MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

SUMMER PRACTICE REPORT
CENG 400

Student Name & Number: Arda Alper Gölbaşı 2580595

Organization Name: Ikon Arge Teknoloji A.Ş

Total Working Days: 30

Address: ODTÜ Teknokent Silikon Blok Bodrum Kat No 10 Çankaya/Ankara.

Student's Signature

Organizational Approval

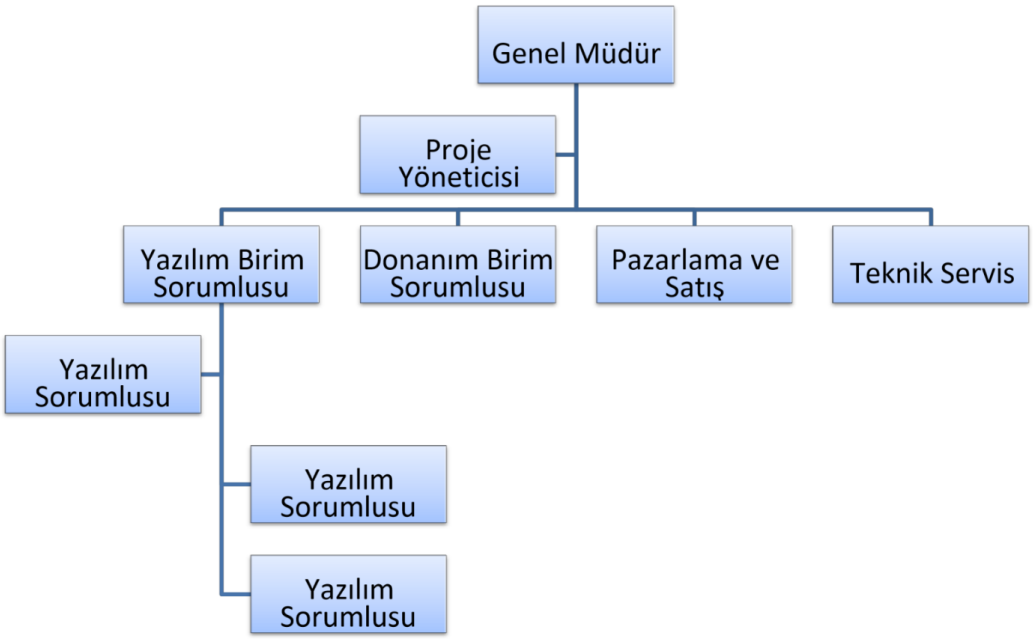
Contents

1Description of the Company

1.1 Company Name: Ikon Arge Teknoloji A.Ş.

1.2 Company Location: ODTÜ Teknokent Silikon Blok Bodrum Kat No 10 Çankaya/Ankara.

1.3 Organizational Structure of the Company



1.4 Number and Duties of Engineers Employed

As of today, IKON Arge Teknoloji A.Ş. employs over 30 engineers, primarily graduates of Electrical-Electronics Engineering and Computer Engineering. These engineers play a crucial role in the company's research and development (R&D) activities, particularly in the fields of mobile device management systems, secure communication solutions, and mobile network infrastructures. Their main duties include:

- Designing and developing secure and efficient software and hardware systems.
- Conducting research and innovation in mobility and network security.
- Implementing and testing enterprise-level solutions for both public and private sector clients.
- Supporting the export activities of the company by tailoring solutions for international markets.

1.5 Main Area of Business

IKON Arge Teknoloji A.Ş. operates in the information and communication technologies (ICT) sector, with a strong emphasis on mobile management solutions and secure communication platforms. The company's core business areas include:

- Mobile Device Management (MDM) systems for enterprises and institutions.
- Secure communication solutions for public and private organizations.
- Mobile networking technologies designed to improve productivity and ensure safe environments for organizations.
- Development of software and hardware products with a focus on scalability, reliability, and security.

The company not only serves the domestic market but also exports its self-developed products abroad, with offices located in Dubai, Singapore, and Ankara.

1.6 A Brief History of the Company

IKON Arge Teknoloji A.Ş. was founded in 2009 with the mission of developing mobile management solutions that enhance organizational productivity and security. Initially

established with a team of young and dynamic engineers, the company positioned itself in the high-tech sector by offering innovative solutions in mobile technologies.

Over the years, IKON expanded its expertise to include secure communication systems and mobile networking infrastructures, strengthening its position as a trusted technology provider. Its operations are currently based in ODTÜ Teknokent (Ankara), where it carries out extensive research and development projects.

With a vision to transform Turkey from a country that primarily consumes technology into one that exports high-tech products, IKON Arge has grown into a company that actively competes in the global market. Today, it generates a significant portion of its revenue from the export of software and hardware solutions, reflecting its success in becoming an international technology exporter

2 INTRODUCTION

This report presents the details of the summer practice carried out at IKON Arge Teknoloji A.Ş. The internship was conducted within the scope of an R&D project, with the primary goal of gaining hands-on experience in the full lifecycle of software development and artificial intelligence systems.

The objectives of the internship included performing requirements analysis and contributing to the technical design, conducting a literature review to identify state-of-the-art approaches, and selecting and applying suitable software development methodologies. Furthermore, the work involved project setup, validation, functional and performance testing, as well as the preparation of project documentation, ensuring active contribution to all phases of the project.

The central focus of the project was the design and implementation of a secretary chatbot, where different language models were tested and compared to determine the most suitable approaches. This provided the opportunity to explore the integration of natural language processing techniques into a practical, user-oriented system.

The expectation from this internship was to gain experience in planning and executing an end-to-end AI project, from initial analysis and design to implementation and testing. Through this process, valuable insights were obtained on how modern AI-driven systems are developed in an industrial R&D environment, bridging theoretical knowledge with real-world application.

3 PROJECT

3.1 Researched and Tested Models

During the internship, several language models and computer vision approaches were explored to determine the most suitable solution for developing an intelligent secretary chatbot capable of handling both text-based interactions and real-time visual analysis. Each model was evaluated in terms of its capabilities, performance, and adaptability to the project requirements.

Rasa

Rasa was the first framework tested. It provided a reliable and structured approach to intent recognition and dialogue management, making it effective for rule-based scenarios. However, while it was efficient for predefined workflows and FAQ-style conversations, it lacked the flexibility and adaptability required for dynamic tool usage. Therefore, although Rasa was fast and robust, it was not sufficient for a project that demanded contextual understanding and tool integration.

GPT-OSS:20b (Ollama)

GPT-OSS emerged as the most effective model for this project. It offered natural and contextually rich responses while being fully operational in an offline environment on company servers. Unlike Rasa, GPT-OSS was able to dynamically select and execute tools (e.g., guest query, cargo query, door control) based on the user's input. Its performance was consistent, with response times averaging around two seconds even when tools were invoked. This made GPT-OSS the backbone of the final chatbot system.

LLaMA (3/4 series)

LLaMA models were also tested due to their open-source availability and strong performance in text generation tasks. They showed potential for fine-tuning and domain-specific customization, offering flexibility in long-context conversations. However, limitations were observed in terms of Turkish language support and more complex tool integration. While promising for research purposes, they were not as seamless as GPT-OSS for production deployment.

GPT-4o

OpenAI's GPT-4o was evaluated as a benchmark for high-quality responses and multimodal capabilities. Its ability to process text, images, and audio made it highly attractive for advanced chatbot systems. However, due to its reliance on API calls, the

system lacked local control and raised concerns about latency, cost, and data security. Although its response times were very fast (around one second), the project's goal of building a locally hosted AI assistant led to the preference for GPT-OSS instead.

YOLOv11

For the visual perception component, YOLOv11 was tested as the primary object detection model. It demonstrated high accuracy and speed in detecting harmful objects such as knives, axes, and scissors. The model was trained and evaluated using the Harmful Objects dataset from Roboflow, achieving superior performance compared to general-purpose datasets like COCO. YOLOv11's ability to perform real-time detection even on personal hardware made it the optimal choice for the security aspect of the project.

Qwen2.5-VL

Qwen2.5-VL was explored as a multimodal alternative, capable of analyzing both images and text simultaneously. While it showed promising results in vision-language tasks such as object recognition and reasoning over visual scenes, its integration into the system introduced significant latency. The simultaneous use of Qwen2.5-VL alongside a language model slowed down response times dramatically, making it impractical for real-time office security applications. Nonetheless, it was found to be a strong candidate for future research and optimized setups.

Summary

The research and experimentation stage provided a clear comparison between traditional dialogue frameworks, open-source large language models, API-based solutions, and multimodal systems. Ultimately, the combination of GPT-OSS:20b for natural language understanding and YOLOv11 for real-time object detection was selected as the most balanced and efficient approach for the secretary chatbot project.

3.2 Implementation and Final Structure

3.2.1 System Architecture

The system is built as a modular, service-oriented stack that combines an offline large language model with real-time computer vision and a lightweight web UI. At a high level:

- LLM & Orchestration: GPT-OSS:20b runs on a company server via Ollama. Conversation logic is orchestrated with LangGraph, which uses two nodes: call_model (response generation) and tool_node (privileged actions via tools).
- Backend API: A FastAPI service mediates all requests, maintains application state, invokes tools, and serves admin/CRUD endpoints.
- Tools Layer: Privileged functions for guest/cargo/staff queries, door control (with password), security dispatch, and email notification.
- Computer Vision Service: A background worker loads YOLOv11 and analyzes the live camera stream for harmful objects; detections flip the alarm state.
- Database: SQLite (data/security_data.db) persists guests, cargos, emergencies, and staff records.
- Frontend: A static web UI (chat panel + status widgets) and a separate admin console for database management.

Runtime topology

- LLM host (Ollama): Launched after SSH on the intra-net server; exposed via OLLAMA_HOST.
- Backend: Runs locally with Uvicorn; communicates with Ollama over LAN and with the browser over HTTP.
- Frontend: index.html for the operator UI (chat, TTS/STT, camera preview, door/alarm status) and admin.html for records management.

State & security model

- Agent state isolation: Operational flags (door/alarm/password attempts) are kept in server state and not directly exposed to the LLM; the backend injects only the minimal context per turn.
- Guarded tools: Functions that reveal sensitive data or change physical state (e.g., which_guest_of_staff, door_control) require correct credentials; failures do not leak information.
- Offline by design: The chosen LLM runs entirely on-prem, aligning with data-privacy and latency goals.

End-to-end data flows

1. Chat / Tooling flow
   UI → /chat → LangGraph call_model → (optional) tool_node → tool execution (DB/email/door) → updated state → response to UI.

2. Vision-triggered alarm flow
   Camera worker → YOLOv11 inference (harmful objects) → set alarm="Aktif" → UI polls /camera_status and updates siren/visuals.
3. Admin CRUD flow
   Admin UI → /admin/login (password check) → /admin/records (read) / /admin/records/add (create) / /admin/records/delete (delete) → tables re-rendered.
4. Notification flow
   Tool send_guest_email → SMTP (TLS) → message composed with inline branding → delivery or error surfaced to chat/admin.

This architecture cleanly separates concerns: the LLM handles language and decision-making; LangGraph governs tool use; the backend enforces policy and state; YOLOv11 delivers low-latency perception; SQLite ensures simple persistence; and the web UIs provide operator and admin touchpoints.

3.2.2 Backend Modules

The backend of the project was implemented as a set of modular Python scripts, each responsible for a specific part of the system. This modular design ensured clear separation of concerns and made the system easier to maintain, extend, and debug.

graph.py
This module defines the LangGraph structure that orchestrates the interaction between the LLM and external tools. It includes two main nodes: call_model, which sends conversation history to the GPT-OSS model and retrieves responses, and tool_node, which manages calls to external functions. The AgentState structure, implemented as a TypedDict, stores the message history as well as operational flags such as door, alarm, and password attempt states. Importantly, these states are not directly visible to the LLM but are injected into prompts by the backend.

```python
def get_graph():
    """Returns the compiled LangGraph application."""
    # Build the graph that defines the flow of the agent.
    graph = StateGraph(AgentState)

    graph.add_node("call_model", call_model)
    graph.add_node("tool_node", tool_node)

    graph.set_entry_point("call_model")

    graph.add_conditional_edges(
        "call_model",
        should_continue,
        {
            "continue": "tool_node",
            "end": END,
        }
    )

    graph.add_edge("tool_node", "call_model")

    return graph.compile()
```

tools.py

This file contains the critical set of functional tools that the LLM can call through the LangGraph orchestration. Tools include:

- guest(query): retrieves guest records from the database.
- which_guest_of_staff(query, password): queries staff's guest list with authentication.
- cargo(query): searches cargo records by personnel name or cargo ID.
- security(query): dispatches security in emergencies.
- door_control(query, password): manages door opening/closing, secured with a password.
- staff_info(query): queries personnel information.
- send_guest_email(...): sends email notifications to staff when a guest arrives.

```python
@tool
def cargo(query: str) -> str:
    """
    Kargolarla ilgili bilgileri aramak ve listelemek için kullanılır.
    Bu araç, şirkete gelen veya gönderilen kargoların durumunu sorgulamak için uygundur.
    Sadece kargo ile ilgili sorular için kullanılmalıdır. Diğer konular için kullanılmamalıdır.

    Sağlayabileceği bilgiler:
    - Kargonun teslim edilip edilmediği veya mevcut durumu
    - Kargo ID'si üzerinden takip
    - Belirli bir personel adına gelen kargolar
    - Hangi şirketten (kargo firması) geldiği bilgisi

    Args:
        query (str): Kargo hakkında aranacak anahtar kelime veya bilgi
                     (örneğin personel adı, kargo ID'si, şirket adı veya kargo durumu).

    Returns:
        str: Eşleşen kayıtların listesi. Her satırda şu bilgiler bulunur:
             Personel adı, Kargo ID, Kargo firması, Kargonun durumu.
             Eğer kayıt bulunmazsa "Kargo bilgisi bulunamadı." döner.
    """
    cursor = sqlite3.connect('./data/security_data.db').cursor()
    cursor.execute("SELECT personnel_name, cargo_id, company, status FROM cargos WHERE personnel_name LIKE ? OR cargo_id LIKE ? OR company LIKE ?
                   ('%'+query+'%', '%'+query+'%', '%'+query+'%', '%'+query+'%'))
    matches = cursor.fetchall()

    if not matches:
        return "Kargo bilgisi bulunamadı."

    result = []
    for match in matches:
        result.append(f"Personel: {match[0]}, Kargo ID: {match[1]}, Şirket: {match[2]}, Durum: {match[3]}")

    return "\n".join(result)
```

database.py

This script manages SQLite database operations. It creates and initializes the database with tables (guests, cargos, emergencies, staff) if they do not exist, and provides CRUD functions:

- get_all_records(): fetches all entries from the database in dictionary form.
- add_record(table_name, record_data): adds a new record with SQL injection protection.
- delete_record(table_name, record_id): deletes a record by ID.

```python
def add_record(table_name, record_data):
    """Belirtilen tabloya yeni bir kayıt ekler."""
    conn = get_db_connection()
    cursor = conn.cursor()

    # SQL enjeksiyonunu önlemek için güvenli parametre kullanımı
    columns = ', '.join(record_data.keys())
    placeholders = ', '.join(['?' for _ in record_data])
    values = tuple(record_data.values())

    try:
        # Sorguyu yazdırmak hata ayıklamada yardımcı olabilir
        print(f"Executing SQL: INSERT INTO {table_name} ({columns}) VALUES ({placeholders}) with values: {values}")
        cursor.execute(f"INSERT INTO {table_name} ({columns}) VALUES ({placeholders})", values)
        conn.commit()
        return {"success": True, "message": "Kayıt başarıyla eklendi."}
    except sqlite3.OperationalError as e:
        # Hata mesajını daha açıklayıcı yapın
        return {"success": False, "message": f"Kayıt ekleme hatası: {e}. Gönderilen veriler: {record_data}"}
    except Exception as e:
        return {"success": False, "message": f"Beklenmedik bir hata oluştu: {e}"}
    finally:
        conn.close()
```

mailsender.py

This module handles email communication via SMTP. The send_email() function builds a multipart message (text + inline image) and connects to Gmail's SMTP server over

TLS. If successful, it delivers guest notifications to the intended staff member. Errors are caught and logged, ensuring reliability.

main.py

This is the central backend service, built with FastAPI. It exposes endpoints for chat (/chat), camera status (/camera_status), and admin functions (/admin/login, /admin/records, /admin/records/add, /admin/records/delete). It integrates with LangGraph for dialogue orchestration, maintains application state, and runs YOLOv11 inference in a background thread. Pydantic models are used for request validation, ensuring robust API communication.

Together, these backend modules form the operational core of the project: LangGraph and the LLM for reasoning, Tools for real-world actions, Database for persistence, Mailsender for notifications, and FastAPI for orchestration.

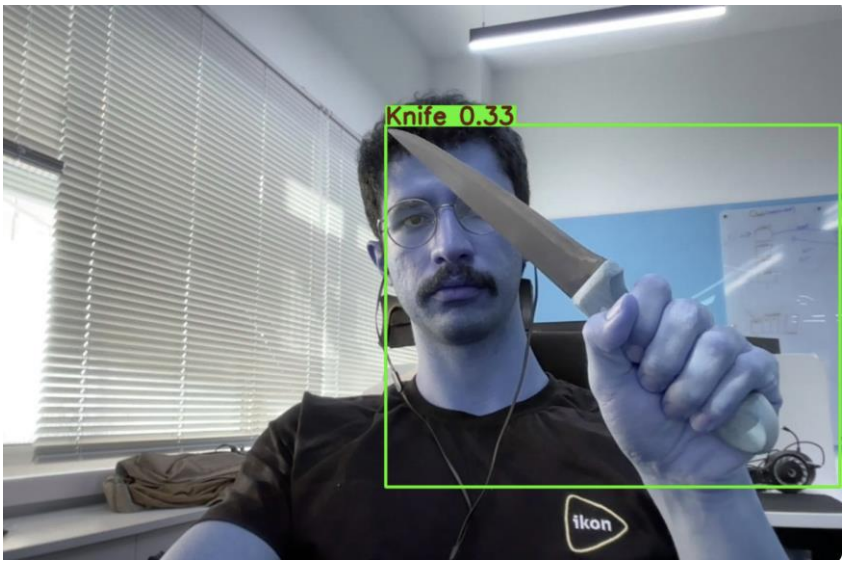
```python
@app.post("/admin/records/add")
async def add_security_record(record_data: RecordData):
    """
    Belirtilen tabloya yeni bir güvenlik kaydı ekler.
    """
    try:
        # Pydantic modeli sayesinde gelen veriyi otomatik olarak doğru formata alıyoruz
        table_name = record_data.table_name
        record = record_data.record

        result = add_record(table_name, record)

        if result["success"]:
            return {"message": result["message"]}
        else:
            raise HTTPException(status_code=400, detail=result["message"])
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Kayıt ekleme hatası: {str(e)}")
```

3.2.3 Computer Vision Integration

A critical component of the project was the integration of computer vision to enable the system to detect threats in real time and trigger automated security responses. This functionality was achieved using the YOLOv11 object detection model, combined with a continuously running camera feed.

Model Selection and Dataset

The YOLOv11 model was chosen due to its high accuracy and real-time performance on commodity hardware. It was trained and tested using the Harmful Objects dataset from Roboflow, which contains annotated images of dangerous items such as knives, axes, scissors, and hammers. Compared to general-purpose datasets like COCO, this domain-specific dataset produced significantly better results in detecting potential threats relevant to an office environment.



Integration into the Backend

The computer vision module was implemented as a background thread in the backend (visionYOLO_test.py). This thread continuously processes frames from the camera stream using the YOLOv11 model. The detection pipeline operates as follows:

1. Capture frames from the video feed.
2. Run YOLOv11 inference to identify objects.
3. Check whether any detected objects belong to the "harmful" class list.
4. If a harmful object is detected, set the alarm state to "Aktif."
5. The alarm state is then exposed through the /camera_status API endpoint, allowing the frontend to update visuals and play an alarm sound.

Alarm Mechanism

When a harmful object is detected:

- The backend updates the global state to reflect an active alarm.
- The script.js module on the frontend periodically polls the /camera_status endpoint.
- If the alarm status is active, the user interface automatically displays a warning, plays the siren audio file, and updates the alarm icon to a red "active" state.

Performance and Testing

The YOLOv11 model consistently achieved real-time inference, processing frames within milliseconds even on a personal laptop. This responsiveness ensured that alarms were triggered almost instantaneously upon object detection. Testing scenarios included:

- Simulated office environments with knives and scissors.
- False-positive checks using harmless objects with similar shapes.

- Stress tests with multiple frames per second to ensure stability over extended runtime.

By combining YOLOv11 with the backend orchestration, the system gained a reliable autonomous perception layer, transforming the chatbot from a purely conversational assistant into a real-time security agent capable of responding to physical threats.

3.2.4 Frontend and User Interfaces

The frontend of the project was designed to provide both operational control for end users and administrative management for supervisors. It consisted of two separate interfaces:

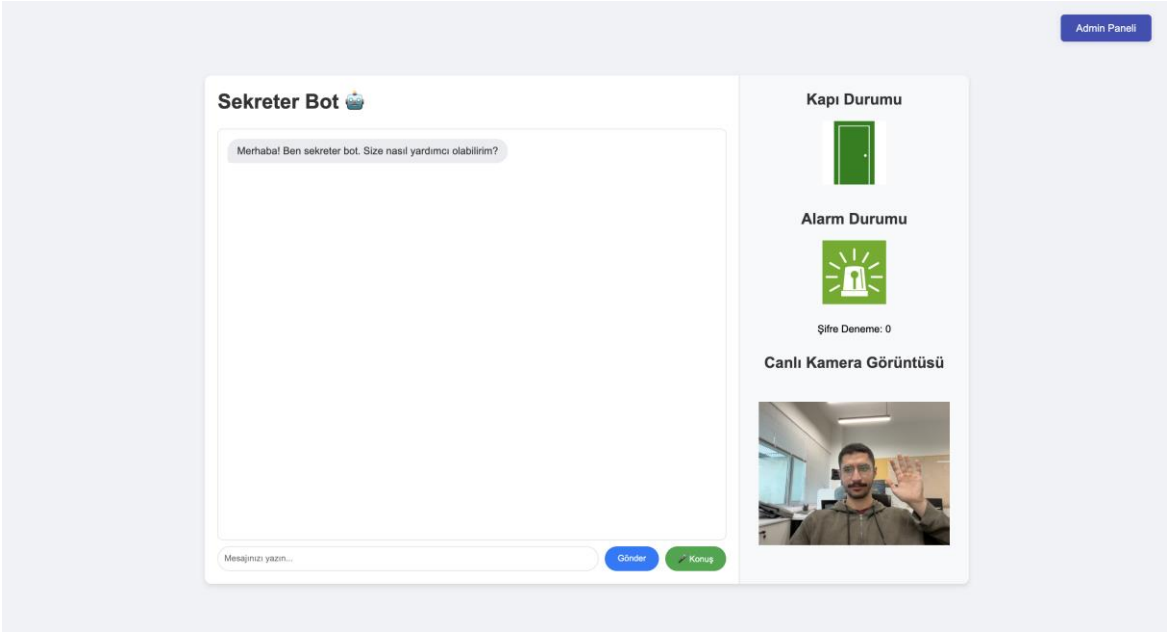
User Interface (index.html & script.js)
The index.html file served as the main user-facing panel. It integrated a chatbot interface, real-time status indicators, and multimedia elements:

- Chat area: Users could type messages, send them with a button, or use speech-to-text (STT) functionality to interact by voice.
- TTS responses: AI responses were spoken aloud via text-to-speech, improving accessibility.
- Door status display: An image dynamically switched between "open" and "closed" states depending on system responses.
- Alarm status display: Visual icons (green = passive, red = active) and an audio siren were automatically triggered when harmful objects were detected.
- Password attempts counter: Displayed the number of incorrect admin login attempts.
- Live camera stream: A video element displayed the real-time feed from the local webcam, synchronized with the backend's computer vision module.

The script.js file contained the logic for:

- Handling user input and sending it to the backend /chat endpoint.
- Managing state updates for door, alarm, and password attempts.
- Polling /camera_status every second to detect threats and trigger the alarm if necessary.
- Handling admin panel access through a password modal and securely redirecting authenticated users.

This interface transformed the chatbot into a multimodal assistant, bridging text, audio, and visual interaction.



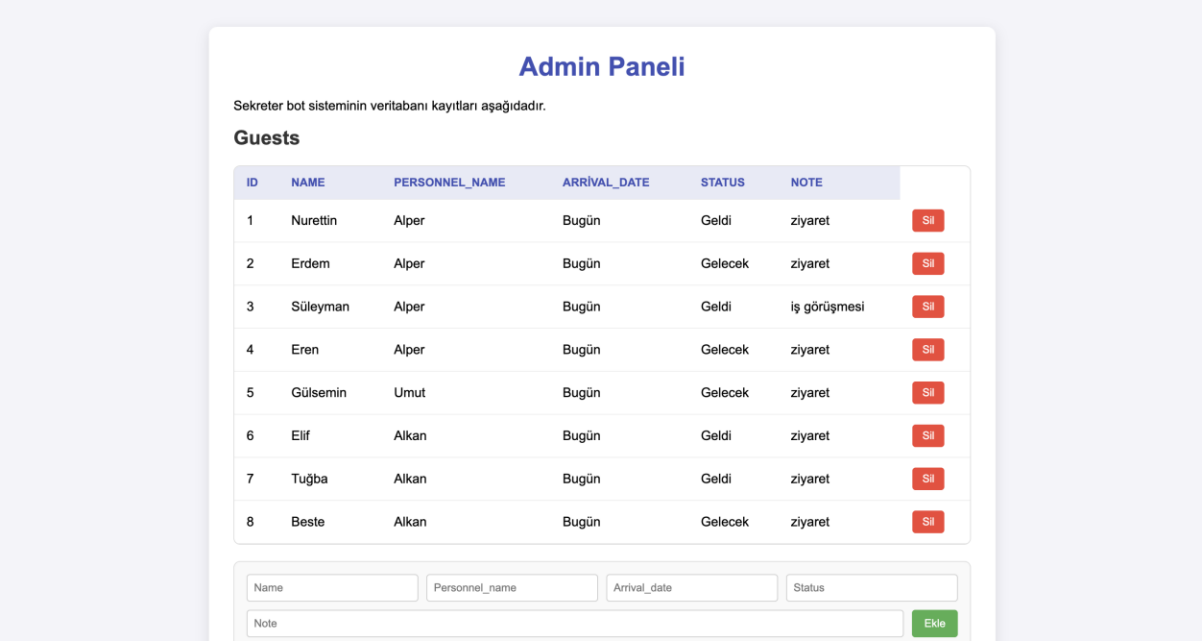
Admin Interface (admin.html & admin.js)

The admin.html file provided a database management console for supervisors. Its main elements included:

- Dynamic tables: Displayed data from the guests, cargos, emergencies, and staff tables.
- Add form: A form dynamically generated input fields depending on the table schema, allowing administrators to insert new records.
- Delete buttons: Each row included a delete option for removing outdated or incorrect entries.
- Status messages: Informational and error messages displayed in response to backend operations.

The admin.js file implemented logic to:

- Fetch all database records via the /admin/records API.
- Render tables dynamically with headers and rows.
- Handle record creation via /admin/records/add.
- Manage record deletion with confirmation prompts through /admin/records/delete.
- Provide success/error feedback with temporary messages that disappear after a few seconds.



Security and Usability

- Access to the admin panel required password verification, enforced by the /admin/login API.
- Both UIs were designed with responsive layouts and simple visuals to ensure usability even in time-sensitive scenarios.
- By separating end-user and admin functionalities, the system ensured that operational controls and database management remained isolated, minimizing the risk of accidental interference.

4 ORGANIZATION

4.1 Organization and Structure

IKON Arge Teknoloji A.Ş. is structured as a research and development company operating within the ODTÜ Teknokent ecosystem. The organization is composed of specialized teams focusing on software and hardware development for mobile management systems, secure communication platforms, and mobile network infrastructures.

The company maintains an agile structure with cross-functional collaboration. Each project team typically includes:

- Software engineers, responsible for system design, implementation, and integration.
- Hardware engineers, focusing on embedded systems and device-level solutions.
- R&D specialists, conducting feasibility studies, prototyping, and technology validation.
- Project managers, overseeing timelines, resource allocation, and customer requirements.

This organizational model enables IKON Arge to balance innovation and efficiency, allowing multiple projects to run in parallel. While each team operates independently on its assigned tasks, collaboration between units is common, particularly when integrating software and hardware components into complete solutions.

4.2 Methodologies and Strategies Used in the Company

IKON Arge Teknoloji A.Ş. follows an R&D-driven development model supported by agile practices. Projects typically begin with requirements analysis and literature research, followed by design, implementation, and testing phases. The company emphasizes rapid prototyping, iterative validation, and documentation at each stage, ensuring both compliance with standards and adaptability to client needs.

This strategy enables IKON Arge to deliver innovative, secure, and scalable solutions while maintaining flexibility in response to evolving technologies and market demands.

5 Conclusion

The internship at IKON Arge Teknoloji A.Ş. provided a comprehensive experience in both research and implementation within the scope of an R&D project. The primary goal of designing and testing a secretary chatbot system was successfully achieved by combining large language models with real-time computer vision and database-driven management tools. This project not only allowed me to apply my academic knowledge but also gave me the opportunity to work with state-of-the-art technologies such as GPT-OSS and YOLOv11 in an industrial setting.

Throughout the internship, I was able to explore and compare multiple models and frameworks, including Rasa, GPT-OSS, LLaMA, GPT-4o, YOLOv11, and Qwen2.5-VL. This research phase deepened my understanding of the strengths and weaknesses of

each approach and helped me identify the optimal balance between flexibility, performance, and practicality. The final choice of GPT-OSS for language processing and YOLOv11 for object detection demonstrated how open-source tools can be integrated into real-world applications effectively.

The implementation stage was equally valuable, as it involved the complete development of a modular system. From building the backend with FastAPI and LangGraph, to integrating a secure SQLite database, designing functional tools, and creating user-friendly interfaces, the project represented a full cycle of software engineering. Additionally, the inclusion of an admin panel and automated email notifications highlighted the importance of system maintainability and usability in real-world environments.

Overall, this internship experience allowed me to gain insights into the workflow of a technology-focused company and taught me how to transform theoretical concepts into practical solutions. By contributing to every stage of the project—from requirements analysis to testing and documentation—I not only improved my technical expertise but also strengthened my problem-solving, teamwork, and project management skills. This experience has significantly prepared me for future roles in the fields of artificial intelligence, computer vision, and secure software development.

6 APPENDIX