System Test Document

Student 1 – Lorena Gomez – 21734359

Student 2 – Darragh Manning - 21506373

Supervisor - Tomas Ward

Project Name – TimelineXtract

Last modified on: 01/05/2025

Abstract - TimelineXtract is a system designed to automate the extraction of questionnaire schedules from complex clinical trial protocols. Clinical trial PDFs are often lengthy, inconsistently structured, and difficult to navigate, making manual data extraction slow and error prone. Our solution combines Adobe PDF Extract API for precise document structure recognition, GPT-4 for semantic data extraction, Django and React.js for processing and presentation, and MongoDB for secure storage.

Through extensive prompt engineering, vector similarity matching, and real world testing using protocols from ClinicalTrials.gov, TimelineXtract delivers clean, structured outputs that streamline clinical research workflows.

Table of Contents

Introduction

This document outlines the testing strategy, techniques, and results used to validate the quality and robustness of the *TimelineXtract* system. Given the nature of the project, a web-based application for extracting structured data from complex clinical trial PDFs, we employed a combination of **unit testing**, **integration testing**, **system testing**, **ad hoc testing, and formal testing** to ensure correctness, reliability, and usability.

Testing Strategy

Testing Type	Purpose
Unit Testing	Ensure individual components (e.g., database insertion, file handling) are correct
Integration Testing	Confirm end to end flows work correctly across components
System Testing	Validate that complete user workflows function as intended
Ad Hoc Testing	Real world manual testing to catch unexpected issues during development
Similarity Evaluation Testing	Quantitatively evaluate output accuracy against a manually created ground truth

Testing Tools and Frameworks

Tool	Purpose
Python unittest	Unit and integration testing
Postman	API endpoint testing
Django Test Client	Backend testing automation
Sentence Transformers	Semantic similarity scoring
React.js Manual Validation	Frontend UI manual testing

Unit Testing

Key Components Tested

- **PDF Extraction Utility:**
  extract_and_classify_tables() tested with simple and complex PDFs.

→ Example code (src/timelinextract/srcExtractor/tests/test_pdf_processing.py):

```
class PDFProcessingTests(TestCase):

    @patch("os.makedirs")
    @patch("builtins.open", new_callable=mock_open, read_data=b"fake pdf content")
    @patch("srcExtractor.utils.pdf_processing.extract_tables",
return_value={"success": True})
    @patch("srcExtractor.utils.pdf_processing.classify_all_tables_in_folder",
return_value={"success": ["table1.csv", "table2.csv"]})
    @patch("srcExtractor.utils.data_processing.convert_valid_files_to_json")
    def test_extract_and_classify_tables_success(self, mock_convert_json,
mock_classify, mock_extract, mock_file, mock_mkdir):
        """Test successful extraction and classification of tables from a PDF."""

        result = extract_and_classify_tables("sample.pdf", "sample")

        self.assertIn("valid_files", result)
        self.assertEqual(result["valid_files"], ["table1.csv", "table2.csv"])

    @patch("builtins.open", new_callable=mock_open, read_data=b"fake pdf content")
    @patch("srcExtractor.utils.pdf_processing.extract_tables",
return_value={"error": "Extraction failed"})
    def test_extract_and_classify_tables_extraction_error(self, mock_extract,
mock_file):
        """Test handling of errors during table extraction."""
        result = extract_and_classify_tables("sample.pdf", "sample")
        self.assertIn("error", result)
        self.assertEqual(result["error"], "Extraction failed")

    @patch("builtins.open", new_callable=mock_open, read_data=b"fake pdf content")
    @patch("srcExtractor.utils.pdf_processing.extract_tables",
return_value={"success": True})
    @patch(
        "srcExtractor.utils.pdf_processing.classify_all_tables_in_folder",
        return_value={"error": "No Schedule of Events table found in the protocol."}
    )
    def test_extract_and_classify_tables_no_valid_files(self, mock_classify,
mock_extract, mock_file):
        """Test when no valid tables are found after classification."""
        result = extract_and_classify_tables("sample.pdf", "sample")
        self.assertIn("error", result)
        self.assertEqual(result["error"], "No Schedule of Events table found in the
protocol.")

    @patch("builtins.open", new_callable=mock_open, read_data=b"fake pdf content")
    @patch("srcExtractor.utils.pdf_processing.extract_tables",
return_value={"success": True})
    @patch("srcExtractor.utils.pdf_processing.classify_all_tables_in_folder",
side_effect=Exception("Unexpected error"))
```

- **PDF Processing and Validation:**
  handle_pdf_upload() tested for normal, missing, and invalid API responses.

  → Example code (src/timelinextract/srcExtractor/tests/test_pdf_validation.py):

```
class PDFUploadTests(TestCase):

   @patch("os.path.getsize", return_value=1024)
   @patch("srcExtractor.utils.pdf_validation.extract_text_from_pdf",
return_value={"text": "Sample text from PDF"})
   def test_handle_pdf_upload_success(self, mock_extract_text,
mock_getsize):
       """Test successful PDF file upload and text extraction."""

       result = handle_pdf_upload("test.pdf")

       self.assertIn("success", result)
       self.assertEqual(result["success"], "test.pdf")

   def test_handle_pdf_upload_no_file(self):
       """Test when no PDF file is provided."""
       result = handle_pdf_upload("")
       self.assertIn("error", result)
       self.assertEqual(result["error"], "No PDF file in data.")

   @patch("os.path.getsize", return_value=1024)
   def test_handle_pdf_upload_invalid_file_type(self, mock_getsize):
       """Test when a non-PDF file is uploaded."""
       result = handle_pdf_upload("test.txt")
       self.assertIn("error", result)
       self.assertEqual(result["error"], "Only PDF files are allowed.")

   @patch("os.path.getsize", return_value=0)
   def test_handle_pdf_upload_empty_pdf(self, mock_getsize):
       """Test when an empty PDF file is uploaded."""
       result = handle_pdf_upload("empty.pdf")
       self.assertIn("error", result)
       self.assertEqual(result["error"], "Uploaded PDF file is empty.")

   @patch("os.path.getsize", side_effect=Exception("File not found"))
   def test_handle_pdf_upload_file_not_found(self, mock_getsize):
       """Test when the provided PDF file does not exist."""
       result = handle_pdf_upload("non_existent.pdf")
       self.assertIn("error", result)
       self.assertEqual(result["error"], "Error validating the file: File
not found")
```

- **MongoDB Database Operations:**
  Functions like add_user(), add_pdf() tested for edge cases (e.g., missing fields, duplicate entries).

  → Example code (src/timelinextract/srcExtractor/tests/test_user_mongodb.py):

```
class UserMongoDBTests(TestCase):

   def test_is_valid_email(self):
       self.assertTrue(user.is_valid_email("user@example.com"))
       self.assertTrue(user.is_valid_email("user.name+tag@example.co.uk"))
       self.assertFalse(user.is_valid_email("not-an-email"))
       self.assertFalse(user.is_valid_email("missing@domain"))
       self.assertFalse(user.is_valid_email("user@.com"))

   def test_add_user_missing_email(self):
       """Should return error if email is missing"""
       result = user.add_user({})
       self.assertIn("error", result)
       self.assertEqual(result["error"], "Missing required field: email.")

   def test_add_user_invalid_email(self):
       """Should return error if email format is invalid"""
       result = user.add_user({"email": "invalid-email"})
       self.assertIn("error", result)
       self.assertEqual(result["error"], "Invalid email format.")

   @patch("srcExtractor.services.mongodb.user.mongodb.get_collection")
   def test_add_user_already_exists(self, mock_get_collection):
       """Should detect existing user and return message"""
       mock_collection = MagicMock()
       mock_collection.find_one.return_value = {"_id": "abc123", "email":
"user@example.com"}
       mock_get_collection.return_value = mock_collection
       result = user.add_user({"email": "user@example.com"})
       self.assertIn("message", result)
       self.assertEqual(result["message"], "User 'user@example.com' already
exists.")
       self.assertEqual(result["user_id"], "abc123")

   @patch("srcExtractor.services.mongodb.user.mongodb.get_collection")
   def test_add_user_success(self, mock_get_collection):
       """Should create new user if not found"""
       mock_collection = MagicMock()
       mock_collection.find_one.return_value = None
       mock_collection.insert_one.return_value.inserted_id = "newid456"
       mock_get_collection.return_value = mock_collection
       result = user.add_user({"email": "newuser@example.com"})
       self.assertIn("message", result)
       self.assertEqual(result["message"], "User 'newuser@example.com' created
successfully.")
```

```
        self.assertEqual(result["user_id"], "newid456")
```

Integration Testing

Key End to End Tests:

- Backend:
  - Uploading a PDF → GPT data extraction → Adobe API table extraction → MongoDB storage → Display on frontend.

- Frontend:
  - Authentication → Token validation → File upload → Results presentation.

Result: All integration points functioned correctly. Minor JSON validation bugs were fixed during development.

System Testing

Core System Test Cases:

Test Case	Description	Expected Outcome	Status
TC-001	Upload valid clinical protocol	JSON extracted and displayed	Passed
TC-002	Upload non-PDF file	Error message shown	Passed
TC-003	Login with non-DCU email	Access denied	Passed
TC-004	View extracted timeline	Correct UI rendering	Passed

Example System Test:

Input: Clinical trial PDF with multiple questionnaire schedules.
Action: Upload through the web interface.
Expected Output: Structured JSON with questionnaire names, types, and schedules.
Actual Output: As expected, extracted questionnaires were correctly structured.

Ad Hoc Testing

→ Uploaded incomplete, corrupted, and oddly formatted PDFs.

→ Manually checked outputs.

→ Tested UI under different browsers and screen sizes.

Similarity Evaluation & Metrics

One major part of the validation was quantitative testing against a manually built ground truth:

- We manually extracted questionnaires, types, and timelines from **50 real clinical protocols** from ClinicalTrials.gov.
- Used a custom semantic similarity script to compare predictions vs ground truth.
- Metrics computed:
    - **Precision** = True Positives / (True Positives + False Positives)
    - **Recall =** True Positives / (True Positives + False Negatives)
    - **F1-Score** = 2 x (precision x recall) / (precision + recall)

Result:

Overall Metric	Avg Precision	Avg Recall	Avg F1-Score
Questionnaire Evaluation	72.60%	76.12%	72.78%
Timepoints Evaluation	62.80%	61.88%	57.21%

User Validation Testing

User Access Tests:

Test Case	Description	Expected Outcome	Status
UV-001	Login with DCU email (@mail.dcu.ie)	Successful login	Passed
UV-002	Login with non-DCU email	Rejected with error message	Passed
UV-003	Missing email field in token	Error response from backend	Passed
UV-004	Re-login existing user	No duplicate entry in database	Passed

UV-005	Email faking attempt	Caught and rejected	Passed

Robustness and Error Handling Testing

Test Case	Description	Expected Outcome	Status
RE-001	Upload corrupted PDF	Error handled gracefully	Passed
RE-002	Adobe API timeout	Retry or error displayed	Passed
RE-003	GPT-4 API failure	Informative error message shown	Passed
RE-004	MongoDB connection loss	Proper error reporting/logging	Passed

Data Integrity Testing

Test Case	Description	Expected Outcome	Status
DI-001	User uploads multiple PDFs	PDFs linked correctly to users	Passed
DI-002	Same file uploaded twice	Unique filenames ensured	Passed
DI-003	MongoDB linking across collections	Users, PDFs, Queries, and Outputs correctly linked	Passed

Testing Results Summary

Category	Outcome
Upload functionality	100% Passed
Authentication	100% Passed
Data extraction accuracy	Medium (Precision/Recall > 60%)
Data integrity	No missing links detected
UI usability	Smooth under real usage

Error handling	System stable under faults

Future Testing Improvements

To further validate and strengthen the performance of TimelineXtract, we propose the following future testing initiatives:

→ **Expand Ground Truth Dataset:** Extend the manually constructed evaluation set from 50 to 100+ protocols, including more diverse therapeutic areas and formats. This will improve statistical reliability and help identify edge cases.

→ **High-Load Simulations:** Conduct stress testing with multiple users uploading large PDF files simultaneously to evaluate system stability, performance under load, and scalability of backend services.

→ **Automated Frontend Testing:** Integrate tools like **Cypress** or **Selenium** to automate UI regression testing, especially around file uploads, user authentication, and result rendering.

Limitations and Accuracy Challenges

Despite achieving decent performance, our system's accuracy and recall on timepoints extraction is not perfect due to several technical challenges:

→ **Unpredictability of AI Output:** GPT-4, while powerful, is inherently non-deterministic. Achieving 100% precision and recall requires extensive prompt tuning and context control, which is possible but time consuming. Additional research into few-shot or retrieval-augmented prompting could help.

→ **Complexity of Table Formats:** The table extraction process is the most error prone stage due to:

- **Merged cells**, which confuse the row/column alignment.
- **Vertical or rotated text**, which Adobe's Extract API struggles to interpret.
- **Multipart or broken tables**, where questionnaire procedures are split across multiple disconnected tables.

Tables with vertically written procedures or nested structures often led to empty or misaligned timepoints being extracted. Example:

Study Visit (Month)	Screening	Baseline		Month 1		Month 2	Month 3	Month 4	Month 5	Month 6	Month 7	Month 8	Month 9	AT F/U[b,c]
		Treatment Period												
		Onset Period		Efficacy Period										
												EOT	EOS/ ET[a,b]	
Study Day (±Visit Window)	-60 to -1	Day 1	15 ±3	29 ±7	43 ±3	57 ±7	85 ±7	113 ±7	141 ±7	169 ±7	197±7	225±7	253 ±7	281 ±7
Thrombophilia Screening	X													
Serum Chemistry[o]	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Liver Function Tests[p]	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Anti-Drug Antibodies[k,n,q]	X	X		X			X					X	X[m]	
Hematology[o]	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Hepatic Tests[p]	X													
Exploratory Biomarkers[k]		X					X						X	
Exploratory Circulating RNA (Optional)[k]		X					X						X	
Exploratory DNA Sample (Optional)[k]		X												
Plasma PK[k,r]		X		X			X				X	X		
Urinalysis	X	X											X	
Urine Collection for Biomarkers[k]		X					X						X	
HJHS[s]		X											X[s]	
Patient Resource Use[s]		X											X[s]	
EQ-5D[s]		X											X[s]	

This is another example of a *Schedule of Events* table where the structure is hard to read even for a human:

Table 2. Schedule of Events: Double-blind Phase (Continued)

Study Procedures	Screening Visit[q]	Run-in Period[q]	Double-blind Treatment Period[a]						Double-blind End of Treatment[b]/ Early Termination	Discontinuation Period
	Day -28 to Day -7	Day -7 to Day 1	Week 1			Week 2 to 12			First Day of Week 13	DP Days 2[b] -14
Visit Days →			M/Tu	W/Th	F/Sa	M/Tu	W/Th	F/Sa		
	-28 to -7	-7 to 1	1	3	5	8	10	12	85	85 to 98
						15	17	19		
						22	24	26		
						29[m]	31	33		
						36	38	40		
						43	45	47		
						50	52	54		
						57[m]	59	61		
						64	66	68		
						71[m]	73	75		
						78	80	82		
Safety and efficacy evaluations										
Patient training on PRO worksheets		X[h,i,j]	X[j]						X[i]	
Worst Itching Intensity NRS (daily)[k]		X	Record on an ongoing basis						X	X[l]
Skindex-10 Scale, 5-D Itch Scale[m]			X			X[m]			X[m]	
Patient Global Impression of Change									X	
Patient Health Assessment (ShOWS)[k] and Observer Health Assessment (OOWS) worksheets									X[n]	X[n]
Record dose of ESA and IV iron	X		Record on an ongoing basis						X	
Record number of missed dialysis visits and reason(s)			Record on an ongoing basis							
IV administration of study drug			Record on an ongoing basis							
Inflammatory biomarker samples[o]			X						X	
Adverse event monitoring	X	X	Record on an ongoing basis						X	X
Concomitant medications (including antipruritic medications)[p]			X	Record on an ongoing basis					X	X
Structured Safety Evaluation[r]		X		X			X			X

DP = Discontinuation Period; ESA = erythropoiesis-stimulating agent; F = Friday; IV = intravenous; M = Monday; NRS = numerical rating scale; OOWS = Objective Opiate Withdrawal Scale; PRO = patient-reported outcome; Sa = Saturday; ShOWS = Short Opiate Withdrawal Scale; Th = Thursday; Tu = Tuesday; W = Wednesday

→ Text-based Timelines Not Counted in Metrics:

Our evaluation only measured the time points extracted from tables. However, in some protocols, the correct schedule was found by GPT-4 in the narrative text and not in a table.

Example:


```
{
    "longName": "Godin Leisure-Time Exercise Questionnaire",
    "shortName": "GLTEQ",
    "type": "PRO",
    "questionnaireSchedule": "Baseline and at end of study visit",
```

"questionnaireTiming": [
"Questionnaires (Informed Consent)(Procedures)"
]
}

In this case, the system correctly extracted the schedule from the narrative text (“questionnaireSchedule”), but the associated table (“questionnaireSchedule”) failed to give usable timepoints. This discrepancy lowers the recall score, even though the system ultimately produced accurate information.

How to Improve Accuracy and Precision

To improve accuracy in future versions of *TimelineXtract,* if we had more time, we could:

→ **Table Extraction Upgrades:**

- Use layout aware models like **LayoutLMv3** or **PubLayNet** trained object detectors for better understanding of complex table structures.

- Implement post processing logic to identify and normalise rotated or merged cell content.

→ **Enhanced Prompt Engineering:**

- Introduce dynamic prompt templates that adapt based on protocol structure and table contents.

→ **Domain Specific Synonym Matching:**

- Add a medical synonym dictionary to better match varied questionnaire names (e.g., “EuroQol 5D” vs. “EQ-5D”).

→ **Feedback Loop for Active Learning:**

- Use clinician or user feedback to retrain or refine matching heuristics based on validated matches.

Conclusion

Testing was a very important part of *TimelineXtract’s* development. Through a combination of structured unit testing, integration testing, comprehensive system testing, semantic output evaluation, and ad hoc exploratory validation, we built a reliable and robust system capable of extracting high quality clinical trial data from complex, real world documents.

We validated not only the correctness of individual components (such as file parsing and API integration) but also the end to end behavior of the full pipeline, from upload to AI processing to final output display. Using a manually curated ground truth dataset of 50 clinical protocols, we implemented semantic similarity scoring with sentence transformers to evaluate precision, recall, and F1 score, giving us measurable insights into the effectiveness of our AI extraction pipeline.

In parallel, we tested system robustness with malformed files, edge cases in user login, and challenging PDF layouts. These helped us detect and mitigate potential failures early. Our authentication system, database structure, and UI/UX flow were also tested and refined with the user in mind, ensuring security, traceability, and clarity throughout the workflow.

While some components such as table extraction and timepoint recognition remain challenging due to the variability of clinical documents, we identified clear paths for improving accuracy through better data fusion, smarter models, and broader validation sets.

Ultimately, our testing approach gave us strong confidence that *TimelineXtrac*t is not just functional, but reliable and secure. Future enhancements to testing (including load simulations, expanded ground truths, and automated UI validation) will allow the system to scale and maintain its performance across more demanding use cases.