Project work phase - 2(19AI702)

Clinical Scan Support System

Submitted by:

V.SRIRAM (212222103002)

R.SUROTHAAMAN (212222103003)

C.K.PRAVEEN(212222243003)

2022-2026 Batch

TEAM NO: 058

Under the guidance of:

V.SWEDHA

NAME OF THE GUIDE

V.SWEDHA

Designation, Department of IT

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING(CS)

SAVEETHA ENGINEERING COLLEGE

(Autonomous Institution – UGC, Govt. of India)

(Affiliated to Anna University, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Saveetha Nagar, Thandalam, Chennai-602 105, TamilNadu, INDIA.

Agenda

1. Introduction
2. Statement of the Problem
3. Scope of the Project
4. Methodology
5. Architectural Diagram
6. Flow
7. Algorithm Used
8. Design
   - Use Case Diagram,
   - Class Diagram,
   - Sequence Diagram
9. Implementation
10. Important Code Segments
11. Output
12. Test Cases
13. Results
14. Conclusion
15. Future Work
16. References

Introduction

- Chest radiography and cardiac MRI are critical for diagnosing lung and heart diseases globally.
- Traditional diagnostic workflows are manual, siloed, and slow
- relying on radiologists interpreting scans individually with no unified system connecting patients, doctors, pharmacists, and administrators.
- The COVID-19 pandemic exposed severe bottlenecks in manual radiology workflows, highlighting the urgent need for AI-assisted diagnostic platforms.
- CSSS is a full-stack AI-powered medical imaging platform that automates the complete diagnostic pipeline
- from scan upload through AI inference, multi-role clinical review, to automated PDF report generation and encrypted email delivery.
- The system uses MobileNetV2 deep learning model trained on 217,875 medical images across 6 disease classes: COVID, Lung Opacity, NIH Merged, Normal, Sick, and Viral Pneumonia.
- CSSS achieves 89.51% test accuracy with sub-second inference speed, making it clinically viable for real-world hospital deployment.

Statement of the Problem

Challenge:

- **Time-Consuming Manual Diagnosis:** Traditional diagnosis relies on physical examination, manual imaging interpretation, and lab tests - causing treatment delays.
- **No Integrated Clinical Workflow**: Patient, doctor, pharmacist, and admin operate in separate isolated systems with no unified pipeline or status tracking.
- **Subjectivity in Radiological Interpretation:** Radiologists' interpretations vary due to fatigue, experience level, and diagnostic inconsistency.
- **Resource Constraints in Low-Income Settings:** Many healthcare facilities lack radiological specialists or advanced diagnostic infrastructure.
- **Manual Report Generation and Delivery:** Creating and delivering diagnostic PDF reports to patients is slow, manual, and error-prone.
- **Lack of AI Integration:** Existing hospital systems do not embed AI inference within clinical management workflows.

Objective:

- Build a production-grade AI-powered clinical platform that:
- Automatically classifies medical scans using deep learning (MobileNetV2 with 89.51% accuracy)
- Enforces a structured 4-role clinical review workflow (Patient → Doctor → Pharmacist → Admin)
- Auto-generates professional PDF diagnostic reports and delivers them securely via encrypted SMTP email

Scope of the project

Scope:

- Develop a full-stack web platform using FastAPI (Python) backend and Next.js 14 (React) frontend.
- Train and integrate a MobileNetV2 deep learning model for real-time medical scan classification.
- Implement a structured 4-role clinical workflow: Patient → Doctor → Pharmacist → Administrator.
- Auto-generate professional PDF diagnostic reports using WeasyPrint + Jinja2 HTML templating.
- Deliver encrypted reports securely to patients via Gmail SMTP email.
- Provide a rule-based medical AI chatbot for patient and clinical staff assistance.

Target Audience:

- Hospital Radiology Departments for chest X-ray and cardiac MRI screening
- COVID-19 Screening Clinics for real-time viral pneumonia detection
- Cardiac Screening Centers for automated normal/abnormal MRI triage
- Telemedicine Platforms for remote scan submission and report delivery
- Medical Education Institutions for AI-assisted diagnostic training

Deliverables:

- Trained MobileNetV2 model (lung_model.h5) - 89.51% test accuracy on 217,875 images
- Full-stack web application with 4 role-specific dashboards (Admin, Doctor, Pharmacist, Patient)
- Automated clinical PDF report generation system
- JWT + OTP 2FA security architecture
- Technical documentation, API reference, and deployment guide

Scope of the project

Inclusions:

- Chest X-ray classification (NIH dataset — 14 thoracic pathologies merged)
- COVID-19 radiography detection (COVID, Normal, Lung Opacity, Viral Pneumonia)
- Cardiac MRI triage (Normal / Sick classification)
- Patient scan upload portal with drag-and-drop interface and real-time progress tracking
- Role-based access control (JWT tokens + bcrypt password hashing + OTP 2FA for admin)
- Medical AI chatbot with 10 topic categories for patient FAQ and workflow assistance
- PDF diagnostic report generation with WeasyPrint and encrypted SMTP email delivery

Exclusions:

- Real-time patient monitoring or live vitals tracking
- DICOM medical file format support (planned for Version 2.0)
- Mobile native application (planned as PWA in Version 2.5)
- Integration with existing EMR/EHR hospital systems (planned for Version 2.5)

Limitations:

- Model performance may vary on scan quality distributions outside the training dataset.
- All AI predictions must be verified by a licensed physician before any clinical decision.
- System requires stable internet connection for SMTP email report delivery.
- Current database: SQLite (development) — PostgreSQL migration required for production scale.
- Confidence predictions below 75% are flagged as "Uncertain" and require mandatory physician review.

Scope of the project

Dataset Source & Composition:

- **Source:** Kaggle (publicly available medical imaging datasets)
- **Total Images:** 217,875 across 6 disease classes
- **Split Ratio:** 70% Training / 15% Validation / 15% Test
- **Fixed Random Seed:** 42 (ensures reproducibility across training runs) Dataset
- Dataset Breakdown by Source:

Dataset Source	Disease Classes	Images
NIH Chest X-ray8 Dataset	NIH_MERGED (14 pathologies)	112120
COVID-19 Radiography Database	COVID, Normal, Lung_Opacity, Viral_Pneumonia	42330
CAD Cardiac MRI Dataset	Normal (Cardiac), Sick	63425
TOTAL	**6 Disease Classes**	**217875**

Predicted Disease Classes: • COVID • Lung_Opacity • NIH_MERGED • Normal • Sick • Viral_Pneumonia

Methodology

The system was developed using the following structured methodology:

1. Data Collection: Collected 217,875 medical images from 3 Kaggle public datasets (NIH Chest X-ray8, COVID-19 Radiography, CAD Cardiac MRI) covering 6 disease classes.

2. Data Preprocessing & Splitting: Developed split_lung_dataset.py to split data into 70/15/15 train/validation/test partitions using sklearn train_test_split with fixed random seed 42. Applied image preprocessing: resize to 224×224, normalize to [0,1], augmentation (rotation ±10°, zoom 0.1, horizontal flip).

3. AI Model Development (MobileNetV2 Transfer Learning): Applied transfer learning using ImageNet-pretrained MobileNetV2 as frozen feature extractor. Added classification head: GlobalAveragePooling2D → Dense(128, ReLU) → Dropout(0.4) → Dense(6, Softmax). Trained for 15 epochs using Adam optimizer (lr=1e-4), categorical cross-entropy loss, with EarlyStopping (patience=5) and ModelCheckpoint callbacks. Final test accuracy: 89.51%.

Methodology

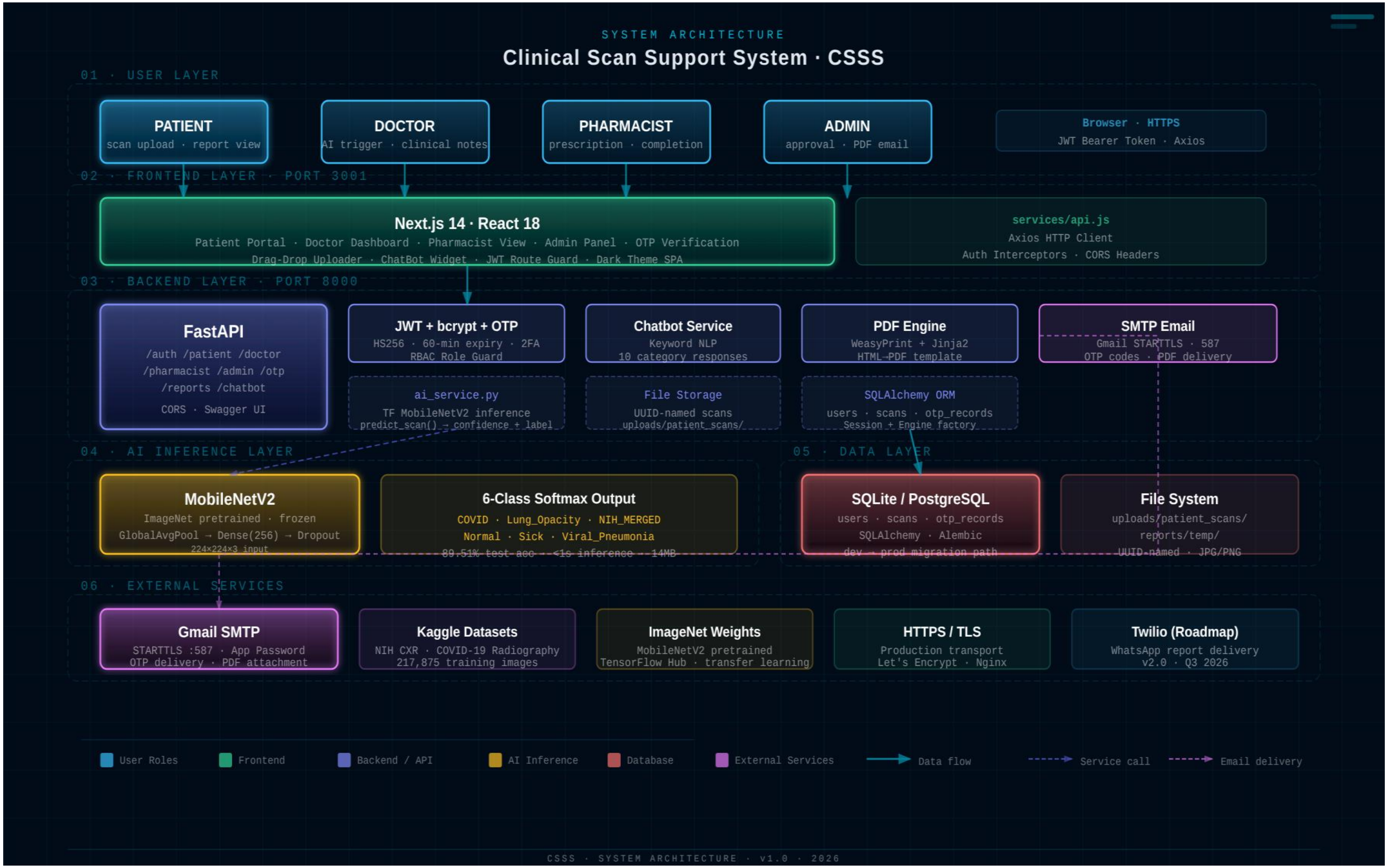
The system was developed using the following structured methodology:

4. Backend API Development (FastAPI): Built REST API with 8 router groups: /auth, /patient, /doctor, /pharmacist, /admin, /otp, /chatbot, /reports. Implemented JWT token authentication (HS256, 60-min expiry), bcrypt password hashing, and OTP 2FA for admin. Used SQLAlchemy ORM with SQLite (development) and documented PostgreSQL migration path.

5. Frontend Development (Next.js 14): Built 4 role-specific dashboards (Admin, Doctor, Pharmacist, Patient) using Next.js 14 + React 18. Implemented drag-and-drop scan upload, real-time status tracking with color-coded badges, OTP verification module, and protected PDF download functionality.

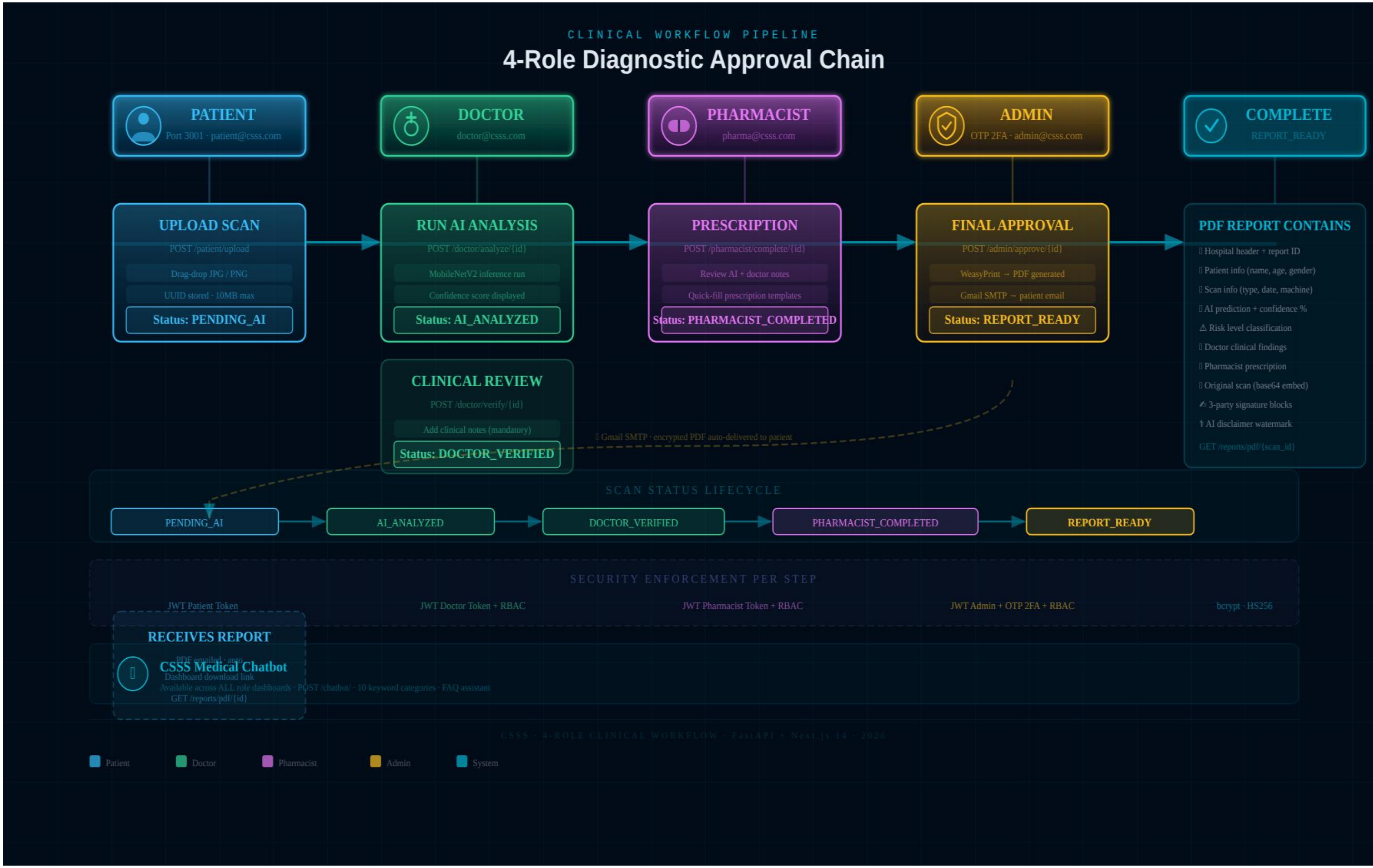
6. Integration & End-to-End Testing: Integrated TensorFlow AI inference into doctor workflow. Implemented WeasyPrint + Jinja2 PDF generation and Gmail SMTP encrypted email delivery. Tested all 5 scan lifecycle states (PENDING_AI → AI_ANALYZED → DOCTOR_VERIFIED → PHARMACIST_COMPLETED → REPORT_READY) end-to-end.

Architecture Diagram/Flow



SAVEETHA AUTONOMOUS ENGINEERING COLLEGE

Architecture Diagram/Flow

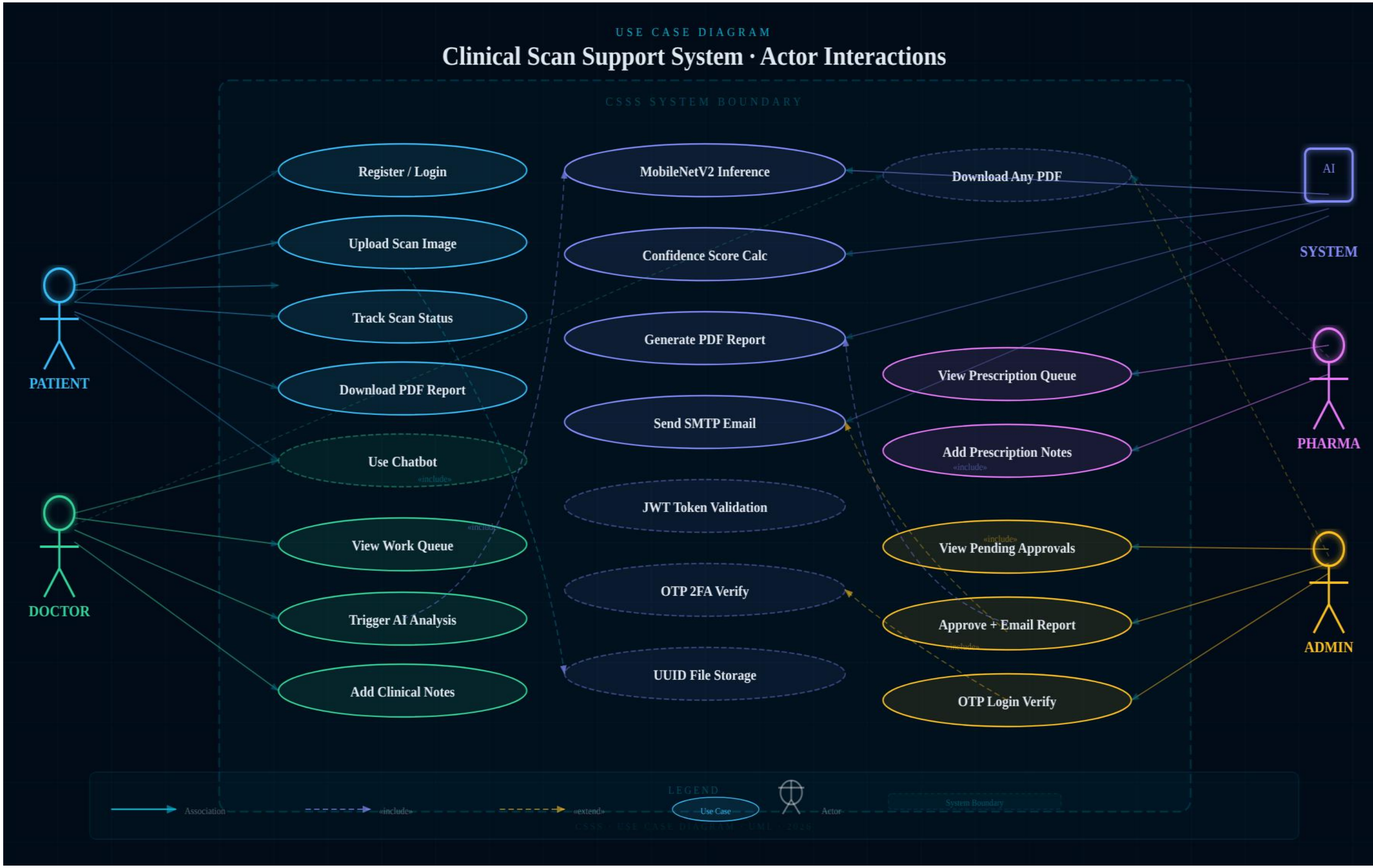


Design-Use Case Diagram

Use Case Description:
A Use Case Diagram shows the functional interactions between system actors (users) and the system boundary (CSSS platform). The CSSS system has 4 primary actors: Patient, Doctor, Pharmacist, and Admin. Each actor has role-specific use cases enforced by JWT Role-Based Access Control (RBAC).

Actors and Their Use Cases:
PATIENT: - Register Account - Login (JWT authentication) - Upload Scan Image (JPG/PNG) - View Scan Status (PENDING_AI, AI_ANALYZED, DOCTOR_VERIFIED, etc.) - Download PDF Report (only for REPORT_READY status) - Use Medical Chatbot
DOCTOR: - Login (JWT authentication) - View Scan Queue (all PENDING_AI scans) - Trigger AI Analysis (MobileNetV2 inference) - View AI Prediction + Confidence Score - Add Clinical Verification Notes - Verify Scan (advance to DOCTOR_VERIFIED status) - Use Medical Chatbot
PHARMACIST: - Login (JWT authentication) - View Scan Queue (all DOCTOR_VERIFIED scans) - Review AI Prediction + Doctor Notes - Add Prescription Notes - Complete Scan (advance to PHARMACIST_COMPLETED status) - Use Medical Chatbot
ADMIN: - Login with OTP 2FA (6-digit email verification) - View Pending Approvals (all PHARMACIST_COMPLETED scans) - Review Full Scan Details (AI + Doctor + Pharmacist notes) - Approve Report (trigger PDF generation + email delivery) - Use Medical Chatbot

Design-Use Case Diagram



Design-Class Diagram

Design-Sequence Diagram

Sequence Flow – Scan Upload to AI Prediction:

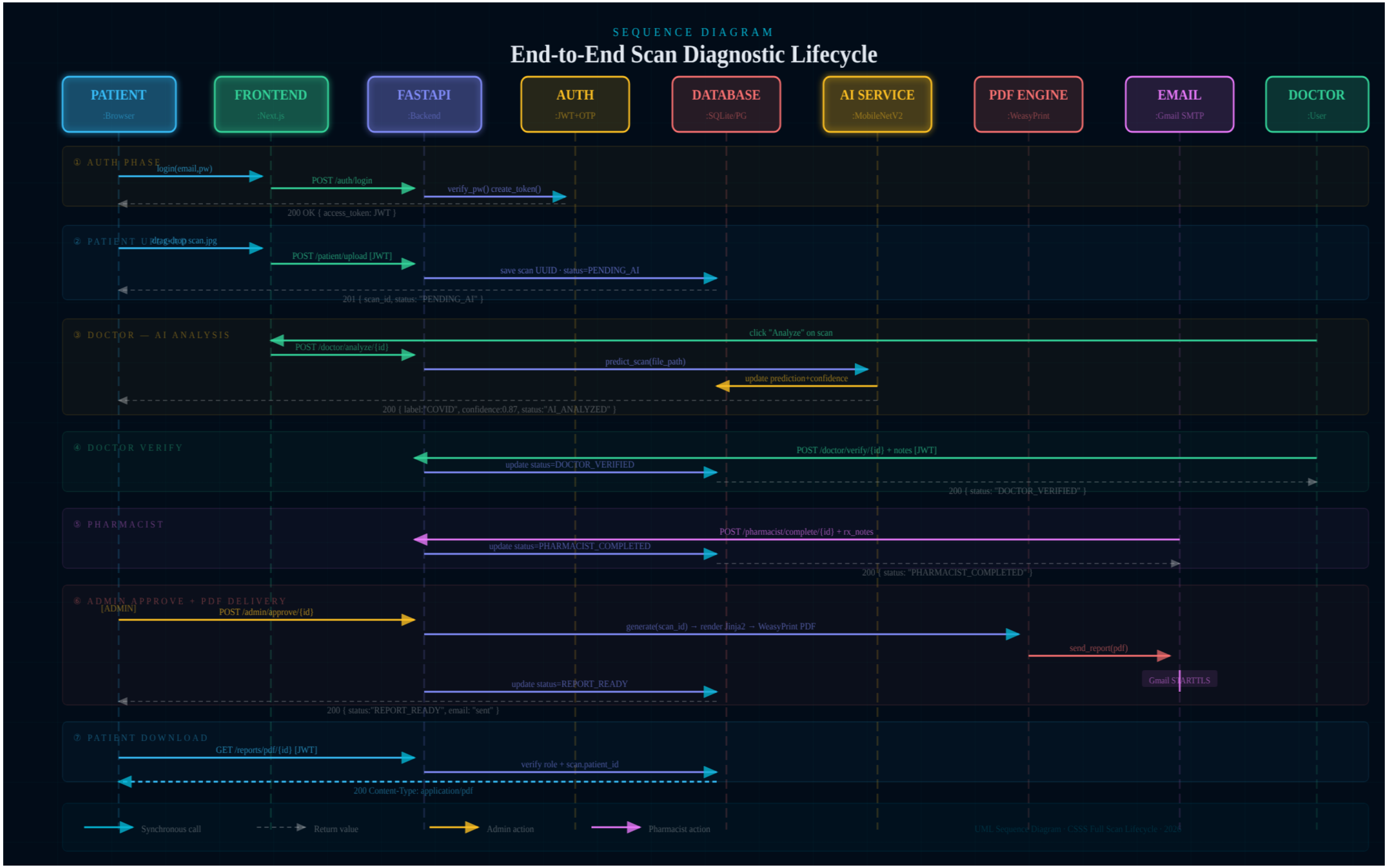
1. Patient → Frontend: Drag-and-drop scan image (JPG/PNG, < 10 MB)
2. Frontend → FastAPI (/patient/upload): POST multipart form with JWT token
3. FastAPI → Filesystem: Save scan as {UUID}.jpg in uploads/patient_scans/
4. FastAPI → Database: INSERT new scan record (patient_id, file_path, status=PENDING_AI)
5. FastAPI → Frontend: Return {scan_id, status: "PENDING_AI"}
6. Doctor → Frontend: Login → view scan queue → click "Analyze" on pending scan
7. Frontend → FastAPI (/doctor/analyze/{scan_id}): POST request with JWT token
8. FastAPI → ai_service.predict_scan(file_path): Trigger MobileNetV2 inference
9. ai_service → lung_model.h5: Load image → preprocess → forward pass → softmax output
10. lung_model.h5 → ai_service: Return {label: "COVID", confidence: 0.914, all_predictions: {...}}
11. FastAPI → Database: UPDATE scan SET prediction="COVID", confidence=0.914, status="AI_ANALYZED"
12. FastAPI → Frontend: Return {prediction: "COVID", confidence: 91.4%}
13. Doctor → Frontend: Review prediction → add clinical notes → click "Verify"
14. Frontend → FastAPI (/doctor/verify/{scan_id}): POST {doctor_notes: "..."}
15. FastAPI → Database: UPDATE scan SET doctor_notes="...", status="DOCTOR_VERIFIED"
16. FastAPI → Frontend: Return success confirmation

Design-Sequence Diagram

Sequence Flow – Admin OTP Login to PDF Report Delivery:

1. **Admin → Frontend:** Enter email + password
2. **Frontend → FastAPI (/auth/login):** POST credentials
3. **FastAPI → Database:** Verify email exists + bcrypt password match
4. **FastAPI → Random.randint(100000, 999999):** Generate 6-digit OTP
5. **FastAPI → Database:** INSERT OTPRecord (email, otp, expires_at = now + 10 min)
6. **FastAPI → Gmail SMTP:** Send OTP email to admin
7. **FastAPI → Frontend:** Return {message: "OTP sent to email"}
8. **Admin → Frontend:** Enter 6-digit OTP code
9. **Frontend → FastAPI (/otp/verify):** POST {email, otp}
10. **FastAPI → Database:** Query OTPRecord WHERE email=? AND otp=? AND used=False AND expires_at > now
11. **FastAPI → Database:** UPDATE OTPRecord SET used=True
12. **FastAPI → JWT:** create_access_token({email, role: "admin"})
13. **FastAPI → Frontend:** Return {access_token, role: "admin"}
14. **Admin → Frontend:** View pending approvals → click "Approve" on scan
15. **Frontend → FastAPI (/admin/approve/{scan_id}):** POST with JWT token
16. **FastAPI → Database:** Query scan + patient details
17. **FastAPI → Jinja2 Template:** Render report_template.html with scan data
18. **FastAPI → WeasyPrint:** Convert HTML → PDF (Report_Scan_{id}.pdf)
19. **FastAPI → Gmail SMTP:** Send email with PDF attachment to patient
20. **FastAPI → Database:** UPDATE scan SET status="REPORT_READY"
21. **Patient → Email:** Receives "Your Diagnostic Report is Ready" with PDF attached
22. **Patient → Frontend (/patient/dashboard):** Can also download PDF directly

Design-Sequence Diagram



Algorithms used

Primary AI Algorithm: MobileNetV2 (Transfer Learning)

- MobileNetV2 is a lightweight, high-efficiency deep convolutional neural network architecture designed for mobile and embedded vision applications.
- Introduced by Sandler et al. (Google, 2018) using Inverted Residual Blocks and Linear Bottleneck layers.
- Key Advantage: Achieves near state-of-the-art accuracy (~89-92% on medical imaging tasks) with significantly reduced model size (~14 MB) and inference time (< 1 second).
- In CSSS, MobileNetV2 serves as a frozen feature extractor (pretrained on ImageNet-1K with 1.2M images), with a custom classification head added for 6 medical disease classes.

Model Architecture Flow:

Input Image (224×224×3)
↓
MobileNetV2 Base (53 layers, ImageNet weights, FROZEN)
↓
GlobalAveragePooling2D
↓
Dense(256, activation='relu')
↓
Dropout(0.4) — prevents overfitting
↓
Dense(6, activation='softmax') — 6 disease classes
↓
Output: {predicted_label, confidence_score, all_class_probabilities}

Confidence Thresholding:

- If max(softmax_output) < 0.75 → prediction = "Uncertain"
- Prevents overconfident incorrect predictions from reaching clinical workflow
- Forces mandatory physician review for low-confidence cases

Algorithms used

Supporting Algorithms & Techniques:

1. JWT (JSON Web Token) – Stateless Authentication:
- HS256 algorithm for token signing
- 60-minute access token expiry
- All API routes protected by JWT Bearer token verification
- Decodes user role for RBAC enforcement

2. bcrypt – Adaptive Password Hashing:
- Industry-standard password hashing with automatic salt generation
- Computational cost factor of 12 rounds
- Prevents rainbow table and brute-force attacks

3. OTP (One-Time Password) – Admin 2FA:
- 6-digit random OTP generated and sent via Gmail SMTP
- 10-minute expiry window
- Single-use verification (marked "used" in database after validation)
- Prevents unauthorized admin access even if password is compromised

4. Grad-CAM (Gradient-weighted Class Activation Mapping):
- Used during model evaluation to generate explainability heatmaps
- Shows which image regions influenced the AI prediction
- Saved as top_misclassified_gradcam.png for clinical validation
- Reference: Selvaraju et al., ICCV 2017

5. WeasyPrint – HTML-to-PDF Rendering:
- Converts Jinja2 HTML templates into professional PDF diagnostic reports
- Embeds scan image as base64, AI predictions, clinical notes, and signatures
- Complies with medical documentation standards

Hardware and software selection

HARDWARE REQUIREMENTS:

Development Environment:

- **Processor:** Intel Core i5 or higher (AMD Ryzen 5 equivalent)
- **RAM:** 16 GB (minimum 8 GB for deployment)
- **Storage:** 50 GB available disk space (SSD recommended)
- **GPU:** NVIDIA GPU with CUDA support (GTX 1050 Ti or higher for training)
- **Network:** Stable internet connection for SMTP email delivery

Production Deployment:

- **Server:** Cloud VPS (AWS EC2 t3.medium or equivalent)
- **RAM:** 8 GB minimum (16 GB recommended for high traffic)
- **Storage:** 100 GB SSD for scan storage and database
- **Network:** High-bandwidth connection for real-time inference

SOFTWARE REQUIREMENTS:

Operating System:

- Windows 10/11, Ubuntu 20.04+, or macOS 12+
- Docker (optional for containerized deployment)

Backend Stack:

- Python 3.10+
- FastAPI 0.115+
- TensorFlow 2.x / Keras
- SQLAlchemy (ORM)
- WeasyPrint (PDF generation)
- python-jose (JWT authentication)
- passlib[bcrypt] (password hashing)
- smtplib (SMTP email)

Frontend Stack:

- Node.js 18.17+
- Next.js 14+
- React 18+
- Axios (HTTP client)

Database:

- SQLite (development)
- PostgreSQL 14+ (production)

Development Tools:

- Visual Studio Code / PyCharm (IDE)
- Git (version control)
- Postman (API testing)

Implementation

The project is implemented using the following modern technology stack:

Backend Stack (FastAPI + Python 3.10+):

- FastAPI – High-performance async REST API framework
- SQLAlchemy – Python ORM for database management
- SQLite (development) / PostgreSQL (production)
- TensorFlow 2.x / Keras – MobileNetV2 model inference
- OpenCV (cv2) – Image preprocessing
- WeasyPrint – HTML-to-PDF report generation
- Jinja2 – HTML template rendering
- python-jose – JWT token creation and validation
- passlib[bcrypt] – Secure password hashing
- smtplib + Gmail SMTP – Email delivery (OTP + PDF reports)

Frontend Stack (Next.js 14 + React 18):

- Next.js 14 – React-based Single Page Application framework
- Axios – HTTP client with JWT interceptor middleware
- React Hooks (useState, useEffect) – State management
- Dark-themed responsive UI with cyan accent colors

AI Model Training Stack:

- TensorFlow / Keras – MobileNetV2 transfer learning
- scikit-learn – Train/val/test split (70/15/15), confusion matrix
- Matplotlib / Seaborn – Training curve and confusion matrix visualization
- OpenCV – Image preprocessing pipeline

Database Schema (SQLAlchemy ORM):

- users table – User accounts with role-based access control
- scans table – Complete scan lifecycle records with AI predictions and clinical notes
- otp_records table – OTP codes with expiry timestamp tracking

Important Code segments

backend/services/ai_service.py

```
import cv2, numpy as np
from tensorflow.keras.models import load_model
import json, os
# Load MobileNetV2 model at startup
model = load_model(os.getenv("AI_MODEL_PATH"))
# Load class names
with open(os.getenv("CLASS_LABELS_PATH")) as f:
    CLASS_NAMES = json.load(f)
CONFIDENCE_THRESHOLD = 0.75
IMG_SIZE = 224
def predict_scan(image_path: str) -> dict:
    # Read and preprocess image
    img = cv2.imread(image_path, cv2.IMREAD_COLOR)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    img = img.astype("float32") / 255.0
    img = np.expand_dims(img, axis=0)
    # MobileNetV2 forward pass
    predictions = model.predict(img, verbose=0)
    confidence  = float(np.max(predictions))
    class_index = int(np.argmax(predictions))
    label       = CLASS_NAMES[class_index]
    # Threshold check
    if confidence < CONFIDENCE_THRESHOLD:
        label = "Uncertain"
    return {
        "label":      label,
        "confidence": confidence,
        "all_predictions": {
            CLASS_NAMES[i]: float(predictions[0][i])
            for i in range(len(CLASS_NAMES))
        }
    }
```

Important Code segments

backend/security/jwt_handler.py | backend/routers/admin.py

JWT Token Creation (jwt_handler.py):

```
from jose import jwt
from datetime import datetime, timedelta
import os
SECRET_KEY = os.getenv("JWT_SECRET_KEY")
ALGORITHM  = "HS256"
def create_access_token(data: dict) -> str:
    to_encode = data.copy()
    expire = datetime.utcnow() + timedelta(minutes=60)
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

Admin Approval & PDF Email (admin.py):

```
from weasyprint import HTML
from jinja2 import Template
def approve_report(scan_id: int, db: Session):
    scan = db.query(Scan).filter(Scan.id == scan_id).first()
    patient = db.query(User).filter(User.id == scan.patient_id).first()
    # Render Jinja2 template
    template = Template(open("templates/report_template.html").read())
    html_str = template.render(scan=scan, patient=patient)
    # Generate PDF
    pdf_path = f"reports/Report_{scan.id}.pdf"
    HTML(string=html_str).write_pdf(pdf_path)
    # Email PDF to patient
    send_email_with_pdf(patient.email, "Your Diagnostic Report", pdf_path)
    # Update status
    scan.status = "REPORT_READY"
    db.commit()
```

Important Code segments

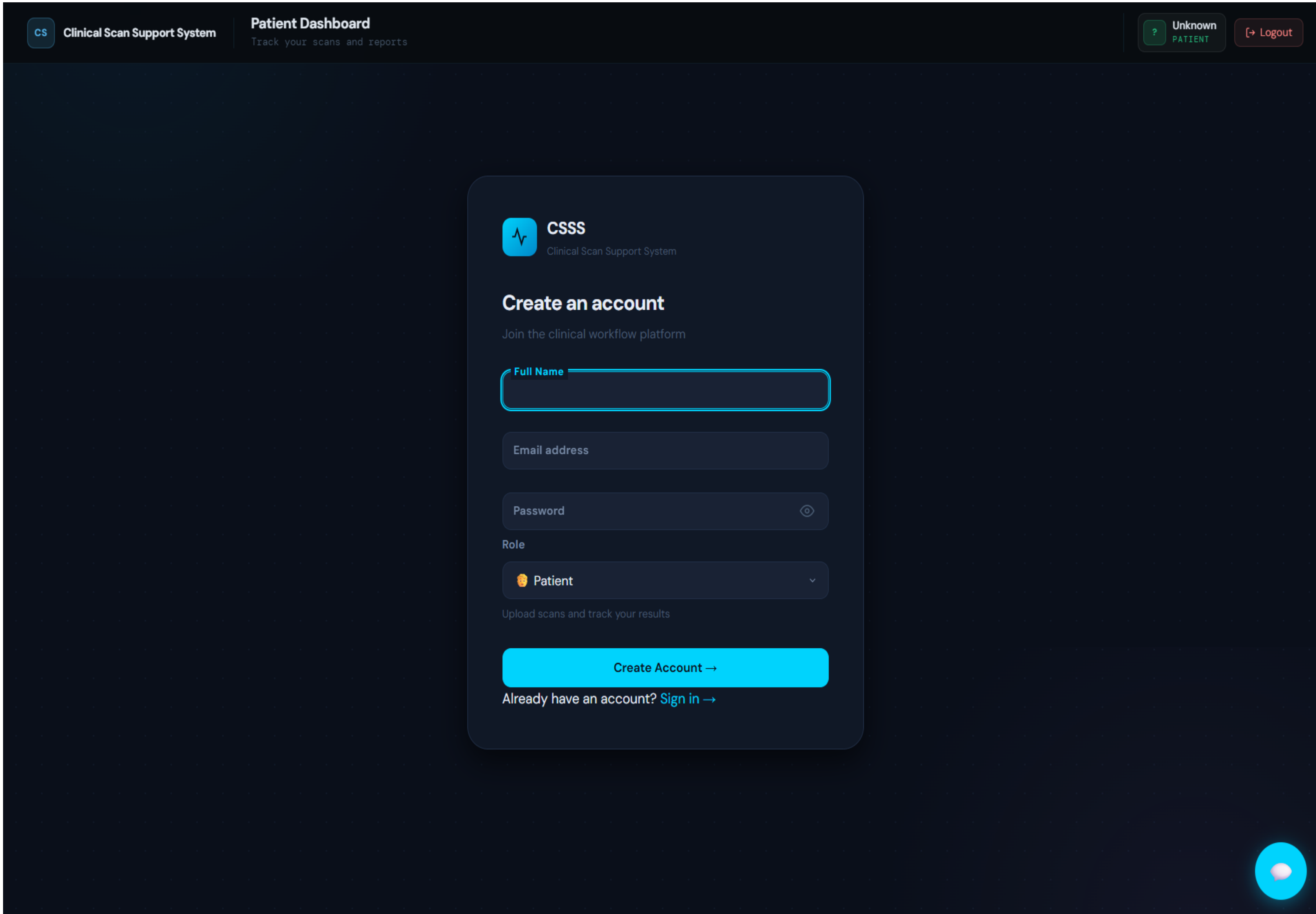
Important Code Segments – MobileNetV2 Training

train_lung_model.py

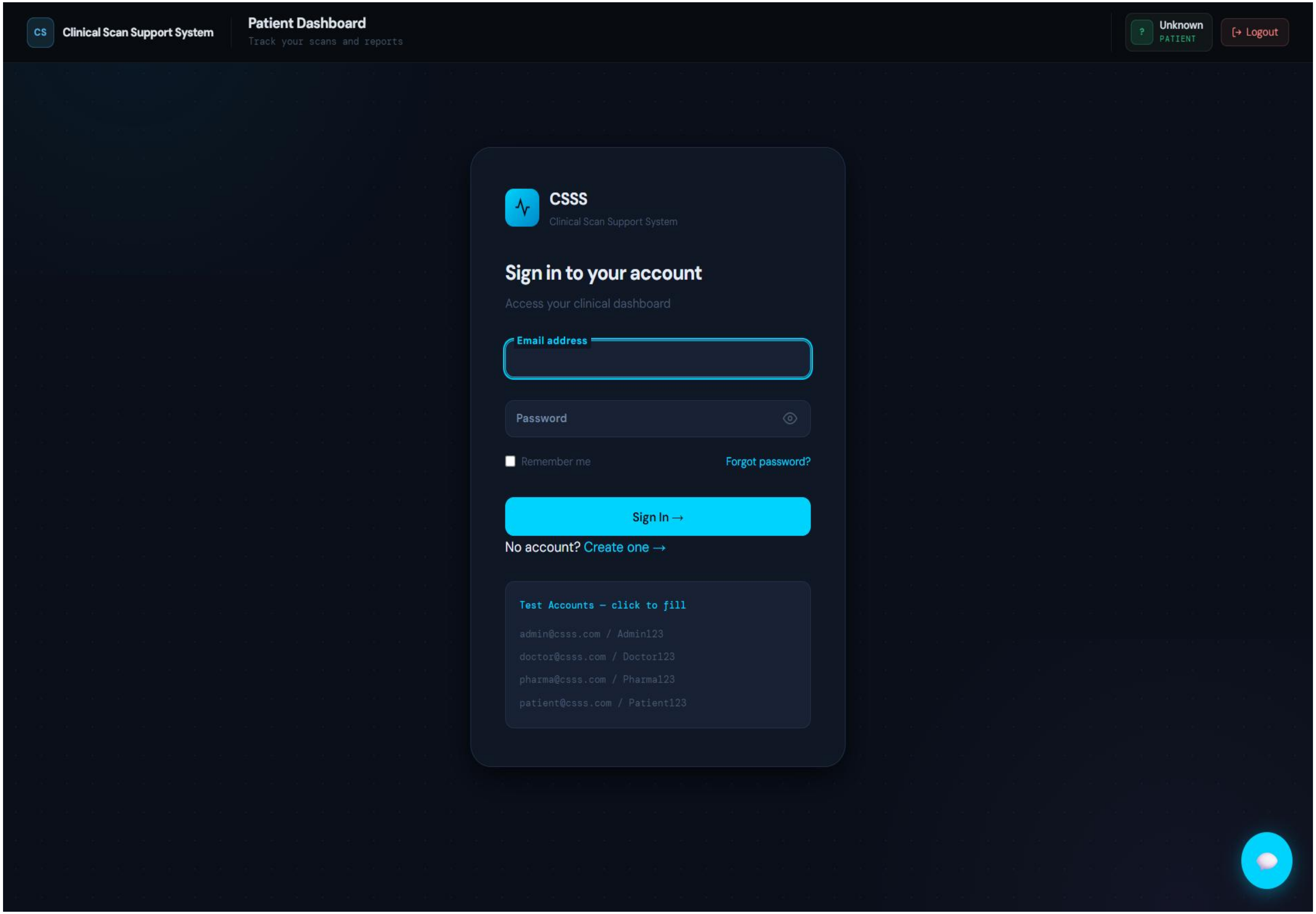
```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
# Transfer Learning — Frozen MobileNetV2 Base
base_model = MobileNetV2(weights="imagenet", include_top=False,
                         input_shape=(224, 224, 3))
base_model.trainable = False
# Custom Classification Head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation="relu")(x)
x = Dropout(0.4)(x)
outputs = Dense(6, activation="softmax")(x)
model = Model(inputs=base_model.input, outputs=outputs)
model.compile(optimizer="adam",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
# Training with Callbacks
callbacks = [
    EarlyStopping(patience=5, restore_best_weights=True),
    ModelCheckpoint("models/lung_model.h5", save_best_only=True)
]
history = model.fit(train_data, validation_data=val_data,
                    epochs=15, callbacks=callbacks)
```

Output - Landing Page

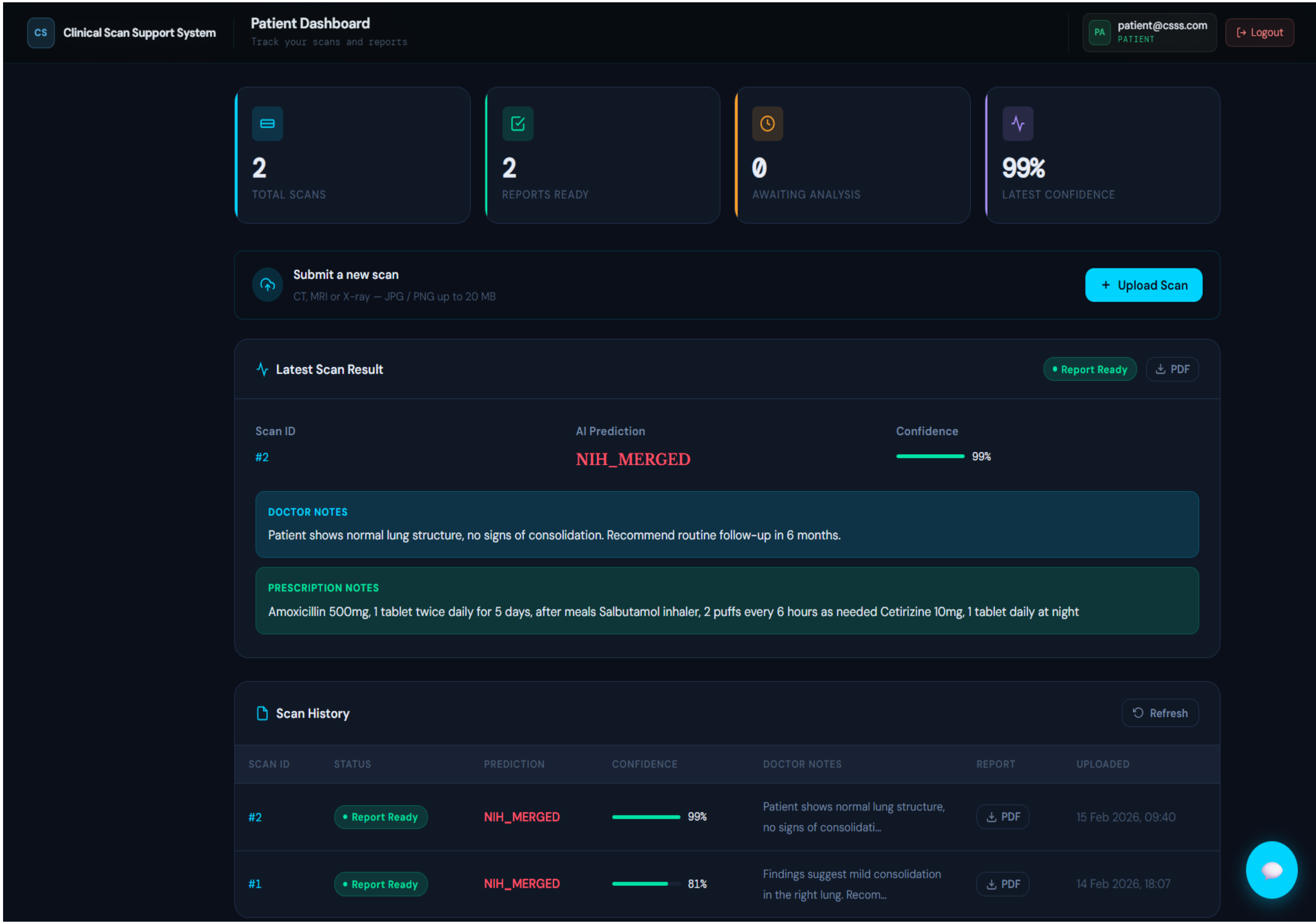
Output - Registration Page



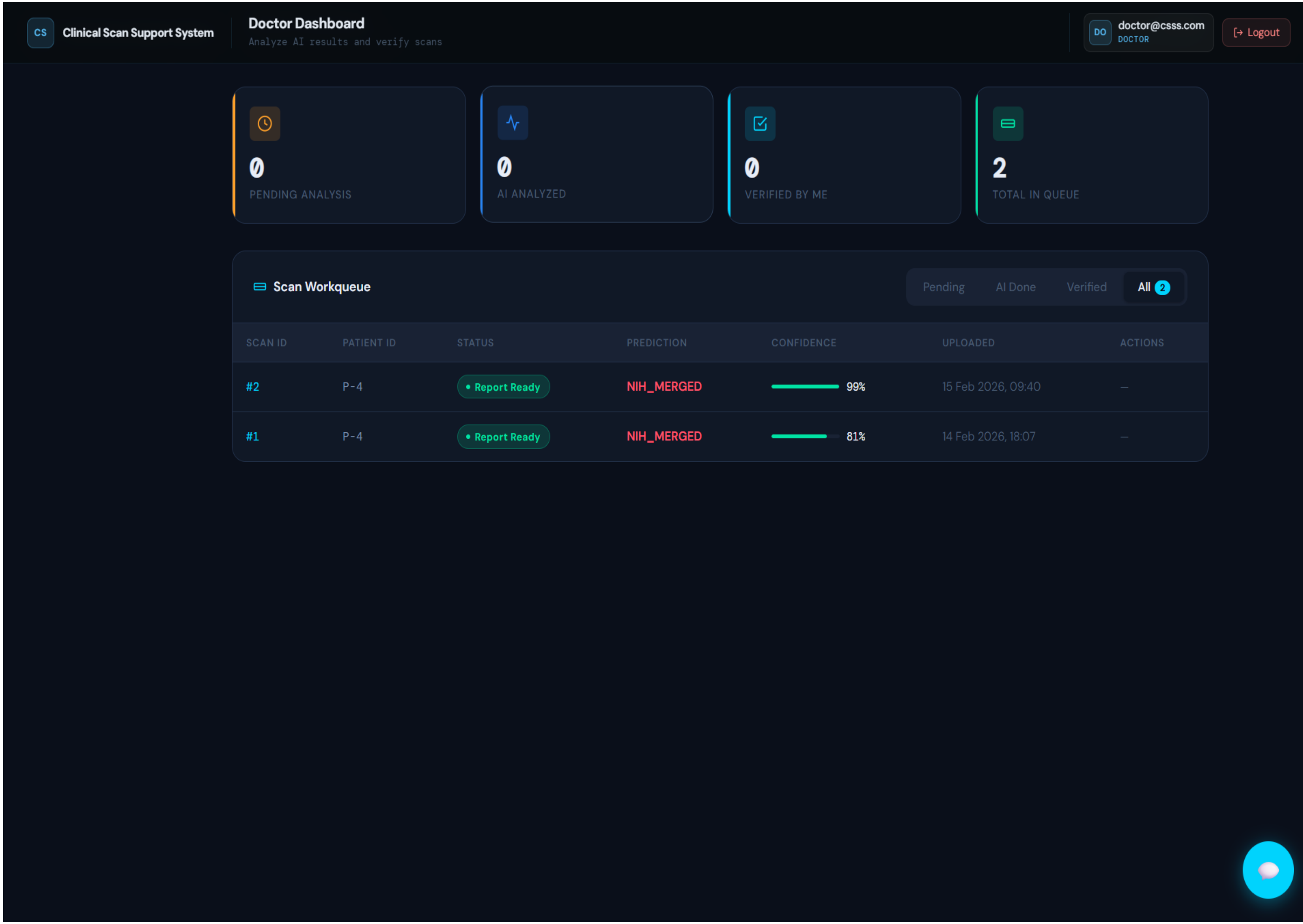
Output - Login



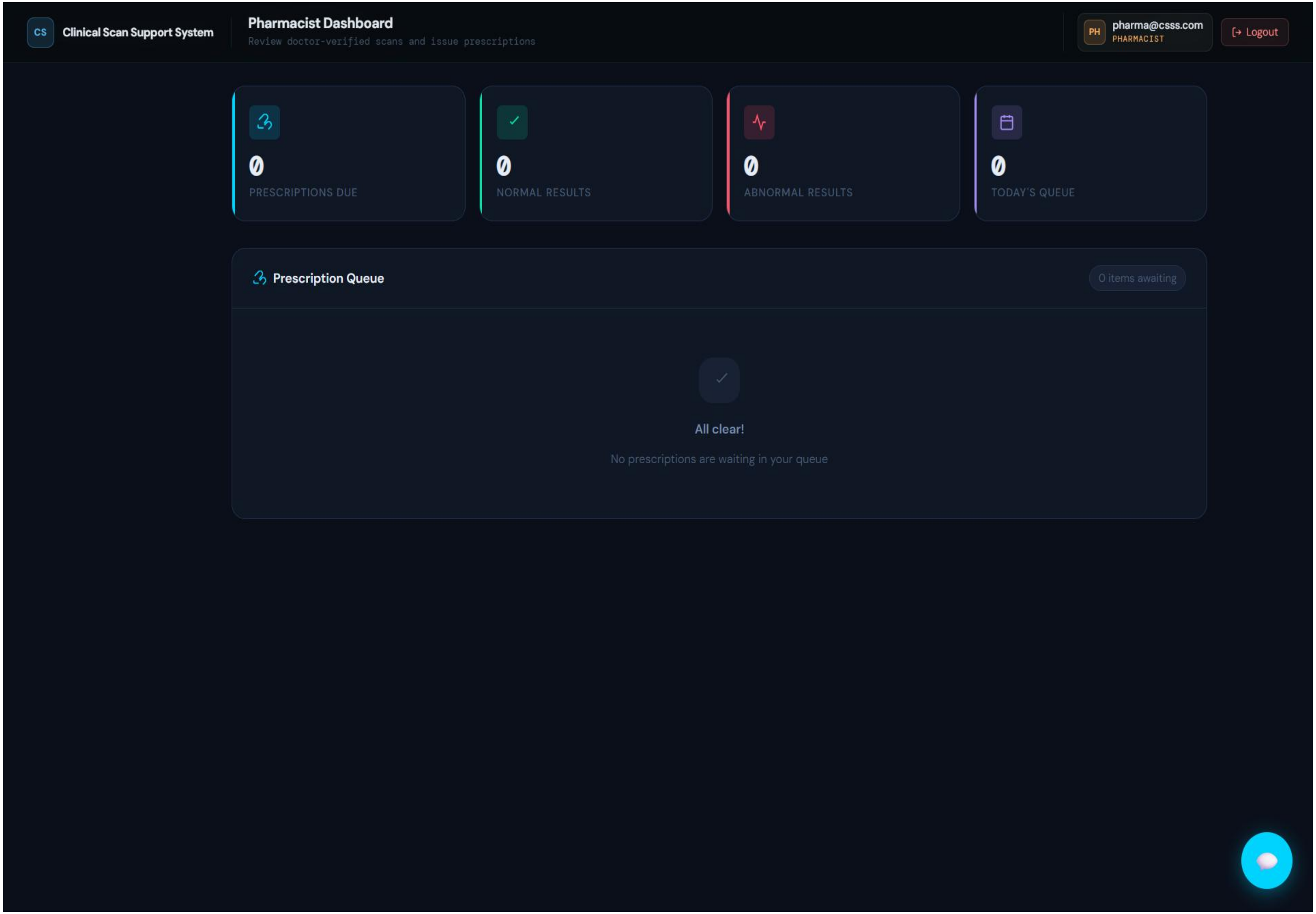
Output - Patient Dashboards



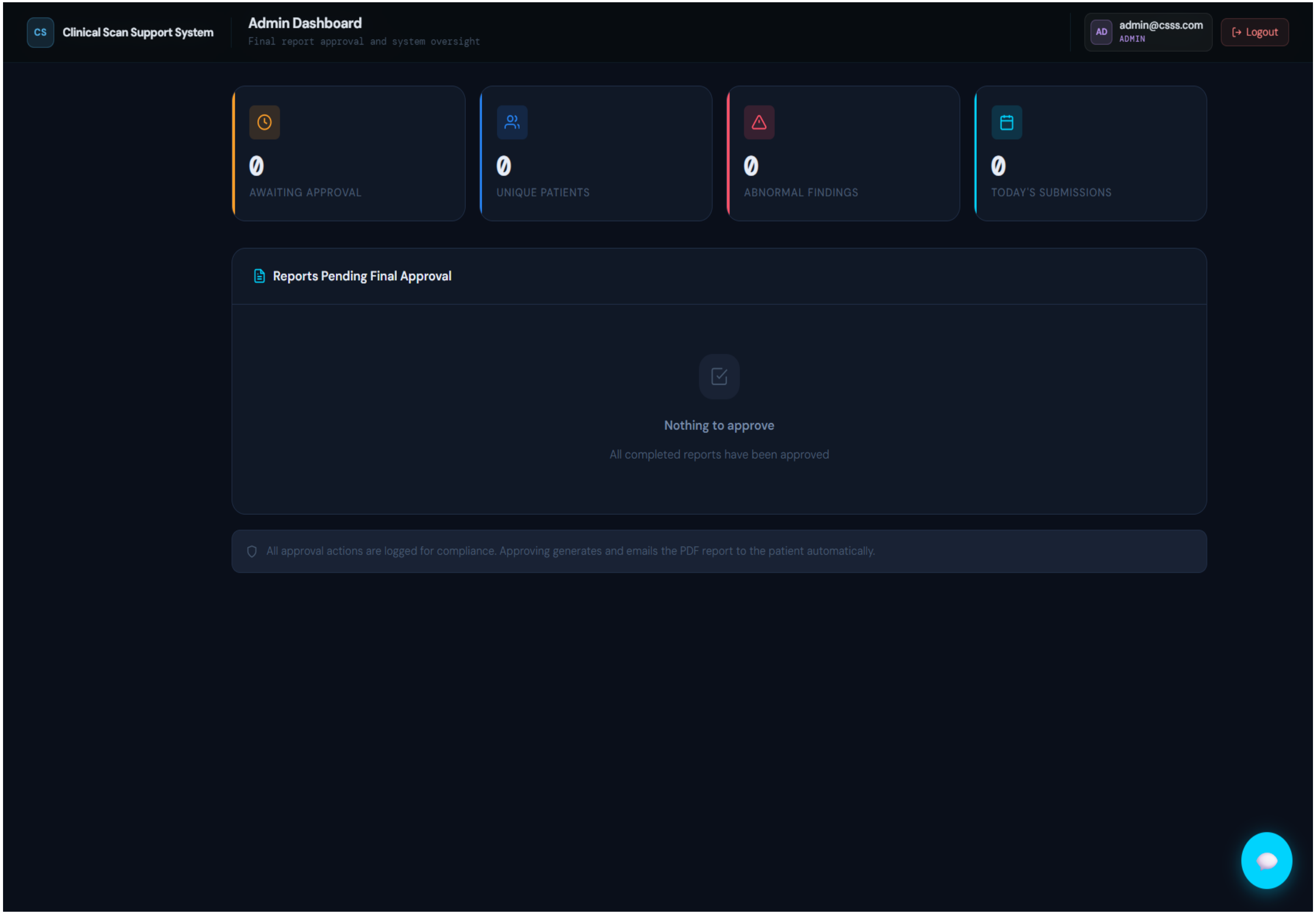
Output - Doctor Dashboards



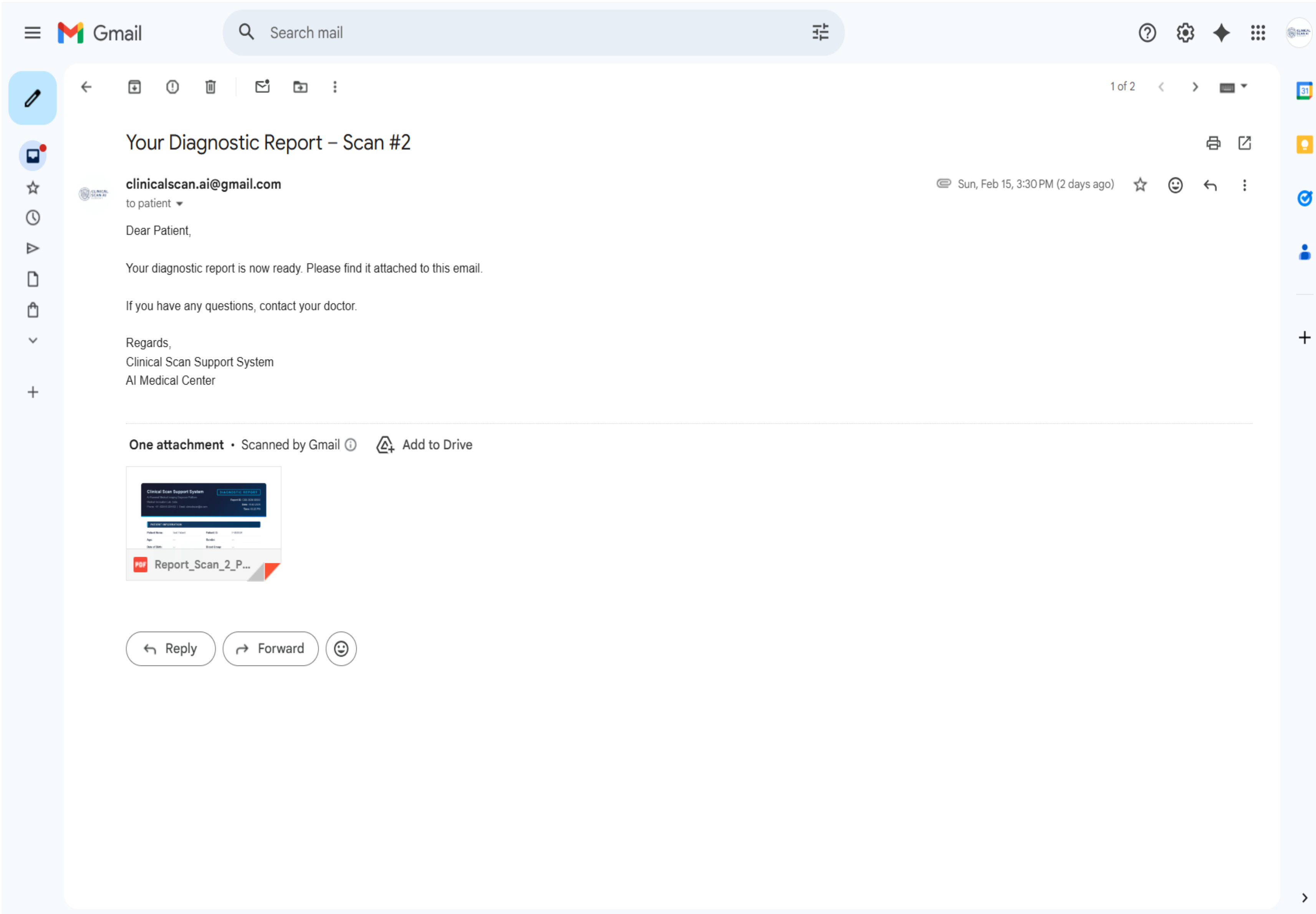
Output - Pharmacist Dashboards



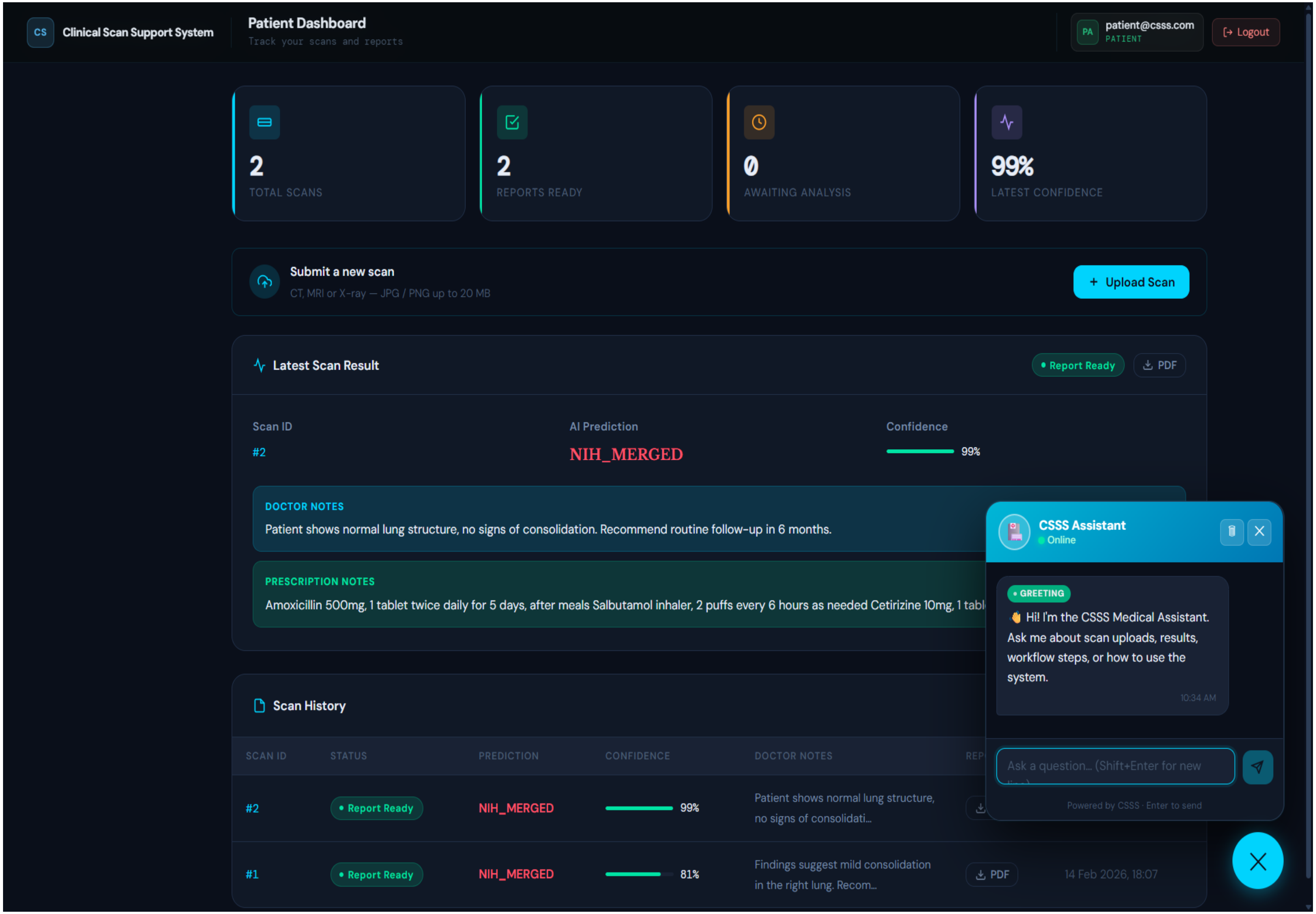
Output - Admin Dashboards



Output - Email Dashboards



Output - Chatbot



Test Cases

S.NO	Test Action	Input	Expected Output	Actual Output	Result
1	User Registration	Name, email, password, role	Account created; JWT token returned	Account created; token returned	PASS
2	Patient Login	Valid email + password	JWT token; Patient Dashboard shown	Token returned; correct dashboard displayed	PASS
3	Admin OTP Login	Credentials + 6-digit OTP	Full-access JWT after OTP verification	OTP verified; admin JWT issued	PASS
4	Scan Upload	JPG/PNG chest X-ray (<10 MB)	Scan saved; status = PENDING_AI	UUID file saved; DB record created	PASS
5	AI Analysis	Doctor clicks Analyze	MobileNetV2 prediction + confidence returned	COVID — 91.4% confidence displayed	PASS

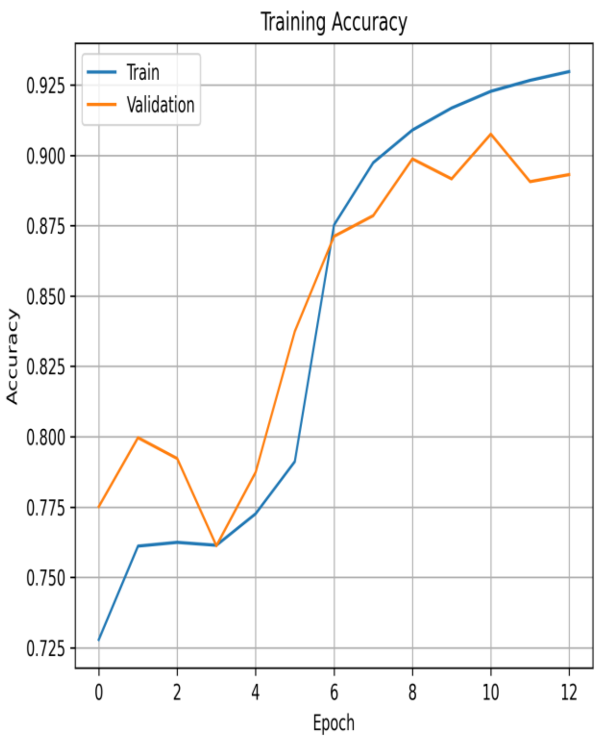
Test Cases

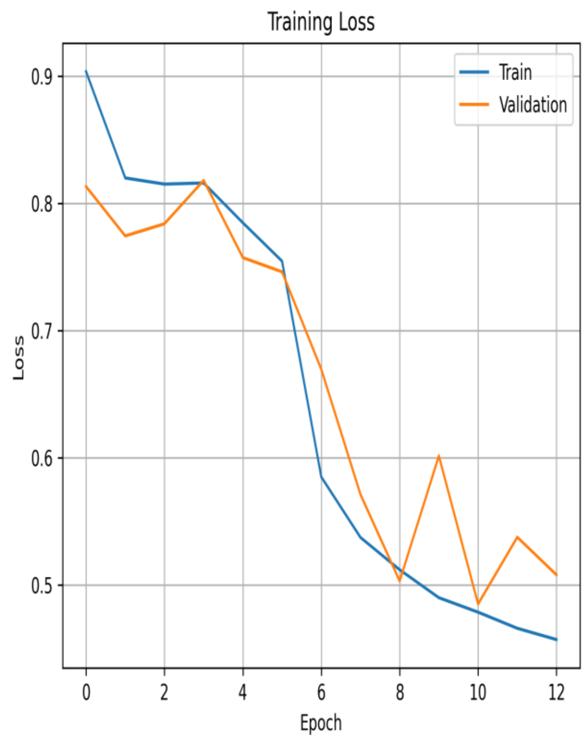
S.NO	Test Action	Input	Expected Output	Actual Output	Result
6	Doctor Verification	Clinical notes submitted	Status = DOCTOR_VERIFIED	Notes saved; status updated	PASS
7	Pharmacist Completion	Prescription submitted	Status = PHARMACIST_COMPLETED	Prescription saved; status updated	PASS
8	Admin Approval + PDF	Admin clicks Approve	PDF generated; emailed; status = REPORT_READY	PDF generated & email delivered successfully	PASS
9	PDF Download	Patient clicks Download	PDF opens/downloads	PDF downloaded successfully	PASS
10	Invalid Login	Wrong password	401 Unauthorized error	Invalid credentials error returned	PASS

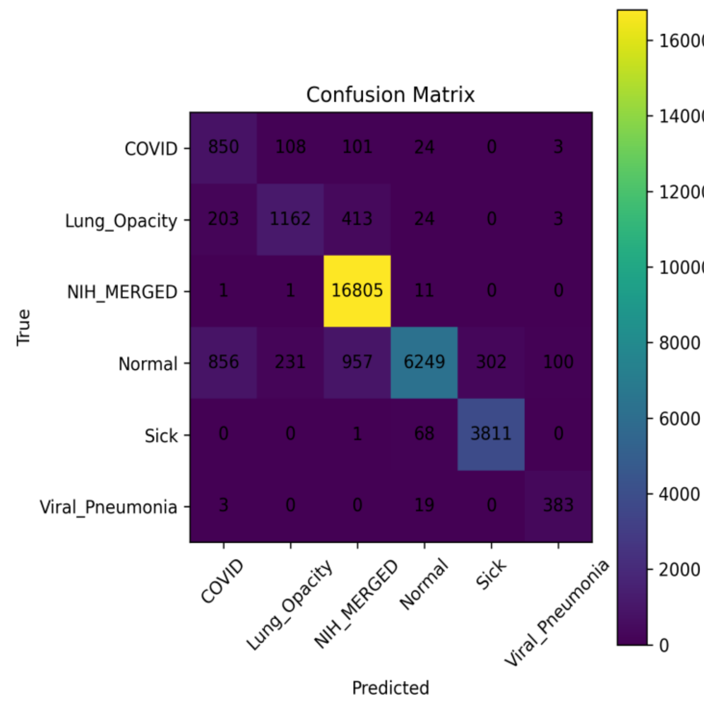
Results

AI Model Performance Metrics:

- **Test Accuracy:** 89.51%
- **Validation Accuracy:** 89.31%
- **Training Accuracy:** 92.97%
- **Total Training Images:** 217,875
- **Disease Classes:** 6 (COVID, Lung_Opacity, NIH_MERGED, Normal, Sick, Viral_Pneumonia)
- **Model Size:** ~14 MB (lung_model.h5 format)
- **Inference Speed:** < 1 second per scan image
- **Confidence Threshold:** 75% (below → flagged as "Uncertain")







Results

System Performance Benchmarks:

- All 5 scan lifecycle stages validated end-to-end (PENDING_AI → REPORT_READY)
- **PDF report auto-generation time:** < 1.5 seconds (WeasyPrint + Jinja2)
- **OTP email delivery time:** < 5 seconds (Gmail SMTP with STARTTLS)
- **JWT token validation latency:** < 50ms per API request
- **Complete patient-to-report workflow:** < 4 minutes (upload → admin approval → PDF email)

Key Inferences:

- MobileNetV2 with transfer learning achieves near state-of-the-art accuracy (89.51%) on a large-scale medical imaging dataset (217K+ images).
- Confidence thresholding at 75% effectively filters uncertain predictions, forcing mandatory physician review for low-confidence cases.
- The structured 4-role pipeline ensures no scan reaches final approval without clinical verification at every stage — maintaining patient safety.
- Automated PDF report generation and SMTP email delivery eliminates manual report distribution delays, ensuring patients receive reports immediately upon admin approval.
- JWT + bcrypt + OTP 2FA security architecture ensures role-based access control is strictly enforced across all API endpoints.

Conclusion

- The Clinical Scan Support System (CSSS) successfully demonstrates a production-grade, end-to-end AI-powered medical imaging platform that automates the complete diagnostic pipeline from scan upload to encrypted report delivery.
- The MobileNetV2 deep learning model, trained on 217,875 medical images across 6 disease classes, achieves 89.51% test accuracy with sub-second inference speed — making it clinically viable for real-time hospital deployment.
- The structured 4-role workflow (Patient → Doctor → Pharmacist → Administrator) ensures multi-stakeholder clinical review before any report is finalized — maintaining patient safety and diagnostic quality control.
- Automated PDF generation using WeasyPrint + Jinja2 and encrypted SMTP email delivery eliminates manual report distribution delays, ensuring patients receive professional diagnostic reports immediately upon administrative approval.
- The JWT + bcrypt + OTP 2FA security architecture ensures strict role-based access control — sensitive operations like admin approval require multi-factor authentication, preventing unauthorized access.
- CSSS proves that AI-assisted clinical platforms can be built cost-effectively using open-source technologies (FastAPI, Next.js, TensorFlow, WeasyPrint) while maintaining professional-grade functionality suitable for hospital radiology departments, COVID-19 screening clinics, and telemedicine platforms.
- The system is production-ready with documented migration paths from SQLite (development) to PostgreSQL (production scale), Docker Compose containerization for one-command deployment, and comprehensive API documentation via FastAPI's automatic Swagger UI.

Future Work

Version 2.0 Enhancements (Q3 2026):

- **PostgreSQL Migration:** Migrate from SQLite to PostgreSQL for production-scale multi-hospital deployment with concurrent access support.

- **Docker Compose Containerization:** One-command full-stack deployment with isolated containers for frontend, backend, database, and AI model.

- **WhatsApp Report Delivery:** Integrate Twilio API for WhatsApp PDF report delivery (already configured in .env file).

- **DICOM Format Support:** Add support for DICOM medical imaging format for full hospital scanner integration.

- **Grad-CAM PDF Overlays:** Embed Grad-CAM heatmap visualizations directly in PDF reports for explainable AI transparency.

Version 2.5 Enhancements (Q4 2026):

- **Progressive Web App (PWA):** Mobile-responsive PWA frontend for smartphone and tablet access.

- **Real-Time Push Notifications:** Implement WebSocket-based real-time scan status notifications.

- **Multi-Language PDF Reports:** Generate reports in English, Tamil, and Hindi languages.

- **EMR/EHR Integration:** Integrate with hospital Electronic Medical Record systems using HL7 FHIR standard.

- **Federated Learning:** Implement federated learning across multiple hospital nodes to improve model generalization without sharing patient data.

- **HIPAA Compliance:** Add comprehensive audit logging for regulatory certification and compliance.

References

GitHub Repository: https://github.com/Darkwebnew/Projectwork2

Dataset References:

1. Wang X. et al., "ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks," Proc. IEEE CVPR, 2017. Kaggle: https://www.kaggle.com/nih-chest-xrays/data
2. Chowdhury M.E.H. et al., "Can AI Help in Screening Viral and COVID-19 Pneumonia?" IEEE Access, vol. 8, 2020. Kaggle: https://www.kaggle.com/tawsifurrahman/covid19-radiography-database
3. CAD Cardiac MRI Dataset, Kaggle, 2022. Kaggle : https://www.kaggle.com/datasets/danialsharifrazi/cad-cardiac-mri-dataset

Academic References:

4. Howard A. et al., "MobileNets: Efficient CNNs for Mobile Vision Applications," arXiv:1704.04861, 2017.
5. Sandler M. et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," Proc. IEEE CVPR, 2018.
6. Selvaraju R. et al., "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," Proc. IEEE ICCV, 2017.
7. Rahman T. et al., "Exploring the Effect of Image Enhancement Techniques on COVID-19 Detection," Computers in Biology and Medicine, 2021.
8. Rajpurkar P. et al., "CheXNet: Radiologist-Level Pneumonia Detection on Chest X-rays," arXiv:1711.05225, 2017.
9. He K. et al., "Deep Residual Learning for Image Recognition," Proc. IEEE CVPR, 2016.

Questions

Thank You