**Detailed Sprint Backlog and Execution Plan**

**Sprint 1: (2.5-3 weeks)**

**Goal:** Implement user authentication and basic file upload functionality.

* **User Story:** As a user, I want to log in to the system so that I can securely access my data.
  + **Tasks:**
    1. **Design the User Authentication Interface:**
       - **Tools:** React.js, HTML, CSS.
       - **Execution:** Create login and registration forms with appropriate validation.
    2. **Implement Authentication Backend:**
       - **Tools:** Python, Flask/Django, JWT for authentication tokens.
       - **Execution:** Set up user model, create endpoints for registration, login, and logout, implement JWT authentication.
    3. **Integrate Authentication with Database:**
       - **Tools:** PostgreSQL/MySQL.
       - **Execution:** Set up database schema for user accounts, connect backend to the database, and ensure secure storage of passwords using hashing.
    4. **Test User Authentication:**
       - **Tools:** Postman for API testing, Selenium for end-to-end testing.
       - **Execution:** Write and execute test cases to validate the authentication flow.
* **User Story:** As a user, I want to upload various file types so that I can process different formats.
  + **Tasks:**
    1. **Develop File Upload Interface:**
       - **Tools:** React.js, HTML, CSS.
       - **Execution:** Create a drag-and-drop file upload component, add validation for file types and sizes.
    2. **Implement Backend for File Handling:**
       - **Tools:** Python, Flask/Django.
       - **Execution:** Set up endpoints for file upload, configure storage for uploaded files.
    3. **Validate File Upload:**
       - **Tools:** Python, libraries for file type validation.
       - **Execution:** Implement server-side validation for file types (PDF, DOCX, XLSX) and file sizes.
    4. **Test File Upload Functionality:**
       - **Tools:** Postman for API testing, Jest for React component testing.
       - **Execution:** Write and execute test cases to ensure file uploads work correctly.

**Sprint 2: (2.5-3 weeks)**

**Goal:** Develop and integrate content recognition algorithms.

* **User Story:** As a user, I want the tool to recognize and extract data from uploaded files so that I can analyze the contents.
  + **Tasks:**
    1. **Research and Select Libraries:**
       - **Tools:** OpenCV for computer vision, Tesseract for OCR, SpaCy for NLP.
       - **Execution:** Evaluate and choose the best libraries for content recognition.
    2. **Implement Initial Content Recognition Algorithms:**
       - **Tools:** Python, OpenCV, Tesseract, SpaCy.
       - **Execution:** Write code to process images and text from uploaded files, integrate OCR for text extraction.
    3. **Integrate Content Recognition into Backend:**
       - **Tools:** Python, Flask/Django.
       - **Execution:** Create endpoints for content recognition, connect algorithms to backend services.
    4. **Test Content Recognition:**
       - **Tools:** Sample files, unit testing frameworks (unittest, pytest).
       - **Execution:** Test algorithms with a variety of file types to ensure accurate recognition.

**Sprint 3: (2.5-3 weeks)**

**Goal:** Implement data extraction for PDFs and begin DOCX extraction.

* **User Story:** As a user, I want to extract data from PDF files so that I can analyze the information provided.
  + **Tasks:**
    1. **Develop PDF Data Extraction Logic:**
       - **Tools:** PyMuPDF, PDFMiner.
       - **Execution:** Write code to extract text and data from PDF files, handle different PDF structures.
    2. **Integrate PDF Extraction with Content Recognition Module:**
       - **Tools:** Python, Flask/Django.
       - **Execution:** Connect PDF extraction logic to existing content recognition endpoints.
    3. **Test PDF Data Extraction:**
       - **Tools:** Sample PDF files, unit testing frameworks.
       - **Execution:** Validate extraction accuracy with various PDF formats.
* **User Story:** As a user, I want to extract data from DOCX files so that I can analyze the information provided.
  + **Tasks:**
    1. **Develop DOCX Data Extraction Logic:**
       - **Tools:** python-docx.
       - **Execution:** Write code to extract text and data from DOCX files, handle different DOCX structures.
    2. **Integrate DOCX Extraction with Content Recognition Module:**
       - **Tools:** Python, Flask/Django.
       - **Execution:** Connect DOCX extraction logic to existing content recognition endpoints.
    3. **Test DOCX Data Extraction:**
       - **Tools:** Sample DOCX files, unit testing frameworks.
       - **Execution:** Validate extraction accuracy with various DOCX formats.

**Sprint 4: (2.5-3 weeks)**

**Goal:** Develop data summarization module.

* **User Story:** As a user, I want the tool to summarize extracted data so that I can easily understand the contents.
  + **Tasks:**
    1. **Design Data Summarization Interface:**
       - **Tools:** React.js, HTML, CSS.
       - **Execution:** Create a user-friendly interface for displaying summarized data.
    2. **Implement Summarization Logic:**
       - **Tools:** Python, Pandas, NumPy.
       - **Execution:** Write code to process extracted data and generate summaries.
    3. **Integrate Summarized Data Display with UI:**
       - **Tools:** React.js, Flask/Django.
       - **Execution:** Connect backend summarization logic to the frontend interface.
    4. **Test Data Summarization:**
       - **Tools:** Sample extracted data, unit testing frameworks.
       - **Execution:** Validate the accuracy and usability of data summaries.

**Sprint 5: (2.5-3 weeks)**

**Goal:** Create the interface for reviewing and editing extracted data.

* **User Story:** As a user, I want to review and edit extracted data so that I can ensure its accuracy.
  + **Tasks:**
    1. **Design Review and Edit Interface:**
       - **Tools:** React.js, HTML, CSS.
       - **Execution:** Create a user-friendly interface for reviewing and editing data.
    2. **Implement Functionality to Edit and Save Changes:**
       - **Tools:** React.js, Flask/Django.
       - **Execution:** Develop features to edit and save changes to the extracted data, ensure real-time updates.
    3. **Add Reject Functionality:**
       - **Tools:** React.js, Flask/Django.
       - **Execution:** Implement the ability to reject incorrect data and provide feedback.
    4. **Test Review and Edit Functionality:**
       - **Tools:** Sample extracted data, Jest for React component testing.
       - **Execution:** Validate the accuracy and usability of the review and edit interface.

**Sprint 6: (2.5-3 weeks)**

**Goal:** Enable exporting of data to CSV and JSON formats.

* **User Story:** As a user, I want to export processed data to various file types so that I can use it in other applications.
  + **Tasks:**
    1. **Implement Export Functionality for CSV and JSON:**
       - **Tools:** Python, Pandas.
       - **Execution:** Write code to export data in CSV and JSON formats, ensure data integrity.
    2. **Integrate Export Options in UI:**
       - **Tools:** React.js, Flask/Django.
       - **Execution:** Add export options in the frontend interface, connect to backend export logic.
    3. **Validate Exported Data:**
       - **Tools:** Sample data, CSV and JSON validators.
       - **Execution:** Ensure exported data is accurate and properly formatted.
    4. **Test Export Functionality:**
       - **Tools:** Sample data, unit testing frameworks.
       - **Execution:** Validate the export functionality with various data sets.

**Sprint 7: (2.5-3 weeks)**

**Goal:** Add error handling and validation for file uploads.

* **User Story:** As a user, I want to be notified of errors during file upload so that I can correct issues and proceed.
  + **Tasks:**
    1. **Implement Error Handling for File Uploads:**
       - **Tools:** Python, Flask/Django.
       - **Execution:** Write code to handle errors such as unsupported file types and file size limits.
    2. **Display Error Messages to Users:**
       - **Tools:** React.js.
       - **Execution:** Develop UI components to display error messages and guidance to users.
    3. **Validate Error Handling:**
       - **Tools:** Sample files, unit testing frameworks.
       - **Execution:** Test error handling with various file upload scenarios to ensure accuracy.
    4. **Test Error Handling Functionality:**
       - **Tools:** Postman for API testing, Jest for React component testing.
       - **Execution:** Validate the error handling flow from the frontend to the backend.

**Sprint 8: (2.5-3 weeks)**

**Goal:** Integrate the tool with existing firm systems.

* **User Story:** As a user, I want the tool to integrate with existing systems so that I can seamlessly use the data.
  + **Tasks:**
    1. **Identify Necessary Integrations:**
       - **Tools:** System documentation, API specifications.
       - **Execution:** Determine the required integrations with firm systems and databases.
    2. **Develop and Test Integration APIs:**
       - **Tools:** Python, Flask/Django.
       - **Execution:** Write and test APIs to facilitate data exchange between the tool and existing systems.
    3. **Ensure Data Flow Between Systems:**
       - **Tools:** Data mapping and transformation tools.
       - **Execution:** Validate data consistency and flow between the tool and integrated systems.
    4. **Validate Integration with Real Data:**
       - **Tools:** Sample and real data sets.
       - **Execution:** Test integration with actual data to ensure seamless operation.

**Sprint 9: (2.5-3 weeks)**

**Goal:** Optimize performance for handling large datasets.

* **User Story:** As a user, I want the tool to perform efficiently with large datasets so that I can work without delays.
  + **Tasks:**
    1. **Profile the Application:**
       - **Tools:** Profiling tools (cProfile for Python, React Profiler).
       - **Execution:** Identify performance bottlenecks in both the frontend and backend.
    2. **Optimize Data Processing Algorithms:**
       - **Tools:** Python, Pandas, NumPy.
       - **Execution:** Refactor and optimize algorithms for better performance.
    3. **Test Performance with Large Datasets:**
       - **Tools:** Sample large datasets, performance testing tools.
       - **Execution:** Conduct performance testing to validate optimizations.
    4. **Implement Caching and Other Improvements:**
       - **Tools:** Redis/Memcached for caching.
       - **Execution:** Add caching mechanisms and other optimizations to improve performance.

**Sprint 10: (2.5-3 weeks)**

**Goal:** Refine the user interface for a consistent and intuitive experience.

* **User Story:** As a user, I want the interface to be user-friendly so that I can easily use the tool.
  + **Tasks:**
    1. **Conduct User Interface Testing:**
       - **Tools:** Usability testing tools, user feedback sessions.
       - **Execution:** Gather feedback from users on the UI design and functionality.
    2. **Refine React.js Components:**
       - **Tools:** React.js, HTML, CSS.
       - **Execution:** Improve the design and consistency of React.js components based on feedback.
    3. **Implement UI Improvements:**
       - **Tools:** React.js, HTML, CSS.
       - **Execution:** Apply changes to the UI to enhance usability and consistency.
    4. **Test the Interface with Various Scenarios:**
       - **Tools:** Sample data, usability testing tools.
       - **Execution:** Validate the improved interface with different user scenarios to ensure it meets user needs.

**Sprint 11: (2.5-3 weeks)**

**Goal:** Finalize backend implementation and ensure seamless integration with the front-end.

* **User Story:** As a user, I want the backend to be robust and reliable so that the tool works smoothly.
  + **Tasks:**
    1. **Finalize Backend Logic:**
       - **Tools:** Python, Flask/Django.
       - **Execution:** Complete the implementation of all backend features and ensure robustness.
    2. **Ensure Integration with Front-end:**
       - **Tools:** React.js, Flask/Django.
       - **Execution:** Verify that all backend APIs are correctly integrated with the frontend.
    3. **Conduct Thorough Testing of Backend APIs:**
       - **Tools:** Postman, unit testing frameworks.
       - **Execution:** Write and execute comprehensive test cases for all backend APIs.
    4. **Fix Any Identified Issues:**
       - **Tools:** Python, Flask/Django.
       - **Execution:** Address any bugs or issues found during testing.

**Sprint 12: (2.5-3 weeks)**

**Goal:** Conduct thorough testing and finalize the tool for deployment.

* **User Story:** As a user, I want the tool to be thoroughly tested so that it is reliable and bug-free.
  + **Tasks:**
    1. **Conduct Unit Tests and Integration Tests:**
       - **Tools:** Unittest, PyTest, Selenium.
       - **Execution:** Write and run unit tests and integration tests for the entire application.
    2. **Perform User Acceptance Testing:**
       - **Tools:** UAT sessions with end users.
       - **Execution:** Gather feedback from users and validate that the tool meets their needs.
    3. **Address Any Bugs or Issues:**
       - **Tools:** Bug tracking tools (JIRA, Bugzilla).
       - **Execution:** Fix any bugs or issues identified during testing.
    4. **Prepare Deployment Documentation:**
       - **Tools:** Documentation tools (Confluence, Markdown).
       - **Execution:** Document the deployment process and user guides.
    5. **Deploy the Tool to Production:**
       - **Tools:** CI/CD tools (Jenkins, GitLab CI).
       - **Execution:** Deploy the final version of the tool to the production environment and conduct final validation.

**Total Project Duration:**

* Each Sprint: 2.5 to 3 weeks
* Total Sprints: 12
* Total Duration: 30 to 36 weeks (approximately 7.5 to 9 months)