1) Project Title:

Dynamic DICOM Endpoints

2) Abstract / Project Summary:

DICOM (Digital Imaging and Communications in Medicine) is a widely used standard for medical imaging storage and communication. Traditionally, DICOM endpoints—such as PACS servers and imaging modalities—are statically configured, which can lead to bottlenecks, inefficiencies, and potential data loss under heavy load.

This project aims to enhance DICOM image routing by introducing dynamic endpoint selection based on system load (CPU usage, instance count) and auto-scaling middleware to handle increased traffic. Additionally, it will extend the probe component to function as a PACS—enabling users to retrieve images and send them to a database. The implementation will be tested with multiple Orthanc servers in a federated setup, ensuring scalability and reliability.

By the end of the project, a fully functional prototype will demonstrate dynamic DICOM endpoint selection, adaptive scaling, and an enhanced probe with PACS functionality, providing a foundation for efficient teleradiology and federated learning use cases.

---

3) Contributor Name:

Bhumin Manodra

4) Contributor Email and GitHub ID:

- Email: manodrabhumin@gmail.com
- GitHub ID: blitzboah

5) Potential Mentor(s):

- Ananth Reddy (bananthreddy30@gmail.com)
- Pradeeban Kathiravelu (pkathiravelu@alaska.edu)

---

6) Personal Background (Brief CV)

I am a pre-final year Computer Engineering student at Shah and Anchor Kutchhi Engineering College, Mumbai. My journey into programming began with a passion for game development, which later expanded into a broader interest in software development.

I have developed multiple games that people loved, built websites, implemented a ray tracer, and even built a Git-like version control system.

Over time, I delved deeper into backend development while also exploring frontend technologies. Currently, I am focused on strengthening my understanding of computer fundamentals, distributed systems, and advanced backend development.

---

7) Project Goals

- Develop a prototype implementation that supports dynamic DICOM endpoints.
- Improve DICOM image routing middleware by incorporating CPU-based load balancing.
- Enhance the probe to function as a PACS, allowing image retrieval and forwarding to a database.
- Integrate Niffler with the probe to enhance DICOM image retrieval and metadata extraction.
- Improve the web app’s UI/UX
- Write unit and integration tests for the middleware and probe components.

- Implement auto-scaling for the middleware to handle increased traffic efficiently with help of Kubernetes.
- Document setup, usage, and contributions for future developers.

7.1) Workflow + Design

The general workflow for my proposed implementation is as follows:

1. **Ingestion**: The process starts with a sender transmitting a DICOM image, which is received by the middleware.

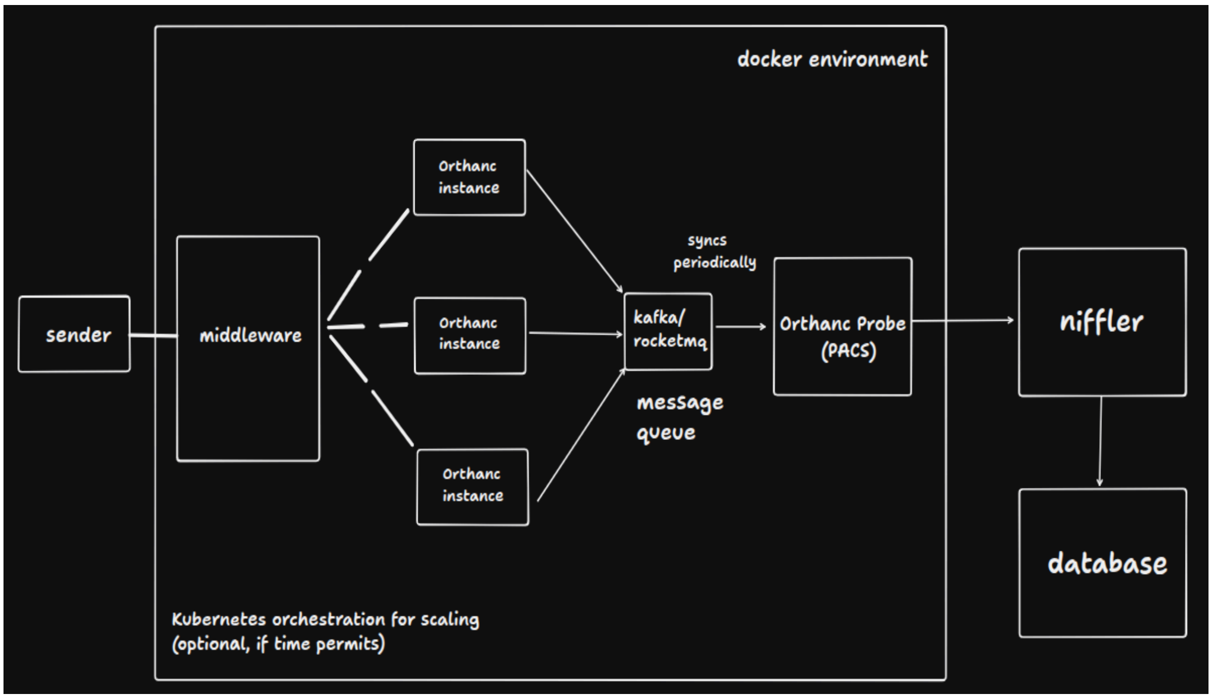
2. **Middleware**: The middleware acts as an intermediary, evaluating the incoming image and determining where it should be stored. It considers factors like CPU load, queue length, and overall availability of storage instances before forwarding the image.

3. **Orthanc Instances**: Based on the decision made by the middleware, the image is forwarded to one of the available Orthanc instances for storage. These instances act as temporary storage units, ensuring that images are stored efficiently without overwhelming a single system.

4. **Message Queue (MQ)**: Instead of directly sending images to a PACS, Orthanc instances push metadata or image references to a message queue. This will prevent overload and will ensure reliable communication between Orthanc instances and the Orthanc Probe.

5. **Orthanc Probe (PACS Functionality)**: The Orthanc Probe acts as a PACS and periodically pulls images from the Orthanc instances using the message queue as a reference. This allows for dynamic and efficient retrieval of images.

6. **Metadata Extraction and Local Storage**: Once the images are retrieved, Niffler can be used for extracting metadata and processes it for further use. The extracted metadata can be then stored in a local database, enabling quick queries and long-term data persistence.



This workflow ensures a dynamic, load-balanced routing of DICOM images with robust storage, retrieval, and metadata management capabilities.

---

8) Project Schedule

8.1) Community Bonding Period (May 8 – June 1, 2025)

- Engage with mentors to finalize the technical approach.
- Refine middleware design for improved load balancing.
- Plan probe enhancements to support PACS functionalities and integration with Niffler.
- Explore message queue options (Kafka/RocketMQ) for improved reliability.
- Outline improvements for the web app.

8.2) Development Phase (June 2 – August 25, 2025)

Week 1 (June 2 – June 8): Middleware Enhancements

- Improve middleware logic for smarter image routing.
- Optimize load balancing based on instance availability.
- Implement basic message queue integration for reliable image ingestion.

- Write unit tests for middleware logic.

Week 2 (June 9 – June 15): Middleware Finalization & Message Queue Testing

- Ensure middleware can push image metadata to the message queue.
- Improve error handling and logging.
- Conduct performance testing under different loads.
- Fix middleware-related issues and ensure stability.

Week 3 (June 16 – June 22): Enhancing Probe for PACS & Niffler Integration

- Modify the probe to periodically retrieve images from the message queue.
- Integrate Niffler with the probe for efficient metadata extraction and filtering.
- Modify probe APIs to leverage Niffler's query capabilities.
- Ensure seamless communication between Niffler and Orthanc for optimized retrieval.

Week 4 (June 23 – June 29): Probe Testing & Message Queue Optimization

- Conduct integration testing for the probe, message queue, and Niffler.
- Write test cases to verify probe functionality.
- Optimize message queue processing to prevent bottlenecks.

Week 5 (June 30 – July 6): Web App Enhancements

- Improve the web app interface for image upload & retrieval.
- Add better visualization and user controls.

Week 6 (July 7 – July 13): Web App Testing & Performance Optimization

- Conduct usability testing and refine UI/UX.
- Optimize API responses for faster image loading.
- Fix bugs and improve error handling.

Week 7 (July 14 – July 20): Integrate Niffler

- Integrate Niffler's for efficient retrieval.
- Ensure message queue, probe, and Niffler integration work under various conditions.
- Submit progress report & midterm evaluation.

Week 8 (July 21 – July 27): Testing and Optimization

- Fine-tune message queue and probe parameters for performance and reliability.
- Refactor any inefficient code.

Week 9 (July 28 – August 3): Kubernetes Auto-Scaling Implementation (Optional, if time permits)

- Implement auto-scaling logic for middleware based on CPU load.
- Deploy auto-scaling using Kubernetes.
- Test scalability under increasing loads.

8.3) Project Completion, Testing, and Documentation (August 4 – August 11, 2025)

- Fix any last-minute bugs.
- Finalize all documentation and prepare the final report.
- Ensure seamless integration between middleware, probe, message queue, Niffler, and web app.
- Submit the final project.

---

9) Planned GSoC Work Hours

- Project size: Full-time (350 hours)
- Availability: 35 hours per week
- Time zone: IST (Indian Standard Time, UTC+5:30)
- Preferred meeting times: Flexible; weekly meeting via Zoom / Skype / Google Meet / Discord.

---

10) Planned Absence/Vacation Days & Other Commitments

I have end semester exams scheduled during the community bonding period, likely in the last weeks of May and the first week of June. To avoid falling behind, I plan to start coding during the community bonding period itself. Aside from this, I have no other commitments during the GSoC period and will be able to dedicate my full attention to the project.

If any unexpected commitments arise, I will inform my mentors in advance and adjust my schedule accordingly.

---

11) Skill Set

Relevant Skills:

- Languages: Python, Java, JavaScript
- Backend: Flask, Spring Boot, REST APIs
- Frontend: React Js
- Containerization & Scaling: Docker, Kubernetes
- DICOM & Medical Imaging: Orthanc
- Database: PostgreSQL, MySQL
- Testing: PyTest, JUnit, Postman

Previous Work:

- **Dynamic DICOM Endpoints Sample Project:** Developed a robust middleware designed to route incoming DICOM images to the least loaded Orthanc instance, ensuring efficient load distribution and preventing bottlenecks. To enhance data retrieval and synchronization, implemented a probe that periodically queries all Orthanc instances, retrieves newly stored images, and processes them for further use. Additionally, integrated both a Command-Line Interface (CLI) and a web-based interface to provide seamless user interaction, allowing users to monitor Orthanc server loads, track image ingestion, and manage retrieval operations with ease

Link - https://github.com/blitzboah/dynamic-dicom-endpoints

- Discussion on Dynamic DICOM Endpoints - Discussed ideas and technical aspects with mentors regarding project implementation.

  Link - https://github.com/KathiraveluLab/Diomede/discussions/8