## **Requirements Analysis: Risk Prediction Dashboard**

### **1. Purpose**

Provide clinicians and data scientists with a simple, in-house web dashboard to look up a patient by ID, review clinical details, visualize the model’s top‑5 radiomics features, and obtain a real-time risk score.

### **2. Functional Requirements**

*Before diving into each feature, users flow from 1) entering a Patient ID → 2) viewing clinical details → 3) inspecting radiomics features → 4) interpreting the risk score.*

1. **Patient Lookup**
   * **What:** A text field where users type a Patient ID.
   * **Why:** Immediate access to all data linked to that patient without extra steps.
   * **Tools:** React (v18+) + Axios (or native Fetch API) for effortless API calls.
   * **Difficulty:** Easy
2. **Patient Lookup**
   * **What:** A text field where users type a Patient ID.
   * **Why:** Immediate access to all data linked to that patient without extra steps.
   * **Tools:** React + Axios for effortless API calls.
   * **Difficulty:** Easy
3. **Display Clinical Data**
   * **What:** A neatly formatted table showing key demographics and clinical details (e.g., age, diagnosis, treatment regimen).
   * **Why:** Presents essential context at a glance, reducing cognitive load on the user.
   * **Tools:** React Table component + FastAPI reading CSV via Pandas.
   * **Handling Edge Cases:** Validate CSV columns on load; display “Data unavailable” in table cells for missing or malformed values.
   * **Difficulty:** Easy
4. **Top-5 Radiomics Features**
   * **What:** A horizontal bar chart highlighting the five most predictive radiomics features and their values.
   * **Why:** Helps users understand which imaging biomarkers drive the model’s decision.
   * **Tools:** Chart.js for quick, interactive visuals; endpoint serves feature metadata from JSON/CSV.
   * **Enhancements:** Include axis labels and tooltips on hover for clarity; handle absent feature data gracefully.
   * **Difficulty:** Easy
5. **Risk Score Prediction**
   * **What:** A clear, prominent display of the probability (e.g., 0–100%) that the patient is high-risk.
   * **Why:** Delivers actionable intelligence instantly, aiding clinical decision-making.
   * **Tools:** FastAPI with a loaded scikit-learn model (pickle) to compute risk on demand.
   * **Difficulty:** Easy
6. **Page Navigation**
   * **What:** Smooth, intuitive routing between the lookup page and the dashboard view, with browser history support.
   * **Why:** Ensures a cohesive user journey without full page reloads.
   * **Tools:** React Router DOM.
   * **Difficulty:** Easy

### **3. Non-Functional Requirements**

* **Error Handling**
  + Provide a clear fallback UI for failures (e.g., display “Unable to retrieve data” when an error occurs) and use standardized HTTP status codes (200, 400, 500) so the frontend can handle each scenario gracefully.
* **Performance**
  + Target overall API response under 500 ms for lookup and prediction.
  + Implement Python’s functools.lru\_cache or lightweight in-memory caching for repeat requests.
  + **Difficulty:** Easy
* **Security**
  + Serve the dashboard over HTTPS (configure via NGINX reverse proxy).
  + Optionally integrate basic HTTP auth or connect to institutional LDAP for credential checks.
  + Sanitize all inputs to prevent injection attacks.
  + **Difficulty:** Moderate
* **Maintainability**
  + Host code on GitHub with clear README and contribution guidelines.
  + Use Docker Compose to standardize local development environments.
  + Employ semantic versioning for API changes.
  + **Difficulty:** Easy

### **4. Technical Stack Overview**

* **Data Layer**
  + **Storage:** CSV files on an on-premise network share, updated via a scheduled ETL script.
  + **Processing:** Python & Pandas handle cleaning, joining, and validation in a cron-driven job.
  + **Difficulty:** Easy
* **API Layer**
  + **Framework:** FastAPI with Uvicorn, serving endpoints to fetch patient data and compute risk.
  + **Model Loading:** Deserialize scikit-learn artifact at startup for immediate inference.
  + **Difficulty:** Easy
* **Frontend Layer**
  + **Framework:** React, styled with Tailwind CSS (or plain CSS) for rapid UI building.
  + **Components:** React Table for clinical data, Chart.js for radiomics charts, React Router for navigation.
  + **Hosting:** Static build served by NGINX on a Linux VM.
  + **Difficulty:** Easy
* **DevOps & CI/CD**
  + **Local Development:** Docker Compose defines services for API and front end.
  + **CI/CD:** GitHub Actions automate linting, testing, Docker image builds, and deployment.
  + **Difficulty:** Easy

### **5. Environment & Infrastructure**

* **Network Share:** Host CSVs and model files securely behind institutional firewall.
* **Server:** Single Linux VM (+ Docker) running both API and static web server.
* **Version Control:** GitHub repository with protected branches.
* **Logging:** Simple text logs; optionally forward to a lightweight ELK stack or local file monitor.

### **6. Assumptions & Constraints**

* No external cloud dependencies—all components run on-premise.
* CSV schemas remain stable; any changes require ETL update.
* Define a simple CSV versioning scheme or maintain a JSON schema file to streamline future ETL adjustments if formats change.
* User access restricted to authenticated institutional personnel.
* Models and feature metadata updated periodically by development team.