,000	Maximum vocabulary size
	oov_token	<OOV>	Token for unknown words
	max_len	100	Maximum sequence length
	padding	post	Padding position
Architecture	Embedding dim	128	Dimension of word embeddings
	BiLSTM layer 1	64 (tanh)	Units in first BiLSTM layer
	BiLSTM layer 2	32 (tanh)	Units in second BiLSTM layer
	Dropout	0.3	Dropout rate to reduce overfitting
	Dense hidden	64 (ReLU)	Units in dense layer
	Output layer	5 (Softmax)	Units in output layer
Training	optimizer	Adam	Optimization algorithm
	loss	sparse_categorical_crossentropy	Loss function
	epochs	10	Maximum number of training epochs
	batch_size	32	Number of samples per batch
	early stopping patience	3	Epochs to wait before stopping
	restore_best_weights	True	Restore weights of best epoch

Table 3.3: Hyperparameters of the BiLSTM model

- **BERT Base**

BERT is a pretrained transformer model developed by Google that captures contextual meaning of words using bidirectional attention across 12 transformer layers [7]. BERT Base is fine-tuned on our ticket classification dataset by adding a classification head on top of the pretrained model. the table 3.4 presents hyperparameters of the BERT Base model:

Component	Hyperparameter	Value	Description
Tokenizer	model	bert-base-uncased	Pretrained tokenizer used
	max_length	128	Maximum token sequence length
	truncation	True	Truncate sequences exceeding max length
	padding	True	Pad sequences to max length
Architecture	base model	BertForSequence Classification	Pretrained model used
	transformer layers	12	Number of transformer layers
	dropout	0.1	Dropout rate for regularization
Training	optimizer	AdamW	Optimization algorithm
	learning_rate	2e-5	Initial learning rate
	epochs	5	Maximum number of training epochs
	batch_size	16	Number of samples per batch
	lr_scheduler	linear	Learning rate scheduling strategy
	loss	CrossEntropyLoss (weighted)	Loss function
	early stopping patience	2	Epochs to wait before stopping

Table 3.4: Hyperparameters of the BERT Base model

- **DistilBERT**
  DistilBERT is a lightweight version of BERT created through knowledge distillation [8], retaining about 97% of BERT's performance while being 40% smaller and 60% faster, making it suitable for deployment. Like BERT Base, it is fine-tuned on our ticket classification dataset.
  DistilBERT shares the same training configuration as BERT Base (table 3.4), with the differences summarized in table 3.5:

Component	BERT Base	DistilBERT
Tokenizer	bert-base-uncased	distilbert-base-uncased
Base model	BertForSequenceClassification	DistilBertForSequenceClassification
Transformer layers	12	6

Table 3.5: Differences between BERT Base and DistilBERT

3.6 Evaluation and metrics

To evaluate and compare the performance of all models, we used the classification metrics presented in table 3.6:

Metric	Formula	Description
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Proportion of correctly classified instances out of all instances.
Macro Precision	$\frac{TP}{TP+FP}$	Average precision across all categories equally.
Macro Recall	$\frac{TP}{TP+FN}$	Average recall across all categories equally.
Macro F1	$\frac{2 \cdot P \cdot R}{P+R}$	Harmonic mean of precision and recall, averaged equally across all categories.

Table 3.6: Classification metrics for evaluation

- **TP** (True Positives): instances correctly predicted as positive.
- **TN** (True Negatives): instances correctly predicted as negative.
- **FP** (False Positives): instances incorrectly predicted as positive.
- **FN** (False Negatives): instances incorrectly predicted as negative.
- **P** (Precision): proportion of correctly predicted positive instances among all predicted positives.
- **R** (Recall): proportion of correctly predicted positive instances among all actual positives.

Table 3.7 summarizes the results of all four models evaluated on the test set:

Model	Accuracy	Macro Precision	Macro Recall	Macro F1
SVM	97.07%	96.92%	97.12%	97.00%
BiLSTM	95.69%	96.00%	96.00%	96.00%
BERT Base	97.00%	97.00%	97.00%	97.00%
DistilBERT	**97.91%**	**97.86%**	**97.88%**	**97.87%**

Table 3.7: Overall comparison of model performance on the test set

Figure 3.6 illustrates these results visually:

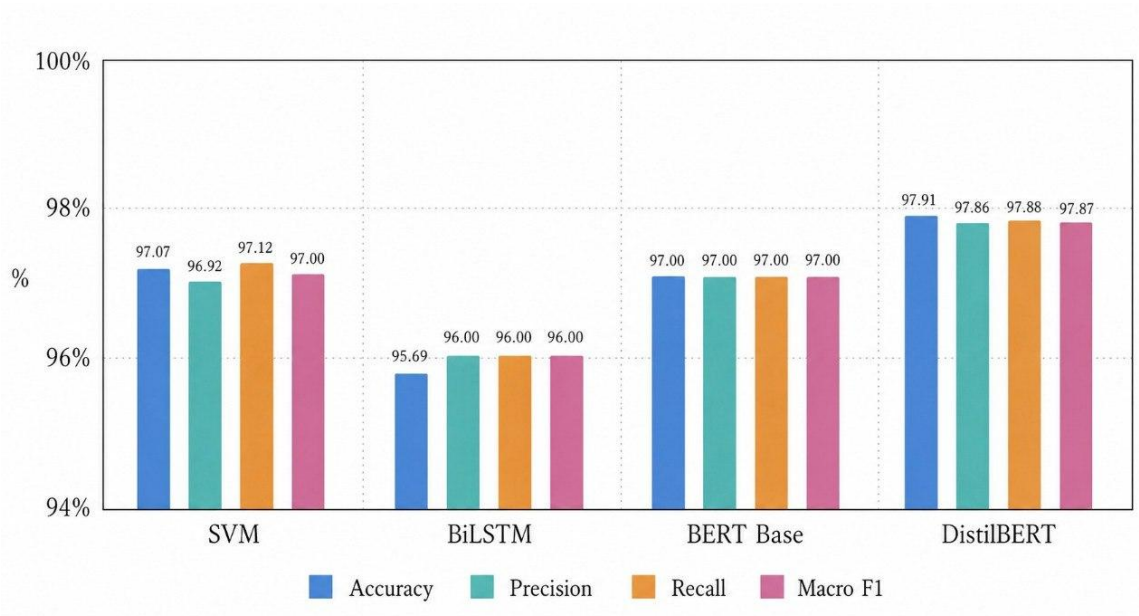


Figure 3.6: Model performance comparison on the test set

3.7 Best model selection

After comparing all models based on the results presented in Table 3.7 and Figure 3.6, we selected DistilBERT as the best-performing model for ticket category classification. It achieved the highest scores across all metrics, with 97.91% accuracy, 97.87% Macro F1, and 97.88% recall. Unlike SVM, which relies on word frequency vectors, DistilBERT captures the meaning and context of words more effectively, making it more robust to different ticket formulations. It is also 40% smaller and 60% faster than BERT Base, making it more suitable for production environments. Its balanced performance across all five ticket categories supports our final choice.

3.8 Translation model

The classification model is trained exclusively on English text. To support French inputs, we added a translation step for tickets written in French. When a ticket's title and description are in French, they are first translated into English and then passed to the classification model. Tickets already written in English are passed directly to the model without translation.

We used the **Helsinki-NLP/opus-mt-fr-en** model for French-to-English translation. It is based on the MarianMT architecture [11], an encoder-decoder Transformer model with attention mechanisms derived from the Transformer framework [12], trained on large-scale French–English parallel corpora.

The model uses subword tokenization and is optimized for neural machine translation of short and medium-length texts such as support tickets. It is lightweight and open-source, requiring no additional training.

3.9 Conclusion

This chapter presented the development and evaluation of an automatic ticket classification module. After dataset construction and preprocessing, several models were trained and evaluated. DistilBERT achieved the best performance, making it the most suitable for integration into our platform, including a translation step for French tickets before classification. The next chapter presents the implementation of our application.

Chapter 4
Implementation

4.1 Introduction

This chapter presents the final phase of our project, which is the implementation. We introduce the hardware and software environment used to develop the application, its deployment diagram, and some of the application interfaces.

4.2 Development environment

4.2.1 Hardware environment

The project was developed on four machines, whose specifications are presented in table 4.1:

Component	**Machine 1**	**Machine 2**	**Machine 3**	**Machine 4**
Model	Lenovo IdeaPad Flex 5 14A	Lenovo ThinkPad T460	Fujitsu	Lenovo ThinkPad T460
Processor	AMD Ryzen 7 5700U with Radeon Graphics	i5-10310U CPU @ 1.70GHz (2.21 GHz)	Intel Core i5-6300U @ 2.40 GHz	Intel Core i7-6600U CPU @ 2.60GHz 2.81 GHz
RAM	16 GB	16 GB	8 GB	8 GB
OS	Windows 10 Pro	Windows 11 Pro	Linux Mint 22 (Wilma)	Windows 10

Table 4.1: Hardware environment

4.2.2 Software environment

This section presents the programming languages, frameworks, libraries, database, and development tools we used to build the application:

- **Programming languages:**
  - **JavaScript**: A programming language used to create interactive web applications. It is used in both frontend (React.js) and backend (Node.js)[21].



Figure 4.1: JavaScript

  - **Python**: A general-purpose language widely used in AI and data science. It is used to develop the classification module[22].



Figure 4.2: Python

- **HTML**: A markup language used to structure web pages[23].



Figure 4.3: HTML

- **CSS**: A stylesheet language used to style web pages[24].



Figure 4.4: CSS

- **SQL**: A language used to manage and query relational databases[25].



Figure 4.5: SQL

- **Frameworks and libraries:**
  - **React.js**: A JavaScript library for building user interfaces[26].

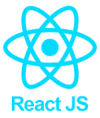


Figure 4.6: React.js

  - **Tailwind CSS**: A utility-first framework for fast UI design[27].



Figure 4.7: Tailwind CSS

  - **Node.js**: A runtime for executing JavaScript on the server[28].



Figure 4.8: Node.js

  - **Express.js**: A framework for building REST APIs[29].



Figure 4.9: Express.js

  - **Prisma**: An ORM used to access PostgreSQL using JavaScript[30].



Figure 4.10: Prisma

  - **FastAPI**: A Python framework used to build high-performance APIs[31].



Figure 4.11: FastAPI

  - **HuggingFace Transformers**: A library for pretrained NLP models[32].



Figure 4.12: HuggingFace Transformers

  - **PyTorch**: A deep learning framework used for model training and inference[33].



Figure 4.13: PyTorch

  - **Additional libraries**: NumPy, scikit-learn, Uvicorn, langdetect, sentencepiece, and sacremoses[34].
- **Database:**
  - **PostgreSQL**: A reliable open-source relational database[35].



Figure 4.14: PostgreSQL

- **Development tools:**
  - **Visual Studio Code**: A lightweight code editor used for development[36].



Figure 4.15: Visual Studio Code

  - **Git / GitHub**: Tools used for version control and collaboration[37].



Figure 4.16: Git / GitHub

– **Google Colab**: A cloud platform used for training machine learning models with GPU[38].



Figure 4.17: Google Colab

4.3 Deployment diagram

A deployment diagram is a type of UML structural diagram that illustrates the physical runtime configuration of a software system[2].

The figure 4.18 presents the deployment diagram of our application:

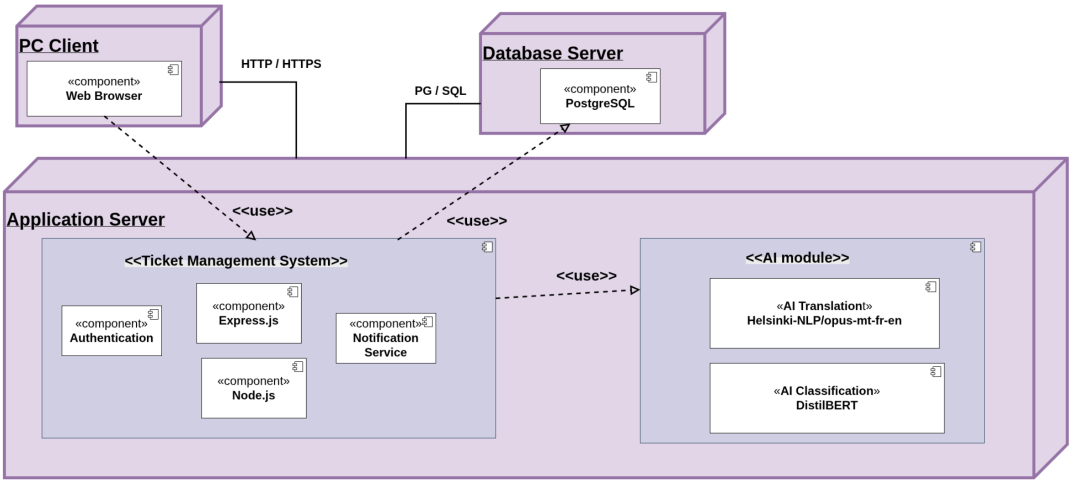


Figure 4.18: Deployment diagram

4.4 Application interfaces

User interfaces enable smooth interaction between users and the system. The following section presents some application interfaces.

4.4.1 Ticket creation interface for employee

The ticket creation allows employees to submit IT-related requests or issues, as shown in figure 4.19:

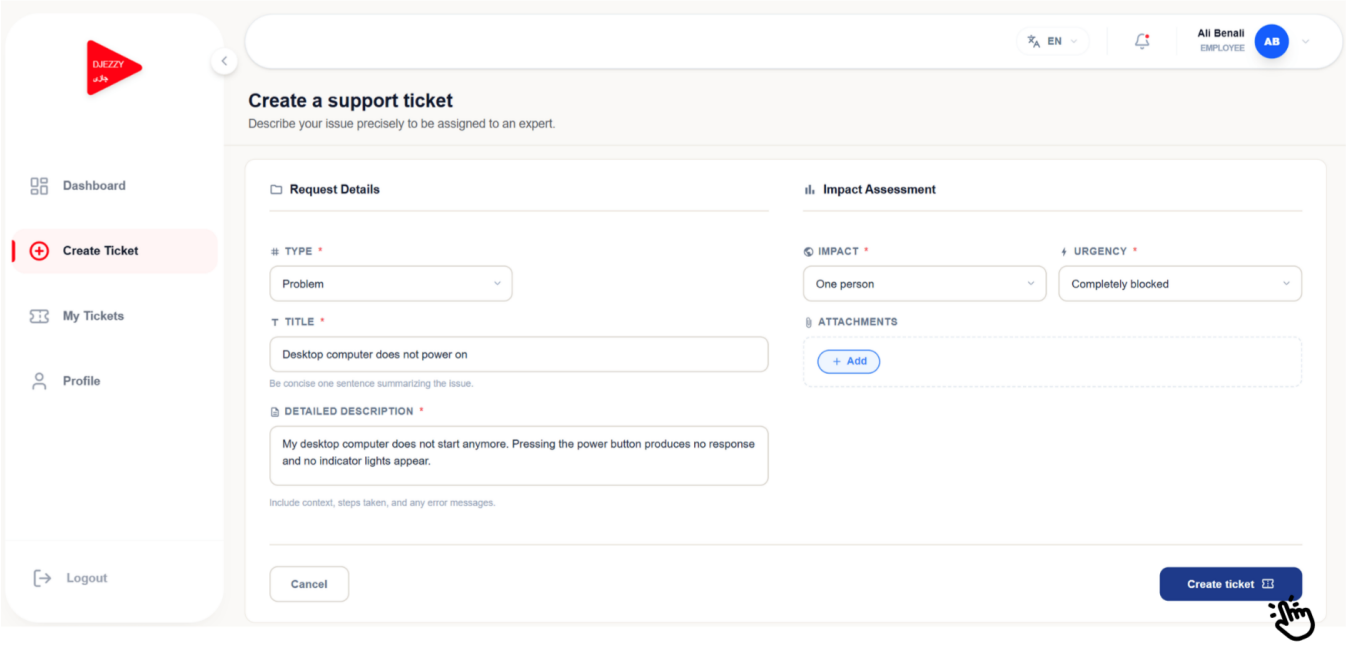


Figure 4.19: Ticket creation interface

After submission, the AI module classifies the ticket by predicting its category. The ticket is then mapped to the appropriate service based on the predicted category, and a success message is displayed to confirm its creation.

Figure 4.20 shows the success page displayed after ticket creation, including the category predicted by the AI module and the service to which the ticket was assigned. The result corresponds to the ticket filled in figure 4.19:

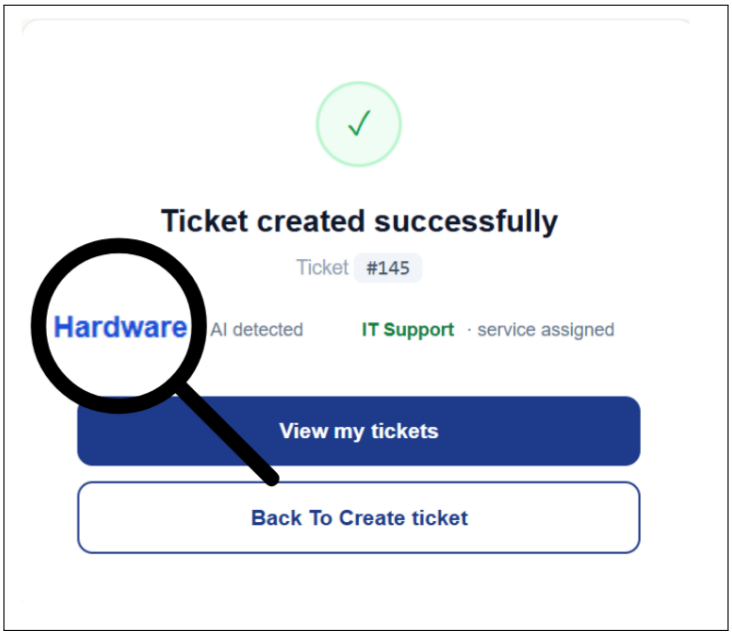


Figure 4.20: Ticket submission success form

Once created, ticket #145 is added to the IT Support service ticket list, where it becomes visible to the manager and technicians of that service.

4.4.2 Service tickets interface for technician

This interface displays all tickets related to the technician's service, including assigned and unassigned tickets, and allows technicians, after selecting a ticket, to view its details and take charge of it if it has not already been assigned. Figure 4.21 presents the service tickets interface, where ticket #145 created in the previous step now appears in the list, awaiting assignment:

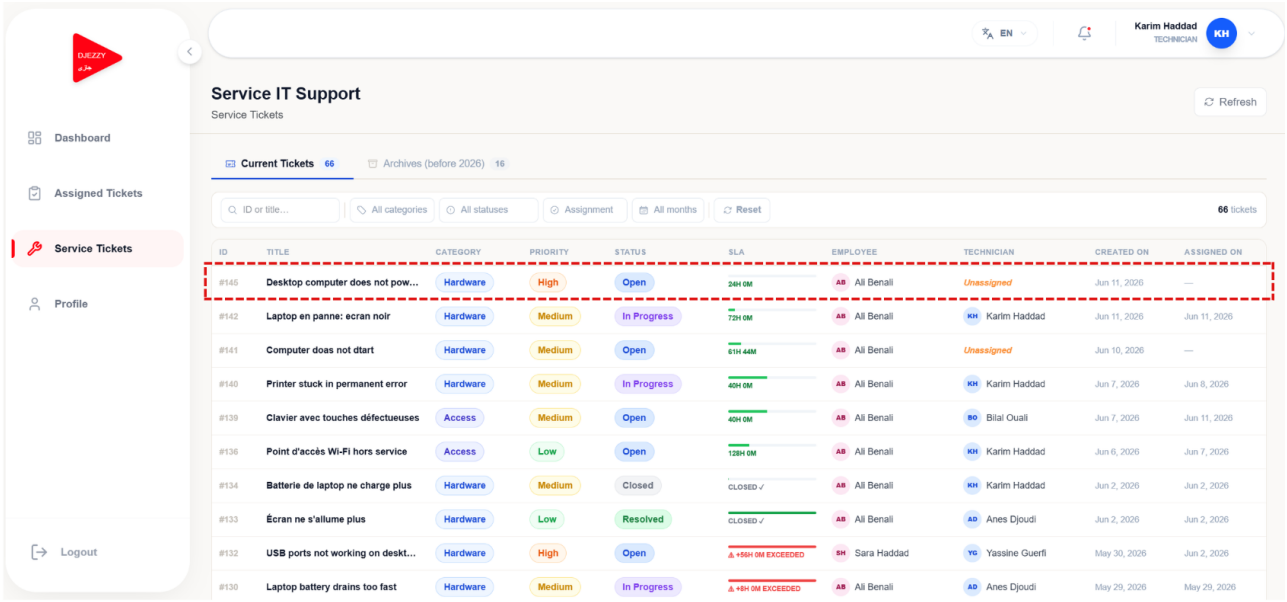


Figure 4.21: Service tickets interface

This interface (figure 4.21) is also used by the service manager, who, unlike technicians, can assign tickets to technicians within the service.

4.4.3 Ticket assignment for manager

The assignment interface allows the manager to view ticket details after selecting a ticket from the service ticket list (figure 4.21) and assign it to a technician for handling if it has not already been taken in charge. In this interface, the manager selects ticket #145 and clicks the "Assign ticket" button, outlined in red in figure 4.22, to proceed with the assignment:

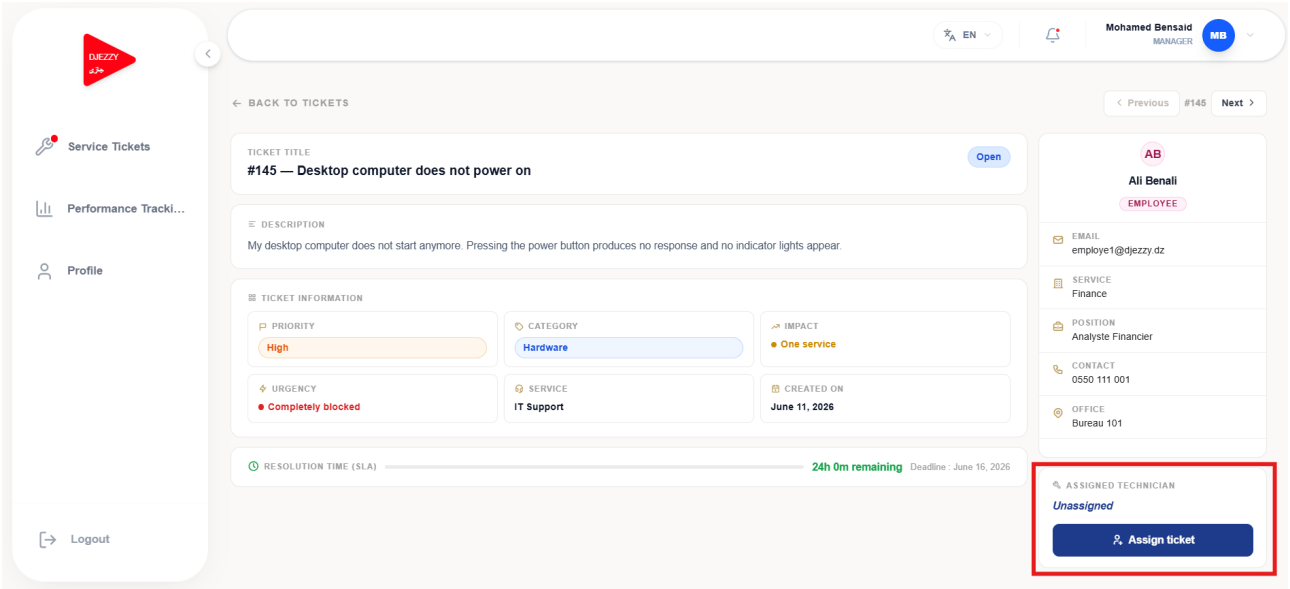


Figure 4.22: Ticket assignment interface

After clicking “Assign ticket”, a list of available technicians is displayed, allowing the manager to select the appropriate technician for the ticket, as shown in figure 4.23. Here, the manager assigns ticket #145 to technician Karim Haddad:

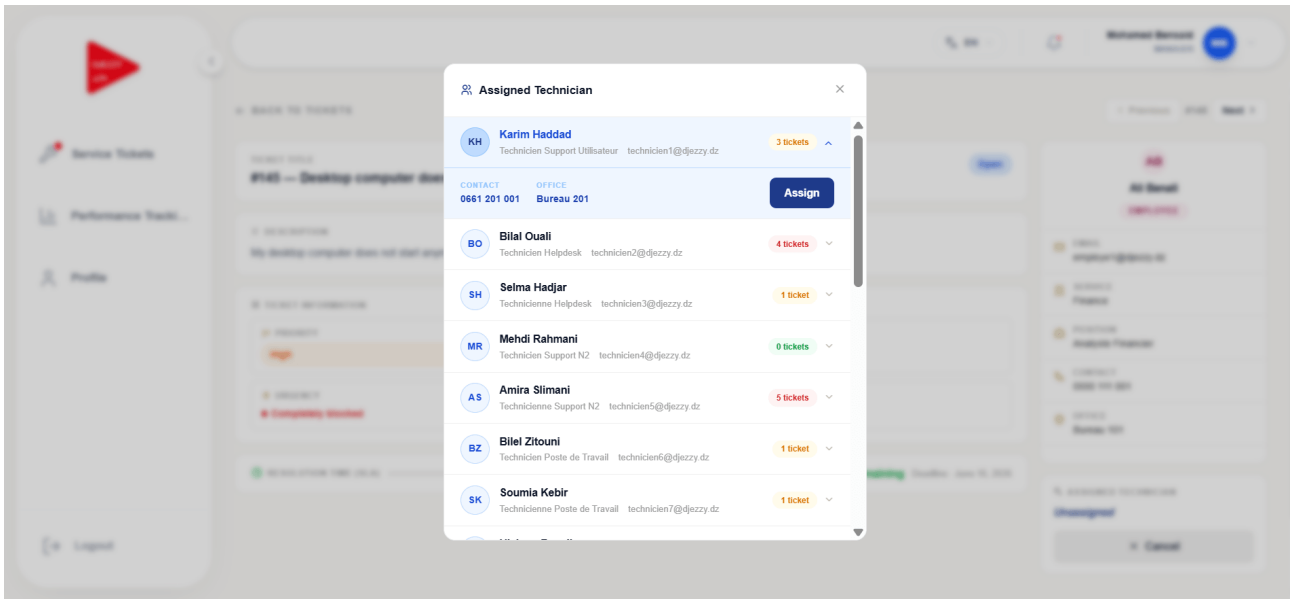


Figure 4.23: Technician selection list

4.4.4 Ticket details interface for technician

After selecting a ticket from the assigned tickets list, the technician can view the ticket details, employee information, attached files, and a workspace divided into tabs that allow the technician to manage communication and perform ticket actions. Figure 4.24 illustrates the communication tab, which allows the technician to exchange messages with the employee and provide solution steps.

In this interface, technician Karim Haddad opens ticket #145, sends resolution steps for the employee to follow, and asks them to confirm whether the issue has been resolved:

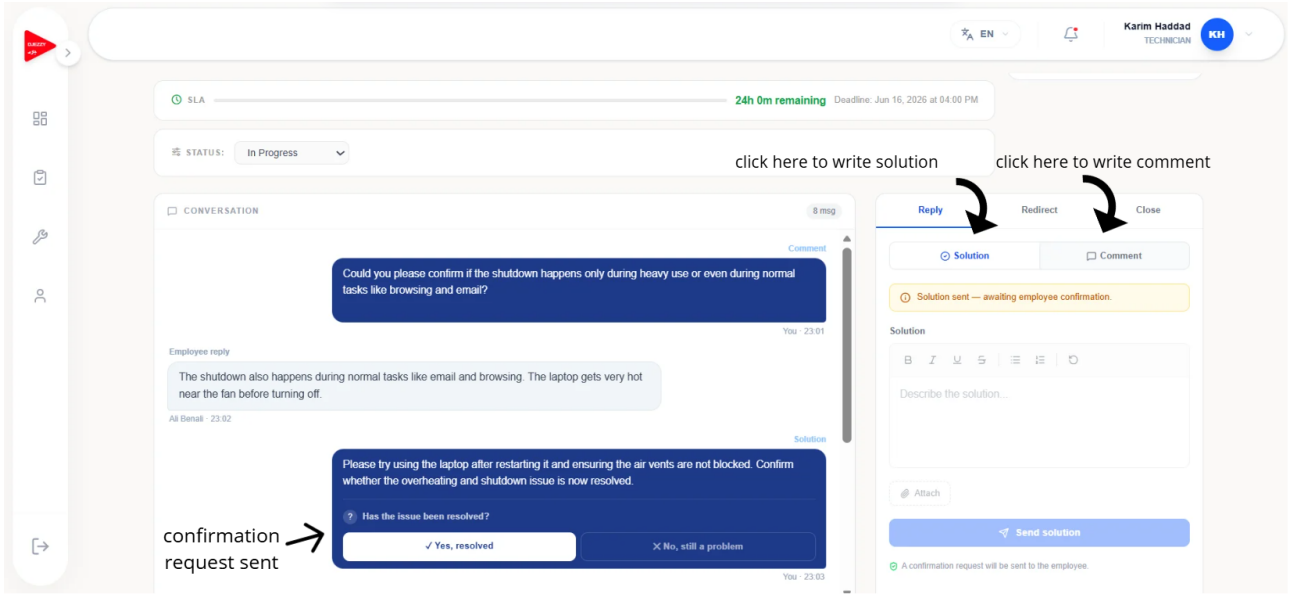


Figure 4.24: Technician communication and ticket action interface

4.4.5 Ticket details interface for employee

This interface allows the employee to view their ticket details and communicate with the assigned technician. Figure 4.25 presents the employee ticket details interface, showing only the communication section.

In this interface, the employee confirms that the issue reported in ticket #145 has been resolved:

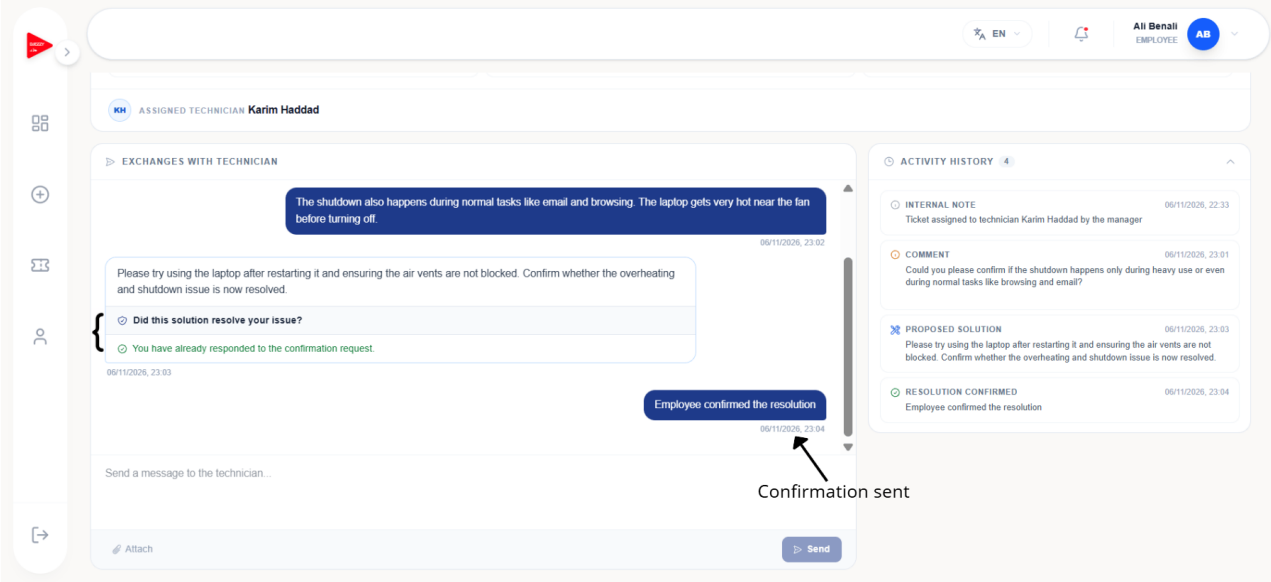


Figure 4.25: Employee ticket details interface

4.4.6 Saving the solution and closing the ticket

Once the employee confirms the resolution, the technician can save the solution and close the ticket. Figure 4.26 shows the technician's close tab after the employee's confirmation for ticket #145:

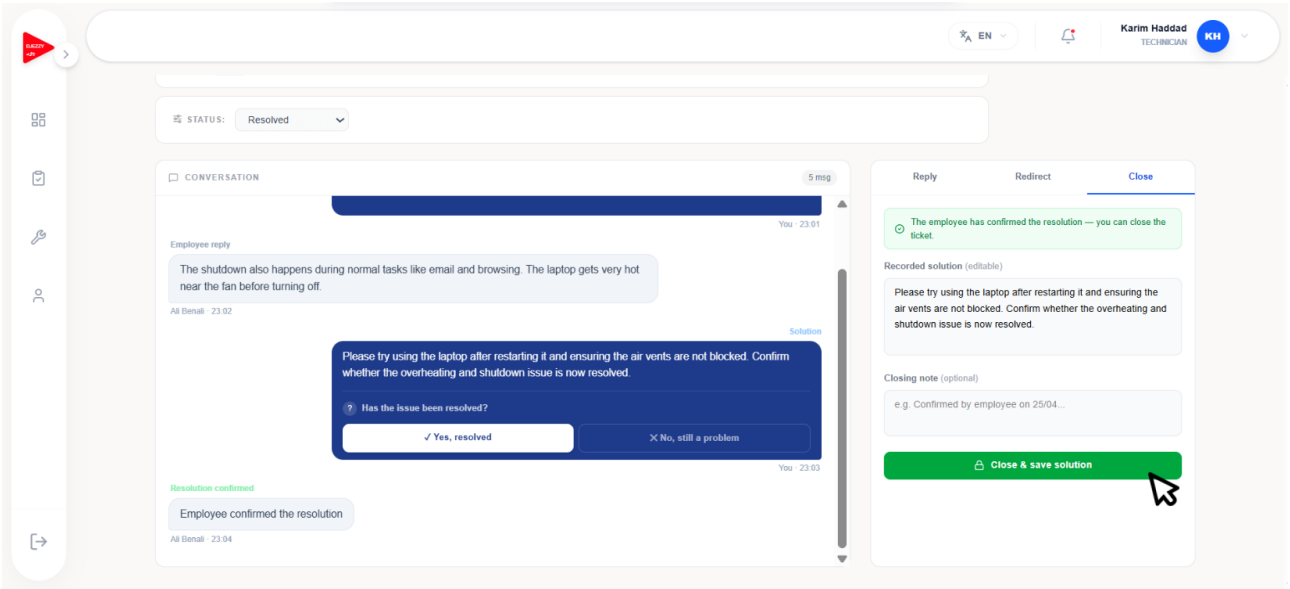


Figure 4.26: Technician close tab after employee confirmation

Figure 4.27 shows the final state of ticket #145 after it has been closed:

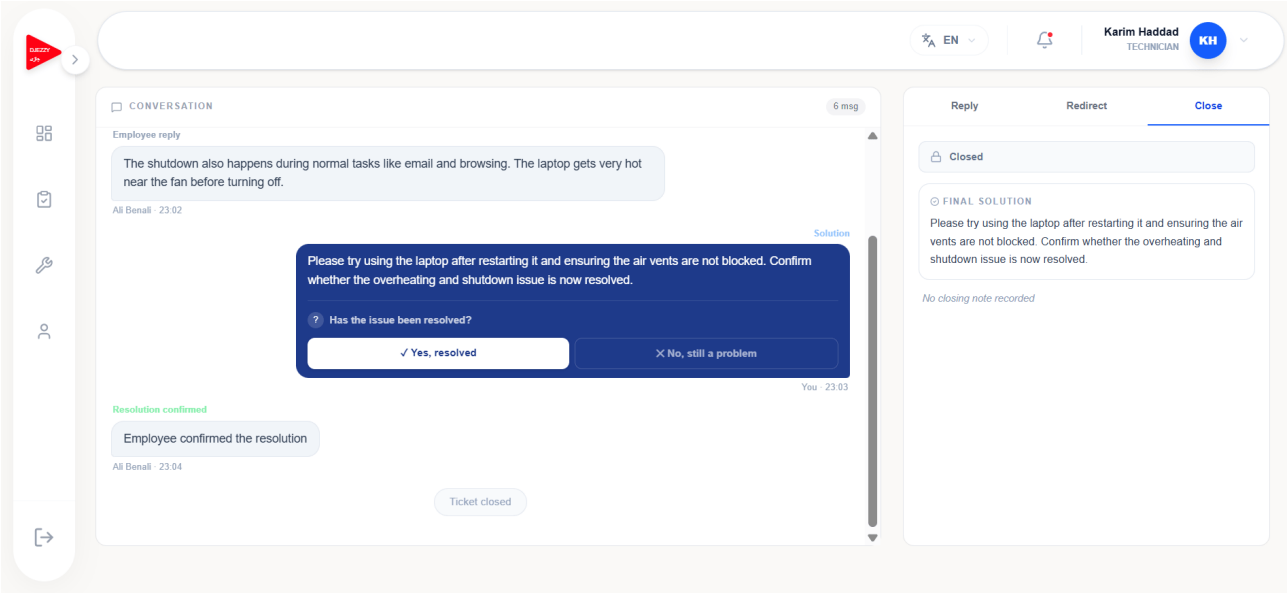


Figure 4.27: Closed ticket with final solution

4.4.7 Alternative ticket handling actions for technician

Besides the manager assignment process presented above, the platform offers alternative actions for handling a ticket.

A technician can also take charge of an unassigned ticket directly from the service ticket list (figure 4.21), without waiting for the manager to assign it.

Figure 4.28 shows the take charge interface:

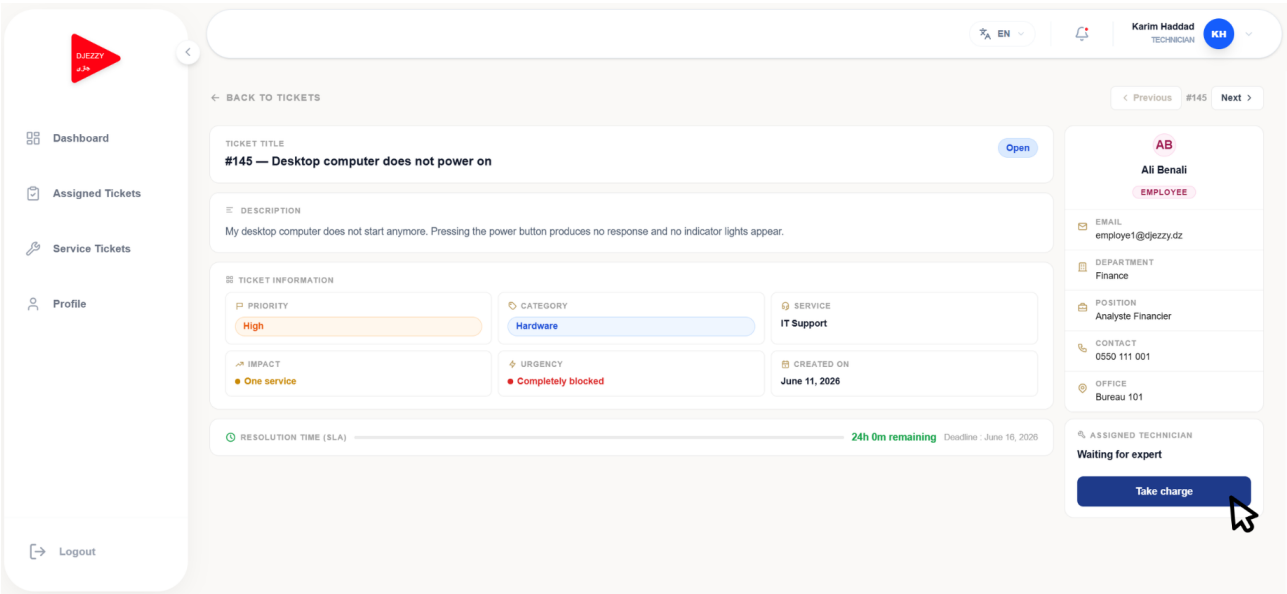


Figure 4.28: Take charge interface

The technician's workspace also includes tabs for ticket redirection and manual ticket closing. Figure 4.29 illustrates the ticket redirection process, while Figures 4.30 and 4.31 show the manual ticket closing process, where the technician selects the closing action and fills in the required information without employee confirmation, used when the confirmation is performed externally:

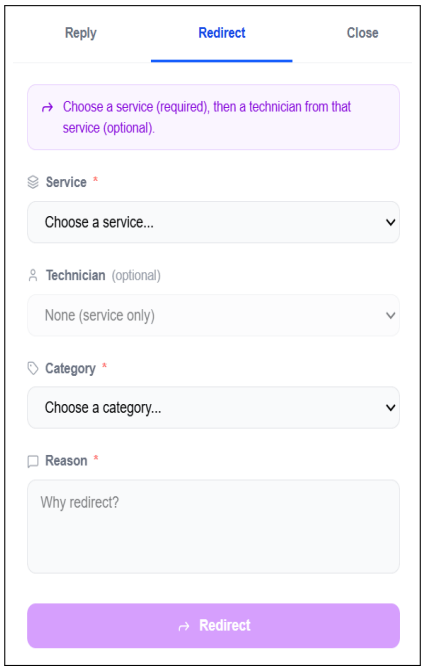


Figure 4.29: Ticket redirection form

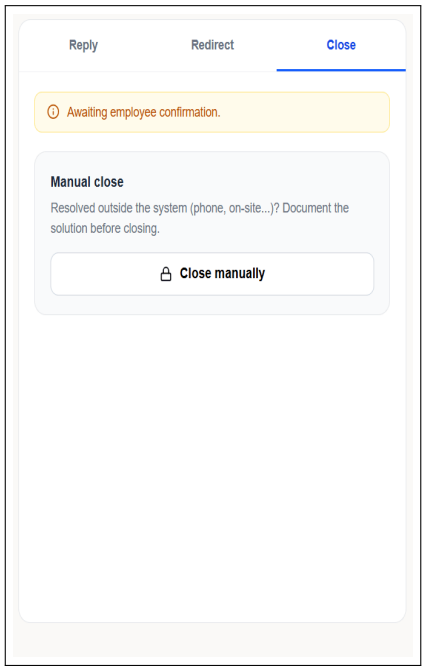


Figure 4.30: Ticket closing action

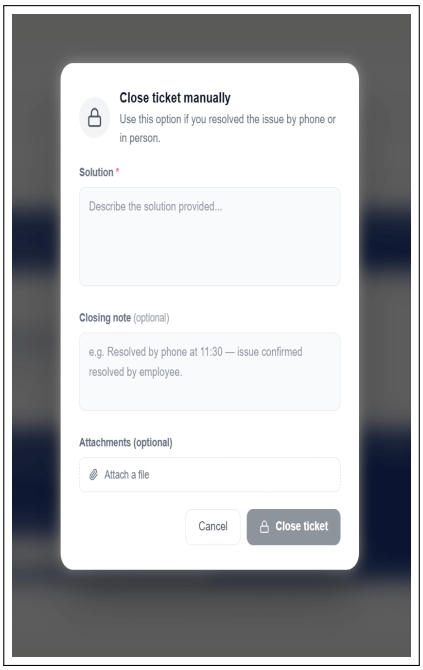


Figure 4.31: Manual ticket closing form

4.4.8 Performance tracking for manager

This interface allows the manager to view statistics about the tickets of his service and the performance of his technicians. He can filter the data by month or by year and export reports as PDF or Excel. The interface is divided into several sections displayed by scrolling down the page.

Figure 4.32 shows the summary cards section, where the manager can view the total number of tickets, the number of resolved tickets, the resolution rate, and the number of tickets where the SLA was respected or not:

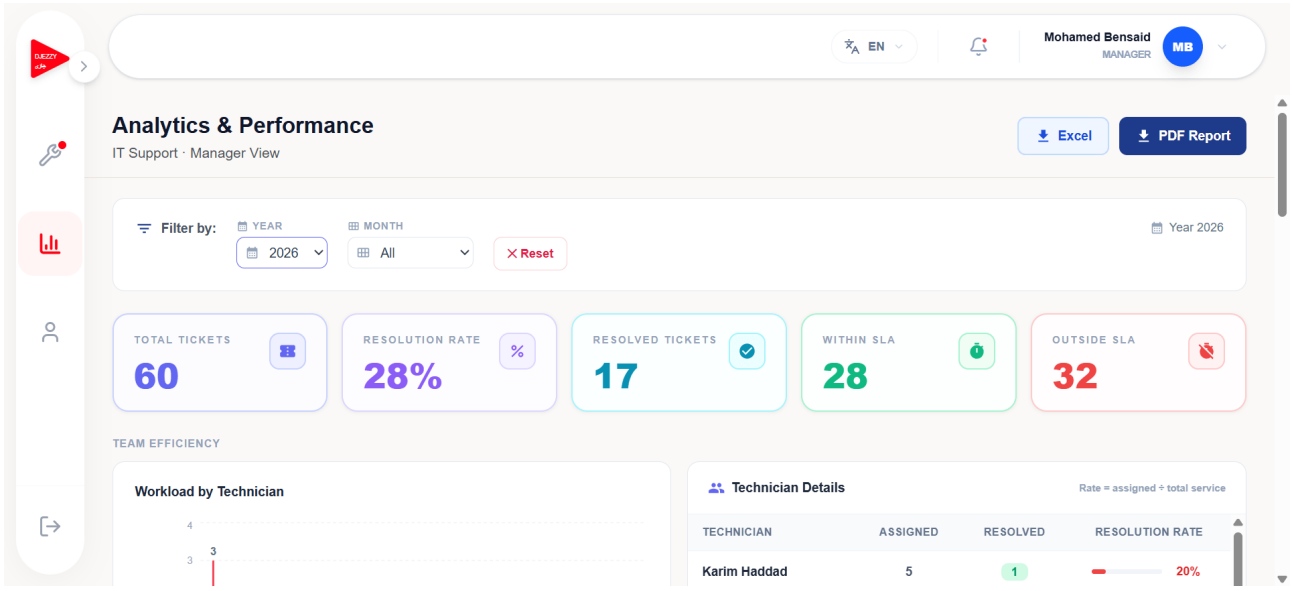


Figure 4.32: Performance tracking summary cards

Figure 4.33 shows the team efficiency section, where the manager can view the performance of each technician, including the number of assigned and resolved tickets:

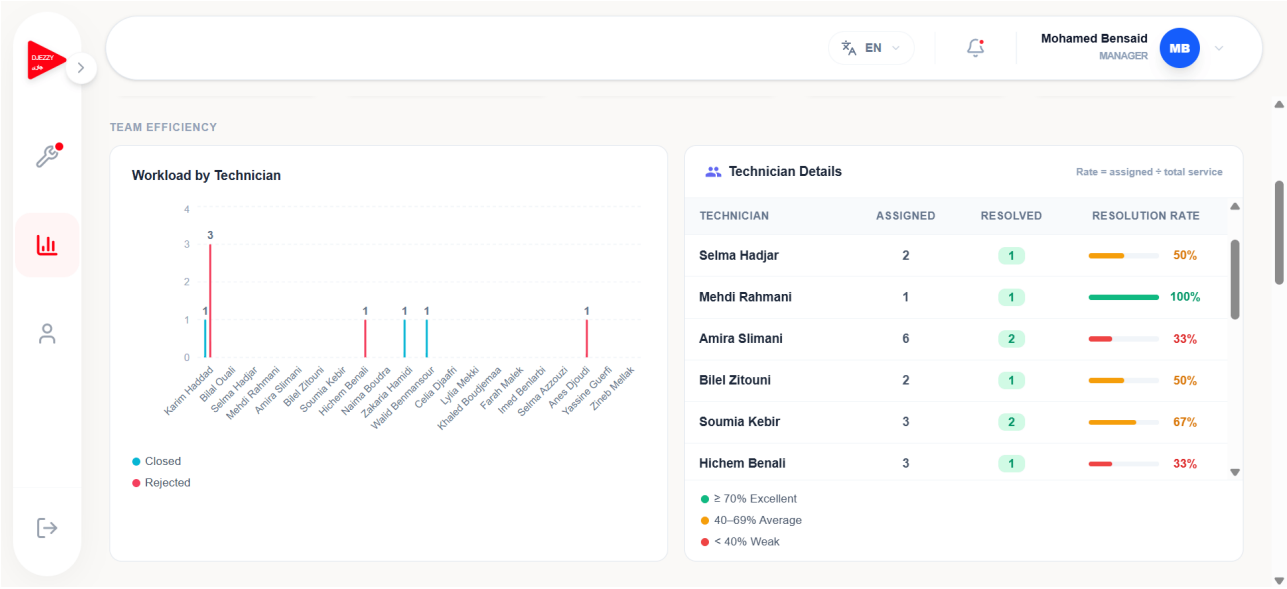


Figure 4.33: Performance tracking team efficiency

Figure 4.34 shows the ticket distribution section, where the manager can view the number of tickets grouped by status and by type:

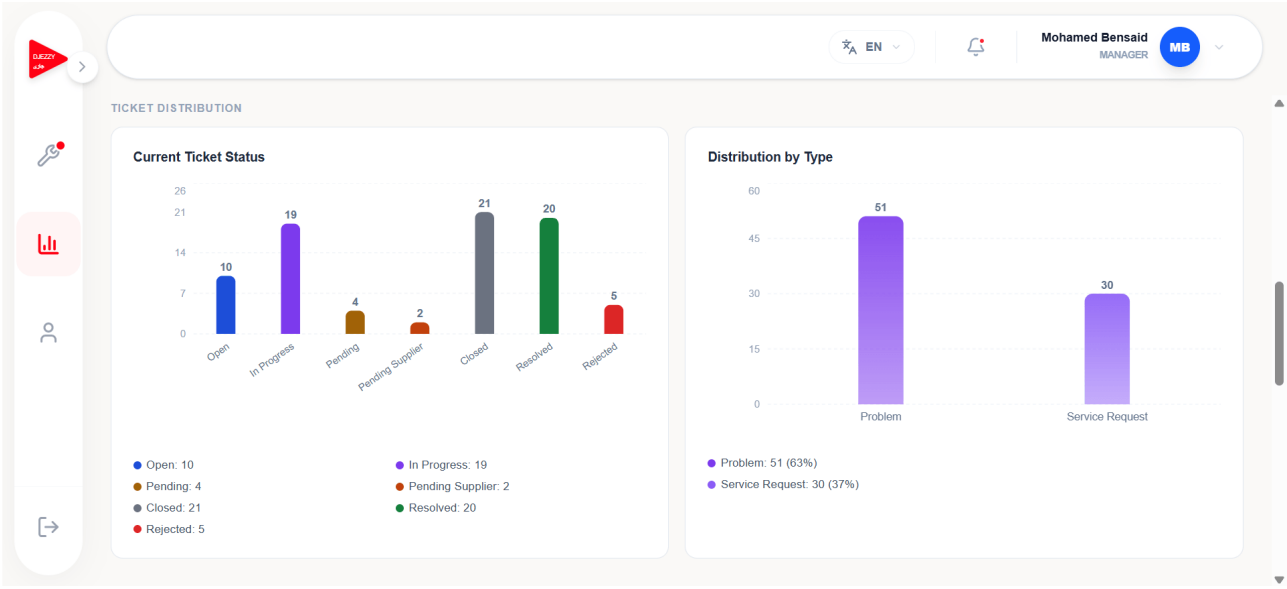


Figure 4.34: Performance tracking ticket distribution

Figure 4.35 shows the trends section, where the manager can view the number of tickets per month or per day depending on the selected filter, and the number of tickets per priority level:

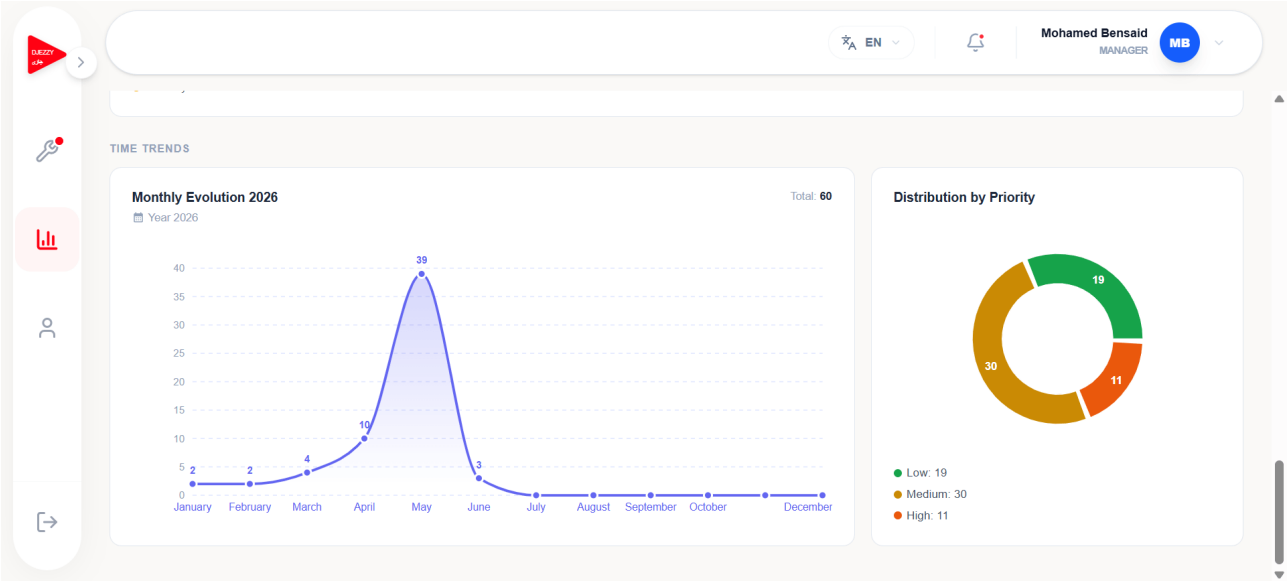


Figure 4.35: Performance tracking trends and priority distribution

4.4.9 User management for admin

This interface allows the admin to manage users. He can view the list of all users, and by clicking on a user he can view more information and edit them. He can also add a new user.

Figure 4.36 shows the user management interface:

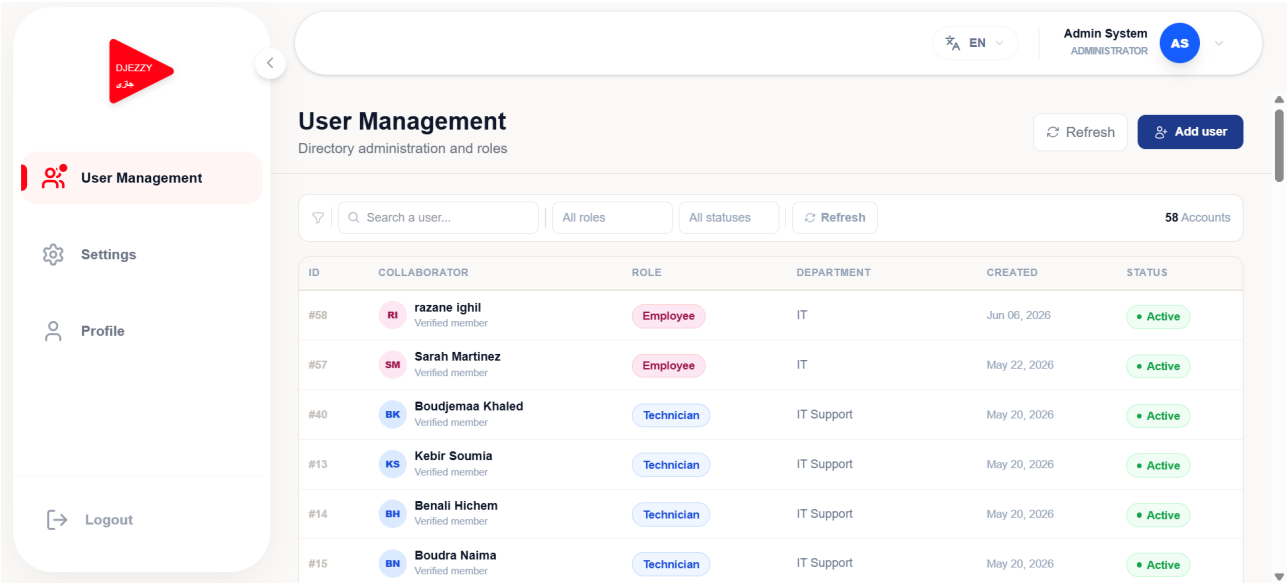


Figure 4.36: User management interface

Figure 4.37 and Figure 4.38 show the forms used to add a new user and to view, update, activate, or deactivate an existing user:

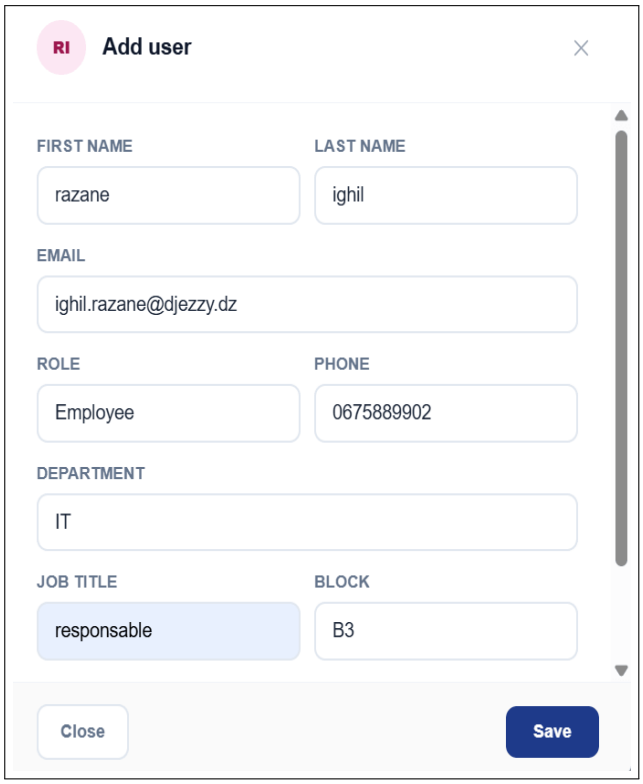


Figure 4.37: Add new user

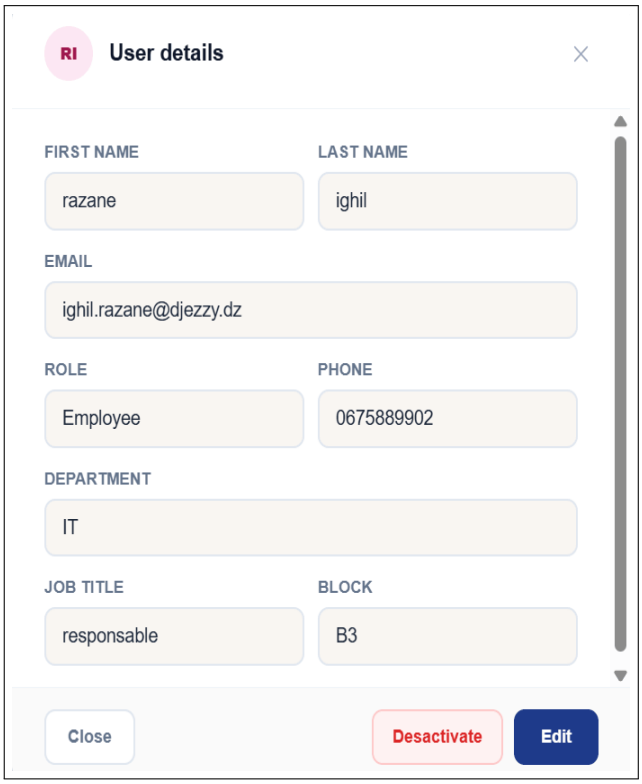


Figure 4.38: View and update user

4.5 Conclusion

In this chapter, we presented the hardware and software environment used to develop our platform, the deployment diagram, and some interfaces of the application.

General conclusion

In this project, we developed a new intelligent IT ticketing solution for the IT department of Optimum Telecom Algeria Djezzy in order to overcome the limitations of the existing system, which relied heavily on manual intervention for ticket assignment and management. This manual process caused delays, reduced efficiency, and complicated the resolution of incidents and employee requests.

The main objective of our project was to design and develop a new smarter and more efficient complete ticket management platform capable of automating certain tasks, improving ticket management, and facilitating communication between employees and IT technicians. To achieve this, we first studied the current system and analyzed how tickets were managed from their creation until their resolution. This step allowed us to better understand the company's needs, identify the existing limitations, and propose improvements adapted to the real working environment.

After the system study phase, we designed the proposed solution using UML diagrams, including use case diagrams, sequence diagrams, activity diagrams for the new ticket resolution workflow, and a class diagram, in order to build a structured architecture for the application and the database. After this step, we moved to one of the most important contributions of this project, which was the integration of an artificial intelligence module for automatic ticket classification. This was also the most challenging part of the project, as it represented our first practical experience in the field of artificial intelligence. We worked on data collection, model training, testing several classification models, and selecting the most suitable model for integration into the system. We also addressed the challenge of handling French tickets by integrating a translation process to support both English-trained models and the company's working language.

Finally, we completed the project by implementing the intelligent ticketing platform with features such as automatic ticket classification, centralized ticket management, a better workspace for technicians, and integrated communication between employees and technicians. This project allowed us to strengthen our technical skills, particularly in software development, discover new technologies and learn how to work with them, as well as improve our abilities in project management and artificial intelligence. It also allowed us to gain concrete experience in the field of artificial intelligence and to understand how classification models work through learning from data. In addition, we gained valuable experience in a real working environment, especially by applying our practical knowledge. This real-world experience enabled us to better understand enterprise environment and to propose improvements that were appreciated by the host company.

Although the proposed solution successfully addressed several limitations of the existing system, there are still many possible improvements for future work. These include adding advanced statistics and dashboards, extending language support especially for Arabic, integrating an intelligent chatbot to assist employees in resolving simple issues, and improving the AI model with more specific categories and larger training datasets to achieve better classification performance in the future.

Bibliography

[1] Djezzy. It department overview. Technical report, Djezzy Optimum Telecom Algeria, 2024. Internal document. Not publicly available.

[2] Benoît Charroux, Aomar Osmani, and Yann Thierry-Mieg. UML 2 : Pratique de la Modélisation. Pearson Education France, 2nd edition, 2007.

[3] Pascal Roques. UML 2 par la pratique : Études de cas et exercices corrigés. Eyrolles, 5th edition, 2006.

[4] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. IEEE Computational Intelligence Magazine, 2018.

[5] Corinna Cortes and Vladimir Vapnik. Support-vector networks. Machine Learning, 20(3):273–297, 1995.

[6] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In Proceedings of the European Conference on Machine Learning (ECML), 1998.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of NAACL-HLT, pages 4171–4186, 2019.

[8] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, 2019. arXiv preprint arXiv:1910.01108.

[9] Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11):2673–2681, 1997.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.

[11] Marcin Junczys-Dowmunt et al. Marian: Fast neural machine translation in c++. arXiv preprint arXiv:1804.00344, 2018.

[12] Ashish Vaswani et al. Attention is all you need. arXiv preprint arXiv:1706.03762, 2017.

Webography

[13] Djezzy. Official website, 2026. Available at: https://www.djezzy.dz.Accessed on April 13, 2026.

[14] Karthikeyan Palaniv. It helpdesk tickets – synthetic dataset, 2024. Kaggle,filehelpdesk_tickets.csv. Available at: https://www.kaggle.com/datasets/karthikeyanpalaniv/it-helpdesk-tickets-synthetic-dataset.Accessed on April 2,2026.

[15] Kameron Brooks. Synthetic it call center tickets v1, 2024. Kaggle, file: synthetic-it-call-center-tickets.csv. Available at: https://www.kaggle.com/datasets/kameronbrooks/synthetic-it-call-center-tickets-v1. Accessed on April 1, 2026.

[16] Adison Goh. It ticket classification dataset, 2024. Kaggle. Available at: https://www.kaggle.com/datasets/adisongoh/it-service-ticket-classification-dataset. Accessed on April 1, 2026.

[17] Noise144. ticket_classification_it_en, 2024. HuggingFace. Available at: https://huggingface.co/datasets/Noise144/ticket_classification_IT_EN. Accessed on April 1, 2026.

[18] Console-AI. It-helpdesk-tickets, 2024. HuggingFace. Available at: https://huggingface.co/datasets/Console-AI/IT-helpdesk-synthetic-tickets. Accessed on April 1, 2026.

[19] Alezzandro. itsm_tickets, 2024. HuggingFace. Available at: https://huggingface.co/datasets/alezzandro/itsm_tickets. Accessed on April 1, 2026.

[20] Hugging Face. Hugging face models, 2024. Available at: https://huggingface.co/models. Accessed on May 7, 2026.

[21] Mozilla Developer Network. Javascript – mdn web docs, 2024. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript. Accessed on April 18, 2026.

[22] Python Software Foundation. Python programming language, 2024. Available at: https://www.python.org. Accessed on May 7, 2026.

[23] W3C. Html hypertext markup language, 2024. Available at: https://www.w3.org/html. Accessed on April 15, 2026.

[24] W3C. Css cascading style sheets, 2024. Available at: https://www.w3.org/Style/CSS. Accessed on April 15, 2026.

[25] ISO/IEC. Iso/iec 9075-1:2023 – information technology – database languages – sql, 2023. Available at: https://www.iso.org/standard/76583.html. Accessed on May 7, 2026.

[26] Meta Platforms, Inc. React – a javascript library for building user interfaces, 2024. Available at: https://react.dev. Accessed on April 15, 2026.

[27] Tailwind Labs. Tailwind css documentation, 2024. Available at: https://tailwindcss.com/docs. Accessed on April 15, 2026.

[28] OpenJS. Node.js, 2024. Available at: https://nodejs.org/en/docs.Accessed on April 15,2026.

[29] OpenJS. Express.js, 2024. Available at: https://expressjs.com.Accessed on April 15,2026.

[30] Prisma Data, Inc., 2024. Available at: https://www.prisma.io/docs.Accessed on April 15,2026.

[31] Sebastián Ramírez. Fastapi documentation, 2024. Available at: https://fastapi.tiangolo.com. Accessed on April 15, 2026.

[32] Hugging Face. Transformers – state-of-the-art machine learning, 2024. Available at: https://huggingface.co/docs/transformers. Accessed on April 15, 2026.

[33] PyTorch, 2024. Available at: `https://pytorch.org/docs`. Accessed on April 15, 2026.

[34] scikit-learn developers, 2024. Available at: `https://scikit-learn.org`.Accessed on April 15,2026.

[35] The PostgreSQL Global Development Group. Postgresql documentation, 2024. Available at: `https://www.postgresql.org/docs`. Accessed on May 7, 2026.

[36] Microsoft. Vs code, 2024. Available at: `https://code.visualstudio.com`.Accessed on April 15,2026.

[37] GitHub, Inc. Github, 2024. Available at: `https://github.com`. Accessed on April 15, 2026.

[38] Google Colaboratory, 2024. Available at: `https://colab.research.google.com`. Accessed on April 15, 2026.