ÇANKAYA UNIVERSITY

Software Requirements Specification

Skinalyzer: AI-Based Skin Cancer Detection System

Melike Hazal Özcan 202011013,
Bilgesu Fındık 202011407,
Bekir Emrehan Şimşek 202011039,
Pelin Koz 202011048

Table of Contents

List of Figures

1. INTRODUCTION

1.1 Purpose

This document provides a detailed specification of the requirements for a skin cancer detection system. Combining Federated Learning and Incremental Learning techniques, the project aims to assist users in analyzing skin lesions to enable early detection of cancer risk. It serves as a guide for the development, implementation, and testing of the system.

1.2 Scope of Project

This project aims to develop a desktop application that supports skin cancer diagnosis by analyzing skin lesion images. The system processes the images uploaded by users through advanced machine learning models, providing diagnostic results in a clear, accessible, and actionable format. By integrating Federated Learning, the system ensures the privacy of sensitive user data, as all raw data remains on local devices and is never transferred to a central server. This privacy-preserving approach aligns with modern data protection standards and user expectations in the healthcare sector.

Additionally, the system leverages Incremental Learning to continuously update the machine learning model with new data while preserving knowledge from previous datasets. This capability ensures that the model adapts to evolving data distributions without losing prior learning, maintaining high performance and accuracy over time. The combination of Federated Learning and Incremental Learning creates a scalable and secure framework for collaborative model training across multiple devices, addressing the challenges of data heterogeneity and privacy in medical applications.

Designed for healthcare professionals, researchers, and even individual users, the application aims to streamline the diagnostic process by offering a user-friendly interface and actionable insights. By training the model on diverse datasets like ISIC 2019 and HAM10000, the system ensures robust performance across various skin cancer types. This innovative solution has the potential to revolutionize early cancer detection by making cutting-edge AI technology accessible, secure, and reliable for clinical and personal use.

1.3 Glossary

- **Skin Lesion**: An abnormal area of skin that may indicate a skin condition or disease, including potential cancerous growths.
- **Machine Learning (ML)**: A branch of artificial intelligence that enables systems to learn and improve from data without being explicitly programmed.
- **Federated Learning**: A privacy-preserving machine learning method where the model is trained across multiple devices or servers without sharing raw data.
- **Incremental Learning**: A machine learning approach that allows a model to continuously update with new data while retaining knowledge from previous datasets.
- **ISIC 2019 Dataset**: A dataset containing labeled skin lesion images used for research and training in skin cancer detection.
- **HAM10000 Dataset**: A large collection of dermatological images used for diagnosing various skin conditions, including cancer.
- **Client Device**: A computer or device used by participants (e.g., healthcare professionals) to perform local training and interact with the system.
- **Central Server**: A server that aggregates updates from multiple client devices to improve the global machine learning model.
- **Model Update**: The process of incorporating new training data into an existing machine learning model to improve its accuracy and performance.
- **Neural Network**: A computational model inspired by the human brain, used in machine learning to identify patterns and make predictions.
- **Confidence Score**: A metric that indicates the certainty of the model's prediction, often expressed as a percentage.
- **Python**: A programming language commonly used for developing machine learning and AI applications.
- **PyTorch**: An open-source machine learning library widely used for building and training neural networks.
- **Tkinter**: A Python library used for creating graphical user interfaces (GUIs).
- **PyQt5**: A set of Python bindings for the Qt application framework, used to develop advanced GUIs.
- **Kafka-ML**: An open-source tool that facilitates communication and data exchange in federated learning systems.
- **HTTPS**: A secure protocol for communication over the internet, ensuring data encryption between clients and servers.
- **TLS/SSL**: Encryption protocols used to secure internet communications, ensuring data integrity and privacy.
- **Data Preprocessing**: The process of preparing raw data (e.g., skin lesion images) for machine learning by cleaning, resizing, and formatting it appropriately.
- **Error Logging**: The practice of recording errors or failures in a system to facilitate debugging and system improvement.
- **Graphical User Interface (GUI)**: A visual interface that allows users to interact with a system using graphical elements such as buttons and menus.
- **Probability Analysis**: The process of estimating the likelihood of an outcome, such as determining whether a skin lesion is cancerous.
- **Data Integrity**: Ensuring that data is accurate, consistent, and free from unauthorized modifications during transmission or storage.
- **Global Model**: A centralized machine learning model that aggregates updates from client devices in a federated learning system.

1.4 References

- ISIC 2019 Dataset: https://challenge.isic-archive.com
- HAM10000 Dataset: https://www.kaggle.com/kmader/skin-cancer-mnist-ham10000
- Incremental Learning and Federated Learning for Heterogeneous Medical Image Analysis: Incremental learning and federated learning for heterogeneous medical image analysis - UBC Library Open Collections

1.5 Overview of the Document

This document provides a comprehensive specification of the requirements for the skin cancer detection system, ensuring a clear understanding of the project's objectives, functionality, and technical expectations. It begins with an introduction outlining the purpose, scope, and goals of the project, accompanied by relevant references and a glossary of key terms. Following this, the overall description section highlights the primary features of the system, its intended users, and its dependencies on external technologies. The core of the document is the requirements specification, which details both functional requirements, such as the system's ability to analyze images and provide diagnostic insights, and non-functional requirements, including performance, usability, and security considerations. Additionally, the document describes the external interfaces, outlining interactions with hardware, software, and communication protocols, while also emphasizing critical system attributes such as portability, scalability, and adaptability. Finally, the references section provides a curated list of datasets, research articles, and external resources that inform the system's design and development. This document serves as a foundational guide for developers, testers, and stakeholders, supporting all phases of the project from design to deployment.

2. OVERALL DESCRIPTION

2.1 Product Perspective

This system represents an innovative approach to medical diagnostics by combining image analysis with artificial intelligence. It aims to make advanced diagnostic tools accessible to general users, ensuring simplicity and usability. The system addresses the limitations of traditional diagnostic methods by leveraging diverse datasets to ensure comprehensive skin cancer classification.

2.1.1. Development Methodology

The development of this system will follow a User-Centered Design methodology, tailored to general users without extensive collaboration with medical professionals. This approach prioritizes creating an intuitive platform that evaluates the likelihood of skin cancer through AI while minimizing complexity for non-expert users. The methodology involves the following steps:

- **Requirement Analysis**: Collecting input from potential general users to understand their needs and expectations.
- **Design and Prototyping:** Developing user-friendly interfaces for image uploading and result visualization.
- **Implementation**: Integrating pre-trained machine learning models for image analysis and cancer risk evaluation.
- **Testing and Validation:** Conducting thorough tests to ensure prediction accuracy and reliability.
- **Deployment and Maintenance:** Launching a widely accessible web-based platform and updating the system regularly to enhance its performance and usability.

The methodology ensures that the system remains focused on providing a seamless user experience while delivering accurate results. Collaboration with medical professionals is optional and limited to post-evaluation consultation, ensuring that the primary emphasis is on empowering general users with AI-driven assessment.

2.2 User Characteristics

2.2.1. Participants

- **Description:** Participants are the primary users of the system who will upload skin lesion images and analyze the results for medical or research purposes.
- **Characteristics:**
  - **Role:** Typically doctors, dermatologists, or healthcare professionals. In some cases, it may include individual users (patients) who want a preliminary analysis.
  - **Technical Knowledge:** Basic computer skills such as uploading files and interpreting simple analysis results. No advanced technical expertise is required.
  - **Primary Tasks:**
    - Upload skin lesion images for analysis.
    - View and interpret analysis results.
    - Save or download reports for further evaluation.
  - **Access:** Desktop interface.

2.2.2 Client Admin

- **Description**: Client Admins are responsible for managing the local client-side system, ensuring that local models are properly trained and updated, and overseeing client-side interactions.
- **Characteristics**:
  - **Role**: Client-side system administrators who manage local infrastructure and ensure smooth functioning of federated learning processes on client devices.
  - **Technical Knowledge**: Familiarity with machine learning workflows, client-side federated learning setups, and model management tools (e.g., Kafka ML-Flow), along with basic understanding of client-server communication.
  - **Primary Tasks**:
    - Monitor and manage client-side updates to the model.
    - Ensure that local client data is processed securely without compromising privacy.
    - Ensure smooth communication between client devices and the central server.
  - **Access**: Client UI with permissions to manage local models and configurations.

2.2.3 Server Admin

- **Description**: Server Admins are responsible for maintaining the central server, ensuring that federated learning models are properly updated, and managing server-side interactions and data aggregation from multiple client devices.
- **Characteristics**:
  - **Role**: System administrators who maintain the server-side infrastructure and ensure the federated learning system is running smoothly on the central server.
  - **Technical Knowledge**: In-depth knowledge of machine learning workflows, federated learning protocols, system security, and server maintenance. Familiar with server-side management tools such as Kafka-ML, and data aggregation techniques.
  - **Primary Tasks**:
    - Monitor and manage the central server's functionality.
    - Aggregate updates from client devices and apply those updates to the global model.
    - Resolve issues related to client-server communication and system security.

- **Access**: Server UI with advanced settings for managing federated learning processes and server-side configurations.

3. REQUIREMENTS SPECIFICATION

3.1. External Interface Requirements

3.1.1. User interfaces

- The user interface will be designed to work on desktop and web-based platforms.
- The system will have an intuitive and user-friendly design that will facilitate users to upload images, view analysis results, and access past analysis results.
- The user interface will be developed with tools such as Tkinter or PyQt5.

3.1.2. Hardware interfaces

- The system requires a standard USB port on the user's computer for uploading skin lesion images.
- Hardware requirements include a minimum of 8 GB RAM and support for CPUs and GPUs capable of running machine learning models.
- An internet connection (Ethernet or Wi-Fi) is necessary for updating federated learning models.

3.1.3. Software interfaces

The system will be designed to run on major operating systems and should integrate with Python-based machine learning libraries. Secure protocols will be used to interact with external APIs, and the system will utilize federated learning for model training and updates without sharing raw data.

3.1.4. Communications interfaces

The system will use HTTPS with TLS/SSL encryption for secure communication between the server and client devices. Communication will include sending model updates, receiving training status, and downloading the global model.

3.2 Functional Requirements

3.2.1. USER REGİSTER USE CASE

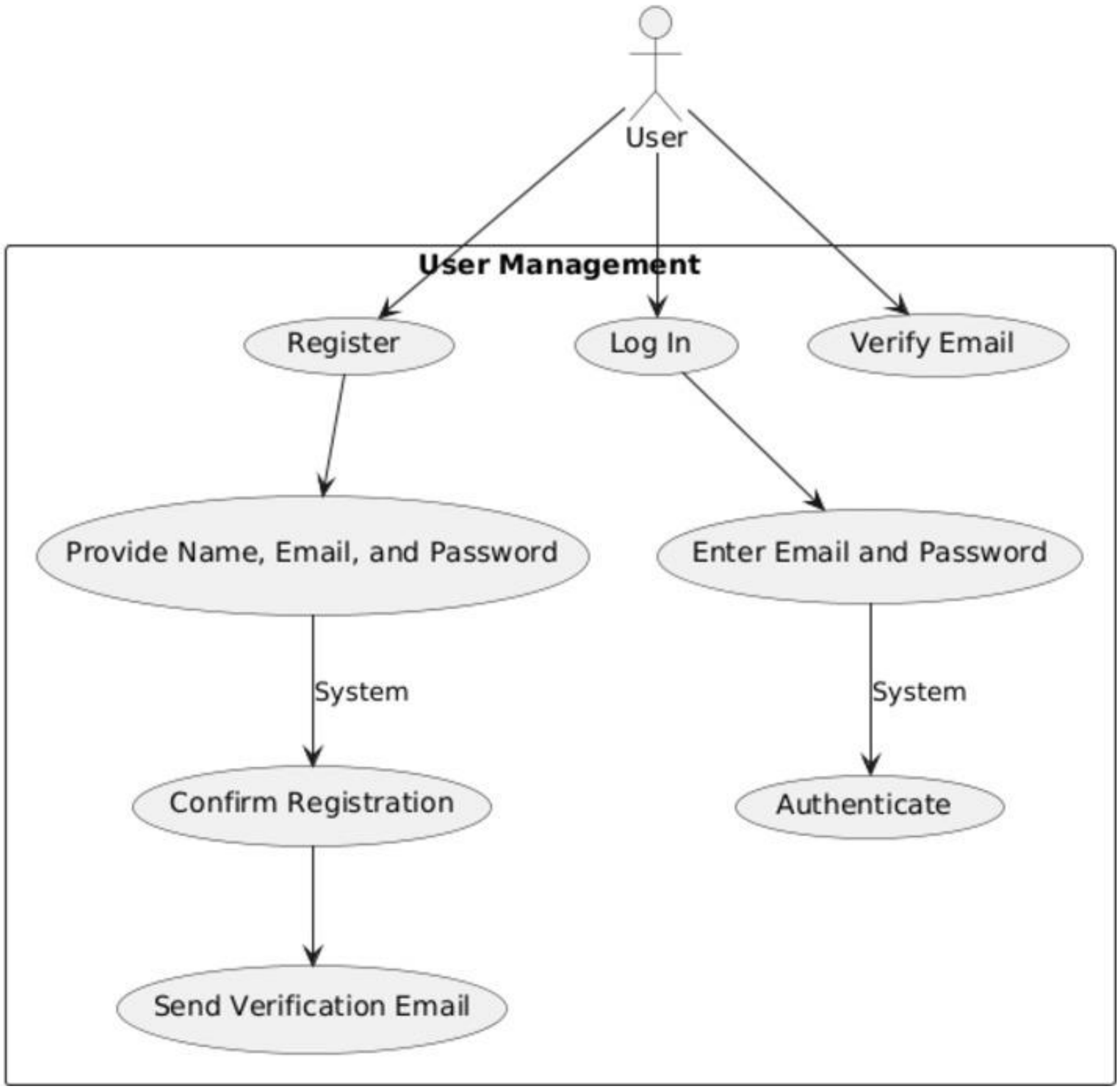

Figure 1

Preconditions:

1. The user must have access to the registration system.
2. All required fields (name, email, password) must be entered correctly.

Brief Description: This use case allows new users to create an account by providing their name, email, and a password. The system validates the provided data, confirms the registration, and sends a verification email.

Steps:

1. **User Action**: The user provides their name, email, and password in the registration form.
2. **System Validation**:
    - The system checks the format and uniqueness of the email.
    - Ensures the password meets the complexity requirements (e.g., minimum length, character types).
3. **Confirmation**:
    - The system confirms the registration and stores the user’s data securely.
4. **Email Verification**:
    - A verification email is sent to the user with a link or code to activate the account.
5. **Activation**:
    - The user clicks the verification link or enters the provided code.
    - The system activates the user account, allowing login and access to additional features.

Postconditions:

4 The user account is successfully created and activated.
5 The system acknowledges the registration and redirects the user to the login page.

3.2.2 USER INTERACTION FLOW USE CASE

User case:

- Login
- Upload image
- Look Past Results
- Give Feedback
- Help

Diagram:

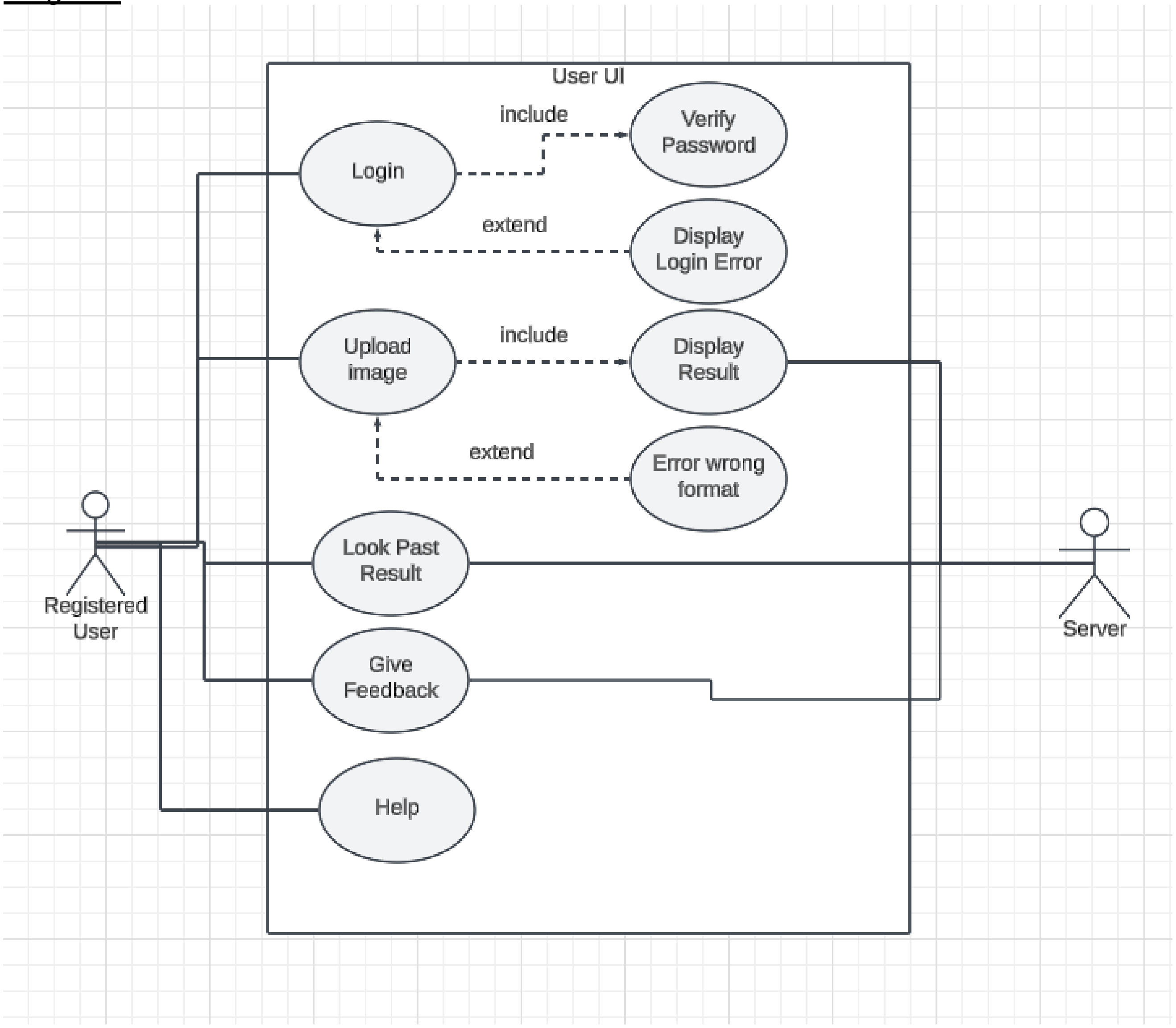

Figure 2

Brief Description: Figure 2 illustrates a Use Case diagram of the user's homepage screen. When the user accesses the homepage, they can take a photo of their skin and upload it to the system to check their likelihood of having skin cancer. The system analyzes the uploaded photo, calculates the cancer probability, and displays the result on the screen. Additionally, the user can view the results of their previous analyses in the "Past Results" section. Moreover, the user can provide feedback about the system to share their experience.

- If the login is successful, the user is redirected to the homepage.
- If the login fails, a message such as "Incorrect password" or "Incorrect email" is displayed on the screen.
- After accessing the homepage, the user is prompted to upload a photo of their skin and click the "Evaluate" button.
    - If the uploaded photo is not in png or jpg format, an error message appears, indicating the incorrect file format.
    - If the photo format is correct, the system proceeds to analyze the image. The user is informed that the analysis is in progress, and once completed, the results are displayed on the screen.
- Once the analysis is complete, the results are displayed on the screen.
- The user can also view their past results by clicking on the "Past Results" section on the homepage, which displays a list of previous evaluations. Each entry includes relevant details such as the date and analysis outcome.
- After completing all tasks, the user can leave feedback by navigating to the "Feedback" section, where they can provide their comments and suggestions.
- If the user needs assistance, they can click on the "Help" section for guidance, which provides instructions and answers to frequently asked questions.

.

3.2.3 CLIENT USE CASE

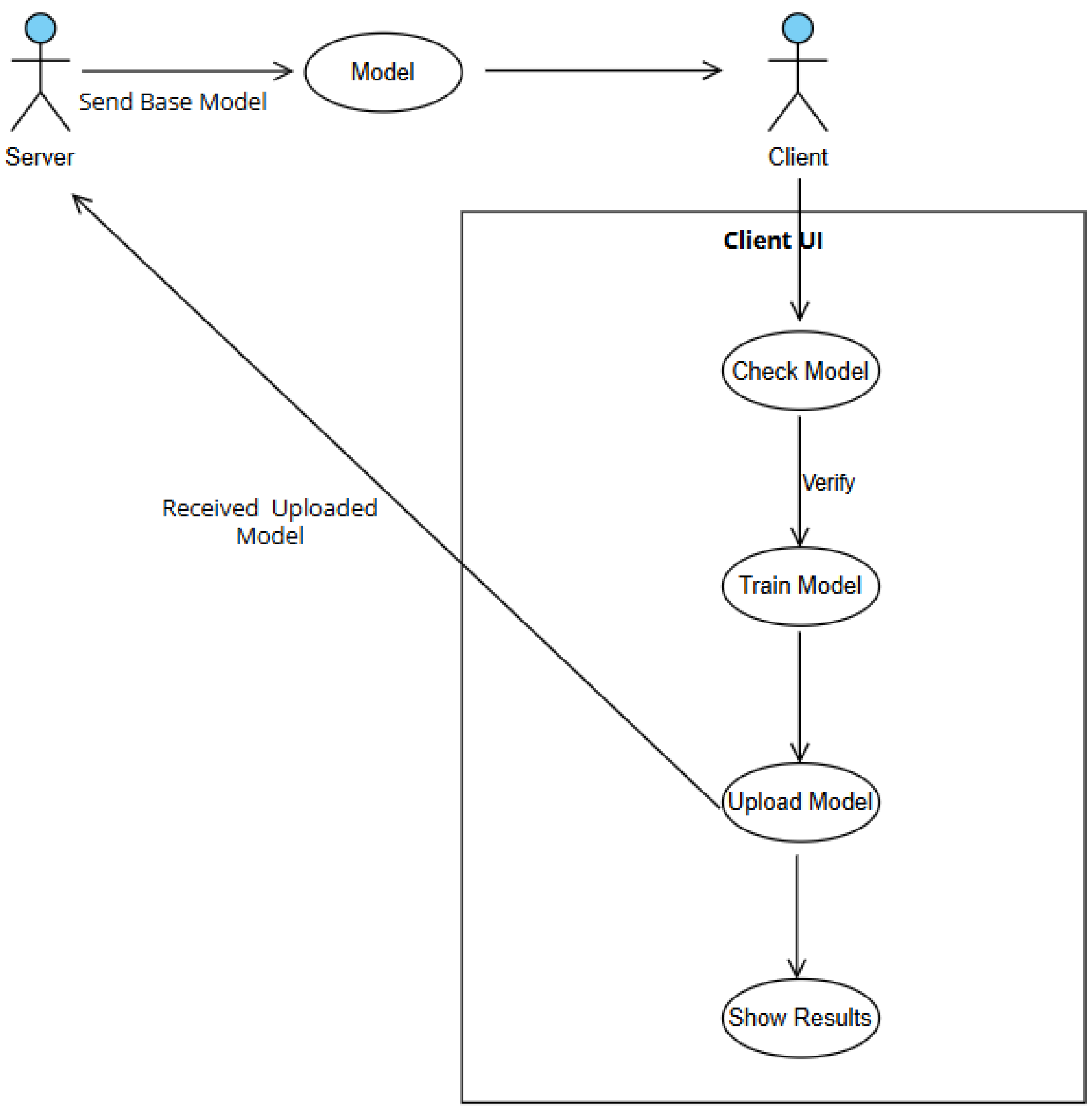

Figure 3

Preconditions

- Base Model Availability: The server must have a base model ready to send to the client.
- Client Connectivity: The client must be connected to the server to receive the base model.
- Data Readiness: Training data must be available on the client-side and properly formatted for model training.
- Client UI Accessibility: The client user interface must be operational for initiating and monitoring the process.
- Resource Availability: The local device should meet the computational requirements for model training (e.g., sufficient memory, CPU/GPU power).

Brief Description: This use case describes the process by which a client user trains a model locally using provided data. Once the model is trained, it is uploaded to a server. The results of the training process are displayed to the user for review.

Steps:

1. Server Sends Model:

    a. When the system runs for the first time: The server sends the **base model** to the client device through the model component.

    b. On subsequent invocations: The server sends the **last updated model** to the client.

2. The client user verifies that the model (base or updated) has been received.

3. The client user initiates a check for the availability and validity of the model and training data through the client UI.

4. The system ensures that the model and training data are present and valid for training.

5. The client user starts the training process through the client UI. (The system utilizes local resources to train the model using the provided data.)

6. The trained model is sent back to the server.

7. Show the results.

Postconditions

- Model Training Completion: The model is successfully trained using local data on the client device.
- Trained Model Uploaded: The trained model is sent to the server for further use, aggregation, or deployment.
- Training Results Displayed: The client user is able to view the results of the training process.

3.2.4 Server Management Use Case

Use Case:

- Send Base Model to Client Devices
- Receive updates from Client Devices
- Merge received updates
- Update central model
- Display update status in Model Management section
- Retry update process if a failure occurs
- Log errors and notify system administrator

Diagram:

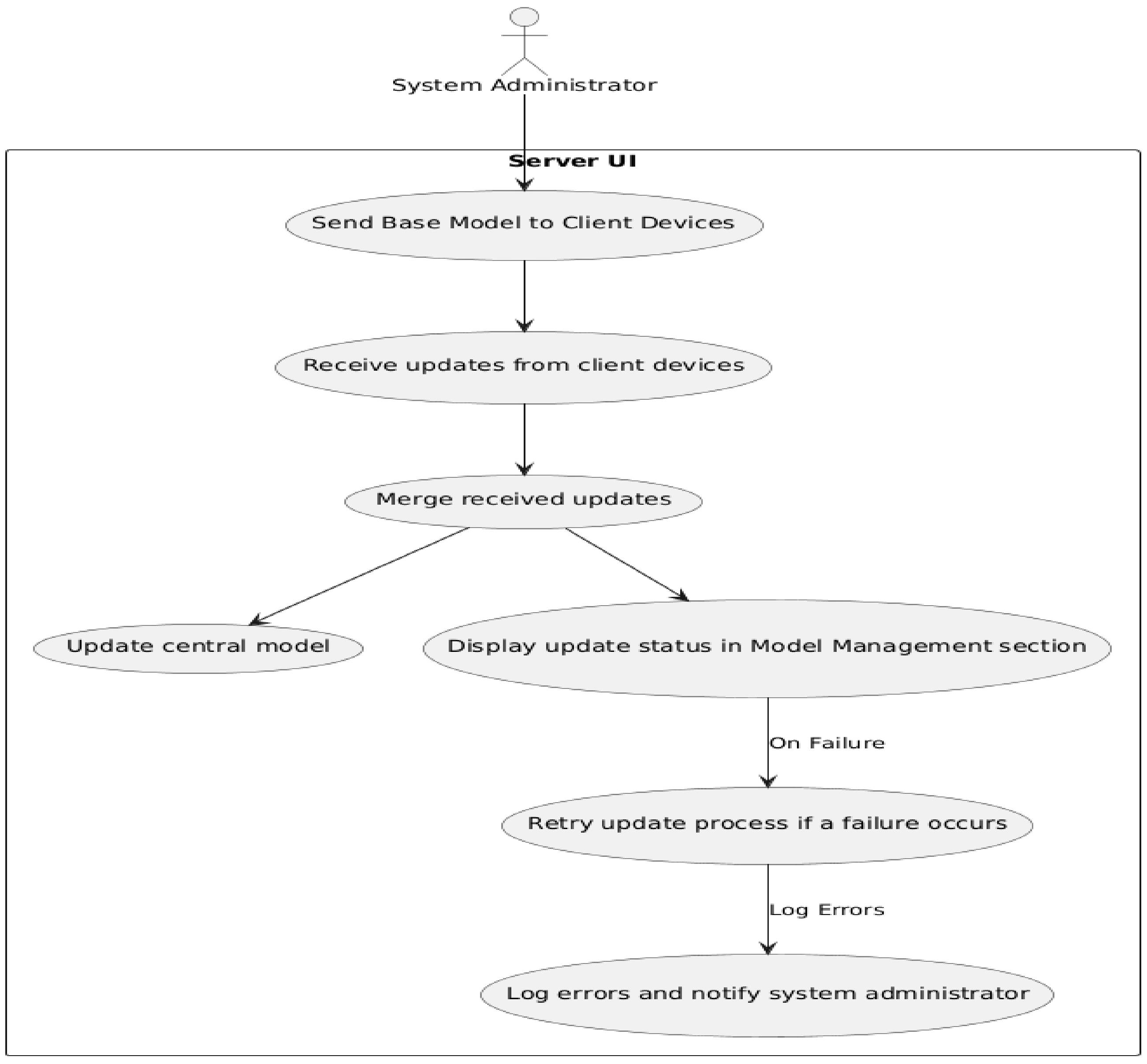
System Administrator
Server UI
Send Base Model to Client Devices
Receive updates from client devices
Merge received updates
Update central model
Display update status in Model Management section
On Failure
Retry update process if a failure occurs
Log Errors
Log errors and notify system administrator

Figure 4

Preconditions

1. **Base Model Must Be Available:**
    - The base model must be ready and accessible on the server before sending it to clients.
2. **Client Devices Must Be Connected**:
    - All participating client devices must be connected to the server and actively transmitting updates.
3. **Central Server Must Be Running**:
    - The central server must be operational and capable of receiving and processing updates.
4. **Valid Update Data**:
    - The updates sent by the client devices must be in the correct format and validated for integrity (e.g., through hash checks).

Brief Description of the Use Case

This use case represents the key operations performed within the **Server UI** in a federated learning system to manage and update the central model effectively. The primary actor is the **System Administrator**, who oversees the process. The use case involves the following steps:

1. **Send Base Model to Client Devices:** The Server UI sends the base model to the client devices. The base model includes initial weights and architecture, which the clients will use to train locally on their datasets.
2. **Receive Updates from Client Devices**: The server collects model updates (e.g., gradients or weights) sent by client devices participating in the federated learning process.
3. **Merge Received Updates**: The collected updates from multiple client devices are merged using a defined aggregation algorithm (e.g., weighted average) to create a unified update for the central model.
4. **Update Central Model**: The merged updates are applied to the central model to improve its performance and accuracy.
5. **Display Update Status**: The system displays the status of the update (e.g., success or failure) in the Model Management section of the Server UI for the administrator to review.
6. **Handle Failures**:
    - If the update process encounters any failures, the system retries the process.
    - In case of persistent failures, the system logs the errors and notifies the system administrator, ensuring transparency and providing insights for troubleshooting.

3.3 Performance Requirement

The system must meet the following performance requirements to ensure accurate and efficient skin cancer detection while providing a seamless user experience:

1. **Image Analysis Speed**: The analysis of a single image should be completed within a maximum of 10 seconds to ensure quick feedback for the users.
2. **Model Update Latency**: Model updates and weight transfers must occur with minimal delay, with a maximum latency of 5 seconds during federated learning processes.
3. **Model Accuracy**: The central model's performance must maintain an accuracy rate of at least 90% to ensure reliable results.
4. **Resource Optimization**:
    - CPU and memory usage should be optimized during model training and image analysis to prevent system overloads.
    - Libraries like TensorFlow [1], PyTorch[2], and NumPy will be used to enhance the computational efficiency of the system.
5. **Low-Latency Communication**:
    - Tools like Kafka-ML [3] will be utilized to ensure low-latency data transfers between client and central servers in the federated learning process.
6. **Hardware Requirements**:
    - Minimum 8 GB RAM to run the application without interruptions.
    - Multi-core processors to support parallel processing during training and inference tasks.

3.4 Software System Attributes

3.4.1. Portability

- The application must run seamlessly on different platforms (Windows, macOS, Linux).
- The user interface must be compatible with both desktop and tablet devices.
- The federated learning system must support client devices with various operating systems (e.g., Windows, Linux).
- Tools like **PyInstaller** can be used for cross-platform distribution, and Docker containers can enhance the portability of the central server.

3.4.2. Performance

- The analysis of a single image must be completed within a maximum of 10 seconds.
- Model updates and weight transfers must occur with minimal delay (maximum latency: 5 seconds).
- The central model's performance must maintain an accuracy rate of at least 90%.
- CPU and memory usage must be optimized during training.
- Libraries like NumPy, TensorFlow, or PyTorch can enhance system performance, and Kafka-ML can ensure low latency during data transfer.

3.4.3. Usability

- The user interface must be intuitive and easy to understand.
- Analysis results must be presented clearly with visual support (e.g., charts, color-coded outcomes).
- Users must receive clear error messages for invalid inputs or uploads.
- A user-friendly help section or tutorial must be provided.
- Frameworks like Tkinter or PyQt5 can be used to create an accessible UI, and user testing can validate the design's intuitiveness.

3.4.4. Adaptability

- The software must support the inclusion of new skin cancer types with minimal changes.
- New federated learning protocols or model architectures must be easy to integrate.
- The system must adapt to new data uploaded by users through incremental learning.
- A modular software architecture must be implemented to ensure flexibility, and incremental learning algorithms should be seamlessly integrated.

3.4.5. Scalability

- The system must be capable of handling updates from multiple client devices simultaneously, with scalability options to increase capacity as needed. Initially, the system should support at least **10-20 clients** for testing and small-scale deployment, with plans to expand based on infrastructure upgrades and user demand.
- The central server must scale to accommodate models working with diverse datasets.
- Analysis processes must remain efficient even with large datasets.

3.4.6. Security

- **Data Privacy and Image Integrity**:
  - The system must ensure that only valid skin lesion images are uploaded by users. If any irrelevant images (e.g., images of non-skin-related objects) are detected, the system must reject them and notify the user with an error message such as "Invalid image format. Please upload a valid skin lesion image."
  - To prevent the misuse of the platform, all images should be checked for file integrity, size, and format (e.g., JPEG, PNG) before processing. The system must not accept any other file types or corrupted images.
  - All user-uploaded images should be processed locally on the client device, ensuring no raw image data is transmitted to the central server. Only model updates (e.g., gradients) should be sent to the central server, preserving user privacy and compliance with data protection regulations (e.g., GDPR).
- **Security Measures for Data Transmission**:
  - All data exchanged between the client devices and the central server must be encrypted using secure communication protocols (e.g., TLS/SSL). This ensures that the data (including model updates) cannot be intercepted or altered during transmission.
  - The system must implement authentication and authorization protocols to ensure that only authorized clients can interact with the server. Each client device should authenticate with a secure token or similar mechanism before sending model updates to the central server.
- **Handling Irrelevant or Malicious Files**:
  - If a user uploads an image that is flagged as irrelevant or malicious (e.g., containing harmful content), the system should immediately stop the analysis process and display a clear message to the user: "The image could not be processed due to security reasons."
  - The system should log any such incidents to provide a record of attempted misuse for system administrators.
- **Handling Timeouts and Long Processing**:

    - If an image analysis takes longer than 10 seconds (e.g., due to system load or large image size), the system should automatically abort the process and prompt the user with a message such as, "The analysis could not be completed within the expected time. Please try again."
    - Users should have the option to either retry the analysis or upload a smaller image.
    - In case of system failure or excessive processing time, the system should not automatically restart. Instead, it should prompt the user with a notification that allows them to either retry or abort the analysis.
    - A timeout mechanism should be implemented to ensure that the analysis process does not hang indefinitely, improving the user experience and preventing system overload.
- **Malware and Threat Detection**:
    - The system should include malware scanning of uploaded files to prevent the risk of virus or malware attacks via file uploads.
    - Any suspicious activity or malicious attempts to upload harmful content should be logged and reviewed by the system administrator for further action.
- **System Recovery in Case of Errors**:
    - If the system encounters an unexpected error during image processing (e.g., timeout, memory overload, or corrupted data), it should gracefully handle the failure by providing a user-friendly error message.
    - The system should include a recovery option, allowing users to retry the operation, but should never force a system restart or cause data loss.

3.5 Safety Requirement

Prevention of Misdiagnosis:

- The system must clearly state that the analysis results are intended as a supportive tool and do not substitute professional medical diagnosis.
- Results should include a disclaimer such as: "This is not a diagnosis, but a probability analysis."

Secure Data Processing:

- Images uploaded by users must be processed only on the local device and never sent to a central server.
- Data must not be shared with other users or systems under any circumstances.

Reliability of Analysis:

- The system must provide the model's accuracy rate and confidence score with every analysis result.
- If the confidence score falls below a certain threshold (e.g., 90%), the user should be notified with a warning.

Protection Against User Errors:

- Unsupported file types (e.g., PDFs or text files) or invalid inputs must trigger error messages to guide the user.

- Before starting an analysis, the system should inform the user about acceptable file formats and requirements.

Backup and Recovery During System Failures:

- If the system crashes or encounters an interruption during analysis, the process must be automatically saved, allowing the user to resume from where they left off.
- Local training processes should also be restartable in case of interruptions.

Critical System Performance:

- The system must be optimized to avoid overheating or crashes under heavy workloads (e.g., analyzing large datasets).
- Performance monitoring tools should regularly check the system's status.

Secure Updates:

- Model updates from the central server must be verified for integrity (e.g., through hash checks) before being applied.
- A backup version of the model must always be available to mitigate risks associated with faulty updates.

Protection Against Misuse:

- The system must validate the type and format of uploaded data to prevent users from uploading irrelevant or inappropriate files.
- For instance, only image files (JPEG, PNG) should be accepted.

4. REFERENCES

1. Federated Learning with TensorFlow: TensorFlow Federated (TFF) Documentation, https://www.tensorflow.org/federated
2. PyTorch Documentation: PyTorch machine learning framework, https://pytorch.org/docs/stable/index.html
3. Kafka-ML: Kafka-ML for Federated Learning, https://github.com/EnzoPerri/kafka-ml