Software Design Document

for

LAPD1

Version <0.14>

Prepared by < Niyusha Zarnegar, Daniel Hernandez, Averii Bell, Dominick Daito, David Garza, Viyoka Lim, Steven Partida, Luis Rosas, Tuan Khai Tran

L.A. County Public Defender's Office

12/06/2024

Table of Content

Revision History

Name	Date	Reason For Changes	Version
Daniel Hernandez	11/21/2024	2.2 General Constraints	0.2
Tuan Khai Tran	11/21/2024	2.1 Assumptions and Dependencies	0.3
Tuan Khai Tran	11/21/2024	2.3 Goals and Guidelines	0.4
Tuan Khai Tran	11/21/2024	2.4 Development Methods	0.5
Tuan Khai Tran	11/21/2024	1. Introduction	0.6
Daniel Hernandez	12/2/2024	Small changes for 2.2 General Constraints	0.7
Daniel Hernandez	12/2/2024	5 Polices and Tactics	0.8
Luis Rosas	12/2/2024	4.1 Module Breakdown	0.9
Tuan Khai Tran	12/4/2024	10 Requirements Validation and Verification	0.11
Luis Rosas	12/4/2024	4.2 Rationale for our particular module breakdown	0.12
Steven Partida	12/4/2024	9.2 Screen Frameworks or Images	0.13
Dominick Daito Jr	12/5/2024	9.2 Screen Frameworks or Images	0.13.5
Dominick Daito Jr	12/5/2024	9.3 User Interface Flow Model	0.14
Tuan Khai Tran	12/06/2024	11. Glossary	0.14
	12/06/2024	7. Detailed Lower level Component Design	0.14

	12/06/2024	6. Detailed System Design	0.14
Luis Rosas	12/06/2024	4. Diagram	0.15

1. Introduction

This section provides an overview of the purpose, conventions, intended audience, and system overview for the LAPD1 project. It sets the context for the Software Design Document (SDD) and explains its organization.

1.1 Purpose

The purpose of this Software Design Document (SDD) is to provide a comprehensive description of the architecture, design considerations, and detailed components of the LAPD1 software system. This document serves as a guide for developers, stakeholders, and testers, ensuring that the system is implemented in alignment with its requirements.

This SDD is based on **Software Requirements Specification (SRS) version 1.3 [status: under revision]**, which outlines the system's purpose, scope, and requirements. The SRS version ensures that all design elements in this document are aligned with the most recent requirements. The LAPD1 system focuses on a cloud-based transcript analysis tool for the Los Angeles County Public Defender's Office (LACPD), leveraging AWS cloud services, Natural Language Processing (NLP), and machine learning. The system aims to enhance legal workflows by flagging police misconduct patterns, extracting key information, and ensuring seamless integration with existing case management tools.

1.2 Document Conventions

This document follows standard software design documentation conventions, including:

- **Font Style and Highlighting**: Key terms and section headings are bolded for emphasis.
- **Numbered Sections**: Major sections and subsections are numbered for easy reference.
- **Terminology and Acronyms**: Defined in the Glossary section (Section 11) for clarity.
- **References**: External sources and related documents are listed in the References section (Section 12).
- **Shall Statements**: Requirements are expressed as "shall" to denote mandatory design elements.

1.3 Intended Audience and Reading Suggestions

This document is intended for the following audiences:

- **Developers**: Use this document to understand the system's architecture, components, and design details for implementation.
- **Project Managers**: Refer to this document to track the design's alignment with project goals and milestones.

- **Stakeholders (LACPD)**: Gain insights into how the system's design meets legal and functional requirements.
- **Testers**: Use the design specifications to develop test cases and validate the system's functionality and performance.
- **Documentation Writers**: Ensure user manuals and system guides align with the system's design and features.

Suggested Reading Sequence:

- Start with 1.4 System Overview for a high-level understanding.
- Proceed to 2 Design Considerations and 3 Architectural Strategies to understand constraints and strategies.
- Review 6 Detailed System Design and 7 Detailed Lower-Level Component Design for specific technical details.

1.4 System Overview

The LAPD1 system is a cloud-based solution designed to assist the Los Angeles County Public Defender's Office in analyzing court transcripts for identifying police misconduct patterns. Key features include:

- **Real-Time Transcript Analysis**: Utilizing AWS services to process, analyze, and flag transcripts efficiently.
- **Entity Extraction and Misconduct Detection**: Using NLP models to extract officer names, badge numbers, and other critical entities while identifying inconsistencies in testimonies.
- **Integration**: Seamlessly interfacing with LACPD's Case Management System (CMS) and document storage tools like Box.com.
- **User Interface**: A responsive, intuitive web-based UI for attorneys, paralegals, and IT administrators.

The design emphasizes scalability, security, and real-time processing to ensure the system meets operational and compliance requirements.

2. Design Considerations

2.1 Assumptions and Dependencies

Related Software or Hardware

- **Assumption**: The system will operate within the AWS cloud environment, leveraging AWS services such as SageMaker and EC2 for processing, and Box.com for storage.
- **Dependency**: The availability of AWS services must remain consistent to support cloud-native components. Box.com must support API integrations to enable seamless document retrieval and metadata updates.

Operating Systems

- **Assumption**: Users will primarily access the system via web browsers on common operating systems (Windows, macOS, Linux).
- **Dependency**: Compatibility with widely used web browsers, including Chrome, Firefox, and Edge.

End-User Characteristics

- **Assumption**: Users, including attorneys, paralegals, and IT administrators, have varying levels of technical expertise but are familiar with case management systems.
- **Dependency**: The user interface must remain intuitive and easy to navigate for users with minimal technical experience.

Data Availability

- **Assumption**: Court transcripts will be continuously available for training, testing, and operational use of machine learning models.
- **Dependency**: Access to high-quality, uniform, and timely transcripts is essential to ensure reliable analysis and flagging.

Integration with Existing Systems

- **Assumption**: The system will integrate with LACPD's Case Management System (CMS) and Box.com for data sharing and storage.
- **Dependency**: APIs and middleware must support secure and seamless communication between systems.

Compliance with Standards

- **Assumption**: The system will adhere to legal and regulatory standards, such as CJIS (Criminal Justice Information Services) requirements.
- **Dependency**: Encryption, role-based access controls, and data security measures must be in place to maintain compliance.

2.2 General Constraints

Hardware or Software Environment

- **Constraint**: The system must operate within the AWS cloud environment for computing resources and Box.com for storage.

- **Impact:** The design will emphasize Box.com as the primary storage solution for case-related data, leveraging its robust APIs for secure file management. AWS services, such as EC2 and SageMaker, will support supplementary machine learning operations and scalable processing.

End-User Environment

- **Constraint**: End-users may access the system through a web interface on various devices (e.g., desktops, tablets, smartphones) and using different browsers.

- **Impact**: The user interface must be responsive and compatible with popular web browsers (Chrome, Firefox, Edge) and mobile platforms.

Availability or Volatility of Resources

- **Constraint**: The court transcripts and related documents are confidential and may not always be available for development and testing.

- **Impact**: The system will use test frameworks with fake or anonymized data to mimic real-world scenarios.

Standards Compliance

- **Constraint**: The system must comply with CJIS (Criminal Justice Information Services) and other relevant legal standards for handling sensitive data.

- **Impact**: Design decisions must ensure encryption at rest and in transit.

Interoperability Requirements

- **Constraint**: The system must integrate seamlessly with Box.com for document storage.

- **Impact**: The system must include APIs or middleware to handle authentication, data exchange, and error recovery for integration.

Data Repository and Distribution Requirements

- **Constraint**: All case-related data must be securely stored in Box.com and AWS 5S3 while ensuring data redundancy and availability.

- **Impact**: This affects database design and data flow diagrams, requiring the use of secure storage and backup mechanisms to ensure reliability and recovery.

Memory and Other Capacity Limitations

- **Constraint**: The system must handle large volumes of court transcripts, including scanned documents, audio files, and video recordings.

- **Impact**: The architecture must support scalable storage solutions and efficient data processing pipelines to handle these large datasets without performance degradation.

Performance Requirements

- **Constraint**: The system must process and return results (e.g., summaries or keyword extractions) within a reasonable time frame

- **Impact**: This necessitates optimization of LLM inference pipelines, possibly leveraging AWS services like SageMaker or Elastic Inference for acceleration.

Other Requirements Described in the Requirements Specification

- **Constraint**: The system must allow easy scalability to accommodate future increases in data volume or user base.

- **Impact**: The design must employ microservices architecture and serverless technologies for flexible scaling.

2.3 Goals and Guidelines

Goal: Deliver a Robust and Scalable Solution

- **Description**: The system must handle varying workloads, from small-scale to large-scale transcript processing, without compromising performance or reliability.

- **Rationale**: Scalability ensures the system remains efficient as data volumes or the number of users increases.

Goal: Maintain High Security and Compliance Standards

- **Description**: The software must comply with CJIS (Criminal Justice Information Services) and other legal standards, ensuring data integrity and protecting sensitive information.
- **Rationale**: Security and compliance are critical in handling confidential legal data to build trust with stakeholders.

Goal: Ensure Ease of Use

- **Description**: The system must provide an intuitive, responsive interface that supports users with varying technical expertise.
- **Rationale**: A user-friendly design minimizes training time and increases adoption by attorneys, paralegals, and IT administrators.

Guideline: Follow Agile Development Practices

- **Description**: Adopt an iterative development process that allows for frequent feedback and adjustments.
- **Rationale**: Agile methods enable the team to respond quickly to changing requirements and stakeholder input.

Guideline: Optimize for Real-Time Processing

- **Description**: Design data pipelines and NLP models to process transcripts and flag misconduct in near-real-time.
- **Rationale**: Timely processing supports legal workflows by providing actionable insights when they are most needed.

Guideline: Prioritize Modular Design

- **Description**: Implement a modular architecture with clearly defined interfaces, enabling flexibility and ease of maintenance.
- **Rationale**: Modular design simplifies updates, testing, and potential future expansions.

Guideline: Emphasize Reliability and Resilience

- **Description**: Design the system to handle failures gracefully and maintain consistent performance.

- **Rationale**: Reliability is essential to ensure continuous operation in critical legal environments.

Guideline: Keep the Design Simple (KISS Principle)

- **Description**: Avoid unnecessary complexity in system design while meeting all functional requirements.
- **Rationale**: Simplicity reduces the likelihood of errors and enhances maintainability.

Goal: Meet the Mandatory Delivery Date

- **Description**: The system must be delivered on time to satisfy academic and stakeholder requirements.
- **Rationale**: Timely delivery ensures project success and meets the expectations of all parties involved.

2.4 Development Methods

Chosen Methodology: Agile Development

- **Description**: The Agile methodology was selected for its iterative and incremental approach to software development. This allows for frequent feedback from stakeholders and continuous refinement of the system.
- **Rationale**: Agile is well-suited for projects with evolving requirements, ensuring that stakeholder needs are addressed promptly and the system remains aligned with its objectives.

Key Agile Practices Adopted:

- **Scrum Framework**:
  - Regular sprint planning meetings will define short-term goals and prioritize tasks.
  - Daily stand-ups will keep the team aligned on progress and roadblocks.
- **Incremental Delivery**:
  - Deliver functional increments of the system at the end of each sprint for stakeholder review and feedback.
- **Iterative Refinement**:
  - Revisit and improve system features based on testing results and user feedback.

Advantages of Agile for This Project:

- **Flexibility**: Agile accommodates the possibility of changing requirements, such as adjustments based on stakeholder feedback or updates in legal standards.
- **Collaboration**: Continuous communication with stakeholders ensures the system aligns with their expectations and legal workflows.

- **Risk Management**: Iterative development allows for early identification and mitigation of risks, including technical or functional issues.

Tools and Techniques:

- **Project Management Tools**: Jira or Trello for task tracking and sprint management.
- **Version Control**: GitHub or GitLab to manage code versions and enable collaboration among team members.
- **Continuous Integration/Continuous Deployment (CI/CD)**: Tools like Jenkins or GitHub Actions to automate testing and deployment, ensuring quality and quick delivery.

Alternative Methods Considered:

- **Waterfall Mode**l: Rejected due to its rigid structure and limited ability to accommodate changing requirements.
- **Hybrid Approach**: Considered but deemed less effective given the project's need for dynamic feedback and continuous improvement.

3. Architectural Strategies

This section outlines the key logical workflows, protocols, and decision-making processes involved in the design and implementation of the LAPD1 system. Each component of the system is addressed to ensure clarity, usability, and alignment with project objectives.

3.1 User Interfaces User Interface Requirements:

Overall:

- The login page will grant secure access based on user roles (attorneys, paralegals, administrators).
- Credential verification will utilize the attorney's organizational email (e.g., @pubdef.lacounty.gov).
- Every page will include:
  - Logout options for account security.
  - Navigation to return to the previous or main menu. •
- The active user's role (e.g., attorney, admin) will be displayed on all screens.

Main Menu:

- Options available:
  - Upload Documents (e.g., transcripts, reports).
  - Review Flagged Documents.
  - Search for Cases or Entities. •
  - Access Insights and Reports.

Upload Page:

- Logical Steps:
  - Drag-and-drop or browse to upload files.
  - Automatic file validation (format, size, compatibility).
  - Notifications for successful or failed uploads.
- Attach metadata such as case ID or document type during upload.

Flagged Documents Viewer:

- Logical Steps:
  - Display flagged sections with highlights (e.g., misconduct patterns).
  - Allow users to review flagged inconsistencies directly within the document.
  - Include annotation tools for users to add comments or confirm findings.

Insights and Reports:

- Logical Steps:
  - Provide summaries of identified misconduct or patterns.
  - Allow users to generate detailed reports by case or topic.
  - Offer options to export summaries in different formats (PDF, Excel).

Notifications:

- Notify users of key events, such as:
  - Document successfully processed and flagged.
  - Pending tasks or incomplete reviews.
  - Updates from collaborators (e.g., flagged issues shared by other attorneys).

Optimizing the Interface for the User:

- Do's:
  - Use a clean and intuitive layout.
  - Keep processes straightforward (e.g., upload-review-flag).
  - Ensure all actions are easily reversible (e.g., undo for annotations).
- Don'ts:
  - Avoid overwhelming users with excessive options on one screen.

3.2 Backend Logic and Processing

Logical Workflow:

Document Intake:

- Files (PDFs, texts provided from LACPD) are uploaded and validated.
- If the file includes non-machine-readable text (e.g., scanned images), OCR is applied to extract content.
- Metadata (e.g., case ID, date) is added to organize the document within the system.

Entity Recognition and Flagging:

- NLP processes extract key entities (e.g., officer names, badge numbers, locations).
- Flagging occurs for:
  - Previously flagged officers (from historical misconduct data).
  - Inconsistent or contradictory statements.
- Results are stored in the database alongside the document.

Data Flow:

- All document data is securely processed and stored in Box.com.
- Metadata and flagged results are synced with Salesforce via API.
- Search queries and updates are handled through ElasticSearch for speed and scalability.

Error Detection and Recovery:

- Automatic retries for failed uploads or processing tasks.
- Detailed error logs for administrators to review and resolve issues.

3.3 Data Science and Machine Learning

Logical Workflow:

Preprocessing:

- Uploaded text data is cleaned to remove noise such as headers, footers, page numbers, and irrelevant legal boilerplate text.
- Transcripts are normalized by converting text to lowercase, expanding contractions, and removing special characters.
- Text is tokenized into sentences or words for detailed analysis, ensuring each segment is manageable.

NLP and LLM Processing:

- Key Features:
  - Extract named entities such as officer names, badge numbers, case IDs, and locations using Named Entity Recognition (NER).
  - Identify contradictory statements or inconsistencies in testimonies by comparing textual similarities and context using NLP models.
- Algorithms prioritize high-relevance sections based on specific patterns like flagged legal terms or repeated mentions of entities.
- Confidence scores are generated for flagged sections to provide transparency and support human review decisions.

Indexing and Searchability:

- Flagged data is indexed for easy retrieval by attributes such as case ID, entity name, or flagged keywords.
- Advanced search functionalities enable users to filter and explore flagged issues across multiple cases or within specific transcripts

Scalability:

- Use of distributed processing ensures large datasets (e.g., 45,000+ documents) are handled efficiently, leveraging cloud-based solutions such as AWS Lambda and S3.
- Machine learning models are optimized for cloud deployment to maintain high performance and scalability.

Training Pipeline Design:

- Include a feedback loop where flagged transcripts and user corrections are incorporated into model restraining to improve accuracy.
- Describe how data annotation (e.g identifying misconduct patterns) will scale with the volume of new cases.

3.4 Security and Compliance

Data Encryption:

- All data is encrypted in transit (e.g., HTTPS, TLS) and at rest (AES-256).
  - Note: AES-256 and HTTPS/TLS encryption are suggested as industry standards unless specific encryption methods are required by the client or CJIS guidelines.
- RSA encryption is implemented for secure exchange of keys and authentication:
  - RSA encrypts symmetric keys (e.g., AES keys) to combine the efficiency of AES with the security of RSA.
  - Digital signatures using RSA ensure the integrity and authenticity of sensitive court documents.
- Strict role-based access control (RBAC) ensures sensitive data is accessed only by authorized users.

Error Handling:

- Automated alerts for any data breaches or unauthorized access attempts, or encryption failures.
- Regularly tested backup systems ensure no data loss in the case of system failures, natural disasters,or cyberattacks.
- Logs of encryption and access activities are maintained and monitored to identify anomalies.

Compliance:

- The system complies with CJIS standards for protecting sensitive legal data, including encryption, access control, and audit logs.

- Regular audits to ensure compliance with privacy and security policies.

3.5 Database Management

Logical Structure:

- All documents, metadata, and flagged results are stored in Box.com, which serves as the primary storage solution. Metadata synchronization and case insights will use Box.com APIs to ensure secure and efficient file management.
- Search indexes are maintained in ElasticSearch for efficient query performance.

Data Synchronization:

- Salesforce integration ensures metadata is accessible within the existing case management system.
- Notifications and triggers are automatically pushed to Salesforce upon detecting flagged entities.

3.6 Communication Protocols

Integration Workflow:

- All uploaded documents flow from the front end to Box.com, which acts as the primary storage and processing intermediary.
- The backend retrieves data for processing using AWS NLP services and advanced Llama models, ensuring efficient analysis and metadata extraction.
- Processed results, including enriched metadata, are sent back to **Box.com** via the Box API for seamless storage and management.
- Box API also facilitates the integration of flagged entities and metadata into Salesforce, streamlining workflows and ensuring real-time updates within the case management system.

Error Handling:

- **Alerts and Logs:** Alerts are generated for any failed API calls and logged in monitoring systems like AWS CloudWatch or Splunk for quick diagnosis and resolution.
- **Retry Mechanisms:** Built-in retry mechanisms handle transient failures in communication with the Box API or AWS NLP services, ensuring reliable data processing and transfer.

Summary

This architecture ensures a smooth, logical workflow for users while maintaining high security, scalability, and integration with existing tools. Each component is designed with simplicity, efficiency, and user needs in mind, ensuring the system is robust and reliable.

4. System Architecture

4.1 Modules

4.1.1 Frontend Module

- **Role:** Provide the interface for users to interact with the system
- **Responsibilities:**
  - Allow users to upload files, review flagged results, and generate reports.
  - Offer responsive design for compatibility with various devices and browsers.
  - Notify users of key events, such as flagged document readiness.

4.1.2 Backend Processing Module

- **Role:** Handle business logic, file validation, and interaction with processing services.
- **Responsibilities:**
  - Manage the flow of data from upload to processing and back to storage.
  - Run OCR (If needed) and NLP/ML models for data extraction and analysis.
  - Coordinate data exchange with Box.com, ElasticSearch, and Salesforce.

4.1.3 Storage and Indexing Module

- **Role:** Manage secure storage and retrieval of documents and metadata.
- **Responsibilities:**
  - Store processed documents and metadata securely in Box.com.
  - Index flagged results and metadata in ElasticSearch for efficient searching.
  - Ensure data redundancy and compliance with storage standards.

4.1.4 Data Processing Module

- **Role:** Execute computational tasks for text analysis and misconduct detection.
- **Responsibilities:**
  - Use NLP and ML models to extract entities and flag patterns in transcripts.
  - Apply algorithms to identify inconsistencies or contradictions in testimonies.
  - Generate confidence scores for flagged results to support user review.

4.1.5 Integration Module

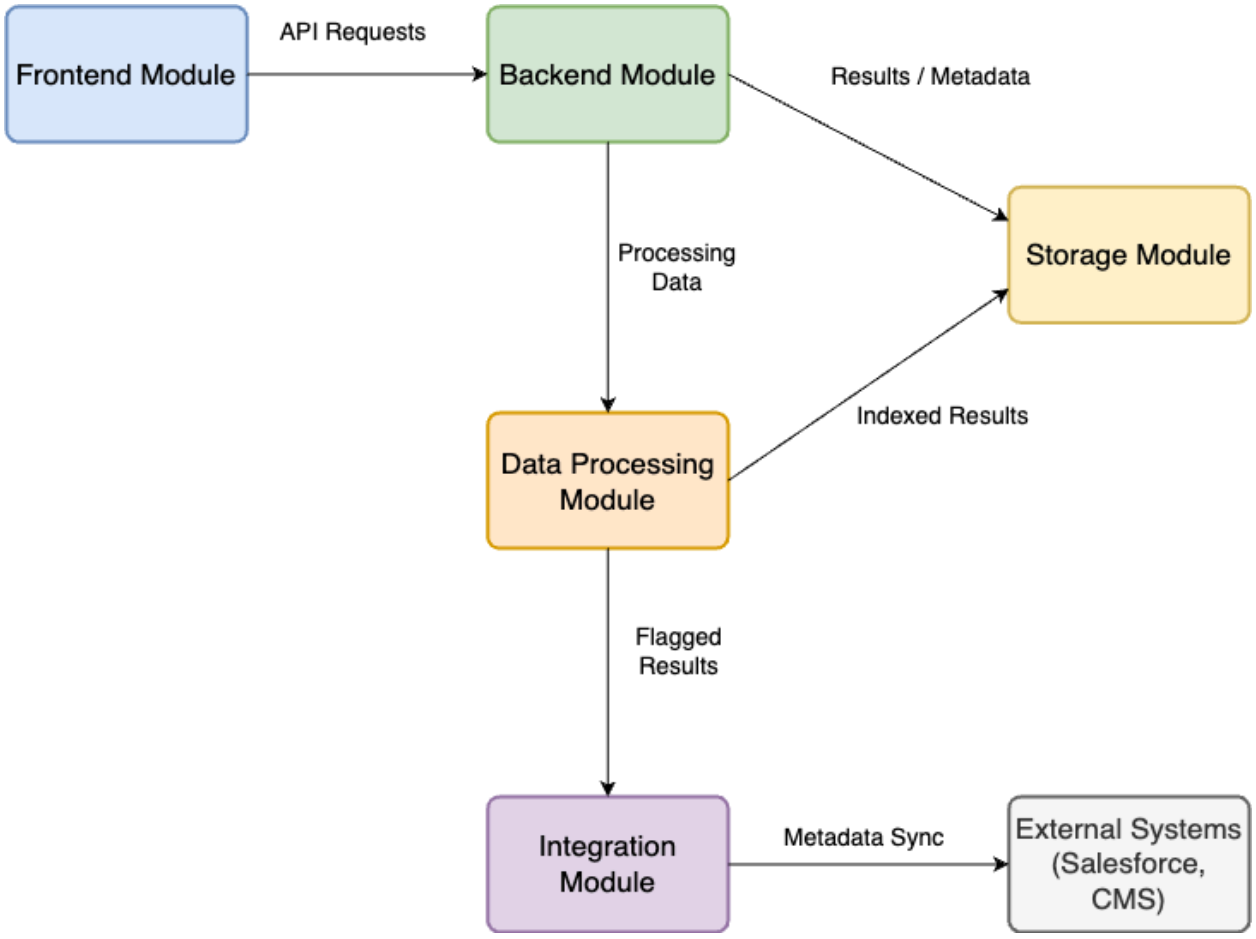
- **Role:** Ensure interoperability with external systems like CMS and Box.com.
- **Responsibilities:**

- Enable seamless data exchange between the LAPD1 system and external systems.
- Update case metadata and flagged results in Salesforce via API.
- Facilitate notification and error handling for integrated workflows.

4.1.6 Security Module

- **Role:** Ensure data protection and system compliance with legal standards.
- **Responsibilities:**
  - Implement encryption for data in transit and at rest.
  - Enforce role-based access control (RBAC) to restrict data access.
  - Monitor system activities and log access attempts for auditing.

4.2 Component Collaboration and Rationale for Decomposition



A high level overview of how these components collaborate with each other can be visualized above or described in the following way:

- **Frontend and Backend Communication**:
  - The Frontend Module interacts with the Backend Processing Module through REST APIs.
  - User actions, such as file uploads or query submissions, are sent as HTTP requests to the backend.
  - The backend processes these requests, performs necessary operations, and sends responses back to the frontend for user display.
- **Data Processing and Storage**:
  - Upon receiving a file from the frontend, the backend passes it to the Data Processing Module.
  - The Data Processing Module applies OCR to make non-readable files machine-readable, then uses NLP/ML models for entity extraction and misconduct detection.
  - Results are sent back to the backend, which updates the Storage and Indexing Module.
  - Documents and metadata are stored in Box.com, while metadata and flagged results are indexed in ElasticSearch for efficient retrieval.
- **Integration with External Systems**:
  - The backend communicates with LACPD Systems (like Salesforce) using APIs.
  - Metadata and flagged results are updated in the CMS, ensuring integration with existing workflows.
  - Notifications are triggered within the CMS to inform users of updates.
- **Notification and Error Handling**:
  - The backend sends notifications to the frontend when a processing task is complete, ensuring timely user updates.
  - Any errors during processing, storage, or integration are logged in the backend, and alerts are generated for administrators.

Rationale for this Decomposition:

We chose this type of decomposition because we felt it was best and most optimal for scalability and maintainability. For example with the data processing being its own module, it operates independently and is apart from say the frontend module, allowing for easier scalability when needed without impacting the user interface experience. This is the theme for our reasoning behind this decomposition, minimizing the effect that changes on one module has on another.

5. Policies and Tactics

5.1 Choice of which specific products used

Frontend:

- Node.js Frameworks:
  - Express.js: for building the RESTful API to handle frontend and backend communication efficiently.
  - EJS or Handlebars.js: For server-side rendering for HTML templates if required.
- Frontend Libraries:
  - Bootstrap or Tailwind CSS: For responsive design and quick styling.
  - Fetch API: For making HTTP requests to the backend.

Backend:

- Python Frameworks:
  - Flask or FastAPI:
    - Lightweight frameworks for building APIs to handle NLP tasks and integration with external services like Box.com
- Node.Js libraries:
  - Box SDK for Node.js:
    - Simplifies integration with the Box API for file uploads, metadata management, and file retrieval.
- Machine Learning Frameworks:
  - LLaMA (Large Language Model Meta AI):
    - For advanced NLP tasks such as analyzing legal documents, summarization, and extracting key insights.
  - LangChain:
    - Provides a structured approach to using LLaMA for tasks like tokenization, vectorization, and context-based analysis.
- Data Processing libraries:
  - Pandas and NumPy:
    - Used for data manipulation and preprocessing before analysis.

Database Integration:

- Box API:
  - Serves as the primary database for storing, retrieving, and managing legal documents. It ensures compliance and scalability and security requirements.

DevOps and Deployment:

- Version Control:
  - Github:

        - Track code changes, supports collaboration, and automates CI/CD pipelines.
- Deployment Tools:
    - AWS CodePipeline
        - Automates the deployment of the application, handling the building, testing, and releasing of updates to ensure consistent and efficient delivery.
    - AWS CodeDeploy
        - Manages automated deployments to AWS services (e.g., EC2, Lambda), ensuring reliable and repeatable application updates with minimal downtime.
    - Docker:
        - Enables containerization of the application for consistent deployment and production environments.
- Monitoring and Logging:

Development Tools:

- API Documentation:
    - Swagger/OpenAPI:
        - Provides clear documentation for RESTful API endpoints to ease development and integration.
- AWS Toolkit for Visual Studio:
    - Integrates with Visual Studio to simplify the development and deployment of AWS services.
    - Features include:
        - **Resource Management:** Allows developers to interact with AWS resources directly from Visual Studio.
        - **CloudFormation Support:** Enables quick deployment and management of AWS infrastructure.
        - **Simplified Testing:** Provides tools to locally test AWS Lambda functions and other resources before deployment.

5.2 Plans for ensuring requirements traceability

Description:

A traceability matrix will be used to link all requirements to specific features, code modulus, test cases, and deliverables. Each requirement will have unique identifiers, ensuring full coverage and easy tracking. Tools like JIRA or Trello will be used for requirement management, and Git version control will log changes to requirements and code, ensuring traceability throughout the project lifecycle.

5.3 Plans for testing the software

Description:

A multi-layered testing approach will be adopted:

- Unit Testing: Verify individual components like LLaMA integration, Box API calls, and NLP tokenization workflows.
- Integration Testing: Ensure proper communication between the frontend (Node.js), backend (Python and Java), and Box API.
- System Testing: Simulate end-user scenarios, such as file uploads, document analysis, and report generation.
- Performance testing: Assess system behavior under high loads and concurrent API calls using tools like Apache JMeter.
- Regression Testing: Validate existing functionality after updates or bug fixes.

5.4 Engineering Trade-offs

Description:

Using Box API simplifies file management but introduces dependency on third-party services. LLaMA provides robust NLP capabilities but requires significant computational resources, potentially increasing costs. Node.js enables a fast, responsible frontend, but Python's backend logic introduces latency due to different runtime environments.

5.5 Coding Guidelines and Conventions

Description:

- Python
- JavaScript
- Naming Conventions
- Documentation

5.6 The Protocol of One or More Subsystems, Modulus, or Subroutines

Description:

- File Management Module: Handles Box API interactions for uploading, retrieving, and managing files.
- NLP Module: Processes documents using LLaMA for tokenization, summarization, and context analysis.
- Frontend API Module: Interacts with backend endpoints to display analysis results and manage user input.

5.7 The Choice of a Particular Algorithm or Programming Idiom (Or Design Pattern) to Implement Portions of the System's Functionality

Description:

- Algorithms: LLaMA is used for NLP tasks like summarization and keyword extraction.
- Design Patterns:

- Factory Pattern: Abstracts API client initialization for Box API and LLaMa,
- MVC (Model-View-Controller): Ensures separation of concerns between frontend backend logic, and data storage.

5.8 Interfaces for End-Users, Software, Hardware, and Communication

Description:

- End-Users: A web interface built with HTML, CSS, and Node.Js for file uploads and viewing analysis results.
- Software: Backend systems interact with the Box API for file storage and LLaMa for NLP.
- Hardware: Hosted on AWS, leveraging docker for consistent environments.
- Communications: Secure via HTTPS and OAuth 2.0 for authentication.

5.9 Hierarchical Organization for the Source Code

Description:

- Frontend Directory: Contains HTML, CSS, and JavaScript files with an organized structure for components and assets.
- Backend Directory: Includes python modules for API handling, NLP processing, and Box API integration.
- Config Directory: Stores environment variables, database configurations, and authentication keys.
- Tests Directory: Holds unit, integration, and system test scripts.

6. Detailed System Design

This section provides a detailed discussion of each system component, focusing on their roles, constraints, subcomponents, interactions, resources, and interfaces. Each component is described with reference to the architectural strategies defined in Section 4 of the SDD.

6.1 Frontend Module

6.1.1 Responsibilities

The Frontend Module provides an intuitive and responsive interface for users to interact with the LAPD1 system. Its main functionalities include uploading transcripts, viewing flagged results, and generating reports. It facilitates secure login, ensures role-based access control, and displays notifications for system events.

6.1.2 Constraints

- The module must support cross-browser compatibility (e.g., Chrome, Firefox, Edge).
- It must be responsive and accessible on desktops, tablets, and smartphones.
- Adherence to WCAG (Web Content Accessibility Guidelines) for accessibility.
- Limited performance when rendering large datasets in older browsers.

6.1.3 Composition

- **Login Component**: Handles secure authentication and displays error messages for failed login attempts.
- **Dashboard Component**: Displays user-specific data, including recent uploads, flagged results, and case summaries.
- **Upload Component**: Facilitates file uploads with progress indicators and metadata tagging.
- **Results Viewer**: Visualizes flagged transcript sections and enables user annotations.
- **Notification System**: Alerts users of processing status or flagged issues.

6.1.4 Uses/Interactions

- Communicates with the Backend Processing Module via REST APIs to send and retrieve data.
- Sends upload requests and metadata to the File Management Module.
- Displays results and flagged data fetched from the Data Processing Module.

6.1.5 Resources

- Uses JavaScript frameworks (Node.js, React.js or similar) for dynamic content rendering.

- Consumes API endpoints for file uploads, transcript analysis, and report generation.

6.1.6 Interface/Exports

- **APIs**: POST /upload, GET /results/:id, POST /annotate.
- **Outputs**: Rendered HTML/CSS elements and JSON responses.

6.2 Backend Processing Module

6.2.1 Responsibilities

Manages data flow between the frontend and backend services. Handles file validation, orchestration of NLP tasks, and integration with Box.com and Salesforce.

6.2.2 Constraints

- Must adhere to Box API limits for file uploads and retrievals.
- Requires efficient handling of simultaneous user requests to avoid bottlenecks.
- Ensures security with OAuth 2.0 authentication.

6.2.3 Composition

- **API Gateway**: Manages incoming API calls and routes them to appropriate services.
- **Validation Service**: Ensures uploaded files meet size and format requirements.
- **Processing Orchestrator**: Coordinates NLP tasks and data synchronization.

6.2.4 Uses/Interactions

- Interacts with the File Management Module for uploading and retrieving files.
- Calls the NLP Processing Module for transcript analysis and flagging.
- Updates case metadata in Salesforce via APIs.

6.2.5 Resources

- Requires access to Box.com for file storage.
- Utilizes AWS Lambda for scalable processing tasks.

6.2.6 Interface/Exports

- **APIs**: POST /process, GET /status/:id, POST /syncMetadata.
- **Outputs**: JSON responses for file validation and processing status.

6.3 NLP Processing Module

6.3.1 Responsibilities

Analyzes transcripts to extract entities, detect inconsistencies, and flag patterns of misconduct. Generates confidence scores for flagged sections to support human review.

6.3.2 Constraints

- Limited by the token capacity of LLaMA models.
- Accuracy depends on the quality of input transcripts and training data.

6.3.3 Composition

- **Entity Extractor**: Identifies officer names, badge numbers, and locations.
- **Pattern Recognizer**: Detects contradictions and recurring entities.
- **Confidence Scorer**: Assigns a confidence score to flagged sections.

6.3.4 Uses/Interactions

- Receives preprocessed text from the Backend Processing Module.
- Outputs flagged results and metadata to the Storage and Indexing Module.

6.3.5 Resources

- Relies on AWS SageMaker for scalable model inference.
- Consumes pre-trained LLaMA models for NLP tasks.

6.3.6 Interface/Exports

- **APIs**: POST /analyzeText, GET /flaggedSections/:id.
- **Outputs**: Flagged entities, confidence scores, and metadata.

6.4 Storage and Indexing Module

6.4.1 Responsibilities

Stores processed documents and metadata securely in Box.com. Maintains search indexes for efficient retrieval of flagged data.

6.4.2 Constraints

- Box.com API rate limits may affect bulk operations.
- Search operations must handle large datasets without performance degradation.

6.4.3 Composition

- **Storage Manager**: Handles file uploads and metadata tagging in Box.com.
- **Indexing Engine**: Creates search indexes in ElasticSearch.

6.4.4 Uses/Interactions

- Integrates with the Backend Processing Module for metadata synchronization.
- Provides indexed search results to the Frontend Module.

6.4.5 Resources

- Requires Box.com API access for file management.
- ElasticSearch for indexing and querying metadata.

6.4.6 Interface/Exports

- **APIs**: POST /storeFile, GET /searchIndex.
- **Outputs**: Metadata tags, indexed search results.

6.5 Integration Module

6.5.1 Responsibilities

Facilitates communication between LAPD1 and external systems, including Salesforce and Box.com. Manages metadata synchronization and error handling.

6.5.2 Constraints

- Must ensure secure data transfer via HTTPS.
- Requires robust error handling for transient API failures.

6.5.3 Composition

- **Metadata Sync Service**: Updates case details in Salesforce.
- **Error Handling Service**: Logs and retries failed API calls.

6.5.4 Uses/Interactions

- Synchronizes flagged data with Salesforce.
- Receives processed results from the Backend Processing Module.

6.5.5 Resources

- Salesforce API for case metadata updates.
- Logging systems (e.g., AWS CloudWatch) for error tracking.

6.5.6 Interface/Exports

- **APIs**: POST /updateMetadata, GET /syncStatus.
- **Outputs**: API call results, error logs.

7. Detailed Lower level Component Design

This section provides detailed descriptions of individual classes or files within the system. Each class or file represents a critical component of the LAPD1 software system and is organized to highlight its role, processing details, interfaces, and constraints.

7.1 NLP Processing Module

7.1.1 Classification

Type: Python module for Natural Language Processing (NLP).

7.1.2 Processing Narrative (PSPEC)

This module applies NLP models, such as LLaMA, to analyze transcripts for named entity recognition (NER) and misconduct detection. The primary functions include tokenization, entity extraction, and confidence score generation.

7.1.3 Interface Description

- **Inputs**: Raw text or scanned documents preprocessed via OCR.
- **Outputs**: Extracted entities (e.g., officer names, badge numbers), flagged sections with metadata, and confidence scores.

7.1.4 Processing Detail

7.1.4.1 Design Class Hierarchy

- **Parent Class**: BaseNLPProcessor (abstract class for NLP tasks).
- **Child Classes**: EntityExtractor, PatternRecognizer.

7.1.4.2 Restrictions/Limitations

- Limited to English-language transcripts.
- Dependent on training data quality for accuracy.

7.1.4.3 Performance Issues

- High computational resource usage for large transcripts.
- Latency during simultaneous multi-user operations.

7.1.4.4 Design Constraints

- Must integrate with AWS SageMaker for scalable inference.
- Constrained by token limits for large language models (LLMs).

7.1.4.5 Processing Detail for Each Operation

- **Tokenization**: Break down transcripts into sentences or words.
- **Entity Recognition**: Extract named entities (e.g., officer names, badge numbers).
- **Flagging**: Identify patterns or inconsistencies in testimonies.

7.2 File Management Module

7.2.1 Classification

Type: Node.js module for handling Box API integration.

7.2.2 Processing Narrative (PSPEC)

Manages uploading, retrieving, and storing case-related files in Box.com. Ensures files are tagged with metadata for search and organization.

7.2.3 Interface Description

- **Inputs**: File upload requests, metadata (e.g., case ID).
- **Outputs**: Confirmation of upload, Box file ID for storage reference.

7.2.4 Processing Detail

7.2.4.1 Design Class Hierarchy

- **Parent Class**: FileHandler (abstract class for file operations).
- **Child Classes**: BoxUploader, BoxRetriever.

7.2.4.2 Restrictions/Limitations

- Dependent on Box API availability.
- API rate limits may impact performance during bulk uploads.

7.2.4.3 Performance Issues

- Slow upload speeds for large files.
- Potential bottlenecks during simultaneous file operations.

7.2.4.4 Design Constraints

- Requires OAuth 2.0 authentication for secure API access.

- Must comply with Box API specifications for metadata tagging.

7.2.4.5 Processing Detail for Each Operation

- **File Upload**: Validate file type and size, upload to Box.
- **Metadata Tagging**: Associate metadata with each uploaded file.
- **File Retrieval**: Query Box for file details using unique IDs.

7.3 User Interface Logic

7.3.1 Classification

Type: JavaScript module for handling UI interactions.

7.3.2 Processing Narrative (PSPEC)

Responsible for managing user input, visualizing transcript analysis results, and interfacing with backend services for data retrieval and updates.

7.3.3 Interface Description

- **Inputs**: User actions (e.g., button clicks, file uploads).
- **Outputs**: Updated UI elements, notifications, or API calls to the backend.

7.3.4 Processing Detail

7.3.4.1 Design Class Hierarchy

- **Parent Class**: UIHandler.
- **Child Classes**: DashboardRenderer, UploadManager.

7.3.4.2 Restrictions/Limitations

- Limited to modern web browsers (e.g., Chrome, Firefox).
- Requires responsive design for mobile compatibility.

7.3.4.3 Performance Issues

- Delays in rendering large datasets or graphs.
- Compatibility issues with older browsers.

7.3.4.4 Design Constraints

- Adherence to accessibility standards (e.g., WCAG).
- Use of reusable React components for consistency.

7.3.4.5 Processing Detail for Each Operation

- **File Upload Interaction**: Trigger drag-and-drop or browse functionality.
- **Analysis Visualization**: Display flagged sections and entity highlights.
- **Notifications**: Provide real-time updates on processing status.

8. Database Design

Table 1: Cases

Column Name	Data Type	Description
case_id	INT (PK)	Unique identifier for the case
case_number	VARCHAR(50)	Case number assigned by the court
case_type	VARCHAR(50)	Type of case (e.g, civil, criminal)
status	VARCHAR(50)	Status of the case (e.g., open, closed)
hearing_data	DATE	Next scheduled hearing date
judge_name	VARCHAR(100)	Name of the judge overseeing the case
created_at	TIMESTAMP	Data and time the case was added
updated_at	TIMESTAMP	Last update to the case record.

Table 2 : Documents

Column Name	Data Type	Description
document_id	INT(PK)	Unique identifier for the document.
case_id	INT(FK)	Foreign key linking to cases
box_field_id	VARCHAR(255)	File ID from box.com for API access
document_type	VARCHAR(50)	Type document (eg., motion, evidence)
uploaded_by	VARCHAR(100)	User who uploaded the document
upload_date	TIMESTAMP	Date and time the document was uploaded
version	INT	Version of the document for tracking edit
is_flagged	BOOLEAN	indicates if the document is flagged

Table 3 : Flags

Column Name	Data Type	Description
flag_id	INT(PK)	Unique identifier for the flag
document_id	INT(PK)	Foreign key linking to document
flag_reason	TEXT	Reason the document was flagged
created_at	TIMESTAMP	Timestamp of when flag was added
resolve	Boolean	Indicates if the flag have been resolved

Table 4: Metadata

Column Name	Data Type	Description
meta_id	INT (PK)	Unique identifier for the flag
document_id	INT (FK)	Foreign key linking to documents
key	VARCHAR(50)	Metadata field name (e.g,.authors)
created_at	TIMESTAMP	Timestamp of when the meta was added
value	TEXT	Value of the metadata field

Table 5: Users

Column Name	Data Type	Description
user_id	INT (PK)	Unique identifier for the user
username	VARCHAR(100)	Username for the user
role	VARCHAR(50)	Role (e.g,.admin, viewer)
email	VARCHAR(100)	Email address for user

created_at	TIMESTAMP	Timestamp of user account creation

9. User Interface

9.1 Overview of User Interface

Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback Information that will be displayed for the user. This is an overview of the UI and its use. The user manual will contain extensive details about the actual use of the software.

System Functionality from the User's Perspective.

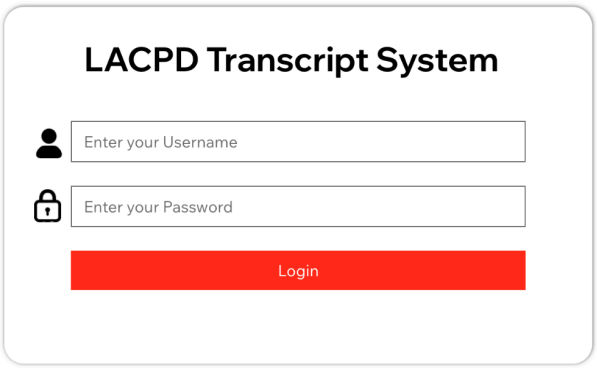
The LACPD Transcript Analysis System will allow users to:

- Login: Secure authentication ensures that only authorized users can access sensitive investigation data.
- Upload Transcripts: Users can upload various transcripts such as audio, text documents, and videos.
- Transcript Analysis: The system will analyze the uploaded transcript for important information such as names, locations, and dates while highlighting relations between patterns.
- Keyword and Entity Search: Users can search for keywords and tags to find relevant information.
- Annotation and Tagging: Users can annotate transcripts with comments or tag specific segments of the transcript for further investigation which could ultimately benefit patterns and relations.
- Pattern Recognition: The system will analyze the transcript for patterns like frequent mentions of events, names, locations, and dates. Visualization will be presented through graphs or summaries.
- Reports: Users can generate custom reports summarizing findings and identified keywords.
- Searchable Database: A searchable database consisting of uploaded transcripts for easy access to past investigations and annotations.
- Alerts: Notifications for key findings.

9.2 Screen Frameworks or Images

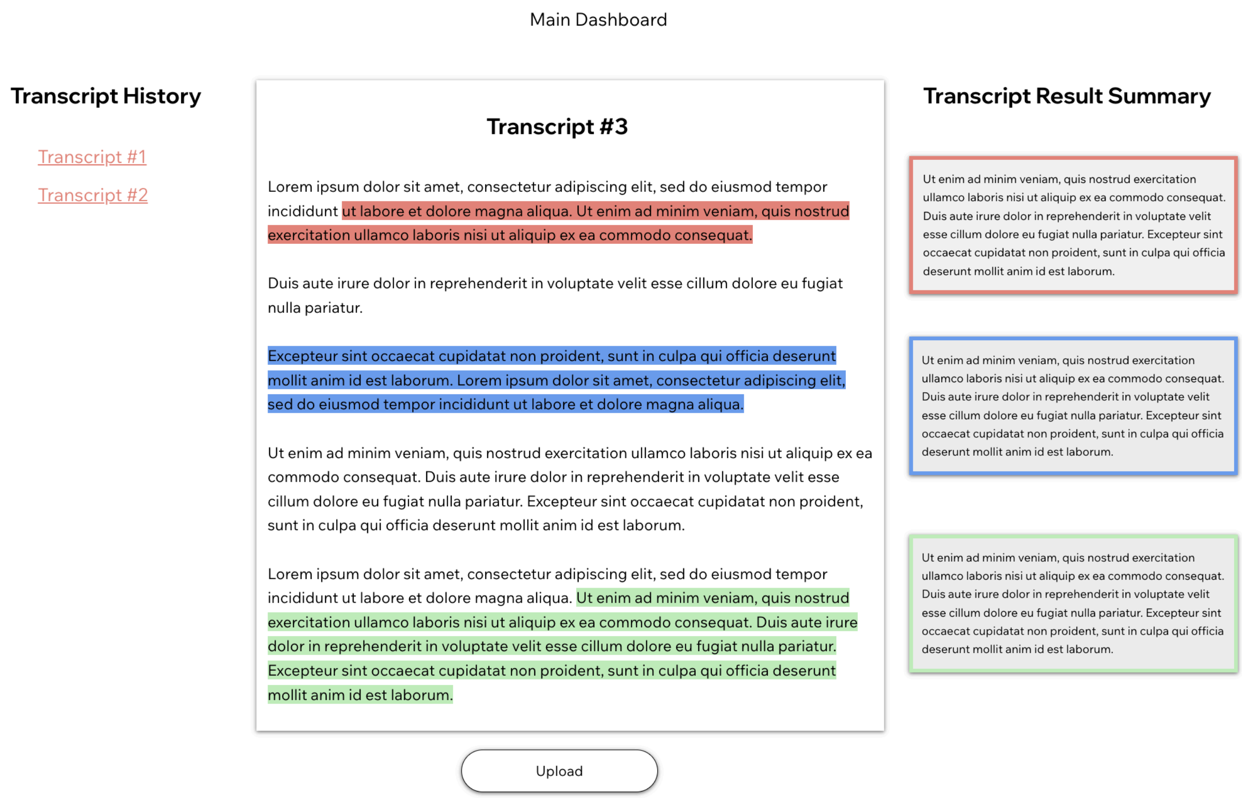
These can be mockups or screenshots of the various UI screens and popups.

Login Page



LACPD Login Page:

1. **Username Input Field:** A textbox is available for users to input their username. This field captures the user's unique identifier for login.
2. **Password Input Field:** A textbox is provided for users to input their password. This field ensures security by masking the input.
3. **Login Button**: A button labeled “Login” is present for users to submit their credentials. Upon clicking, the system verifies the provided username and password.
4. **Forgot Password Link**: A clickable link, “Forgot Password”, is provided for users who have forgotten their login credentials.
5. **Title and System Name**: The page is titled “LACPD Transcript System,” indicating its purpose and affiliation.
   Basic Design: A simple layout with a focus on usability, where the login form is central, and unnecessary distractions are minimal.



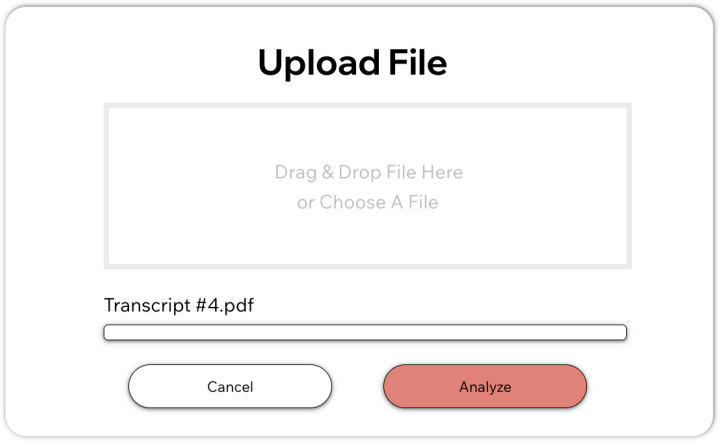
LACPD Main Dashboard:

1. **Transcript History Tab:**
    - **Purpose:** Displays a list of past transcripts that the user has uploaded and analyzed.
    - **Features**:
        - Scrollable list of transcript entries.
        - Clickable entries to view or interact with individual transcripts in detail.

2. **Transcript View and Upload Section:**
    - **Purpose:** Central area for viewing the details of a selected transcript or uploading a new transcript.
    - **Features:**
        - Displays the content or details of a selected transcript from history tab or uploaded.
        - Highlighted sections (with colors such as green, yellow, and red) indicating confidence level of inconsistency.
        - Upload Button: Allows the user to upload a new transcript for analyzing and reviewing.

3. **Transcript Results Summary Section**:
    - **Purpose**: Provides a visual summary of the transcript's results or analysis.
    - **Features**:
        - Colored boxes (green, yellow, red) summarize key inconsistencies.
        - A "Detailed Results" button that would display a Detailed analysis page.

Upload Screen



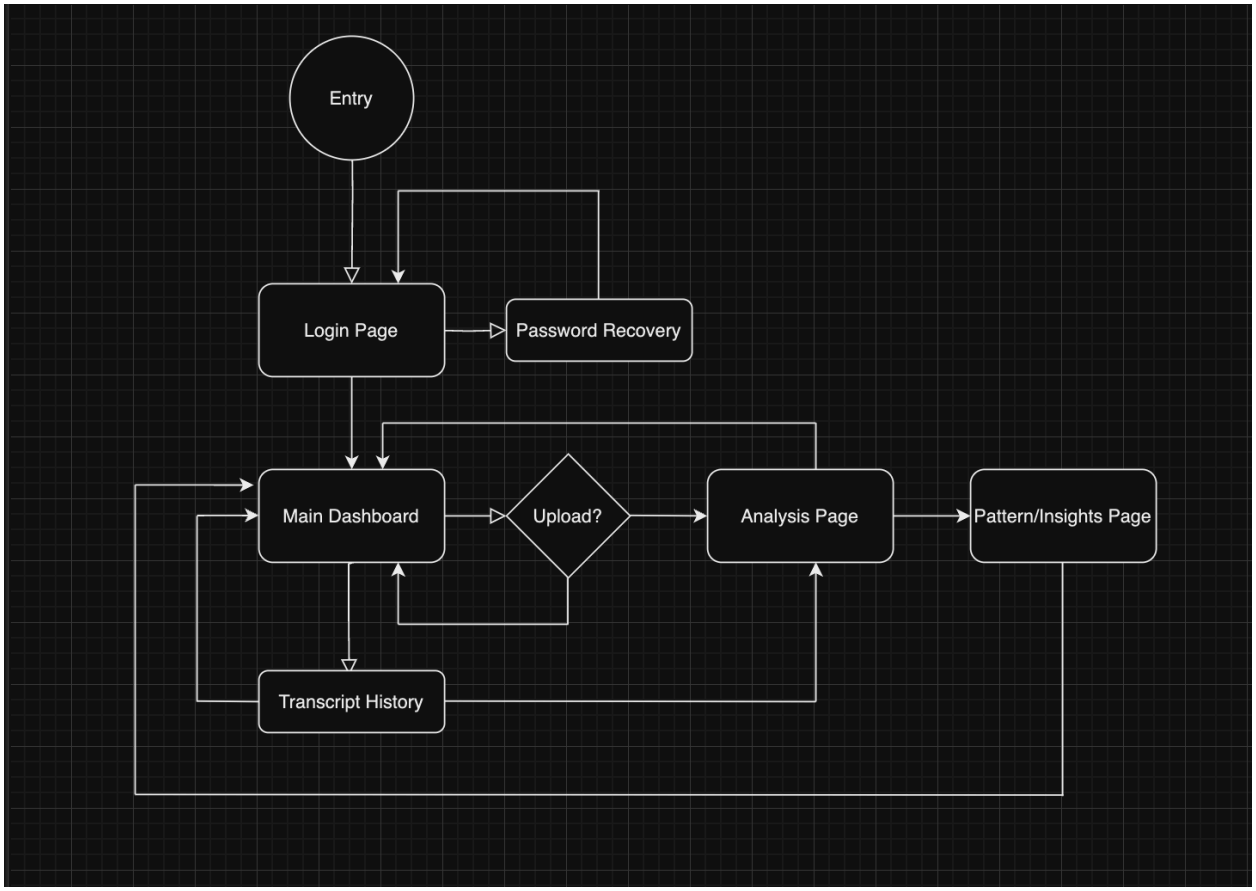
Upload Screen Functionalities:

1. **File Upload Area:**
    - **Drag & Drop Option:** User can drag and drop a file into the designated area to initiate the upload process.
    - **Choose File Option:** Alternatively, users can click on "Choose File" to browse and select a file from their device.
2. **Progress Indicator:**
    - **Progress Bar:** A visual progress bar displays the status of the file upload
3. **File Name Display:**
    - Displays the name of the file being upload for confirmation.
4. **Cancel Button**:
    - Allows users to cancel the upload process before completion and exit the upload screen.
5. **Analyze Button**:

- Once the file is uploaded, users can click the “Analyze” button to initiate the analysis process.

9.3 User Interface Flow Model

A discussion of screen objects and actions associated with those objects. This should include a flow diagram of the navigation between different pages.



10. Requirements Validation and Verification

This section outlines the process for ensuring that the system requirements specified in the SRS are implemented correctly and fully functional within the LAPD1 software system. Each requirement is mapped to corresponding validation and verification techniques.

10.1 Validation Process

The validation process ensures the system meets the needs and expectations of stakeholders. It involves:

- **Stakeholder Reviews**: Regular meetings with the LA County Public Defender's Office to gather feedback on functionality and user experience.
- **Use Case Testing**: Validation of user workflows, such as uploading documents, viewing flagged sections, and exporting reports.
- **Pilot Deployment**: A limited rollout of the system to a small group of attorneys and paralegals to gather real-world feedback.
- **Acceptance Testing**: Formal testing by stakeholders against acceptance criteria defined in the SRS.

10.2 Verification Process

The verification process ensures that the system functions as intended and meets all technical requirements.

Requirement ID	Requirement Description	Validation Method	Verification Method
FR-1	Allow users to upload court transcripts (PDF, DOCX).	User testing with different file formats.	Functional testing using test cases.
FR-2	Perform OCR on non-machine-readable documents.	Cross-validation of extracted text against original content.	Automated OCR unit tests.
FR-3	Detect and flag misconduct patterns in transcripts.	Comparison with manual review by legal experts.	Model evaluation using precision, recall, and F1 scores.
FR-4	Enable secure storage of documents in Box.com.	Verify file access via API after upload.	Penetration tests and security audits.

FR-5	Sync flagged data with Salesforce.	Mock case updates in Salesforce environment.	API integration tests.
NFR-1	Encrypt data at rest and in transit.	Review AWS encryption configurations.	Security compliance testing.
NFR-2	Process large amounts of documents efficiently.	Stress test system with large datasets.	Performance benchmarking.
NFR-3	Ensure role-based access control (RBAC).	Simulate role-specific actions in the UI.	Penetration testing for unauthorized access.

10.3 Testing Plan

- **Unit Testing**: All modules (e.g., OCR, NLP processing, API integrations) will undergo automated unit testing using frameworks such as PyTest.
- **Integration Testing**: Test interactions between modules, including file uploads, NLP analysis, storage, and integrations with Box.com and Salesforce.
- **System Testing**: Validate end-to-end workflows, from user login to document analysis and reporting.
- **Load Testing**: Evaluate system performance under varying workloads to ensure scalability and responsiveness.
- **Regression Testing**: Re-run test cases after updates to ensure no functionality is broken.

10.4 Documentation and Traceability

- **Test Cases and Scenarios**: Maintain a comprehensive list of test cases for each functional and non-functional requirement.
- **Traceability Matrix**:
  - Map requirements to test cases to ensure complete coverage.
  - Example:

Requirement ID	Test Case ID	Verification Status
FR-1	TC-001	Pass
FR-2	TC-002	Pass

FR-3	TC-003	Pending

- **Defect Tracking**: Use tools like Jira or Trello to log, track, and resolve bugs identified during testing.

10.5 Compliance Testing

Ensure that all legal and regulatory requirements (e.g., CJIS compliance, data encryption) are met through regular audits and testing against industry standards.

Summary

Section 10 ensures that all functional and non-functional requirements are validated and verified systematically through well-defined processes and tests. This guarantees the system meets stakeholder expectations and operates reliably in its intended environment.

11. Glossary

Here is a list of key terms used throughout the document, along with their explanations:

1. **CJIS (Criminal Justice Information Services)**
   - **Definition**: A division of the FBI that sets data security and privacy standards for systems handling sensitive law enforcement data.
   - **Relevance**: Ensures that systems dealing with sensitive legal and law enforcement data comply with encryption, access control, and audit requirements.
2. **Apache JMeter**
   - **Definition**: Apache JMeter is a performance testing tool designed to simulate a large number of users or systems making API calls to your application.
   - **Relevance**: A performance testing tool used to simulate high user loads and measure system behavior. It helps evaluate the responsiveness, stability, and scalability of APIs and applications under various conditions.
3. **API Calls (Application Programming Interface Call)**
   - **Definition**: An API Call is a request made by one system to another either get or send data.
   - **Relevance**: A request made by one application to another to send, retrieve, or process data. In this project, API calls handle operations like uploading files to Box, retrieving stored documents, and processing text data using NLP mod
4. **OCR (Optical Character Recognition)**
   - **Definition**: Technology that converts scanned images or non-editable digital documents into machine-readable text.
   - **Relevance**: Used to extract text from court transcripts, police reports, or other scanned documents for analysis.
5. **NLP (Natural Language Processing)**
   - **Definition**: A field of AI that focuses on enabling machines to understand, interpret, and generate human language.
   - **Relevance**: Used in this project to extract entities (e.g., names, badge numbers) and flag inconsistencies in transcripts.
6. **AWS (Amazon Web Services)**
   - **Definition**: A cloud computing platform offering various services, including storage, computation, and AI model deployment.
   - **Relevance**: Used to host the system's processing infrastructure, including NLP workflows, databases, and integration layers.
7. **EC2 (Elastic Compute Cloud)**
   - **Definition:** A scalable cloud computing service provided by AWS, which allows users to run virtual servers for processing tasks and hosting applications.

- **Relevance**: Used to process machine learning workflows, support transcript analysis, and handle scalable workloads.

8. **Llama**
   - **Definition**: An open-source Large Language Model (LLM) used for natural language understanding and generation.
   - **Relevance**: Employed for advanced NLP tasks, such as identifying inconsistencies or extracting meaningful insights from transcripts.
9. **Box.com**
   - **Definition**: A cloud-based content management platform for storing, managing, and sharing files securely.
   - **Relevance**: Used to store documents (e.g., court transcripts) and facilitate integration with the case management system.
10. **Salesforce**
    - **Definition**: A cloud-based platform used for customer relationship management (CRM) and case management.
    - **Relevance**: The Public Defender's case management system, integrated with the project for workflow and data updates.
11. **Entity Recognition**
    - **Definition**: A subfield of NLP that identifies and classifies entities such as names, locations, and dates in text.
    - **Relevance**: Used to extract important metadata from court transcripts (e.g., officer names, badge numbers, locations).
12. **Data Encryption**
    - **Definition**: The process of encoding data to protect it from unauthorized access.
    - **Relevance**: Ensures sensitive legal data is secure, as required by CJIS and other compliance standards.
13. **Tokenization**
    - **Definition**: The process of breaking text into smaller units (tokens) such as words or phrases for analysis.
    - **Relevance**: Used to prepare transcripts for NLP processing and reduce input sizes for large language models.
14. **Role-Based Access Control (RBAC)**
    - **Definition**: A security mechanism restricting access to system resources based on user roles.
    - **Relevance**: Ensures only authorized users can access sensitive data, following CJIS compliance.
15. **Misconduct Flagging**
    - **Definition**: The process of identifying and tagging inconsistencies, errors, or potential misconduct in legal documents.

- **Relevance**: Provides actionable insights for attorneys by highlighting areas of interest in transcripts or reports.

16. **ElasticSearch**
    - **Definition**: A distributed search engine used for full-text search, analysis, and storage.
    - **Relevance**: Can be integrated for indexing and retrieving key metadata from processed documents.
17. **Indexing**
    - **Definition**: The process of organizing and storing data in a way that enables fast retrieval.
    - **Relevance**: Used to organize processed transcripts and link them with metadata for efficient searching and flagging.
18. **Generative AI**
    - **Definition**: AI systems capable of generating content, such as text, based on training data and prompts.
    - **Relevance**: Supports advanced analysis, such as detecting patterns or summarizing transcripts.
19. **Case Management System (CMS)**
    - **Definition**: A platform used to organize, track, and manage legal case data and workflows.
    - **Relevance**: Acts as the core system for managing transcripts, flagged information, and workflow integration.
20. **Metadata**
    - **Definition**: Information that provides details about other data, such as author, creation date, or context.
    - **Relevance**: Extracted from transcripts (e.g., officer name, badge number) and used to organize and search files.

12. References

1. ACM Code of Ethics and Professional Conduct
    - **Description**: Ethical guidelines for computing professionals to ensure responsible decision-making and protect the public good.
    - **Link**: https://www.acm.org/code-of-ethics