Viva Q&A - Blood App

Blood Report & Fitness Evaluation System

Comprehensive Viva Voce Questions & Answers

Section 1: Project Overview & Methodology (Q1-25)

Q1. What is the core objective of your project?
Answer: The main objective is to automate the interpretation of blood test reports using AI, providing users with instant, understandable health insights, disease predictions, and personalized fitness/diet recommendations without immediate doctor intervention.

Q2. What problem does this application solve?
Answer: It bridges the gap between complex medical terminologies in blood reports and common complications. It empowers patients to understand their health status immediately and securely stores their history.

Q3. What is the methodology used in this project?
Answer: We use a hybrid methodology combining Optical Character Recognition (OCR) for data extraction, Machine Learning (ML) for disease prediction, and a Rule-Based System for dietary/fitness recommendations. The workflow is: Image Upload -> Pre-processing -> Text Extraction -> Data Parsing -> ML Classification -> Recommendation Engine -> UI Display.

Q4. Which software development validation model did you use?
Answer: We followed an Agile-like iterative approach, developing the core OCR first, then the ML models, and finally the UI and integration, allowing for continuous testing and refinement of features.

Q5. What are the primary technologies used in the Frontend?
Answer: React.js (library), Vite (build tool), Tailwind CSS/Custom CSS (styling), and Lucide-React (icons).

Q6. What are the primary technologies used in the Backend?
Answer: Python with Flask (web framework).

Q7. What sophisticated AI/ML technologies are involved?
Answer: Tesseract OCR for text extraction, LayoutLMv3 (experimental/planned) for document understanding, and Scikit-learn models (Random Forest, Naive Bayes) for disease prediction.

Q8. How does the system handle database management?
Answer: We use SQLite for the backend database (lightweight, file-based) and LocalStorage for client-side persistence of user sessions and temporary data.

Q9. Is this a web application or a mobile app?
Answer: It is a Progressive Web App (PWA) built with React, which means it runs in a browser but can be installed on mobile devices to look and feel like a native app.

Viva Q&A - Blood App

Q10. What is the 'unique selling point' (USP) of your app?
Answer: The ability to convert a static image of a medical report into actionable fitness and diet plans instantly, combined with secure, offline-capable storage.

Q11. Explain the architecture briefly.
Answer: It follows a Client-Server architecture. The Client (React) handles UI and sends images to the Server (Flask). The Server processes the image, runs ML algorithms, and returns JSON data.

Q12. What are the key modules of the project?
Answer: Authentication, Profile Management, Report Upload & OCR, Disease Prediction Engine, Fitness/Diet Recommender, and Dashboard/Analytics.

Q13. How did you gather requirements for this project?
Answer: We analyzed existing health apps, identified gaps in medical report readability for laypeople, and consulted standard medical reference ranges for accurate logic.

Q14. What were the major challenges faced?
Answer: Accurately extracting text from blurry or low-quality images (OCR accuracy) and mapping various hospital report formats to a standard structure.

Q15. How do you ensure data privacy?
Answer: Data is stored locally or in a secure backend session. We do not sell data. User profiles are isolated in the database.

Q16. What is the 'Confidence Score' in your context?
Answer: It refers to the probability output of our ML models (e.g., "85% confident you are at risk of Anemia") or the OCR's certainty in reading a specific number.

Q17. Did you use any third-party APIs?
Answer: We primarily built our own API using Flask. We use libraries like `tesseract` locally rather than calling external cloud APIs to keep it self-contained and free.

Q18. How does the app handle offline usage?
Answer: As a PWA, it caches static assets (HTML/CSS/JS). However, the OCR/ML processing requires server connectivity unless we migrate models to on-device (TensorFlow.js), which is a future scope.

Q19. What is the target audience?
Answer: General public, fitness enthusiasts, and elderly patients who want quick feedback on their regular blood tests.

Q20. How is the project deployed?
Answer: The frontend can be deployed on Vercel/GitHub Pages, and the backend on Render/Heroku/PythonAnywhere.

Q21. What is a PWA?
Answer: Progressive Web App. It allows a website to be installed on a phone, work offline, and send push notifications.

Q22. Why choose React over Angular or Vue?
Answer: React offers a component-based structure, a vast ecosystem, and Virtual DOM for high performance, which is crucial for our interactive dashboard.

Q23. Why choose Flask over Django?
Answer: Flask is micro, lightweight, and flexible, making it perfect for wrapping our specific ML/OCR scripts into API endpoints without the overhead of Django's monolithic structure.

Q24. What is the role of `vite.config.js`?
Answer: It configures the build process, enabling features like hot module replacement (HMR) during development and optimizing the final bundle for production.

Q25. How do you handle cross-origin requests (CORS)?
Answer: We use the `flask-cors` library in the backend to allow our React frontend (running on a different port) to communicate with the Flask API.

Section 2: Frontend Development (React & UI) (Q26-60)

Q26. What is the entry point of your React application?
Answer: `main.jsx` (or `index.js`), which mounts the `App` component into the DOM.

Q27. Explain the folder structure of `src`.
Answer: `components/` for reusable UI parts, `utils/` for helper functions, `assets/` for images, and `App.jsx` as the main container.

Q28. What are 'Hooks' in React? Which ones did you use?
Answer: Functions that let us use state and lifecycle features. We used `useState` (for managing inputs/results), `useEffect` (for data fetching), and `useRef` (for DOM access).

Q29. How does the routing work in your app?
Answer: We likely use conditional rendering or a router (like `react-router-dom`) to switch between Login, Dashboard, and Analysis views.

Q30. Ideally, how is the 'Upload' feature implemented on the frontend?
Answer: An `<input type="file" />` element captures the image. We then use `FormData` to package it and `fetch` or `axios` to send it to the backend.

Q31. How do you display the charts in the dashboard?
Answer: We can use libraries like `Chart.js` or `Recharts`, or simply use CSS/SVG for custom progress bars to visualize health metrics.

Q32. What is the purpose of `App.css` or `index.css`?
Answer: They contain the global styles, CSS variables (for colors like medical red/blue), and reset rules to ensure consistency across browsers.

Q33. How is the user state managed (logged in vs. logged out)?
Answer: We store a token or user object in `localStorage` or `sessionStorage` upon successful login and clear it on logout.

Q34. What is 'Prop Drilling'? Did you face it?
Answer: Passing data through many layers of components. We avoided it by using Context API or keeping state management efficient.

Q35. How do you make the app responsive?
Answer: Using CSS Media Queries (`@media`) and Flexbox/Grid layouts to adapt to mobile, tablet, and desktop screens.

Q36. What is the Virtual DOM?
Answer: A lightweight copy of the real DOM. React changes this first, compares it to the previous version (diffing), and only updates the actual DOM where necessary (reconciliation).

Q37. Why use `Lucide-React` icons?

Answer: They are lightweight, highly customizable (size, color, stroke), and tree-shakeable SVG icons.

Q38. How do you handle loading states during image processing?
Answer: We use a `useState` boolean (e.g., `isProcessing`). When true, we show a spinner/loader component and disable buttons.

Q39. What is the 'Profile Setup' component for?
Answer: It collects static user data (Age, Gender, Weight, Height) which is essential for calculating BMI and adjusting health thresholds.

Q40. How is BMI calculated in the frontend?
Answer: Formula: `Weight (kg) / (Height (m))^2`. We implement this in a simple utility function.

Q41. Explain the validation you put on the Login form.
Answer: We check if email contains '@' and if the password meets minimum length requirements before allowing the form submission.

Q42. What is `npm` or `yarn`?
Answer: Node Package Manager. It is used to install dependencies listed in `package.json` like React, Flask (via pip), etc.

Q43. What is the `public` folder used for?
Answer: Static assets that don't need compilation, like `robots.txt`, `favicon.ico`, or the `manifest.json` for PWA.

Q44. How does the 'Toast' notification system work?
Answer: It's a temporary popup component triggered by state changes (e.g., "Upload Successful" or "Network Error") that auto-hides after a few seconds.

Q45. What is JSX?
Answer: JavaScript XML. It allows us to write HTML-like syntax directly inside JavaScript code.

Q46. How do you handle errors from the backend in React?
Answer: We use `try-catch` blocks in our async fetch functions. If the API returns a 500 or 400 error, we catch it and display a user-friendly message.

Q47. Can this app capture images using the camera?
Answer: Yes, on mobile devices, the `<input type="file" accept="image/*" capture="environment" />` attribute allows direct camera access.

Q48. How do you ensure the UI looks 'Medical' and professional?
Answer: By using a color palette dominated by white, clean reds, and blues, and using clear, readable typography (sans-serif fonts).

Q49. What is a 'Component' in your project? Give an example.
Answer: A reusable building block. Example: `HealthCard.jsx` which displays a single metric like "Hemoglobin: 13.5 g/dL" with a status icon.

Q50. How do you optimize the React app performance?
Answer: By lazy loading components, optimizing images, and preventing unnecessary re-renders using `useEffect` dependencies correctly.

Q51. What is the difference between `state` and `props`?
Answer: `State` is internal data managed by the component itself (changeable). `Props` are data passed down from a parent component (read-only).

Q52. How do you manage the file upload size limit?
Answer: We can check `file.size` in JavaScript before sending it to the server to prevent uploading overly large images.

Q53. What is the `dist` folder?
Answer: It is the "Distribution" folder containing the optimized, minified production build of the frontend generated by `npm run build`.

Q54. Why use `export default`?
Answer: To allow a component to be imported easily in other files without using curly braces.

Q55. How do you implement the 'logout' functionality?
Answer: By removing the user token/data from LocalStorage and redirecting the user to the Login page.

Q56. What are React 'Fragments' (`<> </>`)?
Answer: They let us group a list of children without adding extra nodes (like `div`) to the DOM.

Q57. How do you format dates in the frontend?
Answer: We might use the built-in `Date` object or specific formatting functions to show dates like "12 Oct 2025".

Q58. What is the role of `package-lock.json`?
Answer: It locks the exact versions of dependencies installed to ensure the project works exactly the same on every machine.

Q59. Did you use any CSS Preprocessors?
Answer: No, we used modern CSS (with variables) or Tailwind, which provides similar power without the need for SASS/LESS compilation configuration.

Q60. How does the Dashboard update dynamically?
Answer: When the backend returns new analysis data, we update the React State. React detects this change and re-renders the Dashboard components with the new values.

Section 3: Backend & Application Logic (Flask/Python) (Q61-100)

Q61. Why did you use Python for the backend?
Answer: Python has the strongest ecosystem for AI/ML (libraries like PyTorch, Scikit-learn, Tesseract), making it the natural choice for our data processing pipeline.

Q62. What is Flask?
Answer: A micro web framework for Python. It processes HTTP requests (GET/POST) from our frontend and returns responses.

Q63. Explain the route `/analyze` in your backend.
Answer: It is a POST route that accepts an image file, saves it temporarily, runs the OCR extraction function, processes the text, and returns the JSON result.

Q64. How do you handle file uploads in Flask?
Answer: Using `request.files['file']` to access the uploaded object and `.save()` to write it to disk for processing.

Q65. What is the structure of the JSON response sent to the frontend?
Answer: It typically contains fields like `{ "hemoglobin": 14.5, "sugar": 90, "risk_prediction": "Normal", "recommendations": [...] }`.

Q66. Did you use a Virtual Environment? Why?
Answer: Yes (`venv` or `conda`). It isolates our project dependencies (like specific versions of Flask or Pandas) from the global system Python installation.

Q67. What is `requirements.txt`?
Answer: A file listing all Python libraries required to run the project. `pip install -r requirements.txt` installs them all at once.

Q68. How do you parse the raw text from OCR?
Answer: We use Regular Expressions (Regex). For example, a regex pattern can look for keywords like "Haemoglobin" followed by a number to extract the value.

Q69. What if the OCR fails to read a value?
Answer: The system handles exceptions and may return "Not Detected" or `null` for that specific parameter, alerting the user to check the image quality.

Q70. Explain the role of `app.py` or `server.py`.
Answer: It is the main entry point where the Flask app is initialized, database connections are set up, and routes are defined.

Q71. How do you enable debugging in Flask?
Answer: By setting `debug=True` when running the app (`app.run(debug=True)`), which provides detailed error logs and auto-reloads on code changes.

Q72. What is an API Endpoint?
Answer: A specific URL (e.g., `http://localhost:5000/login`) that performs a specific function when accessed.

Q73. How is the 'Fitness Plan' logic implemented?
Answer: It is a rule-based logic in Python. `if text.hemoglobin < 12: return iron_rich_diet_plan`.

Q74. Did you use Multithreading?
Answer: Flask handles requests in threads by default. For heavy ML tasks, we process them synchronously in this prototype, but in production, we'd use a task queue like Celery.

Q75. How do you connect Python to the SQLite database?
Answer: Using a robust ORM like SQLAlchemy or the built-in `sqlite3` driver.

Q76. What is the difference between GET and POST?
Answer: GET is for retrieving data (e.g., fetching a profile). POST is for sending data (e.g., uploading a photo or logging in) to keep parameters secure in the body.

Q77. How do you handle concurrent users?
Answer: Flask's development server is not for production. In production, we would use a WSGI server like Gunicorn to handle multiple concurrent requests efficiently.

Q78. What libraries are used for image processing before OCR?
Answer: OpenCV (`cv2`) or Pillow (`PIL`). We use them to convert images to grayscale, threshold them, or resize them to improve OCR accuracy.

Q79. How do you secure the API?
Answer: We can implement JWT (JSON Web Tokens). When a user logs in, they get a token. They must send this token in the header of subsequent requests.

Q80. What is `pycache`?
Answer: A folder containing compiled Python bytecode (`.pyc` files) that makes the program start faster on subsequent runs.

Section 3.5: Advanced Python, Git & Software Engineering (Q81-100)

Q81. What is PEP 8?
Answer: It is the style guide for Python code. It recommends best practices like using 4 spaces for indentation, `snake_case` for function names, and limiting line length to 79 characters for readability.

Q82. What are Python Decorators?
Answer: Functions that modify the behavior of other functions. In Flask, `@app.route('/login')` is a decorator that registers the login function to that specific URL.

Q83. Explain List Comprehension with an example.
Answer: A concise way to create lists. Instead of a loop, we write: `squares = [x2 for x in range(10)]`. It is faster and more readable than standard `for` loops.

Q84. What is a Docstring?
Answer: A string literal used to document a specific segment of code (like a function or class). It starts and ends with triple quotes (`"""`) and explains what the function does.

Q85. What is the difference between `is` and `==` in Python?
Answer: `==` checks for value equality (do they look the same?), while `is` checks for reference equality (are they pointing to the exact same object in memory?).

Q86. What are Python Generators?
Answer: Functions that return an iterator using the `yield` keyword. They generate values one by one on the fly (lazy evaluation) rather than storing the entire sequence in memory, which is efficient for large datasets.

Q87. What is a Lambda Function?
Answer: A small anonymous function defined with the `lambda` keyword. Example: `add = lambda x, y: x + y`. We use it for short, throwaway functions inside `map()` or `filter()`.

Q88. Explain Object-Oriented Programming (OOP) in your project.
Answer: We use Classes to bundle data and functionality. For example, a `User` class might have attributes like `username` and methods like `check_password()`.

Q89. What is Inheritance?
Answer: A mechanism where a new class derives properties from an existing class. Ideally, we could have a base class `Report` and specific classes `BloodReport` and `UrineReport` inheriting from it.

Q90. What is 'Encapsulation'?
Answer: Restricting access to methods and variables to prevent direct data modification. In Python, we denote private variables with an underscore (e.g., `_password`).

Q91. What is Version Control (Git)?
Answer: A system that records changes to a file or set of files over time so that you can recall specific versions later. It allows multiple people to work on the project simultaneously.

Q92. What are the basic Git commands you used?
Answer: `git init` (start repo), `git add .` (stage files), `git commit -m "msg"` (save changes), `git push` (upload to GitHub), and `git pull` (download changes).

Q93. How do you handle Merge Conflicts?
Answer: When two people change the same line of code, Git cannot decide which one to keep. We must manually open the file, choose the correct code, and commit the resolved version.

Q94. What is the difference between specific `requirements.txt` and `Pipfile`?
Answer: `requirements.txt` is the standard, simple list of packages. `Pipfile` (used by Pipenv) is more advanced, managing deterministic builds and distinguishing between dev and prod packages automatically.

Q95. What is REST?
Answer: Representational State Transfer. It's an architectural style for APIs. It uses standard HTTP methods (GET, POST, PUT, DELETE) and stateless communication, typically exchanging JSON data.

Q96. XML vs JSON: Why did you choose JSON?
Answer: JSON (JavaScript Object Notation) is lighter, easier to parse, and native to JavaScript (our frontend), making it much faster for web applications compared to the verbose XML.

Q97. Explain HTTP Status Codes: 200, 401, 403, 500.
Answer: 200 OK: Success. 401 Unauthorized: User is not logged in. 403 Forbidden: User is logged in but doesn't have permission. 500 Internal Server Error: Server crashed.

Q98. Session vs Cookie: What's the difference?
Answer: A Cookie is data stored on the client (browser). A Session is data stored on the server. We usually store a Session ID in a Cookie to link the user to their server-side data securely.

Q99. What is a CSRF Token?
Answer: Cross-Site Request Forgery token. It's a secret value generated by the server to ensure that the request coming from the browser was actually intentionally made by the user, not by a malicious script.

Q100. Encryption vs Hashing: Which is used for passwords?
Answer: Hashing. Encryption is two-way (can be decrypted with a key). Hashing is one-way (cannot be reversed). We hash passwords so that even if the DB is stolen, the attacker cannot read the actual passwords.

Section 4: Machine Learning & OCR (Q81-140)

Q101. What is OCR?
Answer: Optical Character Recognition. It is the technology used to convert different types of documents, such as scanned paper documents, PDF files, or images captured by a digital camera, into editable and searchable data.

Q102. Why did you use Tesseract?
Answer: Tesseract is the most popular, open-source, and accurate OCR engine available. It supports multiple languages and is easy to integrate with Python via `pytesseract`.

Q103. How does Tesseract work?
Answer: It uses a two-step process: line finding (layout analysis) and character recognition (pattern matching and LSTM neural networks).

Q104. What is 'Image Pre-processing' in your app?
Answer: Cleaning the image before passing it to Tesseract. This involves Grayscale conversion (removing color noise), Binarization (black text on white background), and Noise Removal.

Q105. What is the 'Thresholding' technique?
Answer: It converts a grayscale image into a binary image (black and white) by setting a pixel value cutoff. We often use Adaptive Thresholding which calculates thresholds for smaller regions, handling uneven lighting.

Q106. Explain the Disease Prediction Model.
Answer: It is a supervised classification model. We trained it on a dataset where inputs are blood parameters (e.g., Hemoglobin, RBC) and the output is a label (e.g., "Anemia", "Healthy").

Q107. Which algorithm provided the best accuracy?
Answer: We found Random Forest to be very effective because it handles non-linear relationships well and is less prone to overfitting than a single decision tree.

Q108. What is a 'Training Set' vs. 'Test Set'?
Answer: We split our data (e.g., 80/20). The Training Set is used to teach the model. The Test Set is hidden during training and used to evaluate how well the model performs on unseen data.

Q109. What is 'LayoutLMv3' mentioned in your docs?
Answer: It is a state-of-the-art multi-modal Transformer model by Microsoft. Unlike standard OCR, it looks at the *layout* and *visuals* of the document, not just words, making it smarter at understanding structured tables in blood reports.

Q110. How do you improve OCR accuracy?
Answer: By ensuring the user takes a clear photo, cropping the relevant area, and applying image sharpening filters in the backend.

Q111. What is `pandas` used for?

Answer: Data manipulation. We use it to load datasets (CSV), clean data, and prepare logical structures (DataFrames) for feeding into the ML model.

Q112. Explain 'Feature Extraction' in your ML context.
Answer: It involves selecting the most relevant inputs. For us, features are the numeric values extracted from the report (e.g., '13.5' for Hb).

Q113. What is 'Supervised Learning'?
Answer: Learning where the model is provided with labeled data-it is told "Input X leads to Output Y" during training so it can predict Y for new X.

Q114. How do you handle 'False Positives'?
Answer: A False Positive would be predicting a disease when the person is healthy. We tune the model's 'Threshold' to prioritize precision if necessary, but essentially we advise users that this is a *screening tool*, not a diagnosis.

Q115. What is `scikit-learn`?
Answer: A robust machine learning library for Python. We use it for implementing algorithms like Random Forest, splitting datasets, and calculating accuracy metrics.

Q116. What is a 'Confusion Matrix'?
Answer: A table used to evaluate the performance of a classification model. It shows True Positives, True Negatives, False Positives, and False Negatives.

Q117. How does the 'Rule-Based System' for Diet work?
Answer: It's a set of conditional statements. `IF diagnosis == 'Diabetes' THEN recommend ['Low Carb', 'High Fiber']`. It doesn't "learn" but follows expert medical rules.

Q118. What data format does the ML model accept?
Answer: It accepts a numerical array or vector (e.g., `[14.5, 90, 4.5]`). Using `numpy`, we convert our parsed OCR data into this format.

Q119. Did you train your own Tesseract model?
Answer: No, we used the pre-trained English model (`eng.traineddata`). Training Tesseract from scratch is complex and usually unnecessary for standard fonts.

Q120. What is 'pickling' in Python?
Answer: Using the `pickle` or `joblib` library to save a trained ML model to a file (`model.pkl` or `.onnx`). This allows us to load the model instantly without retraining every time the server starts.

Section 5: Database & Security (Q141-160)

Q141. Why SQLite?
Answer: It is serverless, zero-configuration, and stores the entire database in a single file on the disk. It is perfect for development, prototypes, and small-to-medium-scale applications.

Q142. What tables do you have in your database?
Answer: Typically: `Users` (id, name, email, password_hash), `Reports` (id, user_id, date, image_path, data_json), and `Profiles` (user_id, age, height, weight).

Q143. How do you store passwords safely?
Answer: We hash them using a library like `bcrypt` or `werkzeug.security`. We never store plain text passwords. Hashing transforms "password123" into a random string that cannot be reversed.

Q144. What is SQL Injection? How do you prevent it?
Answer: An attack where malicious SQL is inserted into queries. We prevent it by using ORM (Object Relational Mapping) or Parameterized Queries, which automatically escape inputs.

Q145. How does the 'Foreign Key' work in your User-Report relationship?
Answer: The `Reports` table has a `user_id` column that links to the `id` of the `Users` table. This ensures every report belongs to a valid user.

Q146. How do you backup data?
Answer: With SQLite, backing up is as simple as copying the `.db` file. In a real cloud setup, we would rely on the cloud provider's automated backup solutions.

Q147. What is 'Normalization'?
Answer: Organizing the database to reduce redundancy. For example, storing user details in one table and their multiple reports in another, linked by ID, rather than repeating user details in every report.

Q148. Can other users see my reports?
Answer: No. The backend APIs enforce authentication. When fetching reports, the query always filters by the currently logged-in user's ID (`SELECT * FROM reports WHERE user_id = current_user.id`).

Q149. What happens if the database file is deleted?
Answer: Since it's a file-based DB, all data would be lost. This is why production apps use dedicated database servers (PostgreSQL/MySQL) with backups.

Q150. How do you handle image storage?
Answer: We store the *path* (filename) of the image in the database, while the actual image file is stored in a designated `uploads/` folder on the server. Storing large blobs (images) directly in the DB is generally bad for performance.

Section 6: Testing & Deployment (Q161-180)

Q161. What types of testing did you perform?
Answer: Unit Testing (testing individual functions like BMI calc), Integration Testing (testing API + Database), and User Acceptance Testing (UAT) (checking if it works for the end user).

Q162. How did you test the API?
Answer: Using tools like Postman or Thunder Client to send sample requests and verify the JSON responses.

Q163. How do you deploy the frontend?
Answer: We run `npm run build` to create static files (`index.html`, `js/`, `css/`). These can be hosted on GitHub Pages, Vercel, or Netlify.

Q164. How do you deploy the backend?
Answer: The Python Flask app is deployed to a platform like Render or Heroku. We provide a `requirements.txt` and a `Procfile` (or build command) to tell the server how to run it.

Q165. What is 'Continuous Integration' (CI)?
Answer: Automating the testing and building process. We can use GitHub Actions to automatically build the app whenever we push code to the repository.

Q166. How does the app handle different screen sizes?
Answer: We test manually using Chrome DevTools' "Device Mode" to simulate iPhone, iPad, and Desktop views and adjust CSS accordingly.

Q167. What is a '404' vs '500' error?
Answer: 404 means "Not Found" (wrong URL). 500 means "Internal Server Error" (something crashed in our Python code).

Q168. How do you monitor the app's health?
Answer: In production, we'd use logging services (like Sentry or cloud logs) to track errors and uptime. For now, we check server console logs.

Q169. What is the 'Build' process in Vite?
Answer: It takes all our separate `.jsx` and `.css` files, compiles them, removes unused code (tree-shaking), and bundles them into efficient, tiny files for the browser to load.

Q170. Did you face any 'Latency' issues?
Answer: OCR can be slow (1-3 seconds). We mitigate user frustration by showing a "Processing..." animation so they know the system is working.

Section 7: Future Scope & Conclusion (Q181-200)

Q181. What are the limitations of your current system?
Answer: It relies on specific report formats (tables). Use of non-standard, handwritten, or very messy reports might fail. It requires internet connectivity.

Q182. How can you improve the OCR?
Answer: By fine-tuning Tesseract on specific medical fonts or moving to cloud-based Vision APIs (Google Vision/AWS Textract) which are more powerful but costly.

Q183. Can this be extended to other medical reports?
Answer: Yes. The core logic (Image -> Text -> Analysis) applies to X-Rays (using CNNs), MRI, or Urine analysis reports with model retraining.

Q184. How about adding 'Doctor Consultation'?
Answer: We can add a "Book Appointment" feature that connects users to doctors if their report shows "High Risk" flags.

Q185. Can we integrate Wearable data?
Answer: Yes. We can use APIs from Google Fit or Apple Health to pull step counts and heart rate, combining that with blood data for a 360-degree health view.

Q186. What about multi-language support?
Answer: We can use i18n libraries in React for UI translation and Tesseract's language packs to read reports in other languages (e.g., Hindi, Spanish).

Q187. Could you use Blockchain here?
Answer: Yes, for Electronic Health Records (EHR). Blockchain can provide an immutable, secure ledger for patient history that can be shared between hospitals with user consent.

Q188. How would you scale this to 1 million users?
Answer: Move to a cloud architecture (AWS/Azure), use Load Balancers, separate the DB (PostgreSQL), use Redis for caching, and use Docker containers for microservices.

Q189. What is the social impact of this project?
Answer: It democratizes health information, reduces anxiety by explaining reports simply, and promotes proactive health management.

Q190. How would you handle legally sensitive advice?
Answer: We must add a strict Disclaimer: "This is AI-generated advice, not a doctor's diagnosis." We should ensure the app is compliant with regulations like HIPAA or GDPR.

Q191. Can you implement 'Voice' interaction?
Answer: Yes, we can add a Voice Assistant that reads out the report summary for accessibility (using Web Speech API).

Viva Q&A - Blood App

Q192. What are 'Microservices'?
Answer: Breaking the app into small, independent services (e.g., one service just for OCR, one just for Auth). This improves scalability and fault isolation.

Q193. How can you improve the Diet Recommendation?
Answer: By integrating a comprehensive food nutrition database API (like USDA) to give specific meal plans with calorie counts, not just generic advice.

Q194. What is 'Edge AI'?
Answer: Running the AI models directly on the user's phone (using TensorFlow.js or Onnx Runtime) instead of the server. This improves privacy and works offline.

Q195. How do you handle 'Data Drift'?
Answer: If hospital report formats change over time, our model might fail. We need continuous monitoring and periodic retraining of the model with new data.

Q196. Can you generate a PDF report for the user?
Answer: Yes, we can use libraries like `jsPDF` (frontend) or `ReportLab` (backend) to generate a downloadable PDF summary of our analysis.

Q197. What is 'A/B Testing'?
Answer: Showing two different versions of a feature (e.g., two different dashboard layouts) to users to see which one performs better.

Q198. Explain the term 'Full Stack'.
Answer: It refers to working on both the Frontend (Client-side) and Backend (Server-side + Database) of the application.

Q199. What did you learn from this project?
Answer: I learned how to integrate disparate technologies (AI/OCR with Web Dev), handle real-world messy data, and design a user-centric product.

Q200. Summarize your project in one sentence.
Answer: A smart, AI-powered health assistant that decodes your blood reports and guides you towards a healthier lifestyle through personalized, data-driven insights.