Interview Analysis System: Frontend Implementation Specification

System Architecture Document

February 17, 2026

Contents

1 Overview

This document specifies the implementation requirements for a frontend user interface that exposes the outputs of the backend interview analysis pipeline. The frontend must provide temporal evidence inspection, analytical visualization, and pipeline execution control while adhering to strict data integrity constraints.

The backend pipeline consists of six processing stages that analyze video interviews, extract behavioral metrics, transcribe speech, identify question-answer pairs, and generate LLM-based assessments. The frontend must faithfully render all outputs produced by these stages without modification or recomputation.

2 Project Structure

2.1 Directory Layout

```
frontend/
|-- index.html              # Main application entry point
|-- css/
|   |-- styles.css         # Global styles
|-- js/
|   |-- app.js             # Main application controller
|   |-- data-loader.js     # JSON data loading module
|   |-- timeline-view.js   # Temporal evidence view
|   |-- analytics-view.js  # Analytical visualization
|   |-- pipeline-view.js   # Execution control view
|   |-- utils.js           # Utility functions
|-- assets/
|   |-- icons/             # UI icons (SVG)
|-- package.json           # Dependencies
```

2.2 Technology Stack

- **Framework**: Vanilla JavaScript ES6+ (no heavy frameworks)
- **UI Components**: Custom Web Components for reusability
- **Data Visualization**: Canvas API for timeline, SVG for charts
- **Build Tool**: Vite for development and bundling
- **Testing**: Playwright for E2E tests

3 Data Loading Module

3.1 Module Purpose

The data loader is responsible for fetching and validating all JSON artifacts produced by the backend pipeline. It must enforce strict schema compliance and fail explicitly on malformed data.

3.2 API Specification

The data loader exposes the following methods:

1. `constructor(basePath)` - Initialize with path to backend output
2. `loadAll()` - Load all pipeline outputs, returns Promise
3. `loadFile(filename, schema)` - Load single JSON with validation
4. `validate(data, schema)` - Validate against expected schema

State properties:

- `basePath` - Path to JSON files
- `cache` - Map of loaded data
- `loadingState` - 'idle' | 'loading' | 'loaded' | 'error'
- `error` - Error message if failed

3.3 Schema Definitions

The following schemas define the expected structure of each JSON output:

File	Key Fields	Type
timeline.json	timebase, dataset_id, video.fps, video.duration_sec, audio.candidate.duration_sec	Object
speaking_segments.json	segments[].segment_id, start_time, end_time, type	Array
qa_pairs.json	qa_pairs[].question_id, question_text, answer.text, answer.start_time	Array
interviewer_transcript.json	transcription.segments[].start_sec, end_sec, text, words[]	Object
candidate_behavior_metrics.json	segments[].segment_id, audio_metrics, video_metrics	Array
candidate_audio_raw.json	features[].timestamp_sec, rms_energy, pitch_hz	Array
candidate_video_raw.json	frames[].timestamp_sec, face_detected, head_pose, gaze	Array
relevance_scores.json	qa_id, relevance_score, matched_keywords[], justification	Array
candidate_score_timeline.json	checkpoint_entry.checkpoint, competency_scores, incremental_verdict	Array

4 Temporal Evidence View

4.1 Purpose

The temporal evidence view provides synchronized playback of video, audio activity indicators, transcript text, and question-answer boundaries. This is the primary interface for inspecting raw evidence.

4.2 Component Architecture

1. **Video/Audio Player**: Primary playback component
2. **Timeline Canvas**: Visual timeline with markers
3. **Transcript Panel**: Scrolling transcript with time alignment
4. **QA Navigator**: Question-answer pair navigation
5. **Metrics Overlay**: Behavioral metrics display

All components synchronize to a common timebase defined in timeline.json.

4.3 Timeline Visualization

The timeline canvas must render:

1. **Video progress bar**: Current playback position
2. **Speaking segments**: Colored spans indicating candidate speech
3. **Question markers**: Vertical lines with question IDs
4. **Answer spans**: Highlighted regions for each answer
5. **Score indicators**: Color-coded badges for LLM judgments

4.4 Time Synchronization Rules

All temporal elements must use the canonical timebase defined in timeline.json:

- Time unit: seconds (float with millisecond precision)
- Reference: video timeline (24fps for dataset 1)
- Offset: video-to-audio offset is 0.0 (assumed synchronized)
- All timestamps in JSON files use this unified timebase

4.5 Interaction Specifications

Action	Target	Result
Click	Timeline	Seek video to position, update all views
Click	Question marker	Jump to question start time, expand QA panel
Click	Answer span	Jump to answer start time, highlight transcript segment
Hover	QA pair	Show tooltip with scores, keywords, reasoning
Scroll	Transcript	Navigate through interview chronologically
Play/Pause	Video player	Toggle playback, sync timeline position

5 Analytics View

5.1 Purpose

The analytics view renders aggregated and per-question assessments derived from LLM analysis. This includes behavioral metrics, semantic relevance, and candidate progression.

5.2 Component Structure

1. **Summary Dashboard**: High-level metrics and final verdict
2. **Per-Question Analysis**: Individual QA pair evaluations
3. **Behavioral Trends**: Charts showing metrics over time
4. **JD Relevance**: Keyword matching results

5.3 Summary Dashboard

Metric	Source	Display
Final Verdict	candidate_score_timeline.json	Badge: STRONG_HIRE / ADEQUATE / WEAK
Overall Score	candidate_score_timeline.json	0.0-1.0 progress bar
Confidence	candidate_score_timeline.json	HIGH / MEDIUM / LOW
Total Questions	qa_pairs.json	Count with progress indicator
Avg Relevance	relevance_scores.json	0.0-1.0 with trend arrow

5.4 Per-Question Analysis Panel

For each question-answer pair, display:

- Question text (from qa_pairs.json)
- Answer text (from qa_pairs.json)
- Relevance score (from relevance_scores.json)
- Matched keywords (from relevance_scores.json)
- Competency scores (from candidate_score_timeline.json):
  - technical_depth
  - system_design
  - production_experience
  - communication_clarity
  - problem_solving
- Incremental verdict (from candidate_score_timeline.json)
- Reasoning (from candidate_score_timeline.json)

Clicking any question navigates to the temporal evidence view at that question's timestamp.

5.5 Behavioral Metrics Charts

Using data from candidate_behavior_metrics.json:

Metric	**Source Field**	**Chart Type**
Speech Rate	audio_metrics.speech_rate	Line chart over time
Pitch Variance	audio_metrics.pitch_variance	Area chart
Face Presence	video_metrics.face_presence_ratio	Stacked bar
Head Motion	video_metrics.head_motion_mean	Line chart
Gaze Stability	video_metrics.gaze_stability	Line chart
Energy	audio_metrics.energy_mean	Waveform visualization

6 Pipeline Execution View

6.1 Purpose

The pipeline view provides controlled invocation of the backend analysis pipeline with real-time feedback.

6.2 Input Requirements

The UI must accept the following inputs:

1. **Video file**: MP4 format, candidate video recording
2. **Candidate audio**: WAV format, extracted candidate audio

3. **Interviewer audio**: WAV format, extracted interviewer audio
4. **Job Description**: Markdown file containing JD text

Input files must be validated before submission:

- File existence check
- Format validation
- Naming convention: number_candidate_video.mp4, number_candidate_audio.wav, number_interviewer_audio.wav

6.3 Execution States

State	Trigger	UI Response
idle	Initial state	Show input form, disabled run button
validating	Input files selected	Show spinner, disable inputs
running	User clicks Run	Show progress, stream logs, disable inputs
completed	Exit code 0	Show success message, enable results view
failed	Exit code != 0	Show error message, enable retry

6.4 Subprocess Invocation

The pipeline execution uses Node.js child_process.spawn to invoke main.py:

1. Construct argument array from user inputs
2. Spawn python process with cwd set to backend directory
3. Attach stdout and stderr listeners to stream logs to UI
4. On process close, check exit code and update state
5. On success, call discoverOutputs() to scan for new JSON files

Key code points:

- Main script path: ’./backend/main.py’
- Working directory: ’./backend’
- Arguments: –video, –candidate-audio, –interviewer-audio, –jd
- Output directory scanning: ’./backend/output/’

6.5 Output Discovery

After successful completion:

1. Scan ./backend/output/ for JSON files
2. Parse each file to verify validity
3. Update data loader cache
4. Enable temporal and analytics views
5. Display run summary with file list

7 UI-JSON Mapping Tables

7.1 Temporal Evidence Mapping

UI Component	JSON File	Fields Used	Al
Video Player	timeline.json	video.file, video.duration_sec	Pr
Timeline Bar	speaking_segments.json	start_time, end_time	Ma tic
Segment Colors	speaking_segments.json	type (speaking/silence)	Vi
Transcript Panel	interviewer_transcript.json	segments[].text, start_sec, end_sec	Sc
QA Navigator	qa_pairs.json	question_start_time, answer.start_time	Ju
Metrics Overlay	candidate_behavior_metrics.json	segment_id, timestamps	Fr
Audio Waveform	candidate_audio_raw.json	timestamp_sec, rms_energy	Ca
Face Video	candidate_video_raw.json	timestamp_sec, face_detected	Fr

7.2 Analytics Mapping

UI Component	JSON File	Fields Used	Update Rule
Verdict Badge	candidate_score_timeline.json	final_verdict.verdict	On load
Score Gauge	candidate_score_timeline.json	final_verdict.overall_score	On load
Confidence Label	candidate_score_timeline.json	final_verdict.confidence	On load
Question Cards	qa_pairs.json	qa_pairs[]	On load
Relevance Scores	relevance_scores.json	relevance_score	Per question
Matched Keywords	relevance_scores.json	matched_keywords[]	Per question
Competency Bars	candidate_score_timeline.json	competency_scores	Per question
Progress Timeline	candidate_score_timeline.json	checkpoint_entry	Sequential
Behavioral Chart	candidate_behavior_metrics.json	audio/video_metrics	Aggregation

8 State Management

8.1 Application States

The application maintains the following state object:

State Property	Type	Purpose
dataLoading	string	'idle' | 'loading' | 'loaded' | 'error'
dataError	object	Error details if loading failed
pipeline	string	'idle' | 'validating' | 'running' | 'completed' | 'failed'
runLogs	array	Lines from stdout/stderr during execution
exitCode	number	Process exit code after completion
currentView	string	'timeline' | 'analytics' | 'pipeline'
selectedQA	string	Currently selected question ID or null
currentTime	number	Current playback time in seconds
isPlaying	boolean	Video playback state
outputs	object	Cached pipeline output data

8.2 State Transitions

1. Initial load: dataLoading = 'idle', pipeline = 'idle'
2. User selects inputs: pipeline = 'validating'
3. Validation passes: pipeline = 'running', start subprocess
4. Subprocess completes:
   - code = 0: pipeline = 'completed', dataLoading = 'loading'
   - code != 0: pipeline = 'failed'
5. Data loads successfully: dataLoading = 'loaded'
6. User navigates: currentView updates, components re-render

9 Error Handling

9.1 Error Categories

Category	Examples	UI Response
File Missing	timeline.json not found	Show error banner, disable views
Schema Invalid	Wrong JSON structure	Show field-level error details
Pipeline Failed	Non-zero exit code	Show stderr, enable retry

Category	Examples	UI Response
Network Error	Failed to fetch JSON	Show retry button
Video Error	Corrupt video file	Show player error state

9.2 Error Display Requirements

- All errors must be visible, not silent
- Error messages must be specific and actionable
- Error state must prevent further interaction until resolved
- Errors must be logged to console for debugging

10 Execution Flow

10.1 User Workflow

1. **Launch**: User opens index.html
2. **Input Selection**:
   (a) Click file input for video
   (b) Click file input for candidate audio
   (c) Click file input for interviewer audio
   (d) Click file input for JD markdown
3. **Validation**: System validates all inputs
4. **Execution**:
   (a) Click "Run Analysis"
   (b) Watch live logs scroll
   (c) Wait for completion
5. **Analysis**:
   (a) View timeline and playback video
   (b) Click questions to navigate
   (c) Hover for detailed scores
   (d) Switch to analytics view
   (e) Review summary and per-question details

11 Local Run Instructions

11.1 Development Setup

1. Clone and navigate to project directory
2. Run: npm install
3. Run: npm run dev
4. Open browser to: http://localhost:3000

11.2 Backend Prerequisites

- Python 3.8 or higher installed
- Required packages: opencv-python, librosa, whisper, etc.
- Video files location: ./backend/trans/
- JD files location: ./backend/jd/

12 Assumptions and Limitations

12.1 Assumptions

1. **Data Integrity**: JSON files are produced correctly by the backend
2. **Timebase**: All timestamps use video as canonical timebase
3. **File Naming**: Input files follow number_*.ext convention
4. **Web Environment**: Modern browser with ES6+ support
5. **No Auth**: Single-user local deployment

12.2 Limitations

1. **No Offline Processing**: Requires backend to be available
2. **Static JD**: Cannot edit JD after pipeline runs
3. **Read-Only**: Cannot modify analysis results
4. **Single Session**: No multi-interview comparison
5. **No Export**: Cannot download reports (future feature)

13 Validation Checklist

Before deployment, verify:

- [ ] All JSON schemas match actual backend output
- [ ] Timeline synchronization is accurate within 100ms
- [ ] Pipeline execution handles all exit codes correctly
- [ ] Error states display appropriate messages
- [ ] Per-question navigation jumps to correct timestamps
- [ ] Analytics charts render with correct data
- [ ] No console errors on normal operation
- [ ] Works in Chrome, Firefox, Safari (latest versions)

14 References

- Backend main.py: ./backend/main.py
- Output schemas: See JSON files in ./backend/output/
- Design document: Previous system design specification
- Example run results: ./backend/results/006-d04ed4c1/