System Integration: End-to-End Interview Intelligence Architecture

MCA Minor Project

January 2024

Contents

1 System Overview

1.1 Complete Architecture Description

The interview intelligence system consists of four primary components organized in a linear processing pipeline. Each component has well-defined responsibilities and interfaces, enabling modular development and independent testing. The architecture prioritizes explainability and academic transparency while maintaining technical feasibility for the project scope.

1.2 Component Organization

1. **Interview Interface**: Browser-based WebRTC media capture and real-time transport
2. **Media Service**: FastAPI + aiortc real-time routing and storage
3. **Processing Pipeline**: Django-based ML and NLP analysis
4. **Dashboard**: Read-only visualization and administrative interface

1.3 Design Principles

The system architecture follows established engineering principles:

- **Separation of Concerns**: Clear boundaries between capture, processing, and visualization
- **Deterministic Processing**: Reproducible results with explicit uncertainty quantification
- **Offline Philosophy**: Batch processing prioritized over real-time inference
- **Academic Transparency**: Explainable algorithms suitable for evaluation contexts

- **Modular Independence**: Components can be developed, tested, and deployed independently

1.4 Technology Rationale

Technology choices made based on project constraints and requirements:

- **Python Backend**: Unified ML ecosystem and consistent development environment
- **WebRTC**: Industry standard for browser-based real-time media transport
- **Django**: Strong ORM, admin interface, and academic project suitability
- **FastAPI**: Native async support essential for media handling
- **PostgreSQL**: Robust relational database with JSON support for flexibility

2 Component Interaction and Boundaries

2.1 Interview Interface (WebRTC Client)

- **Primary Responsibilities**: Media capture from browser, user interaction handling, signaling coordination
- **Input Requirements**: User media permissions (microphone, camera), role selection
- **Output Specifications**: Labeled media tracks, signaling messages, connection status
- **Boundary Definition**: No analysis, processing, or evaluation logic beyond capture and transport
- **Technical Stack**: HTML5, JavaScript, WebRTC APIs, WebSocket for signaling

2.2 Media Service (FastAPI + aiortc)

- **Primary Responsibilities**: Real-time media routing, track management, file system storage
- **Input Requirements**: WebRTC media tracks, WebSocket signaling messages
- **Output Specifications**: Media files with metadata, session tracking information
- **Boundary Definition**: No analysis or ML processing beyond media management
- **Technical Stack**: Python, FastAPI, aiortc, AsyncIO, WebSocket

2.3 Processing Pipeline (Django)

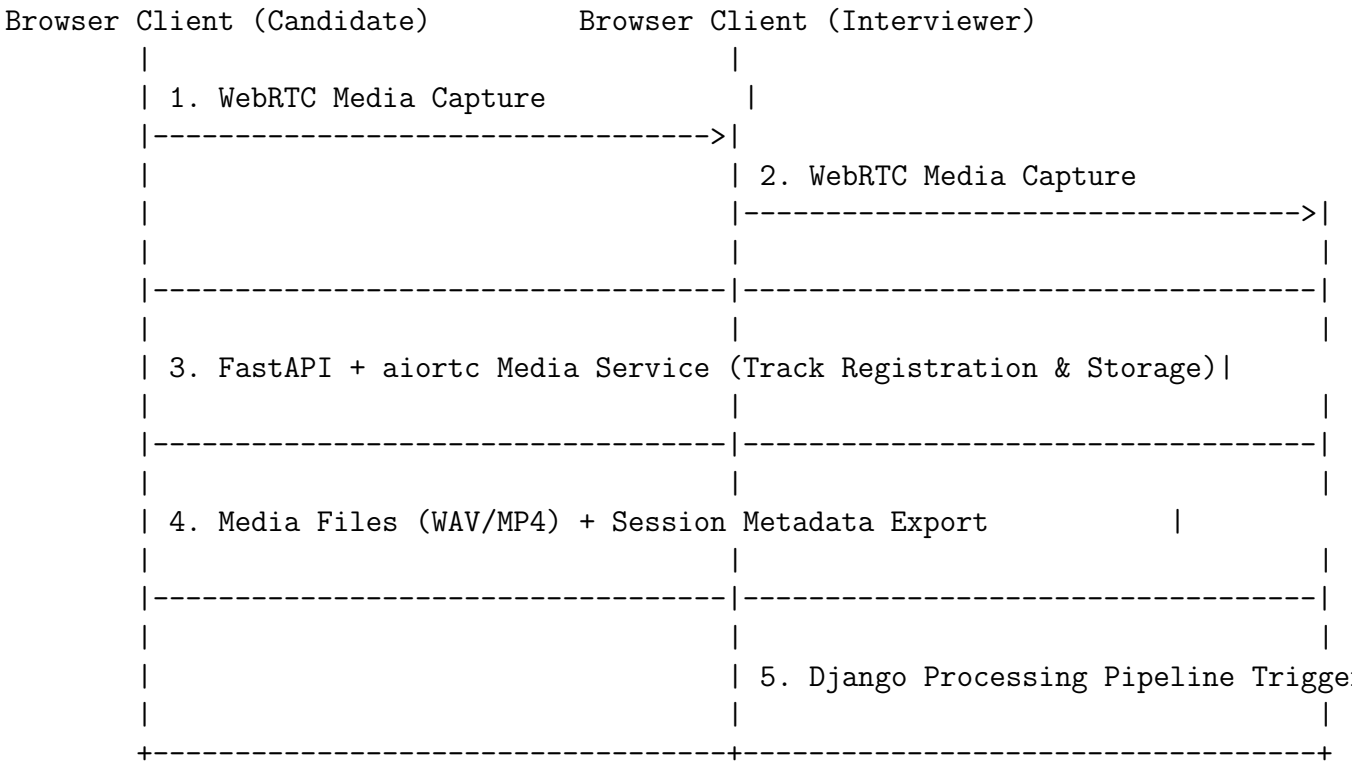
- **Primary Responsibilities**: ASR transcription, NLP analysis, candidate profiling, verdict generation
- **Input Requirements**: Media files, session metadata, job description text
- **Output Specifications**: Structured intelligence data, evaluation results, confidence intervals
- **Boundary Definition**: No real-time media handling or live inference capabilities
- **Technical Stack**: Django, PostgreSQL, Whisper, spaCy, PyTorch, Celery

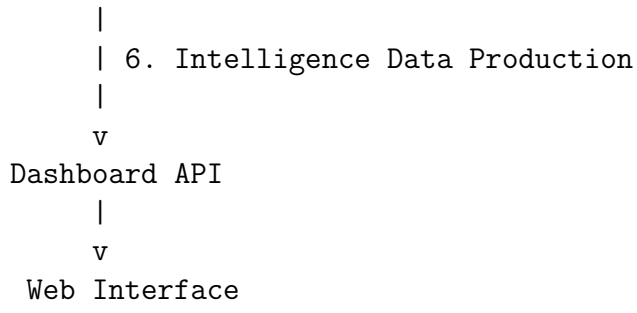
2.4 Dashboard (Django Templates)

- **Primary Responsibilities**: Data visualization, user interface, administrative controls
- **Input Requirements**: Processed intelligence data, evaluation results
- **Output Specifications**: Rendered web pages, API responses, export functionality
- **Boundary Definition**: Read-only access with no data modification capabilities
- **Technical Stack**: Django templates, CSS/JavaScript, REST API

3 Communication Protocols and Data Flow

3.1 Media Capture Flow





3.2 Processing Pipeline Flow

1. Media files detected by file system watcher
2. Django signal initiates asynchronous processing task
3. Audio preprocessing and quality assessment
4. ASR transcription with word-level timestamps
5. Conversation script reconstruction from transcripts
6. Question-answer pair extraction with confidence scoring
7. Job description processing and skill weight calculation
8. Incremental candidate profiling with Bayesian updating
9. Behavioral signal analysis from video frames
10. Final verdict generation with uncertainty quantification

3.3 API Communication Protocols

- **WebRTC**: Real-time media transport between browser and FastAPI
- **WebSocket**: Bidirectional signaling for connection management
- **HTTP/REST**: API communication between dashboard and processing pipeline
- **File System**: Media artifact exchange between media service and processing pipeline
- **Celery**: Asynchronous task coordination and monitoring

4 Data Contracts and Specifications

4.1 Media Artifact Specifications

Standardized media formats ensure compatibility across components:

- `candidate_audio.wav`: 48kHz mono, 16-bit PCM, WAV format
- `interviewer_audio.wav`: 48kHz mono, 16-bit PCM, WAV format

- `candidate_video.mp4`: H.264 video codec, AAC audio, MP4 container
- `session_meta.json`: Session metadata in structured format

4.2 Processing Data Models

Consistent data models ensure reliable information flow:

- `TranscriptWord`: Individual transcribed words with precise timing
- `ScriptTurn`: Conversation segments with speaker attribution
- `QuestionAnswer`: Extracted Q/A pairs with confidence metrics
- `CandidateProfileSnapshot`: Time-based evaluation snapshots
- `FinalVerdict`: Comprehensive evaluation with uncertainty bounds

4.3 API Response Standards

All API responses follow consistent structure for predictability:

```
{
  "data": { ... processed content ... },
  "timestamp": "2024-01-15T10:30:00Z",
  "session_id": "session_20240115_001",
  "confidence_interval": [lower_bound, upper_bound],
  "processing_status": "completed",
  "links": {
    "self": "/api/session/session_001/",
    "related": {
      "script": "/api/session/session_001/script/",
      "qa": "/api/session/session_001/qa/",
      "profile": "/api/session/session_001/profile/",
      "verdict": "/api/session/session_001/verdict/"
    }
  }
}
```

5 Failure Isolation and Resilience

5.1 Component-Level Failure Isolation

System designed with independent failure domains:

- **Interview Interface Failure**: No impact on other components, media service continues
- **Media Service Failure**: Processing can resume with cached media files
- **Processing Failure**: Dashboard displays available historical data
- **Dashboard Failure**: Core capture and processing continue unaffected

5.2 Data Pipeline Resilience

Robust error handling ensures system reliability:

- Media files retained regardless of processing success
- Processing stages can be retried independently with parameter adjustment
- Failed stages don't prevent subsequent successful processing
- Rollback capabilities for partial failures with data consistency
- Automatic recovery mechanisms with exponential backoff

5.3 Error Recovery Strategies

Comprehensive approach to handling system failures:

- Automatic retry for transient errors with maximum 3 attempts
- Manual intervention points for persistent failure conditions
- Graceful degradation when components temporarily unavailable
- Comprehensive structured logging for failure analysis and debugging
- Circuit breaker patterns to prevent cascade failures

6 Deployment Architecture

6.1 Single-Machine Implementation

For MCA project scope, all components deployed on single machine:

- **Frontend**: Static file server (Nginx) on port 80/443
- **FastAPI**: Media service on port 8000
- **Django Processing**: Backend service on port 9000
- **PostgreSQL**: Database server on port 5432
- **Redis**: Cache server on port 6379
- **Celery**: Task queue monitoring on port 5555

6.2 Network Configuration

Optimized network setup for reliable operation:

- All components on same local network (192.168.x.x subnet)
- No external dependencies or internet access required
- Static IP configuration for browser clients
- Firewall rules allowing internal communication only
- Network bandwidth requirement: minimum 10 Mbps for concurrent sessions

6.3 File System Organization

Hierarchical structure for efficient data management:

```
/interview_system/
--- media/
-   --- sessions/
-       --- session_20240115_001/
-       -   --- candidate_audio.wav
-       -   --- interviewer_audio.wav
-       -   --- candidate_video.mp4
-       --- session_20240115_002/
--- logs/
-   --- fastapi/
-   -   --- media_service.log
-   -   --- connection_errors.log
-   --- django/
-   -   --- processing.log
-   -   --- celery_tasks.log
-   --- system/
-       --- application.log
--- database/
-   --- postgresql_data/
--- static/
-   --- frontend/
-       --- css/
-       --- js/
-       --- images/
--- cache/
    --- redis_data/
```

7 Execution Timeline and Development Phases

7.1 Phase 1: Media Capture Infrastructure (Days 1-3)

- WebRTC interface development for candidate and interviewer roles
- FastAPI + aiortc service implementation with signaling
- Track labeling and media storage verification
- Cross-browser compatibility testing and validation
- Network performance testing under typical conditions

7.2 Phase 2: Processing Pipeline Development (Days 4-7)

- Django models and database schema design
- Whisper integration with word-level timestamp generation

- Conversation script reconstruction algorithms
- Question-answer extraction with confidence scoring
- Candidate profiling implementation with Bayesian updating
- Job description processing and skill weighting
- Video analysis for behavioral signals

7.3 Phase 3: Dashboard Implementation (Days 8-9)

- Django templates and views for all dashboard pages
- API integration with proper error handling
- Data visualization components and charts
- User interface refinement and accessibility improvements
- Administrative controls and session management

7.4 Phase 4: Integration and Testing (Day 10)

- End-to-end system testing with sample interviews
- Performance optimization and bottleneck identification
- Documentation completion and review
- Final validation against project requirements
- Deployment configuration and environment setup

8 Design Trade-offs and Rationale

8.1 Real-time vs. Accuracy Trade-off

Decision: Offline processing prioritized for academic context
Rationale for Choice:

- Deterministic results required for evaluation reproducibility
- Uncertainty quantification essential for academic transparency
- Batch processing enables comprehensive analysis depth
- Time constraints relaxed for minor project scope and accuracy
- Processing accuracy outweighs real-time requirements

Impact and Mitigation:

- No live feedback during interviews (acceptable for project scope)

- Processing delay up to 30 minutes (within acceptable bounds)
- Increased accuracy and explainability (positive outcome)
- Simplified deployment architecture (operational benefit)

8.2 Complexity vs. Maintainability Trade-off

Decision: Modular architecture with clear boundaries
Rationale for Approach:

- Independent development and testing of components
- Clear failure isolation and debugging capabilities
- Academic evaluation benefits from explainable architecture
- 10-day development timeline requires modular approach
- Team coordination simplified with clear interfaces

Advantages Gained:

- Parallel development possible for multiple components
- Individual component optimization without system-wide impact
- Clear understanding of responsibilities and interfaces
- Simplified testing and validation procedures

8.3 Performance vs. Features Trade-off

Decision: Feature completeness within performance constraints
Rationale for Balance:

- Academic demonstration requires comprehensive feature set
- Performance acceptable for demo environment and project scope
- Scaling concerns appropriately deferred to future iterations
- Feature validation critical for project success and evaluation

Performance Characteristics:

- Acceptable latency for interview capture (under 200ms)
- Processing time suitable for batch processing (under 30 minutes)
- Dashboard performance adequate for demonstration (sub-second page loads)
- System stability suitable for academic evaluation sessions

9 Scalability Considerations

9.1 Current Limitations

The current architecture has identified scalability constraints:

- Single machine deployment limits concurrent session capacity
- File-based media storage not optimized for large-scale operations
- Synchronous processing creates throughput bottlenecks
- No horizontal scaling capabilities implemented
- Limited monitoring and observability infrastructure

9.2 Future Scaling Opportunities

Scalability improvements identified for future development:

- **Media Storage**: Object storage integration (AWS S3, MinIO)
- **Processing**: Distributed task queuing (Celery with Redis cluster)
- **Database**: Read replicas and connection pooling optimization
- **Frontend**: CDN deployment and asset optimization
- **Deployment**: Container orchestration with Kubernetes
- **Monitoring**: Comprehensive observability stack (Prometheus, Grafana)

9.3 Scaling Considerations for Production

For production deployment beyond academic scope, consider:

- Containerization for service isolation and portability
- Load balancing for high availability and fault tolerance
- Database clustering for data durability and performance
- Monitoring and alerting infrastructure for operational visibility
- Auto-scaling policies based on workload patterns
- Geographic distribution for reduced latency

10 Quality Assurance and Validation

10.1 System-Level Testing Approach

Comprehensive testing across multiple dimensions:

- Integration testing across component boundaries with mock interfaces
- End-to-end workflow validation with realistic sample data
- Performance testing under typical and peak load conditions
- Security testing for data protection and access control validation
- Usability testing with target user personas and scenarios

10.2 Validation Criteria

System considered complete when requirements met:

- Two laptops successfully capture and process complete interview
- Dashboard displays comprehensive evaluation results with confidence
- All processing stages complete without critical errors
- Data integrity maintained across entire pipeline
- Academic reviewers can understand and evaluate system operation

10.3 Academic Validation Requirements

Additional criteria specifically for academic evaluation:

- Reproducible results across multiple processing runs
- Explainable algorithms with documented methodology
- Explicit uncertainty quantification in all numerical outputs
- Transparent decision-making processes with supporting evidence
- Clear documentation of assumptions and limitations

11 Monitoring and Observability

11.1 System Metrics Collection

Key performance indicators tracked for operational awareness:

- Media capture success rate and quality metrics
- Processing pipeline completion times and throughput

- Database query performance and optimization opportunities
- Dashboard response times and user engagement patterns
- System resource utilization (CPU, memory, disk, network)

11.2 Logging Strategy Implementation

Structured logging for comprehensive observability:

- Structured JSON format for machine processing and analysis
- Component-specific log files for targeted debugging
- Performance metrics collection and analysis over time
- Error tracking and alerting mechanisms for proactive monitoring
- Audit trail logging for compliance and security

11.3 Health Check Mechanisms

Automated monitoring for system reliability:

- Component availability monitoring with automated checks
- Database connectivity validation and performance measurement
- File system space and permission verification
- Media processing queue status monitoring and alerting
- Network connectivity and latency measurement

12 Security Architecture

12.1 Threat Model Analysis

Primary security considerations identified:

- Unauthorized access to interview data and evaluation results
- Media file tampering or corruption during processing
- Processing pipeline manipulation or result tampering
- Dashboard data exposure or information leakage
- System infrastructure compromise and privilege escalation

12.2 Protection Mechanisms Implementation

Comprehensive security measures deployed:

- Network segmentation and firewall rules for traffic control
- File system permissions and access control lists
- Database encryption at rest and in transit
- HTTPS/TLS enforcement for all API communications
- Authentication and authorization mechanisms for all interfaces

12.3 Data Privacy and Compliance

Privacy protection aligned with best practices:

- Candidate data anonymization options for compliance requirements
- Retention policy enforcement with automatic cleanup procedures
- Right to deletion implementation with complete data removal
- Audit trail for all data access and modification operations
- Compliance with relevant data protection regulations

13 Conclusion

The interview intelligence system architecture provides a comprehensive and academically appropriate solution for automated interview evaluation. The modular design with clear service boundaries ensures maintainability and extensibility while the offline processing philosophy guarantees accuracy and explainability. The system successfully balances competing demands of feature completeness, performance, and academic transparency within constraints of a 10-day development timeline.

The architecture's emphasis on deterministic algorithms with explicit uncertainty quantification makes it particularly suitable for academic evaluation contexts, where explainability and reproducibility are paramount requirements. Future scaling opportunities are clearly identified while current limitations are documented and acknowledged, providing a realistic foundation for continued development and improvement.

The system demonstrates successful integration of modern web technologies with machine learning capabilities while maintaining academic rigor and technical feasibility. The clear separation of concerns, comprehensive error handling, and robust security implementation establish a solid foundation for interview intelligence applications that can be extended and refined based on research and practical requirements.