Backend Requirements for the System

1) **Database**
    a) Use PostgreSQL 13 or later as the database for storing user information, ensuring secure storage of encrypted passwords.
    b) Utilize an ORM (Object-Relational Mapping) library such as Prisma to interact with the PostgreSQL database, facilitating easier data manipulation and management.

2) **API and Authentication**
    a) Develop RESTful APIs to facilitate communication between the frontend and the authentication backend.
    b) Implement JWT (JSON Web Tokens) for secure session management, generating tokens that expire after a defined time for enhanced security.
    c) Store JWTs in cookies to maintain session integrity.

3) **File Upload Management**
    a) Use Multer for handling file uploads efficiently and securely.

4) **Email Integration**
    a) Utilize Nodemailer integrated with Gmail for sending verification and notification emails to users.

5) **Security**
    a) Store passwords securely using bcrypt hashing with a minimum of 12 rounds.
    b) Ensure all API endpoints are secured and validate user permissions appropriately.

6) **Error Handling**
    a) Implement comprehensive error handling for API requests to provide meaningful feedback and maintain system integrity.

7) **Rate Limiting**
    a) Consider implementing rate limiting on sensitive routes (like login) to mitigate brute force attacks.

8) **Input Validation and Sanitization**
    a) Ensure that user inputs are validated and sanitized to prevent SQL injection and other attacks.

9) **Environment Configuration**
    a) Use environment variables to manage sensitive configurations (like database credentials and JWT secrets) securely.

10) **Architecture**
    a) Follow the MVC (Model-View-Controller) architecture for organizing the codebase, ensuring clear separation of concerns and maintainability.

11) **AI Model Integration**
    a) Integrate an AI model using ONNX (Open Neural Network Exchange) for processing and inference within the application, ensuring compatibility and performance.

Frontend Technology Requirements

1) **HTML**
    a) Structure the layout of the application.
    b) Define semantic elements for accessibility and SEO.

2) **CSS**
    a) Style the application and add basic layout properties.
    b) Manage custom styles not covered by Tailwind CSS.

3) **JavaScript**
    a) Add interactivity and dynamic functionality to the application.
    b) Handle form validation, animations, and other logic.

4) **Tailwind CSS**
    a) Use utility-first classes to style the components.
    b) Customize Tailwind's configuration for colors, fonts, and breakpoints.

5) **React**
   a) Component-based structure for reusable and maintainable UI components.
   b) Manage application state using React's hooks (useState, useEffect, etc.).
   c) Use React Router for routing and navigation (if a multi-page app).
   d) Fetch and display data using React (possibly with fetch or axios).

6) **Additional Libraries (optional)**
   a) React Icons for icons and SVGs.
   b) React Router for client-side routing.
   c) Axios for API requests, if needed.

AI System Requirements

1. **Model Training and Selection**
   a. **Algorithms:** Decision Trees, Random Forests, Support Vector Machines (SVM), or neural networks if needed.
   b. **Requirement:** Models should achieve at least 80% accuracy to be considered reliable for production.
   c. **Technologies**:
      - **Python**: Main programming language for AI model development.
      - **Scikit-learn**: For training models such as Decision Trees, Random Forests, and SVMs.
      - **TensorFlow** or **PyTorch**: For neural network models if complex architectures are needed.

2. **Model Serialization and Compatibility**
   a. **Requirement:** Models must be stored in a standardized format, ensuring smooth loading and inference within the backend environment.
   b. **Technologies**:
      - **Flask** or **FastAPI**: For deploying the inference engine as an API that serves model predictions.
      - **ONNX Runtime**: To load and run ONNX models efficiently in production.

3. **Risk Level Classification Mechanism**
    a. **Requirement:** Thresholds for risk levels must be tunable to allow adjustments based on clinical recommendations or feedback.
    b. **Technologies**:
        - **Python**: To implement classification logic based on model output thresholds.
        - **Scikit-learn** or **Numpy**: For implementing custom thresholding and risk classification mechanisms.
4. **Data Preprocessing Pipeline**
    a. **Requirement:** Pipeline should include feature scaling, encoding of categorical variables, and handling of missing values to match the training data schema.
    b. **Technologies**:
        - **Pandas** and **NumPy**: For data manipulation, cleaning, and feature engineering.
        - **Scikit-learn**: For feature scaling, encoding categorical data, and imputing missing values.

5. **Recommendations Engine**
    a. **Requirement:** Health tips should be stored in a structured database or JSON format, with recommendations aligned to each disease type and risk level.

6. **Result Formatting and API Integration**
    a. **Requirement:** Results must include prediction probabilities, risk classifications, and personalized health tips in a structured JSON response.
    b. **Technologies**:
        - **JSON**: Standard format for structuring and transmitting prediction results, risk levels, and recommendations to the frontend.
        - **Flask** or **FastAPI**: For handling API responses and ensuring structured JSON output for frontend integration.

7. **Model Monitoring and Versioning**

a. **Requirement:** Each model version should be trackable, with a system in place to assess prediction accuracy over time and trigger retraining when necessary.
b. **Technologies**:
   - **MLflow** or **DVC (Data Version Control)**: For versioning models, tracking model metrics, and managing model updates.

8. **Data Security and Privacy Compliance**
   a. **Requirement:** Anonymize or encrypt personally identifiable information (PII) and log usage for monitoring data access within the AI pipeline.
   b. **Technologies**:
      - **bcrypt**: For secure hashing of sensitive information.
      - **SSL/TLS**: To ensure secure data transmission over the network.

10. **Automated Retraining Pipeline (Optional)**
    a. **Requirement:** The pipeline should be able to incorporate new data, retrain models, and redeploy updated versions with minimal manual intervention.