CS 492 - Senior Design Project II

Raporla.ai

Final Report

Kerim Eren - 22103759

Ömer Amir Akdağ - 22102918

Ali Cevat Erçal - 22103341

Göktuğ Serdar Yıldırım - 22103111

Efe Tokar - 22103299

Contents

1. Introduction

In the contemporary world, the health system has many challenges. Most of those challenges also apply to Turkey. When statements of the doctors from Turkey are evaluated, it appears that one of the biggest problems is that with the increasing number of patients, the time that doctors can spend per patient is severely limited. In order to fix this problem, automated technologies can be used.

One of the main solutions that many doctors need is the systems that can automatically make preliminary diagnosis. For example, for the fields like neurology or radiology, test results can be evaluated by image processing to make a preliminary diagnosis to help the doctors by reducing their time that is spent by evaluating the already evident findings. However, both the available data and necessary technology is not enough for this idea to become a senior project, even though it may be efficient.

So, our project Raporla.ai focuses on the other main problem that takes up doctors' time unnecessarily long: reports. Doctors must write formal report documents for their examinations and surgeries. These reports may take up unnecessary times since they should be written for every patient detailly. Especially in the fields like neurology, radiology, and pathology, the formal reports may be long and complex, therefore taking up a long time for doctors.

Furthermore, there is no standardized report format in any field. A doctor may write the same examination type’s report very differently from another doctor. This makes it difficult to understand and evaluate a patient's examinations by different doctors at different times.

By recognizing these challenges of the doctors from Turkey, Raporla.ai introduces an automated, speech-to-text report system that is specialized in Turkish and medical literature. By this, we aim to reduce the time spent on procedural things for doctors, and encourage them to focus on detailed examinations and treatments for the patients. Also, by introducing common templates, we aim to standardize the reports among different hospitals and different doctors. Last but not least, the application will also have a preliminary diagnosis suggestion system to create even more time for doctors by helping them to find out evident diagnosis in advance. This system will be purely based on medical datasets to minimize the model’s bias as much as possible.

2. Requirement Details

2.1 Functional Requirements

- The user can register the system with their credentials and login afterwards.
- The user can record their reports such as examination and surgery reports as audio files.
- The speech-to-text model will convert these recorded audio files to text files in Turkish and medical literature.
- The user can generate standardized reports using predefined templates based on the transcribed text.
- The system will warn the user if there are empty fields, before submitting the report.
- The user can create customized reports by removing unnecessary fields.
- The system provides anticipated diagnosis based on keywords taken from the report and verified against medical datasets like PubMed.
- The user can download the report.
- The user can delete an existing report.
- The user can delete their account.
- The user can search an existing report
- The user can select a report type that already exists.

2.2 Non-Functional Requirements

2.2.1 Usability

- The interface should be easy for common usage and require minimal training for doctors to use effectively.
- The application should be compatible with Turkish language and medical terminology [1].

2.2.2 Reliability

- The system should be performed under any condition without crashing or failing.
- Speech-to-text conversion should maintain high accuracy even with background noise or variation in voices [1].

2.2.3 Performance

- The application should process audio files and generate reports to create an ideal experience.
- Diagnosis suggestions should be retrieved and displayed within feasible and reasonable seconds after report generation [1].

2.2.4 Supportability

- The application should be easy to update for incorporating new medical datasets and templates [1].

2.2.5 Scalability

- The application must handle increasing numbers of users and simultaneous audio uploads without significant performance decline.
- The application should support the addition of new features like integration with hospital management systems [].

3. Final Architecture and Design Details

3.1. Overview

Raporla.ai is structured into four main, interconnected parts to provide a complete medical reporting solution. The Client Layer provides the user interface for doctors and administrators, allowing them to interact with the system. The Communication Layer acts as a secure messenger, handling all data exchange between the user interface and the processing core. The Backend Layer is the "brains" of the operation, responsible for all the processing, including speech-to-text conversion, report generation, user management, and AI functionalities. Finally, the Storage Layer securely stores all user data, generated reports, and the AI models that power the system. These four parts work together seamlessly to provide the functionality of our project.

3.2. Subsystem decomposition

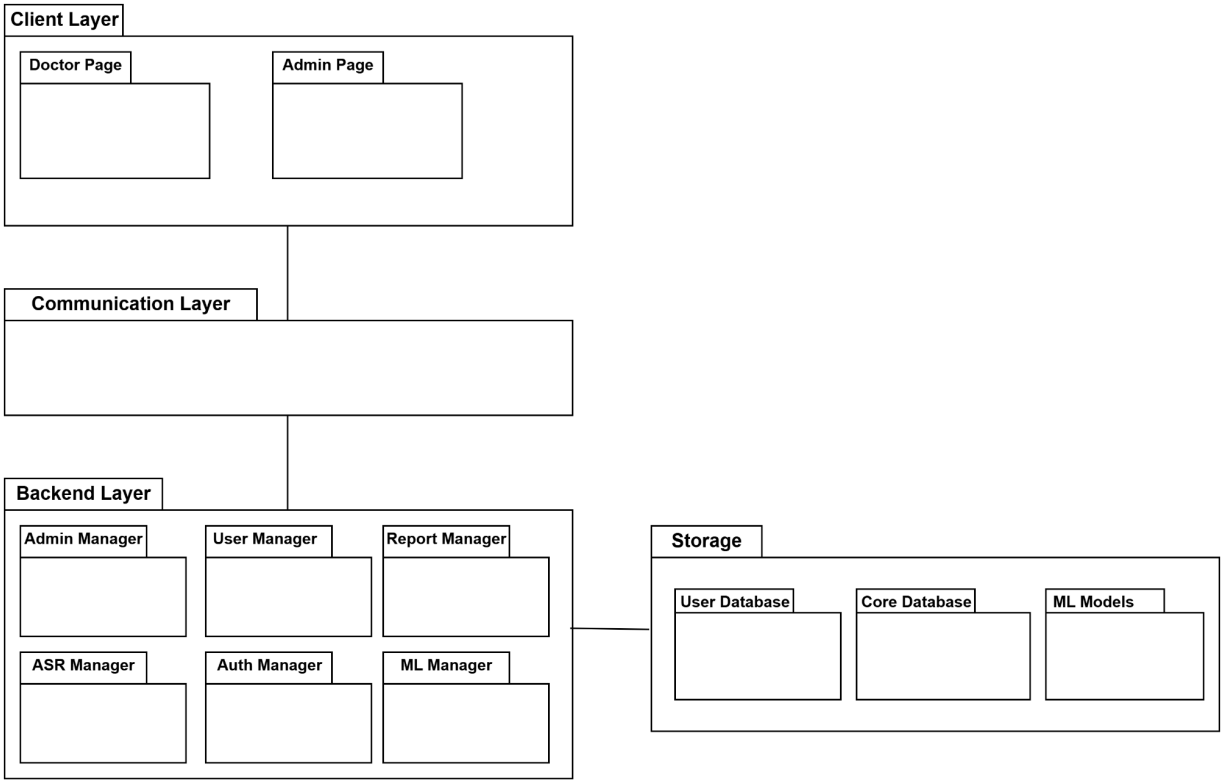

Figure 1: Subsystem Decomposition of our project

The **Client Layer** represents the user interface, tailored for both medical professionals and administrators. The Doctor Page provides a user-friendly environment for doctors to dictate reports via speech-to-text, review generated reports, and manage patient information, ensuring seamless integration into their clinical workflows. Conversely, the Admin Page empowers system administrators with tools to monitor user activity, manage roles and permissions, update system configurations, and access logs and analytics, enabling effective platform oversight.

The **Communication Layer** acts as a vital bridge between the client and backend systems. It manages data exchange, ensuring secure transmission for speech-to-text processing, report generation, and user management. This layer also facilitates API integrations, connecting client actions with backend services such as authentication, report generation, and machine learning tasks, thereby ensuring smooth and efficient data flow and service interaction.

At the core of the system lies the **Backend Layer**, the processing hub responsible for business logic and user request execution. This layer is modularized through several managers. The Admin Manager oversees system-wide settings and configurations, maintaining system integrity and scalability. The User Manager handles user-related functionalities, including profile

management, role assignments, and personalized system behavior. The Report Manager generates and manages medical reports, processing speech-to-text outputs and integrating patient data. The ASR Manager converts audio input into text using fine-tuned machine learning models, optimized for medical terminology. The Auth Manager ensures secure access through robust authentication and authorization processes. Finally, the ML Manager manages machine learning models, ensuring their efficient deployment and relevance.

The **Storage Layer** serves as the data management backbone, organizing and maintaining all essential data. The User Database stores user-specific information, while the Core Database acts as the central repository for system-critical data, including generated reports, system configurations, and operational logs. It also houses admin-specific data. The ML Models component contains trained machine learning models, facilitating version control and efficient deployment for AI functionalities such as Automatic Speech Recognition.

3.3. Hardware/Software Mapping

This section describes how the software components of Raporla.ai are deployed onto physical or virtual hardware. Our architecture is designed to separate the user interface from the core processing and data storage, largely leveraging cloud infrastructure for scalability and accessibility.

- **Client Layer:** The user interface software (Frontend) runs on the user's device (desktop computer, laptop, tablet, or smartphone) within a standard web browser. It requires no specific hardware beyond a device capable of running a modern browser and accessing the internet.
- **Communication Layer & Backend Layer:** The server-side logic, including the API Gateway, user management, report management, ASR processing, and ML functionalities, runs on cloud-based server infrastructure. This allows the system to handle requests from multiple users simultaneously and scale resources as needed. The speech-to-text and AI models (ASR Manager, ML Manager) are processed on this server infrastructure, potentially utilizing specific hardware capabilities provided by the cloud provider, such as GPUs for faster model inference, depending on the final deployment configuration.
- **Storage Layer:** The databases and stored ML Models are located on persistent storage within the cloud environment. This ensures data availability and durability.

In simple terms, the user interacts with the Client (Frontend) on their own device, which communicates over the internet with the Communication and Backend Layers hosted on servers in the cloud. The Backend then interacts with the Storage Layer, also located in the cloud, to retrieve or save data and access the necessary models.

3.4. Persistent Data Management

Persistent data management is crucial for Raporla.ai to store user information, reports, templates, and system configurations reliably and securely. Our approach utilizes a database and file storage, primarily managed within the Storage Layer as described in the architecture.

- **Database:** We use MongoDB which is a NoSQL database to store structured data. This includes:
- **ML Model Storage:** The trained machine learning models are stored as specific model formats within the cloud storage environment, accessible by the ML Manager and ASR Manager.
- **Security and Compliance:** Data is stored securely with access controlled by the Auth Manager and role-based permissions. Passwords are not stored directly but as hashed values using bcrypt. We implement measures to ensure compliance with data protection regulations like KVKK and GDPR, including considering data encryption and limited access to sensitive information.

Essentially, all critical application data that needs to persist beyond a user session or server restart is stored in the cloud-based MongoDB database or dedicated cloud storage for ML models, with security and access controls in place to protect sensitive medical and user information

4. Development/Implementation Details

4.1. Frontend

The user interface of the application is designed using the React library due to its component-based architecture, high performance, and structure that improves the developer experience.

4.1.1 Voice Reporting Tool:

The Voice Reporting Tool (Raporla!) page contains one of the main functions of the Raporla.ai. On this page, the user creates a new report item by selecting a report template. This created report item is automatically filled with the transcript extracted from the received voice recording or the uploaded audio file. The user can also manually change the transcript text as desired and make manual changes to the report (addition, removal, stylization). From this page, the user can also access the reports that have not yet been archived from the list on the left or archive the report of the discharged patient. The user can also delete the report or download it in PDF format. During the download in PDF format, the default information of the patient and the hospital that is in the system but not in the report is added to the report via a table, thus ensuring compatibility with the hospital format. This tool is also available on the mobile side of the

application. In addition, the “Operation Logbook”, developed for a similar purpose, is also available on the mobile side: It converts critical information provided by faculty members in university hospitals during surgery into a digital record by annotating it without losing live narration. In this way, students and residents can instantly access important explanations given during surgery during the surgical training process, review them later, and permanently document what they have learned. In this way, both loss of information is prevented and hospital education standards are raised.

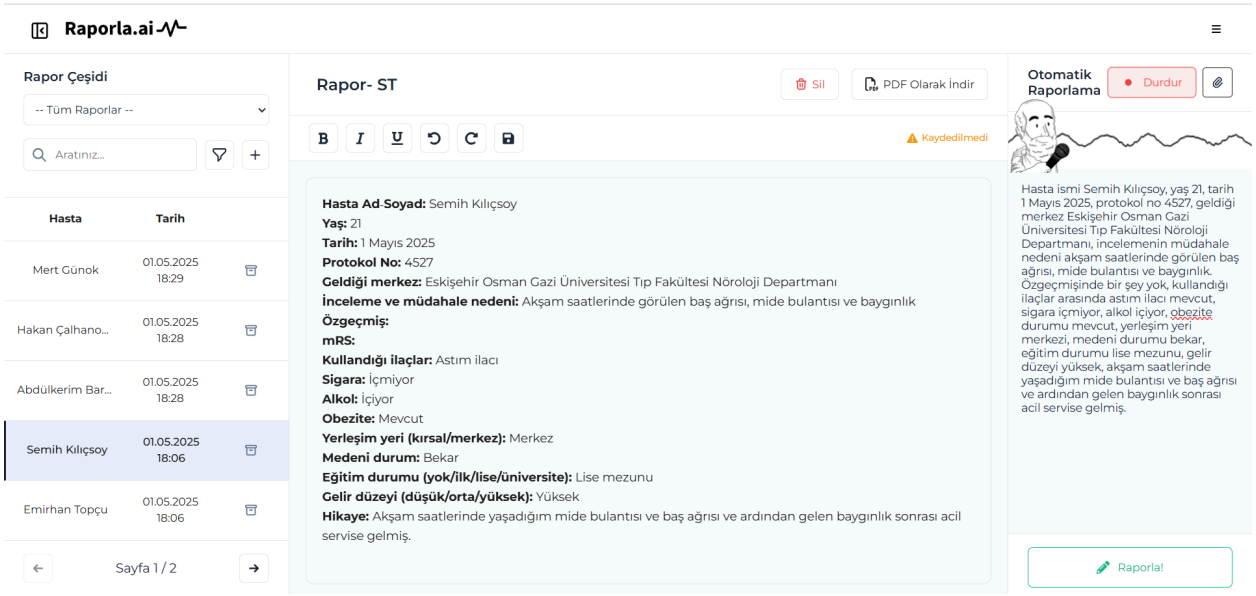

Figure 2: The Voice Reporting Tool

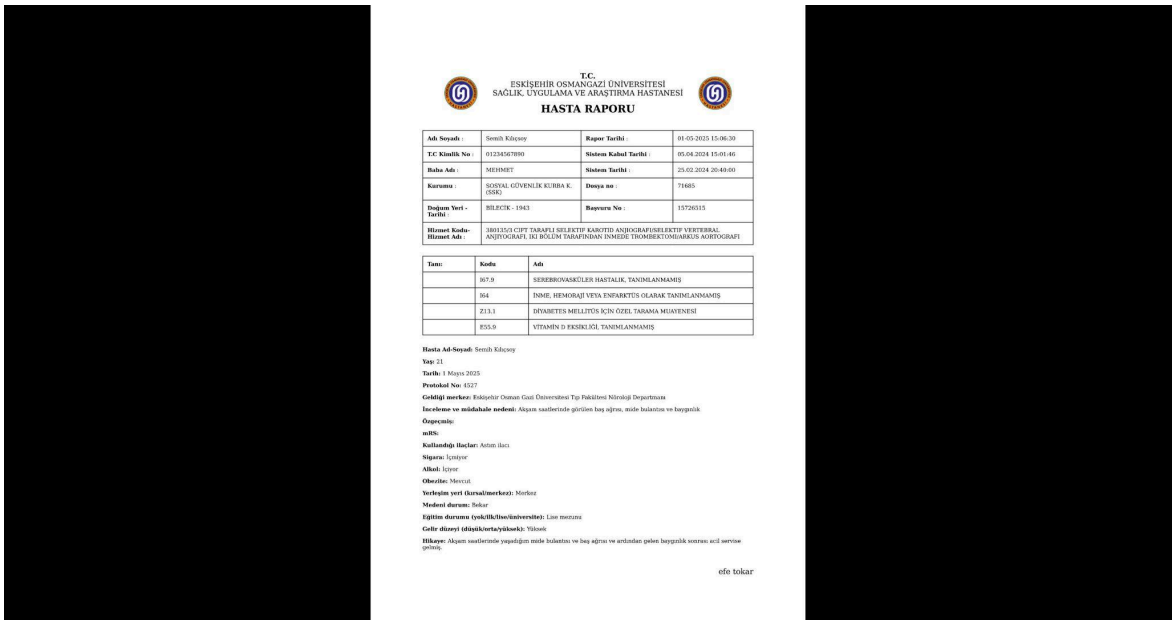

Figure 3: Report sample in PDF format

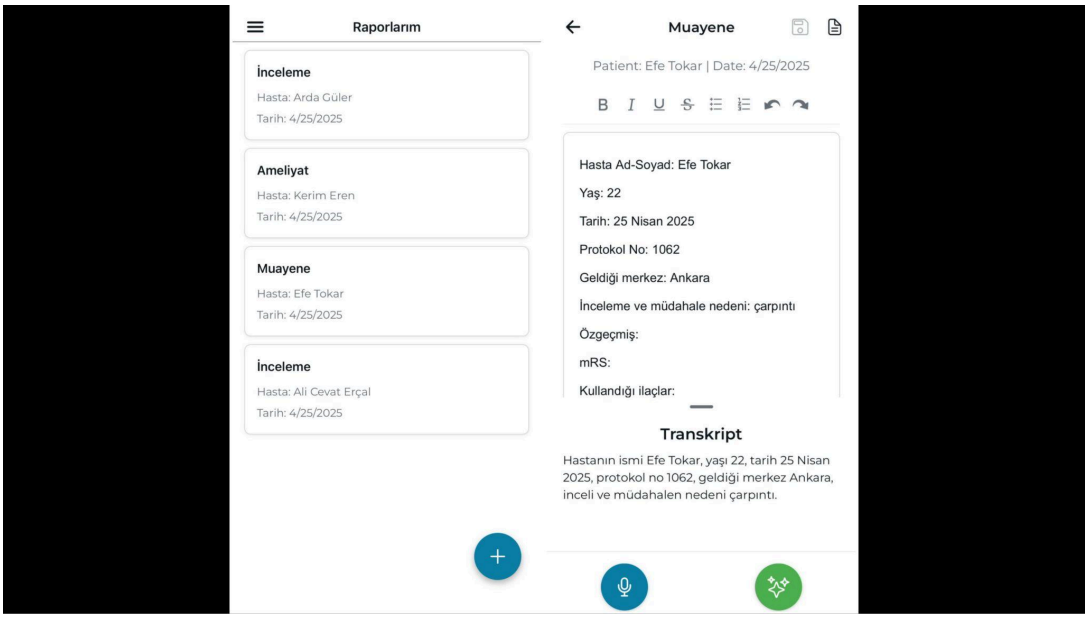

Figure 4: Voice reporting tool (mobile interface)

4.1.2 Group Reports Portal:

In the group reports portal, you can access the active or discharged patient reports of other doctors with whom you are in the same group (in a hospital, this could be the department where this user is located).

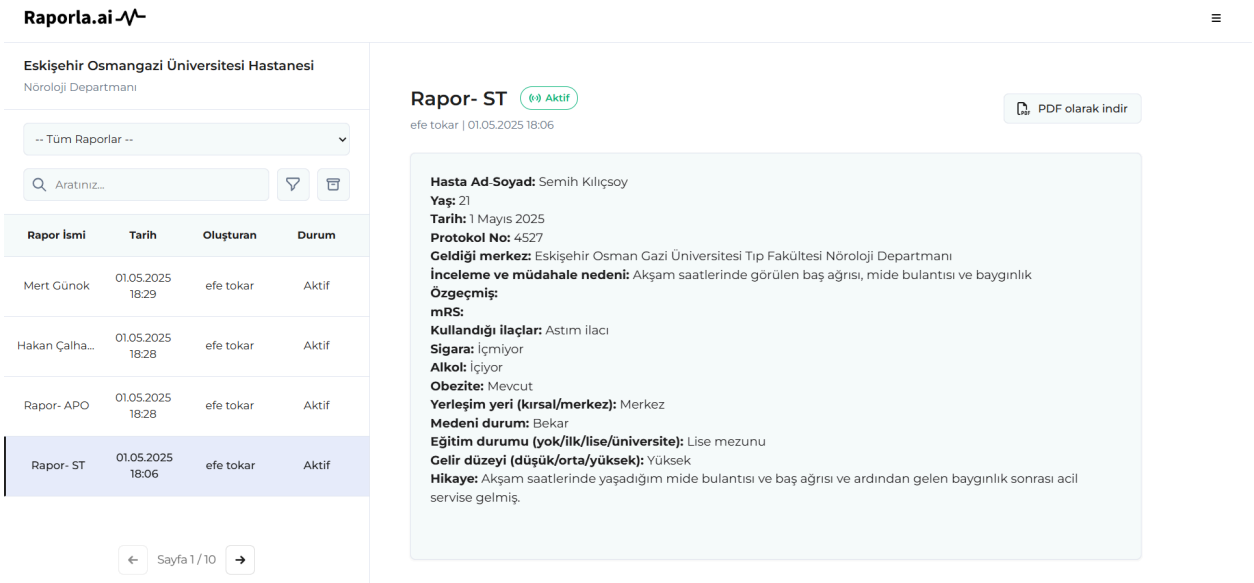

Figure 5: Group reports portal

4.1.3 Discharged Patient Reports Portal:

Since the doctors we worked with during the application development process stated that the reporting process should be completed with the patient being discharged, an archiving logic was developed and a discharged patient reports portal was opened for use so that the user could view their personal archived reports.

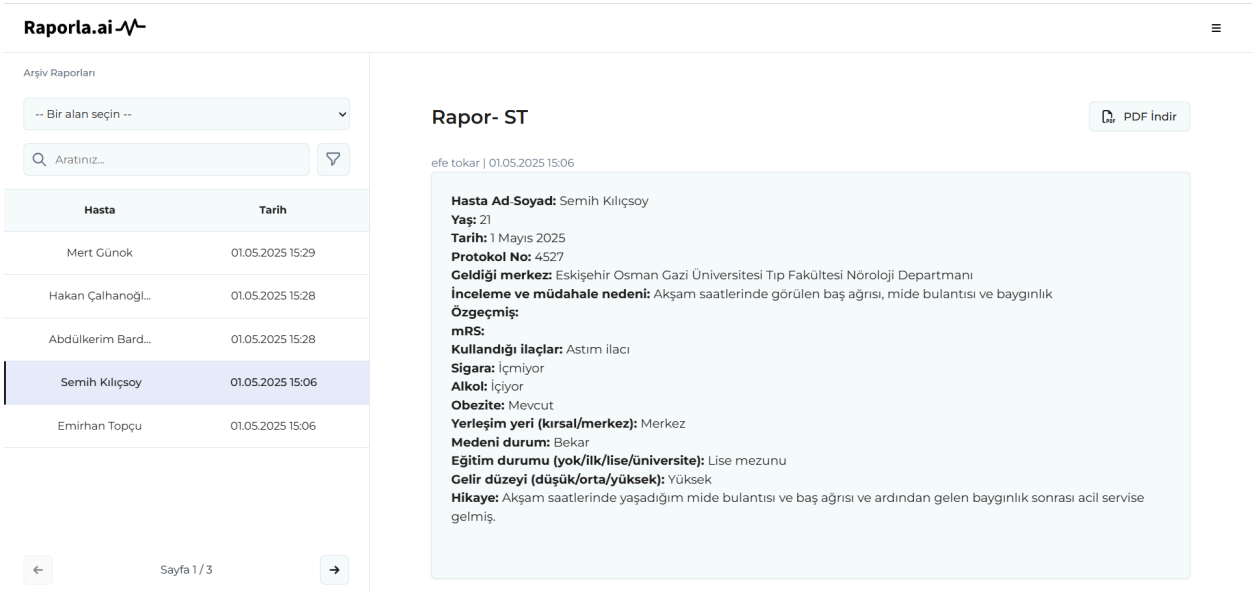

Figure 6: Discharged patient reports portal

4.1.4 Report Template Creation/Editing Tool

Another important functional page of the Raporla.ai application is the report template creation/editing page. On this page, the user can create a new template (for example, with the content order changed or content added/removed) for their own personal use by using the report templates already in the system for their own branch (for example, the default Thrombectomy template). They can use these personal templates in the next report creation process. They can also access the personal templates created by other users in their group and add the ones they like to their own collection and do changes on the already created personal report templates. Drag-drop elements are used on this page to provide ease of movement for the user.

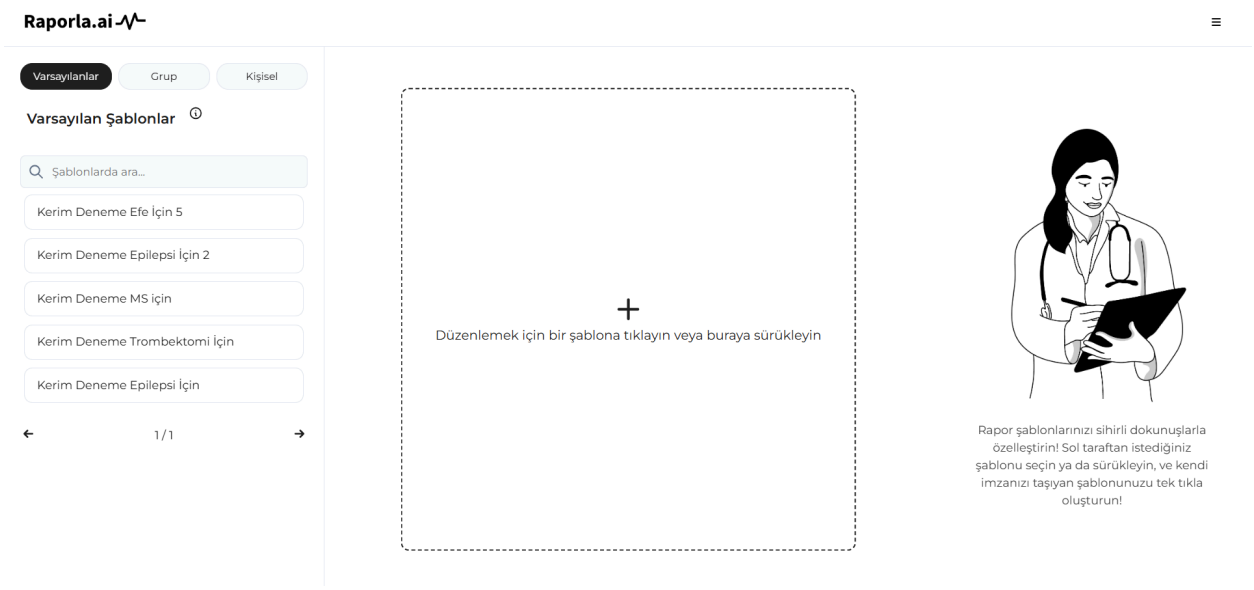

Figure 7: Initial template creation/editing tool

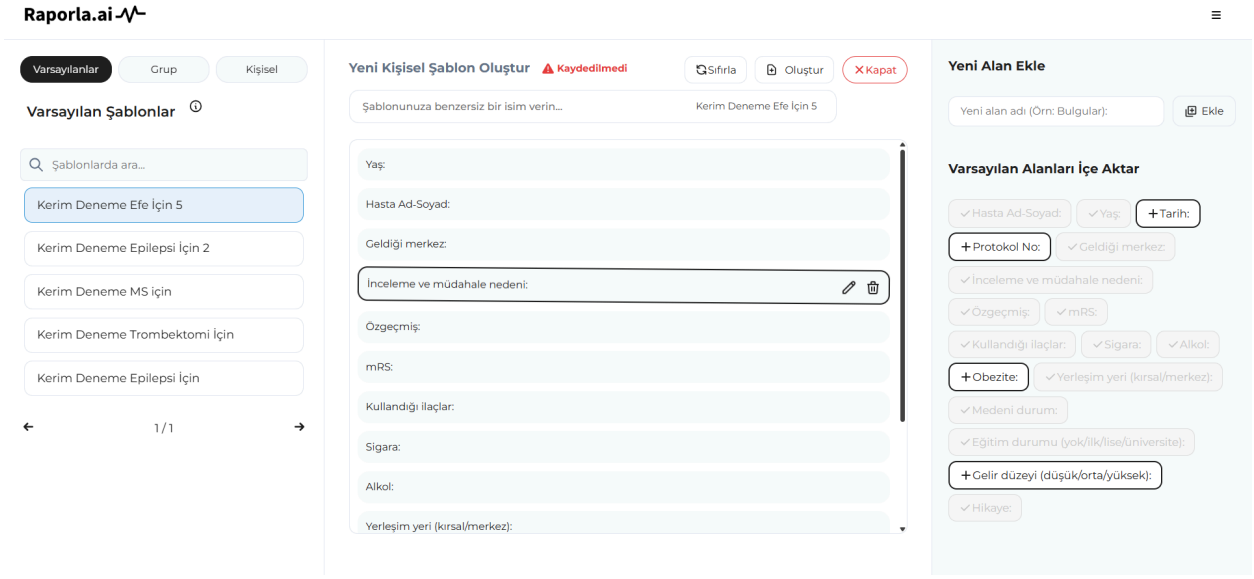

Figure 8: Position changing of an element in a template using drag-drop

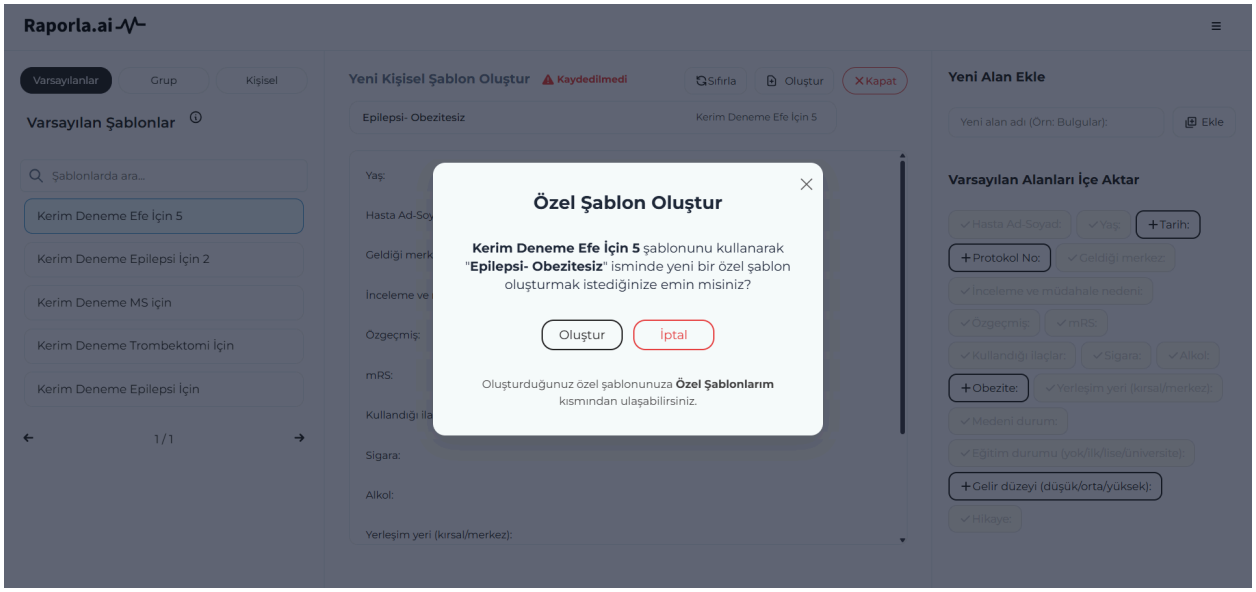

Figure 9: New personal template creation modal

4.1.5 Article Suggestion and Analysis Tool:

Along with the voice reporting tool and template creation/editing tool, another main functional page of the application is the article suggestion and analysis tool. This page is the artificial intelligence-supported implementation of the method used by doctors in the differential diagnosis process. The selected report is searched in PubMed by extracting medical keywords from it, assigning the most relevant 3 as the main keywords and 5 as secondary keywords. The first 5 articles resulting from this search are selected and their titles and abstracts are displayed to the user. In order to provide language convenience, the simultaneous translations of these titles and abstracts are also provided. At the same time, the user is given the opportunity to search in new and different combinations by replacing the keywords in use with other keywords in the pool (they can also add their own new keywords). In addition, the process is accelerated and made easier by discussing reports and articles with the chatBot on the right side and receiving support in the differential diagnosis process. At the same time, with the addition of a name and interactive visual (Hipokrat), the chatBot is given a personality, making it easier for the user to establish a personal connection.

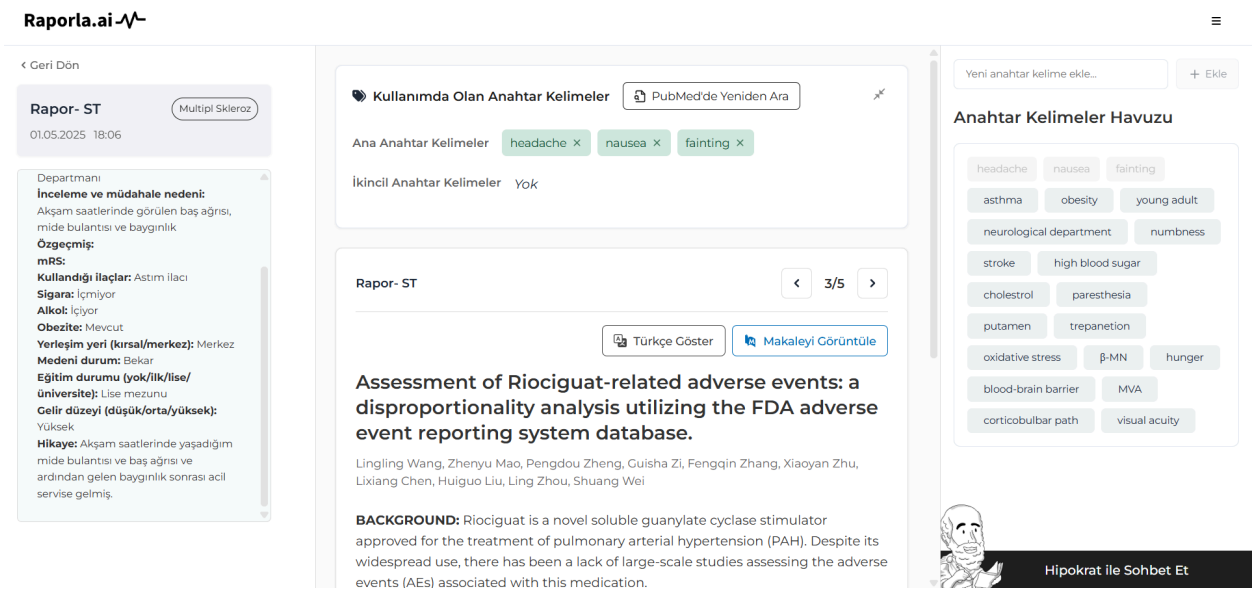

Figure 10: Article suggestion and analysis tool

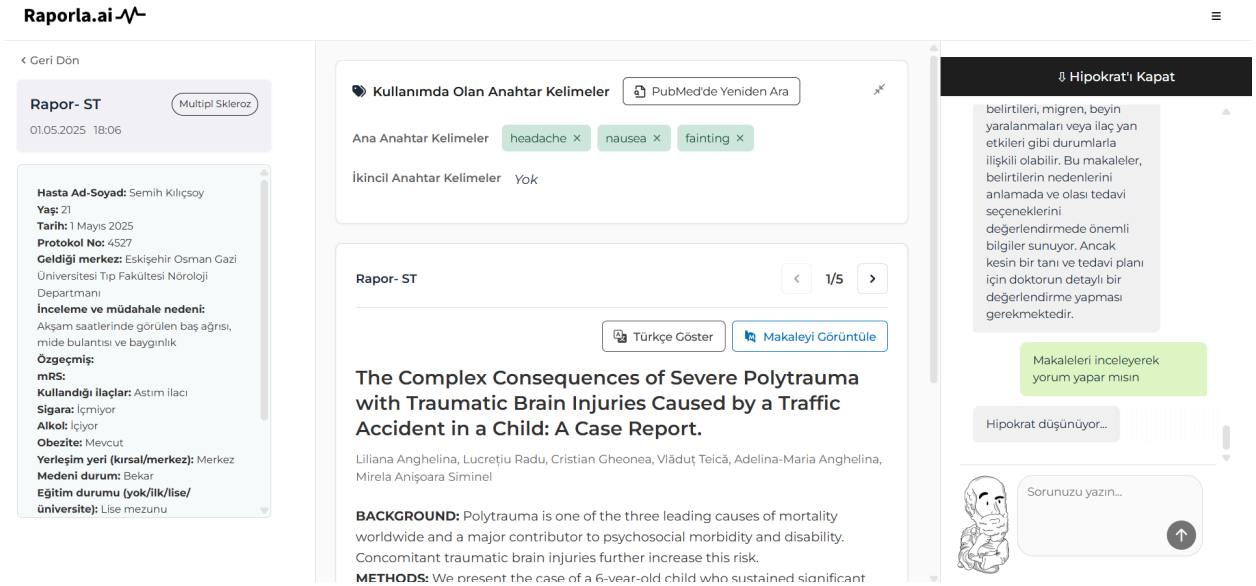

Figure 11: Hipokrat ChatBot

4.1.6 Hopital Administrator Panel:

The application also includes a hospital administrator panel that provides functions for hospital administrators to assume the role of an administrator, such as assigning doctors and groups, and checking reports. The hospital administrator uses this panel to create accounts for doctors working in the hospital. In this way, doctors who are included in a hospital can log in to the system with the institutional authentication information assigned to them without having to deal with the individual registration process (such as the Bilkent SRS system). This panel is only open to those assigned as hospital administrators by the system; ordinary users cannot view it.

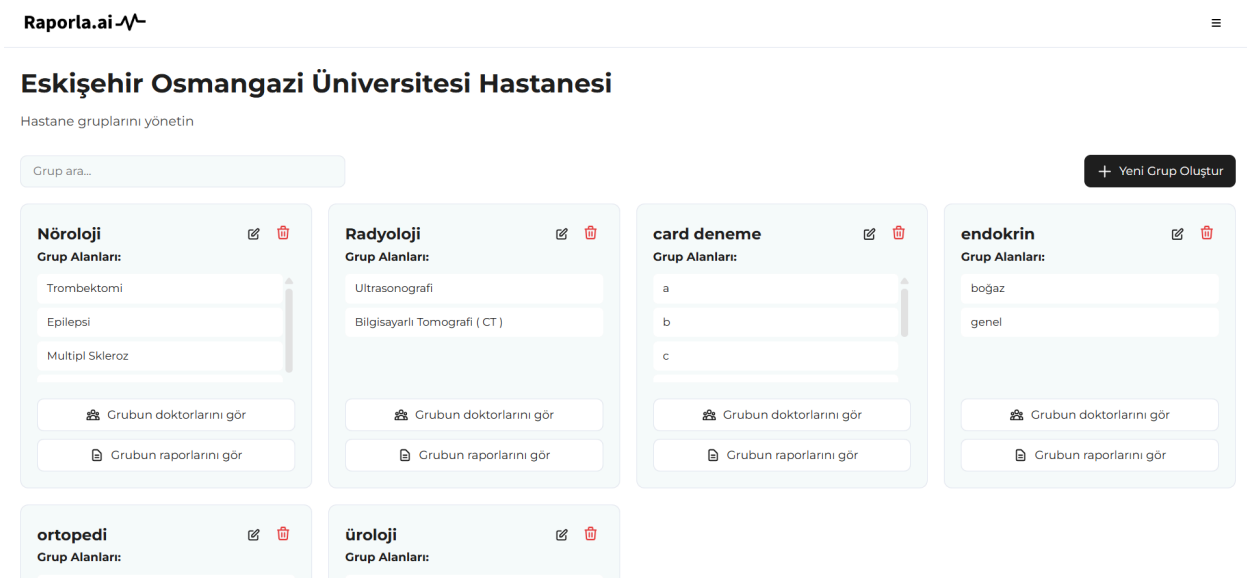

Figure 11: Hospital Administrator Panel

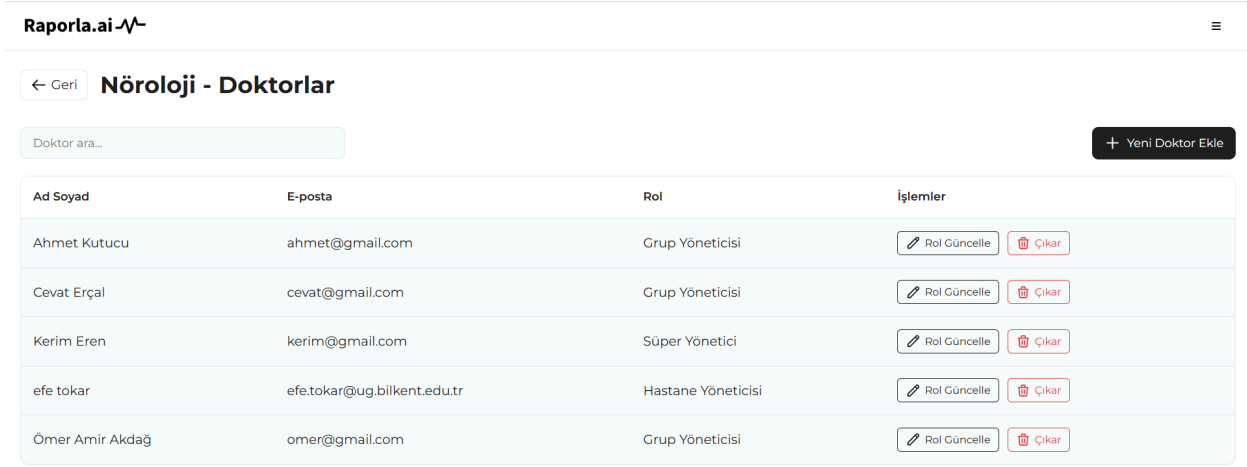

Figure 11: Hospital Administrator Panel (II)

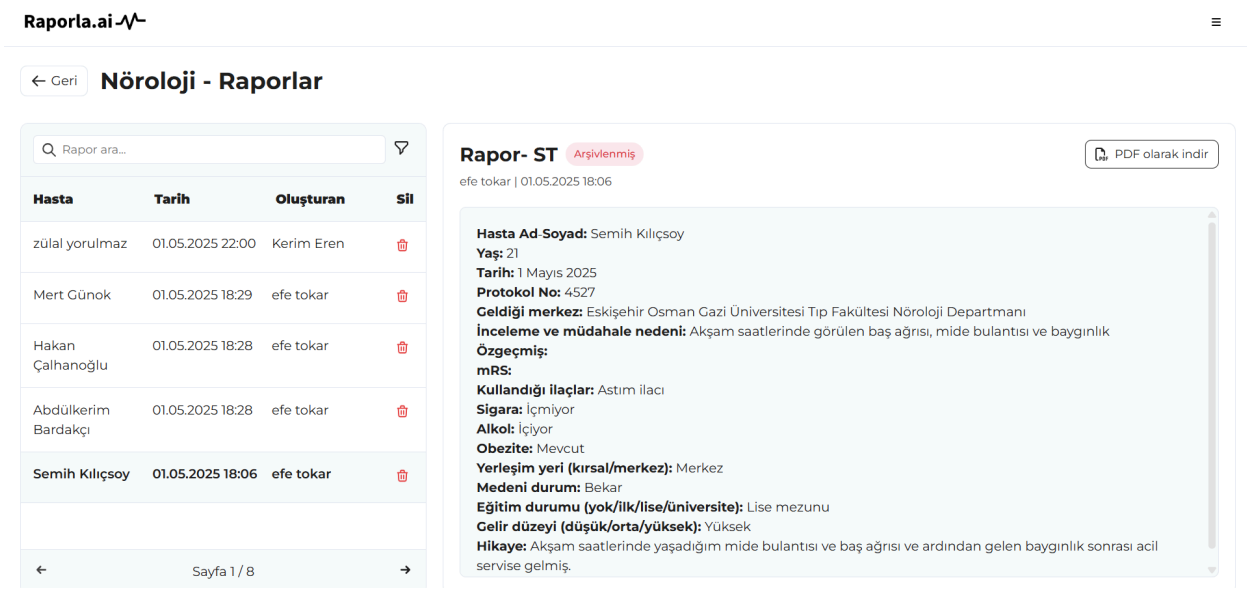

Figure 11: Hospital Administrator Panel (III)

4.2. Backend

The backend for Raporla.ai is developed using the asynchronous Python web framework, FastAPI, providing a robust and high-performance RESTful API for the frontend application. It uses Pydantic for data validation and serialization of request/response models, ensuring data integrity throughout the system. Database interactions are handled asynchronously, using an Object-Document Mapper (ODM) Beanie with MongoDB, allowing for efficient non-blocking I/O operations critical for a responsive application. User authentication and authorization are managed securely using OAuth2 and JWT (JSON Web Tokens), with password hashing implemented for credential protection.

The codebase is structured modularly following a layered architecture pattern to enhance maintainability and separation of concerns. The “endpoints” layer defines the HTTP routes and handles request/response cycles, delegating business logic to the “services” layer. This service layer encapsulates the core functionalities, including user management, CRUD operations for hospitals, groups, reports, and templates, processing transcripts, interacting with AI models for diagnosis suggestions (keyword extraction, article fetching/translation), and managing chat interactions. The “db” layer contains the data models and session management logic, while the “schemas” directory holds the Pydantic models. Common functionalities like authentication helpers, email services, PDF generation, and text utility functions reside in the “utils” directory, and core configurations and security settings are managed within the “core” directory.

This design ensures that API endpoints remain slim, focusing primarily on request handling and validation, while the complex business logic resides within dedicated service modules. Dependency injection, a core feature of FastAPI, is utilized to manage dependencies

like database sessions and authenticated user context efficiently. Middleware is implemented for tasks such as database session scoping and request logging, applied across multiple requests. The overall logic facilitates features like role-based access control (differentiating permissions for Doctors, Group Admins, Hospital Admins, Super Admins), paginated data retrieval with filtering and search capabilities for reports and other resources, and seamless integration of AI and external services.

4.3. Database

For managing our application's data, we primarily utilized MongoDB. As a NoSQL document database, MongoDB was chosen for its flexibility and scalability, which suits the varied structure of medical reports and user data. It serves as the core repository within the Storage Layer, storing essential information such as:

- User profiles and account details.
- The content and metadata of medical reports.
- Report templates (both default and custom).
- System configuration settings.

MongoDB's structure allows for easy handling of report content that might not fit a strict relational schema, aligning well with the dynamic nature of medical reports and templates.

4.4 Model

The core artificial intelligence component of Raporla.ai is the Automatic Speech Recognition (ASR) model. We developed a fine-tuned version of the Whisper model, specifically adapted for the project's needs. This fine-tuning process involved training the model on datasets relevant to Turkish medical language, including reports provided by Eskişehir Osmangazi University and general Turkish speech data from the Mozilla Voice Foundation. The primary function of this model is to accurately convert spoken medical reports in Turkish into text.

4.5 Cloud

Raporla.ai leverages Google Cloud Platform (GCP) for its infrastructure, providing scalability, reliability, and managed services. The key GCP services utilized are:

- **Google Cloud Run**: This serverless platform is used to host and automatically scale our backend services. It allows us to run our containerized application logic efficiently and cost-effectively, scaling up automatically during peak usage and down when idle.
- **Google Cloud Storage**: This is used for storing unstructured data, such as uploaded audio files for transcription and the trained machine learning models. It provides durable and scalable object storage.

Using these cloud services enables us to deploy and manage the application efficiently without needing to manage underlying servers, supporting our goals for scalability and reliability.

5. Testing Details

In order to ensure the quality, reliability and usability of, Raporla.ai was a cruical phase our application. We have done a multi-layered approach, combining different testing methodologies to validate the application on the requirements we indicated in our Detailed Design Report. Our approach prioritized manual and user testing. In the user tests we asked the doctors we worked with to try and use different functionalities in our application.

5.1 Testing Strategy and Methodology

Our methodology involved systematic manual testing, developer team testing, end-user tests and feedback. This allowed us to verify specific functionalities and user workflows thoroughly.

We adopted aspects of different testing levels:

5.1.1 Functional Testing

This is the core of our testing efforts. We executed 54 predefined test cases that has been explained before in our Detailed Design Report covering all major features, including:

- User Authentication and Authorization (Registration, Login, JWT handling, Role-based access - TC37, TC39, TC45, TC47, TC51).
- Report Management (Create, Read, Update, Delete, List, Search, Filter, Pagination - TC01-TC06, TC16-TC18).
- Speech-to-Text (Real-time recording, File upload, Transcription accuracy checks - TC07, TC08, TC19, TC46).
- Text Editor Features (Formatting, Undo/Redo, Copy - TC09-TC13, TC42).
- Template Management (Default/Custom templates, Drag-and-drop, Variable handling, Search, Save, Delete - TC23-TC35, TC50).
- AI Features (AI-assisted report generation, Paper suggestions - TC20, TC21, TC41, TC52).
- General UI/UX (Theme toggling, PDF download, Unsaved changes warnings, Logout - TC05, TC14, TC15, TC36, TC43).

5.1.2 Integration Testing

We manually verified the interactions between different subsystems as defined in our architecture.

- Client Layer communication with the Backend Layer via the Communication Layer (API calls for login, report actions, ASR requests).
- Backend Layer service coordination (e.g., ASR Manager output feeding into the Report Manager, Auth Manager validating requests for other managers).
- Backend Layer interaction with the Storage Layer (Data persistence and retrieval from MongoDB for users, reports, and templates).
- Integration of external services (e.g., handling responses and potential errors from OpenAI API for AI generation or PubMed for paper suggestions - TC52).

5.1.2 System Testing

For the system testing, we performed end-to-end testing by simulating typical user workflows from a doctor's perspective. This involved an imitation of the whole workflow for our application including logging-in,creating/selecting a report, using speech-to-text (both recording and upload), editing the transcript, applying templates, potentially using AI generation, saving the report, and downloading it as a PDF.

5.1.3 Usability Testing

Team members, sometimes those less involved in the specific component's development, conducted informal usability tests (alpha testing). We focused on the intuitiveness of the UI (Doctor Page and Admin Page), ease of navigation, clarity of instructions, and overall user experience, aligning with our Usability design goal.

5.1.4 Security Testing

Beyond functional authentication tests, we manually checked for basic security considerations:

- Ensuring role-based access control prevents unauthorized actions (e.g., standard users accessing admin functions - TC47).
- Verifying session management, including logout functionality and inactivity timeouts (TC43, TC51).
- Confirming sensitive data like passwords are not exposed and are handled securely (hashed storage via bcrypt).

5.1.5 Performance and Reliability Testing

While extensive automated load testing was not performed, we conducted manual checks on critical performance aspects:

- Response times for ASR transcription, AI report generation, and paper suggestions under typical conditions.

- System behavior during concurrent actions, such as multiple users editing different reports or attempting PDF downloads simultaneously (TC48, TC49, TC54), looking for obvious conflicts or failures.
- Handling of simulated errors, such as invalid file uploads or potential external API unavailability (TC19, TC44, TC52), ensuring user-friendly messages instead of crashes, aligning with our Reliability goal (Section 1.2.2).

5.1.6 AI/Model Specific Testing

- ASR: We tested the speech-to-text conversion with various Turkish audio samples, including recorded speech and uploaded files, to qualitatively assess accuracy, particularly with common medical terms.
- ML Manager (Paper Suggestion): The relevance of suggested papers (TC41) was manually evaluated by team members based on the content of sample reports.

5.2 Testing Tools

- Web Browser Developer Tools: Used extensively to monitor network requests (API calls), inspect element states, check console logs for errors, and simulate different network conditions. [2]
- Manual Execution of Test Cases: Systematically going through the documented test cases from the DDR.
- Postman: Used during development for direct API endpoint testing to isolate backend functionality.

Fig 12: Google Website Performance Analysis for raporla-ai.com

6. Maintenance Plan and Details

Maintaining Raporla.ai is essential to ensure it remains reliable, secure, accurate, and useful for doctors over time. This involves keeping the system running smoothly, fixing issues, updating features, and improving performance. Our maintenance plan focuses on several key areas:

Bug Fixing and Software Updates: We need a process to identify and fix any errors or bugs reported by users or found through monitoring. Regularly updating the frontend and backend code is important to improve stability, add small features, and keep libraries and frameworks up-to-date for security and performance. Since our backend runs on Google Cloud Run, deploying updated code is straightforward by deploying new container images.

Database Management: The MongoDB database needs ongoing care. This includes monitoring its performance to ensure fast data access, performing regular backups to prevent data loss, and applying any necessary database software updates. Using a managed MongoDB service can simplify some of these tasks.

Speech-to-Text Model Maintenance: The accuracy of the fine-tuned Whisper model is crucial. As medical terminology evolves or new accents/speaking styles are encountered, the model may need retraining. This involves gathering more relevant Turkish medical audio data, retraining the model on this new data, evaluating the updated model's performance, and deploying the improved version to Cloud Storage for use by the ASR Manager.

Cloud Infrastructure Monitoring: We need to monitor the Google Cloud Platform services to ensure they are operating correctly, managing resource usage, and keeping costs in check. Google Cloud provides tools for monitoring service health and performance.

Security Monitoring and Updates: Protecting sensitive medical data is a top priority. This involves continuously monitoring system logs for suspicious activity, applying security patches to the backend software's dependencies, and reviewing cloud security configurations to ensure they remain robust.

Performance Monitoring: Regularly checking the application's speed and responsiveness is important. This includes monitoring API response times, database query times, and overall system load. If performance drops, we would investigate bottlenecks and potentially adjust Cloud Run scaling settings or optimize code/database queries

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

In developing Raporla.ai, several non-technical factors come into play, including public health, safety, welfare, global, cultural, social, environmental, and economic elements. By understanding these considerations early, the system can be optimized for social good, regulatory compliance, user acceptance, and long-term viability.

7.1.1 Considerations

7.1.1.1 Public Health Considerations

Raporla.ai's primary goal is to improve the medical reporting process, potentially reducing reporting errors and allowing doctors to dedicate more time to patient care. More accurate and standardized reports contribute to better patient outcomes, while faster documentation may help doctors manage their workloads efficiently, improving overall public health standards. Ensuring patient data privacy and compliance with regulations like KVKK also indirectly supports public trust in health services.

7.1.1.2 Public Safety Considerations

Accurate and clearly documented medical reports are essential for safe patient care. By reducing transcription errors and ensuring consistency, Raporla.ai indirectly contributes to public safety. Misinterpretations or incomplete information in reports can lead to delays or mistakes in treatment. Thus, implementing strong error-checking and standardization promotes safer healthcare decisions.

7.1.1.3 Public Welfare Considerations

Although Raporla.ai does not directly influence public welfare policies, it can improve healthcare efficiency, making it easier and faster for patients to receive the correct diagnosis and treatment. This efficiency can positively impact the broader community by reducing wait times, improving patient satisfaction, and raising the overall quality of healthcare services.

7.1.1.4 Global Considerations

While the system initially focuses on Turkish healthcare contexts and language, it can later be adapted for global use. Global expansion would require addressing diverse medical standards, languages, and regulatory requirements. Ensuring multilingual support, compliance with various health regulations, and integrating internationally recognized medical datasets (e.g., PubMed) are considerations for future scalability.

7.1.1.5 Cultural Considerations

Medical practices and reporting expectations vary by region. As Raporla.ai specializes in Turkish medical literature, it respects local medical terminology, report formats, and communication styles. If adapted elsewhere, cultural factors such as different naming conventions, privacy attitudes, or doctor-patient communication norms would need to be considered for broader acceptance and utility.

7.1.1.6 Social Considerations

By easing the documentation burden, Raporla.ai may improve the work-life balance of doctors, potentially reducing burnout. This can enhance doctor-patient relationships and the overall healthcare environment. Moreover, standardizing reports can improve cooperation and understanding between medical professionals, leading to better teamwork and coordination in patient care.

7.1.1.7 Environmental Considerations

Digitizing and automating reporting can reduce the reliance on paper-based systems, indirectly lowering paper consumption and waste. Although the use of cloud-based ML models involves energy consumption, optimizing model size and computation times can help minimize the system's environmental footprint. Over time, balancing performance with energy efficiency will remain an important consideration.

7.1.1.8 Economic Considerations

Efficient medical reporting can potentially reduce administrative costs by minimizing manual transcription and editing time. Hospitals and clinics may find the system economically beneficial as it streamlines documentation, speeds up patient turnover, and reduces error-related costs. Nonetheless, the initial investment in cloud infrastructure, machine learning development, and ongoing maintenance must be managed effectively to ensure affordability and accessibility for a wide range of healthcare institutions.

TABLE 1

THE EVALUATION OF VARIOUS FACTORS' EFFECTS

Factor	Effect Weight (out of 10)	Effect to the Application
Public Health Considerations	8	Improving report accuracy and standardization can enhance patient outcomes, reduce documentation errors, and build trust in healthcare services.
Public Safety Considerations	5	Clearer, more consistent reports can lower the risk of medical errors, indirectly increasing patient safety and promoting responsible healthcare practices.
Public Welfare	4	Efficient documentation may speed access to proper

Considerations		treatments, indirectly supporting overall community health and well-being through streamlined healthcare.
Global Considerations	7	Adapting to international standards, languages, and regulations facilitates broader adoption, requiring flexibility in language models and compliance frameworks.
Cultural Considerations	6	Variations in medical terminology and reporting expectations across regions necessitate customizable templates and NLP models respectful of cultural nuances.
Social Considerations	5	Reduced administrative burdens can alleviate physician stress, improving doctor-patient interactions and possibly fostering better healthcare relationships.
Environmental Considerations	3	Digital documentation reduces paper consumption, but computing resources incur energy costs, prompting optimization for efficiency and sustainability.
Economic Considerations	9	Balancing initial implementation costs against potential long-term savings in time and effort may influence stakeholder adoption and funding opportunities.

7.1.2 Standards

- **Data Security and Privacy:**
    - **KVKK and GDPR Compliance:** Personal and medical data of users will be processed in full compliance with KVKK in Turkey and GDPR in Europe.
    - **Encryption:** Data will be encrypted during transmission and storage.
- **Health Data Integration:**
    - **HL7 (Health Level Seven):** The system will comply with HL7 standards [3], internationally recognized for health data sharing.
- **Machine Learning Accuracy:**
    - **WER (Word Error Rate):** The Speech-to-Text model will be optimized using this metric to enhance accuracy [4].

- **Accessibility:**
    - **WCAG 2.1:** The system will be designed to meet these standards, ensuring easy access for users with disabilities [5].

7.2. Ethics and Professional Responsibilities

As in all project development processes, a structured and respectful environment provided due to the requirements of teamwork during the development process of our project Raporla.ai. For the sake of the project, all team members completed their task appropriately in the designated time interval as a professional. Weekly meetings held in order to stay in touch with each other and maintain our professional manner. Since the data we will use will carry important personal information, minimizing this security vulnerability is vital for the ethical part of the project. Any third party resource used for our project will be cited properly.

Patient Privacy:

- The system will ensure the security of patient data through strong encryption methods.
- All data will be anonymized and accessible only to authorized personnel.
- Full compliance with KVKK and GDPR data protection regulations will be ensured.

Data Usage and Transparency:

- Medical reports and suggestions will only be provided as guidance and will not be regarded as definitive diagnoses. This will be clearly stated to users.

AI Bias:

- Potential biases in training datasets will be analyzed and corrected to achieve fair and impartial results.
- The model will be designed to avoid discrimination based on gender, age groups, or clinical conditions.

Impact of Errors:

- The system will emphasize that the reports and diagnostic suggestions it produces must be verified by doctors.
- Any misinformation or incomplete diagnosis suggestions will be carefully examined, and accountability will be clearly defined.

Ethical AI Usage:

- The system will include appropriate warning mechanisms to prevent users from over-relying on AI-generated reports and suggestions.

- Users will understand that the system is an assistive tool and that medical decisions should always be made by a qualified doctor.

Accessibility:

- The system will be designed to be accessible and easy to use for users with varying levels of technological expertise.
- The application will aim to serve without discrimination for any user group.

Data Sharing and Third Parties:

- Collected data will not be shared with third parties under any circumstances, except with user consent for a specific purpose.
- User consent will be obtained for all stages of data sharing.

Compliance with Standards:

- The system will be developed in compliance with international standards for medical reporting and data management.
- Ethical AI design guidelines such as those from the IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems will be followed.

7.3. Teamwork Details

Effective teamwork was essential for successful Raporla.ai development. Our approach involved open communication, leveraging individual strengths, shared responsibility, and a collaborative environment in achieving project goals. The work was assigned based on members' interests and expertise to ensure comprehensive coverage of the project's requirements.

7.3.1. Contributing and functioning effectively on the team to establish goals, plan tasks, and meet objectives

Each member of the team contributed effectively in various aspects of the project:

Ali Cevat Erçal: Worked solely on the backend framework, including Docker containerization, database setup (MySQL/Alembic), AWS configuration, and implementing the OpenAI API for AI-based report functionality. Contributed to report generation logic at the core level from transcripts, helped in fixing bugs for frontend and backend, performed cost analysis, and managed mobile app prototype development.

Kerim Eren: Contributed to critical backend infrastructure and design elements like cross-platform Docker updates and text-to-speech integration. He implemented the fullstack admin panel, provided frontend support where needed, managed the Docker setup, helped with bug fixing, and served as the Scrum Master, guiding team efforts.

Ömer Amir Akdağ: Contributed across the stack, developing report generation features like building templates, building PDFs, and corresponding API endpoints. He took his contribution to fullstack implementations like building filtering features, integrating the Google Cloud Translation API, market and competitor analysis, performing required testing, and fixing bugs.

Göktuğ Serdar Yıldırım: Was initially focused on frontend development, implementing the main and report customization pages based on Figma designs, and handling endpoint connections. His work was extensively grown to encompass backend development, integration with PubMed for paper recommendations, implementing the logic of the 'Hipokrat' chatbot feature, and risk analysis. He was the core developer for the initial diagnosis/PubMed label prediction feature, too.

Efe Tokar: Led the UI/UX design of the application in Figma and worked on major parts of the frontend, including the main and report customization pages. He managed critical external communications, like the Ethics Board application and correspondence with healthcare providers. His responsibility included rigorous frontend testing followed by bug fixing for UI quality control.

Collaborative Efforts: The team collectively participated in regular meetings, brainstorming sessions, unofficial code reviewing, and project report and presentation preparation to ensure alignment and shared understanding.

7.3.2. Helping creating a collaborative and inclusive environment

We actively worked towards creating a healthy and productive team environment:

Open Communication: We utilized communication media like WhatsApp and periodic online meetings arranged by the Scrum Master (Notion) for frequent updates, discussions, and problem-solving. This ensured technical issues and progress were communicated well.

Knowledge Sharing: Experience was actively shared among team members. Ali, for instance, was more experienced in mobile development and deployment, Ömer had experience in report generation logic in his previous internship, Göktuğ worked frontend and backend development since he had hands-on experience beforehand. Kerim was in charge of the backend and Efe was the one who was the most experienced in frontend. Everyone shared their prior knowledge to other team members. This made no one shy away from asking or learning from others.

Active Participation: Each member was encouraged to participate actively in conversation as well as in decision-making practices, such that inputs from multiple stakeholders were accommodated, notably in areas of UI/UX design and adding features.

Guidance: We assisted each other while debugging programs, fixing defects, and decrypting intricate units, prioritizing collaboration over individual silos.

Respectful Interaction: We maintained a respectful atmosphere where all opinions were valued, creating an inclusive environment where members felt engaged and valued.

7.3.3. Taking lead role and sharing leadership on the team

Though teamwork was essential, individual members took charge and leadership in their own fields of expertise:

Kerim Eren: As Scrum master, held a central leadership role in project coordination and management. He ensured the team stayed on track with regular meetings, sprint planning, and facilitating discussions, maximizing the overall workflow.

Ali Cevat Erçal: Directed backend architecture and implementation as well as the mobile, leading technical decisions related to database management, Docker, cloud deployment (AWS), and core API integrations (e.g., OpenAI).

Efe Tokar: Directed frontend design and user experience, translating requirements into user-centric interfaces using Figma. Directed external communications as well, speaking for the team in relations with the Ethics Board and potential users.

Göktuğ Serdar Yıldırım: Played a key part in unifying frontend and backend implementation, particularly for the key user-facing features and combining AI/ML elements such as PubMed suggestions and the chatbot.

Ömer Amir Akdağ: Had a crucial role in designing and implementing the core report generation system and later expanded his scope to ensure quality fullstack capability, testing, and critical integrations like translation.

This distributed leadership model allowed us to work to individual strengths in an efficient way, and each critical element of the Raporla.ai project was carried forward with professionalism and accountability.

7.3.4. Meeting objectives

Throughout the development cycle of Raporla.ai, the team set and pursued a series of objectives matching the project requirements and design purposes established in the initial planning and Detailed Design Report. Progress toward objectives was tracked with regular meetings, which were coordinated by the Scrum Master.

The primary objective was to create a functional, web-based application that assisted Turkish physicians in automating and simplifying the report writing process using speech-to-text technology. The primary objective was met, and the resultant application embodies:

- Secure user registration, login, and role-based access control (Standard User, Admin levels).
- Simple report management functions: creating new reports, reading/browse old reports, editing text within a rich text editor, saving, finding/searching for reports, and download as PDF.
- Integration with a streamlined Automatic Speech Recognition (ASR) model (Whisper) specifically developed for Turkish health language for transcribing through recording in real time and upload through audio files.
- An extensible template mechanism to enable standardized default template application or even authoring and managing custom templates with dynamic parameters.

Further goals addressed the implementation of AI-enhanced aid and usability improvements, which were largely achieved as well:

- Building of AI-powered report content creation using the OpenAI API.
- Integration of PubMed to propose related scholarly research based on report content.
- Implementation of the 'Hipokrat' chatbot functionality to special user input.
- Deployment of the Google Cloud Translation API to integrate with appropriate text processing needs.

Support for objectives critical to system administration and operation was also attained:

- Development of a general Admin Panel allowing management of administrators, users, and potentially system settings.
- Exemplary deployment of the application backend in an expandable cloud infrastructure using the Google Cloud Platform (Cloud Run, Cloud Storage).

- Development of an initial prototype mobile application in order to continue towards expanding accessibility.

Generally, the team managed to effectively complete the major functional specifications and most non-functional requirements stipulated for the project and deliver a comprehensive proof-of-concept application demonstrating the viability of Raporla.ai.

7.4 New Knowledge Acquired and Applied

The Raporla.ai initiative also served as a learning space, where all members of the team could acquire and apply new technical knowledge and improve their software engineering abilities in several areas.

- **Backend Development (Python, FastAPI, APIs):** The members gained deep expertise in modern backend development using Python and high-performance FastAPI web development framework. This involved developing RESTful APIs, utilizing Pydantic for strong data validation, implementing asynchronous programming patterns required for responsiveness, and managing secure user authentication and sessions through OAuth2 and JWT. (Kerim Eren, Ali Cevat Erçal, Göktuğ Serdar Yıldırım, Ömer Amir Akdağ).
- **Database Management (NoSQL, MongoDB):** Heavy usage of MongoDB provided hands-on practice in NoSQL database design and management. Team members developed data modeling skills suitable for adaptive, document-based data (e.g., reports and user profiles) and employed an Object-Document Mapper (Beanie) to facilitate effective database interaction in the asynchronous FastAPI environment (Ali Cevat Erçal, Kerim Eren, Ömer Amir Akdağ).
- **AI/ML Applications (ASR Fine-tuning, NLP, API Integration):** The strongest learning component included AI/ML applications. This included fine-tuning the Whisper ASR model on domain Turkish medical data sets, involving an understanding of data preparation, training processes, and model evaluation (Göktuğ Serdar Yıldırım, Kerim Eren). Third-party AI service integrations like text generation using the OpenAI API and translation using the Google Cloud Translation API included learning best practices for third-party API consumerization and processing their outputs (Ali Cevat Erçal, Ömer Amir Akdağ). The integration of PubMed included applying the principles of Natural Language Processing (NLP) to keyword extraction and querying non-academic databases (Göktuğ Serdar Yıldırım).
- **Frontend Development (Web Technologies, UI/UX Design):** The team members significantly enhanced their frontend development abilities in the new web technologies (HTML, CSS, JavaScript, React). This comprised converting Figma UI/UX designs into interactive user interfaces, application state management, clear communication with backend APIs, applying features like the rich text editor and drag-and-drop functionality,

and focusing on user experience (Efe Tokar, Göktuğ Serdar Yıldırım, Kerim Eren, Ömer Amir Akdağ - fullstack). Efe Tokar particularly improved his knowledge of UI/UX design principles utilizing Figma and applied systematic frontend testing methods.

- **Cloud Computing and DevOps (GCP, Docker):** The project involved the learning of cloud deployment and DevOps practices. The group gained first-hand experience with Google Cloud Platform, specifically working with Cloud Run for serverless deployment of containers and Cloud Storage for storing objects (models, audio files) (Ali Cevat Erçal, Kerim Eren). Docker use for containerization was crucial for maintaining reproducible development and deployment environments (Ali Cevat Erçal, Kerim Eren). Learning cloud cost management was also covered through cost analysis (Ali Cevat Erçal).
- **Mobile Application Development:** Ali Cevat Erçal acquired theoretical knowledge and practical experience in mobile application development through the development of the Raporla.ai prototype.
- **Project Management and Team Collaboration:** Besides technical skills, handling the sophisticated project also instilled critical soft skills within all project members. Scrum process, under the guidance of Kerim Eren, reinforced task planning, tracking progress, and agile practice. Excellent communication through frequent meetings and virtual media, collaborative problem-solving (especially during debugging and integration phases), and work documentation/presentation were key takeaways.

Learning material was primarily official documentation (FastAPI, Python libraries, GCP, MongoDB, Whisper, OpenAI), online tutorials, technical articles, and large-scale collaborative experimentation and peer-to-peer knowledge sharing among the team. The project provided valuable hands-on experience, relating theoretical knowledge to practical application in building an actual software system.

8. Conclusion and Future Work

The Raporla.ai was developed to reduce the time loss experienced by healthcare professionals in the reporting process and to make this process more efficient, secure and standard. The application has been enriched with modules such as automatic conversion from voice reporting to text, template-based report editing, PubMed integration with artificial intelligence-supported article suggestions and a personalized chatbot called Hipokrat. Thanks to the web and mobile platforms developed throughout the project, an accessible and comprehensive experience has been provided for both desktop and mobile users. The aim is to ease the clinical workload of doctors with user-friendly interfaces, secure authentication systems, customizable templates and AI integrations.

A business plan is being worked on in the future where the application will be used in hospitals across Turkey. The application will be made usable and released to the market after

extensive user testing and feedback during the development phase, which is currently being carried out with Eskişehir Osmangazi University Faculty of Medicine. Also for the later phases, it is planned to expand the system with multilingual support, provide full integration with health data standards such as HL7 and increase the quality of automatic analysis in line with user feedback. In addition, it is aimed to make the Hippocrates chatbot more interactive and contextual, to develop and spread the mobile application natively and to provide deeper integration with the surgery logbook module with faculty members. Thus, Raporla.ai will become an advanced digital health assistant that can integrate with both national and international healthcare systems.

9. Glossary

A:

Admin Panel:
A management interface used by hospital or system administrators to manage user accounts, assign roles and groups, and oversee report activities. Only accessible by users with administrator privileges.

AI (Artificial Intelligence):
Technology that enables machines to simulate human intelligence. In Raporla.ai, AI is used for speech recognition, report generation, keyword extraction, PubMed article suggestions, and chatbot interaction.

ASR (Automatic Speech Recognition):
A machine learning system that converts spoken language into written text. Raporla.ai uses a fine-tuned version of OpenAI's Whisper model tailored for Turkish medical terminology.

Authentication / Authorization:
Mechanisms to verify user identity (authentication) and control access based on user roles (authorization). Raporla.ai uses OAuth2 and JWT for secure login and role-based access.

B:

Backend:
The server-side logic of the application that processes requests, manages data, and handles AI integrations. Implemented using FastAPI in Raporla.ai.

Beanie ODM:
An asynchronous Object-Document Mapper used with MongoDB and FastAPI, facilitating seamless database operations.

C:

Cloud Run:
A serverless compute platform by Google Cloud used to host Raporla.ai's backend services in containerized form, enabling auto-scaling and efficient deployment.

Custom Template:
A user-defined report format that can be created by modifying existing templates or building a new one to suit specific clinical needs.

D:

Discharged Reports:
Archived medical reports for patients who have been discharged. These are view-only and cannot be modified or deleted.

Drag-and-Drop:
A UI interaction pattern that allows users to rearrange elements (such as report fields or template blocks) by dragging them visually.

E:

Encryption:
A security process that protects data by encoding it during storage and transmission. Essential for ensuring compliance with GDPR and KVKK.

F:

FastAPI:
A high-performance Python web framework used to build RESTful APIs. Chosen for its speed and modern asynchronous capabilities.

Frontend:
The client-side interface that users interact with. Raporla.ai features a React-based web frontend and an Expo-based mobile app frontend.

G:

Google Cloud Platform (GCP):
The cloud infrastructure used for hosting Raporla.ai, providing services such as Cloud Run for deployment and Cloud Storage for assets.

H:

Hipokrat (Chatbot):
A personalized, AI-powered chatbot embedded in the application, designed to assist users with suggestions and clarification during the diagnosis process.

J:

JWT (JSON Web Token):
A compact, secure token format used to transmit user identity and permissions after login. Used for session management and access control.

K:

KVKK:
Turkish personal data protection law, similar to the European GDPR. Raporla.ai complies with KVKK for handling user and patient data.

M:

Machine Learning (ML):
A subset of AI that allows systems to learn from data and make predictions. Used in Raporla.ai for keyword extraction and speech recognition.

MongoDB:
A NoSQL document-based database used to store user data, reports, templates, and configuration settings.

O:

Operation Logbook:
A mobile tool that captures and annotates critical explanations made by instructors during surgery, converting them into digital records for educational use.

P:

PDF Export:
The feature that allows users to export finalized medical reports as PDF files, formatted with official headers and structure.

PubMed:
A biomedical literature database used by Raporla.ai to suggest relevant articles during the differential diagnosis process based on extracted keywords.

R:

React:
A JavaScript library used for building user interfaces. Raporla.ai's web frontend is built with React.

RBAC (Role-Based Access Control):
A system that restricts access to certain functionalities based on predefined user roles, such as doctor, group admin, or super admin.

S:

Speech-to-Text:
The core functionality that transcribes spoken reports into text using the ASR model. Supports both live recording and uploaded audio files.

T:

Template:
A pre-structured layout for report creation, customized to clinical branches (e.g., neurology, radiology) or personal user needs.

Token Expiration:
The time limit after which a JWT becomes invalid, requiring users to re-authenticate for security purposes.

U:

User Roles:
Defined roles within the application (doctor, group admin, hospital admin, super admin), each with specific permissions and access levels.

V:

Voice Reporting Tool:
A central feature of Raporla.ai that allows users to create structured reports by speaking into the system, with optional manual editing. Available on both web and mobile platforms.

W:

Whisper:
A multilingual, open-source ASR model by OpenAI. In Raporla.ai, it is fine-tuned with Turkish medical audio to ensure accurate transcription.

10. References

[1] A. Papworth, “Non Functional Requirements: Quick Guide for the business analyst in 2024,” BusinessAnalystMentor.com. https://businessanalystmentor.com/non-functional-requirements/ (accessed Nov. 21, 2024).

[2] “Raporla.ai – PageSpeed Insights (Desktop),” *pagespeed.web.dev*. https://pagespeed.web.dev/analysis/https-raporla-ai-com/dew9mmcm0d?form_factor=desktop (accessed May 2, 2025).

[3] “HL7 Standards – Health Level Seven International,” *hl7.org*. https://www.hl7.org/ (accessed Nov. 21, 2024).

[4] “What is Word Error Rate (WER)?,” *clari.com*. https://www.clari.com/blog/word-error-rate/ (accessed Nov. 21, 2024).

[5] “Web Content Accessibility Guidelines (WCAG) 2.1,” *w3.org*. https://www.w3.org/TR/WCAG21/ (accessed Nov. 21, 2024).