**HeartLen App Documentation**

**1. Function of Each Component**

**CameraFeed**

**Purpose:** Captures video frames from the webcam and displays them on a canvas element

**Functionality:**

1. Accesses the user's camera through the browser's MediaDevices API
2. Extracts RGB values from specific sampling points in each video frame
3. Toggles camera recording on/off and handles flashlight activation when available
4. Provides the raw data needed for PPG signal processing

**ChartComponent**

**Purpose:** Visualizes the PPG signal and detected valleys in real-time

**Functionality:**

1. Displays the PPG signal as a continuous line chart using Chart.js
2. Highlights detected valleys with red markers
3. Provides visual feedback on heart rhythm patterns
4. Includes customized styling for dark/light mode compatibility
5. Updates in real-time as new PPG data comes in

**MetricsCard**

**Purpose:** Displays key metrics such as heart rate, HRV, and signal quality

**Functionality:**

1. Shows the value of metrics with appropriate units
2. Visualizes confidence levels with color-coded progress bars
3. Adapts display based on the type of metric (heart rate, HRV, signal quality)
4. Supports different color themes for visual differentiation between metrics
5. Handles formatting of complex data objects

**SignalCombinationSelector**

**Purpose:** Allows users to select different signal combinations for PPG processing

**Functionality:**

1. Provides a dropdown menu with various signal processing options:
   1. Default (2R - G - B): Standard combination optimized for PPG sensing
   2. Red Only: Uses only the red channel
   3. Green Only: Uses only the green channel
   4. Blue Only: Uses only the blue channel
   5. Red - Blue: Uses a combination of red and blue channels
   6. Custom (3R - G - B): Custom combination with stronger emphasis on red
2. Allows the switch between different signal processing methods

**2. Function of Each State Variable**

In the **app/page.tsx** File:

**isRecording**

**Purpose:** Tracks whether the app is currently recording video frames

**Usage:** Controls when the camera is active and processing frames

**State Management:** Toggled by the START/STOP RECORDING button

**isSampling**

**Purpose:** Controls whether PPG signal processing is active

**Usage:** When true, the app continuously samples and processes frames

**Interaction:** Works with isRecording to manage the sampling pipeline

**isUploading**

**Purpose:** Tracks whether data is being uploaded to MongoDB

**Usage:** Prevents multiple concurrent uploads attempts and manages loading states

**signalCombination**

**Purpose:** Stores the selected signal combination strategy

**Usage:** Determines how RGB channels are combined to create the PPG signal

**Options:** "default", "redOnly", "greenOnly", "blueOnly", "redMinusBlue", "custom"

**ppgData**

**Purpose:** Holds the processed PPG signal data array

**Usage:** Stored as an array of numerical values representing blood volume changes

**Size:** Limited to the most recent 300 samples for performance reasons

**valleys**

**Purpose:** Stores the indices and values of detected valleys in the PPG signal

**Usage:** Used for calculating heart rate and HRV

**Structure:** Array of objects with timestamp, value, and index properties

**heartRate**

**Purpose:** Stores the calculated heart rate and its confidence level

**Usage:** Displayed in the UI and saved to MongoDB

**Structure:** Object with bpm (beats per minute) and confidence properties

**hrv**

**Purpose:** Stores the calculated HRV and its confidence level

**Usage:** Displayed in the UI and saved to MongoDB

**Structure:** Object with sdnn (standard deviation of Normal to Normal intervals) and confidence properties

**currentSubject** and **confirmedSubject**

**Purpose:** Manage the subject identifier for MongoDB records

**Usage:** Track user input and confirmed subject ID for data attribution

**3. Data Processing Pipeline**

**Signal Processing Pipeline overview:**

Video Frame Capture → RGB Extraction → Signal Combination →

PPG Signal → Valley Detection → Heart Rate, HRV Calculation & signal quality prediction using trained ML model

**Frontend Processing**

**Video Frame Capture and RGB extraction**

1. Camera frames are captured ideally at approximately 30 FPS
2. The processFrame function in usePPGProcessing extracts pixel data
   1. Extracts the RGB value form 5 (top-left, top-right, center, bottom-left and bottom- right) from each frame
   2. Values are combined according to the selected signalCombination

**Signal Combination**

|  |  |
| --- | --- |
| Selected Method | Formula |
| Red | R/n |
| Green | G/n |
| Green | B/n |
| Red - Blue | (R-B)/n |
| Custom | (3R-B-G)/n |
| Default | (2R-G-B)/n |

**Where:**

**R, G, B = Sum of respective colour channels**

**n = Number of valid samples**

**Valley Detection**

1. Detects local minima in the normalized signal
2. Uses a moving window and minimum distance constraints
3. Each valley represents a heartbeat

**Metric Calculation**

1. **Heart Rate:** Calculated from the time intervals between valleys

**Formula:**

**Confidence** **Formula:**

**In which,**

1. **HRV (SDNN):** Standard deviation of intervals between consecutive valleys

**Formula:**

**HRV Confidence** combines:

* 1. **Interval Confidence**: proportion of valid RR intervals (scaled to 100%)
  2. **Consistency Confidence:** based on CV (as in heart rate)

**Formula**:

**Signal Quality Assessment**

1. Features are extracted from the PPG signal (Mean, SD, skewness, etc.)
2. TensorFlow.js model classifies signal as "bad", "acceptable", or "excellent"
3. Confidence score is calculated based on model prediction

**Backend Processing**

**Data Submission**

Processed data is submitted to **/api/save-record** endpoint

**MongoDB Storage**

Frontend (click Save Data button) → API Request → MongoDB Connection → Data Validation → Record Creation → Response

**Data Schema**

1. subjectId (required, indexed)
2. heartRate (BPM and confidence)
3. hrv (SDNN and confidence)
4. ppgData array
5. timestamp

Code snippet:

{

  subjectId: { type: String, required: true, index: true },

heartRate: {

    bpm: { type: Number, required: true },

    confidence: { type: Number, required: true },

  },

  hrv: {

    sdnn: { type: Number, required: true },

    confidence: { type: Number, required: true },

  },

  ppgData: { type: [Number], required: true },

  timestamp: { type: Date, default: Date.now },

}

**Data Retrieval**

1. Historical data can be queried by subject ID
2. The API calculates aggregated metrics (average heart rate, average HRV)
3. Last access timestamp is provided for reference

**4. Integration of ML Models**

**Model Initialization**

The model is loaded from the /public/tfjs\_model/model.json path

Loading occurs once when the component mounts

**Feature Extraction**

1. **Signal Preprocessing**

The PPG signal is analysed in windows of at least 100 samples

Statistical features are calculated from the signal:

1. Mean: Average signal value
2. Standard Deviation: Measure of signal variance
3. Skewness: Measure of distribution asymmetry
4. Kurtosis: Measure of "peakedness" of distribution
5. Signal Range: Difference between max and min values
6. Zero Crossings: Number of times signal crosses zero
7. RMS: Root mean square of the signal
8. Peak-to-Peak: Difference between max and min values

**Model Prediction**

1. **Classification Process**
2. Statistical Features are converted to a TensorFlow tensor
3. The model performs inference on the feature tensor
4. Output probabilities for each class (bad, acceptable, excellent) are calculated
5. The class with highest probability is selected as the prediction
6. **Result Handling**
7. The predicted class is displayed in the signal quality card
8. The confidence percentage is visualized as percentage and bar
9. Colour coding of the bar indicates confidence levels (red for low, orange for moderate and green for high)
10. Memory is properly managed by disposing of tensors after use
11. **Integration with UI**

The ML model results are displayed in the UI using the MetricsCard component, which shows:

1. The predicted signal quality class (bad, acceptable, excellent)
2. A confidence percentage indicating the model's certainty
3. Color-coded visual indicators based on the confidence level