

Test Plan

Team: Adrianna Sauro, Bhavik Patel, Michael Wahoski, Nick Blaisdell

Faculty Advisor: Dr. King

Date: 9/8/26

1. Introduction

This Software Test Plan covers the CRESCENT Social Guardian project. This will include the following systems: extension, adapter, text-preprocessing pipeline, fine-tuned BERT detection model, classification and flagging logic, interpretability features, administrative functions, and web-based moderator triage interface. The test effort provides evidence that CRESCENT can reliably process authorized chat text, identify conversations that may warrant human review, and display clear results to authorized users. CRESCENT uses an iterative development and testing approach. Testing proceeds from individual components to integrated data flow and interfaces, to the complete triage workflow, and then to acceptance testing by intended users.

Testing will verify end-to-end functionality by verifying correct routing from extension to web app. CRESCENT is a decision-support system: flags are alerts for human review, not proof of harmful intent or automatic triggers for discipline, reporting, or account action. The effort excludes legal or criminal determinations, autonomous safety decisions or enforcement actions, validation of the internal design of third-party pretrained-model frameworks, enterprise-scale load testing, disaster recovery, and full penetration testing.

1.5: Notation Description

FR- Functional Requirements

A1- Adapter case number 1

E1- Extension 1 test

EL1- Explainability Layer

T1- Transformer 1test

W1- WebApp 1

Example: FR-E1- Browser Compatibility

FR-E2- Dynamic Chat Monitoring

Extension:

FR-E1 – Browser Compatibility

2.1 Test Case Identifier: FR-E1

2.2 Objective

Verify that the browser extension can be installed, enabled, and executed in the supported browser environment.

The test shall confirm that:

1. The extension installs successfully.
2. The extension loads on supported chat webpages.
3. Required permissions are available.
4. The extension does not interfere with unsupported webpages.

Priority: High

2.3 Inputs

Test inputs shall include:

- Supported Chromium-based browser
- Extension package or development build
- Supported test chat webpage
- Unsupported or unrelated webpage

Example supported browser:

Google Chrome

2.4 Outcomes

Expected outcomes:

- Extension installs without error.
- Extension is enabled successfully.
- Content script executes on the supported webpage.
- Required webpage content is accessible according to configured permissions.
- Unsupported webpages do not cause extension failure.

2.5 Environmental Needs

Hardware:

- Internet required

Software:

- Google Chrome or other approved Chromium-based browser
- Current extension build
- Supported test chat webpage

- Browser developer tools

2.6 Special Procedural Requirements

Preconditions:

- Extension development mode is enabled if required.
- Extension manifest is correctly configured.

Postconditions:

- Confirm extension remains enabled.
- Confirm no browser errors were generated.
- Remove or reset test extension configuration if required.

2.7 Intercase Dependencies

FR-E1 shall be completed before executing the remaining extension test cases.

FR-E2 – Dynamic Chat Monitoring

2.1 Test Case Identifier: FR-E2

2.2 Objective

Verify that the extension detects new chat messages as they are added to the webpage.

The test shall confirm that newly rendered messages are identified without requiring manual refresh or reload.

Priority: High

2.3 Inputs

A supported test chat page shall be used.

During testing:

1. Load an initial conversation.
2. Add a new message dynamically.
3. Add multiple messages sequentially.
4. Remove unrelated information from website.

2.4 Outcomes

Expected outcomes:

- Newly added chat messages are detected.
- Multiple new messages are detected individually.
- Detection occurs without page refresh.
- Unrelated webpage changes are not incorrectly processed as chat messages.

2.5 Environmental Needs

Software:

- Supported browser
- Extension
- Dynamic test chat webpage
- Browser developer tools

2.6 Special Procedural Requirements

Preconditions:

- Extension shall be installed and active.
- Supported chat webpage shall be loaded.
- Dynamic message-monitoring functionality shall be enabled.

Postconditions:

- Confirm each new message was detected.
- Confirm no unrelated DOM changes were extracted.

2.7 Intercase Dependencies

FR-E2 depends on successful completion of FR-E1 browser compatibility testing.

FR-E3 – Client-Side DOM Inspection

2.1 Test Case Identifier: FR-E3

2.2 Objective

Verify that the extension correctly identifies DOM elements containing chat messages and extracts their rendered text content.

The test shall verify that the extension:

1. Identifies the appropriate HTML message containers.

2. Extracts visible message text.
3. Preserves message content.
4. Ignores unrelated webpage elements.

Priority: High

2.3 Inputs

A controlled webpage shall contain example chat HTML.

Representative structure:

```
<div class="message">  
  <span class="sender">UserA</span>  
  <span class="message-text">Hello there!</span>  
</div>
```

The extension shall inspect the rendered DOM and extract the required information.

2.4 Outcomes

Expected output:

Sender: UserA

Message: Hello there!

The extracted message shall match the rendered message content.

Unrelated elements such as navigation bars, advertisements, or page controls shall not be interpreted as chat messages.

2.5 Environmental Needs

Software:

- Supported browser
- Extension
- Test chat webpage
- Browser developer tools

2.6 Special Procedural Requirements

Preconditions:

- Extension shall be active.
- Test webpage shall contain known DOM structures.

- Message selectors or identification rules shall be configured.

Postconditions:

- Compare extracted content with expected DOM content.
- Confirm required message text was preserved.

2.7 Intercase Dependencies

FR-E3 depends upon the extension successfully executing within the supported webpage environment. (FR-E2 completes properly).

FR-E4 – Message Tracking and Duplicate Prevention

2.1 Test Case Identifier: FR-E4

2.2 Objective

Verify that the extension tracks previously processed message nodes and prevents the same message from being extracted repeatedly.

The extension shall:

1. Associate an identifier with each processed message.
2. Store processed identifiers for the active session.
3. Check whether a message was previously processed.
4. Skip previously extracted messages.
5. Continue extracting newly detected messages.

Priority: High

Duplicate prevention ensures that the adapter and downstream classification model do not repeatedly process the same chat messages.

2.3 Inputs

Initial conversation:

Message 1: Hello

Message 2: How are you?

After initial extraction, a third message shall be dynamically added:

Message 3: I'm doing well.

Additional DOM updates shall then be triggered without changing the original messages.

2.4 Outcomes

During initial processing:

Message 1 → Extracted

Message 2 → Extracted

The extension shall record identifiers associated with Messages 1 and 2.

When the DOM is checked again:

Message 1 → Previously processed → Skip

Message 2 → Previously processed → Skip

Message 3 → New → Extract

Expected result:

- Existing messages are not extracted again.
- New message is extracted exactly once.
- Repeated DOM changes do not result in duplicate copies of previously processed messages.

2.5 Environmental Needs

Software:

- Supported browser
- Extension
- Dynamic test chat webpage
- Browser developer tools
- Session memory or data structure used by the extension

2.6 Special Procedural Requirements

Preconditions:

- Extension shall be active.
- DOM message extraction shall be operational.
- Tracking data shall begin in a known empty state.

A message shall be extracted and recorded before duplicate detection is tested.

Postconditions:

- Confirm each unique message was extracted once.
- Confirm previously processed messages were skipped.
- Confirm newly added messages continued to be detected.
- Reset temporary tracking data after the test if required.

2.7 Intercase Dependencies

FR-E4 depends on successful operation of:

- FR-E2 – Dynamic Chat Monitoring
- FR-E3 – Client-Side DOM Inspection

FR-A1 – Adapter Input Validation and Rejection

2.1 Test Case Identifier: FR-A1

2.2 Objective

Verify that the adapter correctly validates incoming chat-log data against the format established by the browser extension.

The test shall verify that the adapter:

1. Accepts correctly formatted chat-log input.
2. Detects chat logs that do not comply with the expected format.
3. Rejects invalid chat-log input.
4. Prevents invalid input from continuing through the system.
5. Provides the requesting service with a notification identifying the reason the input was rejected.

2.3 Inputs

Each test case shall use a controlled chat-log payload representing either valid or intentionally invalid data. The exact fields shall conform to the finalized extension-to-adapter chat-log structure.

2.4 Outcomes

- The adapter receives the chat-log payload.
- The adapter determines that the payload conforms to the expected structure.
- The input is accepted.
- The payload is permitted to continue to the next processing stage.
- No validation error is generated.
- The adapter detects the invalid format.
- The input is rejected.
- The invalid payload is not forwarded for further processing.
- The requesting service is notified of the rejection reason.
- Notification of rejection due to missing required data field

2.5 Environmental Needs

Hardware

No specialized hardware is required.

Software

- Supported operating system
- Python
- Javascript
- FastAPI
- Current adapter application
- Current extension-to-adapter schema definition
- Postman For testing

2.6 Special Procedural Requirements

Preconditions:

- Required software dependencies shall be installed.
- The current chat-log schema shall be configured.
- A known-valid chat-log payload shall be available.
- Any service used to receive validation responses shall be available.

FR-A2 – FastAPI Normalization

2.1 Test Case Identifier: FR-A2

2.2 Objective

Verify that the adapter correctly cleanses and normalizes validated chat-log data into the standard internal format required by the system.

The test shall verify that the adapter:

1. Receives chat-log data that has successfully passed input validation.
2. Converts the input into the defined standard structure.
3. Removes or corrects unnecessary formatting inconsistencies.
4. Standardizes values such as timestamps and field structure where required.
5. Preserves the meaning and required content of the original chat messages.
6. Produces data suitable for use by downstream components such as the database and machine-learning model.

2.3 Inputs

- Each test case shall use a valid chat-log payload containing data that requires one or more normalization operations.
- The exact standardized structure shall match the finalized adapter data model.

- Input shall contain valid chat-log data that conforms to the accepted extension format.
- The adapter shall convert the input into the defined internal normalized structure.
- Input shall contain valid fields with unnecessary leading or trailing whitespace.
- Input shall contain a valid timestamp in a supported source format.
- If source chat platforms or extension modules produce supported variations in field naming or structure, the adapter shall map those fields into the system-standard representation.
- Input shall contain valid chat messages containing punctuation, capitalization, and other meaningful text content.

2.4 Outcomes

- The adapter receives the validated chat log.
- The adapter successfully normalizes the data.
- The resulting payload conforms to the system-standard structure.
- The normalized payload is available for downstream processing.
- Removed whitespace from data.
- TimeStamp will be normalized to a standard date/time.
- The adapter shall map supported source fields into the standard internal field names and structure.
- Required message content remains unchanged.
- Meaningful punctuation and text are preserved.
- Normalization shall not alter the semantic meaning of the conversation.
- The normalized data remains suitable for machine-learning analysis.

2.5 Environmental Needs

Hardware

No specialized hardware is required.

Software

- Supported operating system
- Python runtime
- FastAPI
- Current adapter application
- Current standardized chat-log schema
- Postman

2.6 Special Procedural Requirements

Preconditions:

- The adapter shall be running and accessible.
- Input data shall have successfully passed FR-A1 validation.
- The normalization rules and standard output schema shall be configured.

2.7 Intercase Dependencies

FR-A2 depends on successful completion of the applicable FR-A1 validation test because normalization shall only occur after the adapter determines that the incoming chat-log data is valid.

Transformer based classification model:

FR-T1: Preprocessing

2.1: Test Case Identifier: CSG-PRE-001

2.2: Objective

Verify that the preprocessing component accepts an authorized chat-log record from the data adapter and formats the conversation text as required for the model. This includes:

- Removing or masking identifying metadata
- Keep the conversation_id connected to messages
- Tokenize the text into model-ready inputs.
- Restructure the data into a PyTorch tensor

Priority: High

2.3 Inputs

- One valid, authorized chat-log window containing a conversation identifier, ordered message text, and timestamps from adapter.
- The configured Hugging Face tokenizer associated with the DistilBERT model.
- Preprocessing configuration values, including the selected maximum token length, padding strategy, truncation strategy, and return-tensor format.
- A test record containing nonessential metadata, such as a name, username, handle, or user ID, to verify that the model input excludes or masks information that is not required.

2.4 Outcomes

- Successfully accepts the authorized chat-log record
- Excludes or masks identifying metadata such as names, usernames, handles, and user IDs.
- Uses the configured Hugging Face tokenizer to convert the formatted conversation into model-ready PyTorch tensor inputs.
- The output should include, at minimum, input_ids and attention_mask tensors, while continually storing the conversation_id
- The produced tokenized inputs must be compatible with DistilBERT, capable of being passed directly into the model's inference process without type, shape, or missing-input errors, and generated without exposing unnecessary personal information in the model input or test logs.

2.5 Environmental Needs

- Software: Python, PyTorch, and DistilBERT
- Other: Authorized, privacy-minimized test chat logs

2.6 Special Procedural Requirements

- **Precondition:** Confirm that the authorized test chat-log record has been received from the adapter and contains formatted ordered message text with a conversation identifier. The record must be synthetic or otherwise approved for research.

- **Postcondition:** Record the tokenized version of the data, preprocessing configuration, formatted-input length, output tensor, and pass/fail result. Verify that the resulting `input_ids` and `attention_mask` PyTorch tensor are available as the input to the transformer test case.

2.7 Intercase dependencies

- This is dependent on all extension and adapter functional requirements passing.

FR-T2: Transformer

2.1 Test Case Identifier: CSG-TFR-001

2.2 Objective:

Verify that the fine-tuned DistilBERT transformer accepts the input from the testing PyTorch DataLoader produced by the preprocessing component from the pan12 testing dataset and generates contextual text representations for each conversation window. This test confirms that Crescent analyzes message content using conversational context rather than evaluating words as isolated terms.

Priority: High

2.3 Inputs:

- Valid DataLoader consists of tensors made up of `input_ids` and an `attention_mask` produced by CSG-PRE-001.
- A test conversation that contains language whose meaning depends on context across multiple messages, such as a sequence in which one speaker gradually pressures another speaker to keep a conversation secret.

2.4 Outcomes:

- Successfully accepts the `input_ids` and `attention_mask` tensor for one authorized, tokenized conversation window
- Produces a contextualized vector representation of that conversation window. The output shall be a PyTorch tensor
- This tensor shall capture the contextual relationships among the message text, message order, and approved de-identified speaker labels within the submitted window and shall be suitable for use as input to the separate downstream classifier-head component. The transformer shall complete processing without unhandled exceptions, missing-input errors, tensor-shape errors, or processing of unapproved identifying metadata.

2.5 Environmental Needs

- Hardware: Development workstation or approved server with sufficient CPU or GPU memory to load the fine-tuned BERT model and execute inference.
- Software: Python, PyTorch, DistilBERT, the Crescent model interface

2.6 Special procedural requirements:

- **Precondition:** Confirm that CSG-PRE-001 has passed and that the test uses valid `input_ids` and `attention_mask` tensors generated from an authorized, privacy-minimized conversation window.
- **Postcondition:** Record the model output and the pass/fail result. Confirm that the produced conversation tensor is available for the separate classifier-head test case.

2.7 Intercase dependencies

- This is dependent on all extension and adapter functional requirements, and CSG-PRE-001

FR-T3: Classifier Head

2.1 Test Case Identifier: CSG-CRH-001

2.2 Objective

The test verifies that the XGBoost classifier is configured for three-class probability output using `num_class = 3` and the `multi:softprob` objective. For each submitted conversation window, the classifier must:

- Return a NumPy array of scores between 0 and 1
- Return a classification label corresponding to one of the three configured risk classifications.
- Apply the approved threshold and priority rules to the returned probabilities to determine the final classification result for the window.

The test also verifies that the resulting classification, all three probability values, conversation window information, and analysis timestamp are made available to the downstream database component for association with the appropriate `conversation_id`.

2.3) Inputs

- One valid contextualized conversation tensor produced by CSG-TFR-001.
- An approved synthetic or de-identified test conversation window with an expected classification result.

2.4) Outcome(s)

The classifier head's output will be one of the categories mentioned above, along with a NumPy array that contains a score that represents how predatory a window is. The score that is generated is used to determine the overall score of an entire conversation. This score will be stored in the database along with the window it belongs to.

2.5) Environmental needs

- **Hardware:** Development workstation or approved server with sufficient CPU memory to load the fine-tuned DistilBERT transformer output fixture, the trained XGBoost classifier model, and required test data

- **Software:** Supported Linux operating system; Python, PyTorch, DistilBERT, NumPy, XGBoost, the Crescent transformer-inference module, the Crescent XGBoost classifier-head module.

2.6 Special procedural requirements:

- **Precondition:** Confirm that CSG-TFR-001 has passed and that the return type is a PyTorch tensor
- **Postcondition:** Record the conversation_id, tested window or message range, feature-array shape, full probability NumPy array, threshold values applied, final class identification and label, analysis timestamp, and pass/fail result

2.7 Intercase dependencies

- This is dependent on all extension and adapter functional requirements, CSG-PRE-001, and CSG-TFR-001

Explainability Layer:

FR-EL1 – Chat Record Receiving

2.1 Test Case Identifier: TC-FR-EL1

2.2 Objective

Verify that Transformers-Interpret can take in chat logs after the risk threshold is met. If the threshold is not met, the explainability layer is temporarily void.

2.3 Inputs

- Scanned message that is involved in the chat id at or over the threshold
- Chat ID
- Chat risk score
- Threshold value

2.4 Outcomes

- The chat logs are properly ingested
- The logs are staged in transformers-interpret to begin working with Captum

2.5 Environmental Needs

Hardware:

- PC/Laptop/Development Workstation with sufficient GPU and CPU power

Software:

- Pytorch, Captum
- Model Interface backend connection
- Supported Linux OS
- HuggingFace Support
 - Transformers-interpret
 - Transformers

2.6 Special Procedural Requirements

Preconditions:

- The model shall be running
- The model interface shall be running
- The chat logs shall be obtained from the extension and adapter
- Chat logs are properly sanitized and formatted
- Chat logs have accumulated a risk value at or over the threshold

Postconditions:

- Transformers-Interpret shall obtain chat logs
- Transformers-Interpret is installed
- Chat logs are properly staged for use with Captum

2.7 Intercase Dependencies

- FR-EL1 is the first test that must be completed before other tests in EL

FR-EL2-Captum Conversion

2.1 Test Case Identifier: TC-FR-EL2

2.2 Objective:

Validate that Captum can properly generate floating-point tensors of the chat logs

2.3 Inputs

- Ingested chat log package from FR-EL1-2.3

2.4 Outcomes

- PyTorch attribution Tensors

2.5 Environmental Needs

Hardware: PC/Laptop/Workstation with proper CPU power

Software:

- Pytorch, Captum
- Model Interface backend connection
- Supported Linux OS
- HuggingFace Support
 - Transformers-interpret
 - Transformers

2.6 Special Procedural Requirements

Preconditions:

- Captum is installed within Transformers-Interpret
- Valid chat logs have been ingested from the package
- Pytorch math engines are actively running
- The softmax classifier head is active

Postconditions:

- Proper floating-point tensors have been created and are staged for the second stage of Transformers-Interpret
- Temporary data from captum mathematical functions is cleared
- The softmax classifier head remains active

2.7 Intercase Dependencies

- TC-FR-EL1 must be validated

FR-EL3-Tuple Conversion

2.1 Test Case Identifier: TC-FR-EL3

2.2 Objective:

Validate that Transformers-Interpret can take in floating-point tensors and make Tuples from them

2.3 Inputs

- PyTorch attribution floating-point Tensors
- Raw text String

2.4 Outcomes

- List[Tuple[String, Float]] formatting for Transformers-Interpret tuples

2.5 Environmental Needs

Hardware: PC/Laptop/Workstation supporting sufficient CPU power

Software:

- Pytorch, Captum
- Model Interface backend connection
- Supported Linux OS
- HuggingFace Support
 - Transformers-interpret
 - Transformers

2.6 Special Procedural Requirements

Preconditions:

- Transformers-Interpret is active
- Must contain valid, sanitized strings for the Tuple
- Must contain valid floating-point values

Postconditions:

- Transformers-Interpret remains active to receive more tensors from Captum
- The Tuple must be in the correct List formatting for the Tuple Translator

2.7 Intercase Dependencies

- TC-FR-EL2 and previous intercase tests must validate successfully

FR-EL4-Tuple Translator

2.1 Test Case Identifier: TC-FR-EL4

2.2 Objective:

Validate that the Tuple Translator can format the list of Tuples from Transformers-Interpret into a proper string for BART to accept.

2.3 Inputs

- List[Tuple[String, Float]] format from Transformers-Interpret
- Tuple Translator function
- Chat ID
- Token-Threshold for filtering into BART

2.4 Outcomes

- Fully functional format to work with the BART summarizer
- Filtered list of high-risk tokens to allow BART to not reach the token limit
- Cleaned, merged strings

2.5 Environmental Needs

Hardware: PC/Laptop/Workstation with sufficient RAM and CPU power

Software:

- Python
- Pydantic
- Model Interface backend connection

2.6 Special Procedural Requirements

Preconditions:

- A valid Tuple list exists in the format List[Tuple[String, Float]]
- The list can be sent to the formatter
- Floating-Point values fall within the proper range no greater than 1 or less than -1.

Postconditions:

- BART can access the formatted string, containing the relevant information for a summary
- List of Tuples released from RAM for optimization

2.7 Intercase Dependencies

TC-FR-EL3 and previous intercase tests must validate successfully

FR-EL5-BART Summarizer

2.1 Test Case Identifier: TC-FR-EL5

2.2 Objective:

Validate that BART can successfully summarize formatted tuples into easily readable user summaries

2.3 Inputs

- Properly formatted Tuple list from the Tuple Translator
- Generation Command for execution

2.4 Outcomes

- Summary text String
- Chat ID
- Summary timestamps and completion status

2.5 Environmental Needs

Hardware:

- PC/Laptop/Workstation with sufficient RAM and CPU power
- GPU that can run CUDA or other parallel computing for intensive math workloads

Software:

- Pydantic
- Pytorch
- HuggingFace Support
 - BART
 - Transformers

2.6 Special Procedural Requirements

Preconditions:

- Preloaded BART model
- BART tokenizer loaded
- BART evaluation mode must be enabled

Postconditions:

- BART summary result is stored within the model packaging to be sent to the database

2.7 Intercase Dependencies

All previous intercase dependencies must be validated

Wep App: FR-W1 – Model Result Receiving

2.1 Test Case Identifier

Primary Test Case Series: FR-W1

2.2 Objective

Verify that the web application successfully receives classification results generated by the transformer-based classification system and preserves all required information.

The test shall verify that:

1. Classification results are successfully received.
2. Received values match the values produced by the model.
3. Incomplete or failed transmissions are identified.
4. The web application does not display incomplete data as a valid alert.
5. Loss or alteration of model output could cause reviewers to receive inaccurate risk information.

2.3 Inputs

- A controlled classification result shall be generated or simulated.
- The exact fields shall conform to the finalized interface between the classification system and web application.
- Provide a complete and valid model result.
- Provide a known result containing predetermined values that can be compared with the web application's received values.
- Provide a result missing one or more required fields.
- Simulate unavailable communication between the model/backend service and web application.

2.4 Outcomes

- Result is successfully received.
- Required fields are available to the application.
- Result is available for subsequent triage and display.
- Conversation ID matches the transmitted value.
- Risk score matches the transmitted value.
- Priority matches the transmitted value.
- Summary and indicator information remain unchanged.
- Incomplete input is detected.
- The incomplete result is not treated as a complete alert.
- An appropriate error is logged or returned.
- Communication failure is detected.
- The application does not create a false or incomplete alert.
- The application remains operational.

2.5 Environmental Needs

Hardware:

- Development workstation or laptop
- Network connectivity between test components when required

Software:

- Web application
- Web application backend/API
- Transformer model or mocked model service
- Supported browser
- API-testing utility such as Postman

2.6 Special Procedural Requirements

Preconditions:

- Web application shall be operational.
- Model interface or simulated model interface shall be available.
- Test classification result shall be known before execution.

Postconditions:

- Compare transmitted and received values.
- Confirm no required fields were lost or altered.
- Remove test alerts or records when necessary.

2.7 Intercase Dependencies

FR-W1 should be completed before integrity and failure testing.

FR-W2 – Flag Prioritization

2.1 Test Case Identifier

Primary Test Case Series: FR-W2

2.2 Objective

Verify that flagged conversations are organized in the review queue according to their assigned risk level or score.

The highest-priority alerts shall be presented before lower-priority alerts according to the defined prioritization rules.

Prioritization helps reviewers address potentially higher-risk conversations first.

2.3 Inputs

Provide multiple alerts with known risk scores.

Example:

CONV-001 – Risk Score: 34

CONV-002 – Risk Score: 91

CONV-003 – Risk Score: 67

CONV-004 – Risk Score: 48

The system shall use the finalized risk-score or priority rules established by the project.

2.4 Outcomes

Expected queue:

1. CONV-002 – 91

2. CONV-003 – 67

3. CONV-004 – 48

4. CONV-001 – 34

For equal-risk alerts, the defined secondary sorting rule shall be used, such as alert creation time.

When a new higher-priority alert is received, the queue shall update so that the new alert appears in its proper priority position.

2.5 Environmental Needs

Hardware:

- Development workstation or laptop

Software:

- Web application
- Test database
- Supported browser
- Test alert data

Other:

A defined prioritization rule shall exist before testing.

2.6 Special Procedural Requirements

Preconditions:

- Multiple test alerts shall exist.
- Each alert shall contain a known risk score or assigned priority.
- Queue sorting rules shall be defined.

Postconditions:

- Verify displayed queue order.
- Confirm no alerts were removed during sorting.
- Remove test data when required.

2.7 Intercase Dependencies

FR-W2 depends on successful receipt of alert information under FR-W1.

FR-W3 – Reviewer Notification

2.1 Test Case Identifier

Primary Test Case Series: FR-W3

2.2 Objective

Verify that the web application generates a notification when a qualifying alert is received and directs the notification to the designated reviewer.

The notification shall provide sufficient information for the reviewer to identify that an alert requires attention.

2.3 Inputs

A test alert meeting the established notification threshold shall be submitted.

A test email address or test messaging destination shall be configured.

2.4 Outcomes

Expected outcome:

- A notification is generated.
- Notification is sent to the configured reviewer.
- Notification identifies that an alert requires review.
- Relevant alert identification information is included.
- Notification shall not expose unnecessary sensitive conversation content.

2.5 Environmental Needs

Hardware:

- Development workstation
- Network connectivity

Software:

- Web application
- Email / Test email
- Database
- Supported browser

2.6 Special Procedural Requirements

Preconditions:

- Reviewer contact information shall be configured.
- Notification service shall be configured.
- Test alert shall meet the notification criteria.

Postconditions:

- Confirm receipt of notification.
- Verify notification destination.
- Verify notification content.
- Remove test notification data if applicable.

2.7 Intercase Dependencies

FR-W3 depends upon the web application successfully receiving a qualifying alert.

FR-W2 prioritization does not necessarily need to complete before notification occurs.

FR-W4 – KPI Card Display

2.1 Test Case Identifier

Primary Test Case Series: FR-W4

2.2 Objective

Verify that the web application displays alert information using the required card-based structure and that displayed KPI values accurately represent the associated classification result.

Potential card information may include:

- Overall risk score
- Priority level
- Conversation identifier
- Classification summary
- Indicator breakdown
- Individual message or sentence contributions
- Alert date/time
- Review status

The final KPI set shall be determined by the project's approved user-interface design.

2.3 Inputs

Provide a known classification result.

Example:

Conversation ID: CONV-002

Risk Score: 82%

Priority: High

Indicator 1: 17%

Indicator 2: 23%

Indicator 3: 42%

Status: Pending Review

2.4 Outcomes

The displayed alert card shall contain the required KPI values.

Expected values shall match the corresponding stored/model values.

Example:

Conversation: CONV-002

Risk Score: 82%

Priority: High
Status: Pending Review

No incorrect KPI value shall be displayed.

Multiple alerts shall produce separate cards associated with the correct conversation.

2.5 Environmental Needs

Hardware:

- Development workstation or laptop
- Display capable of running the supported browser

Software:

- Web application
- Supported web browser
- Test database
- Test classification results

2.6 Special Procedural Requirements

Preconditions:

- Test alert data shall exist.
- Required KPI fields shall be defined.
- User shall be logged into the web application if authentication is enabled.

Postconditions:

- Compare displayed KPI values against expected source values.
- Verify each card is associated with the correct alert.
- Remove test records when appropriate.

2.7 Intercase Dependencies

FR-W4 depends upon alert data being successfully available to the web application.

FR-W5 – User Authentication and Authorization

2.1 Test Case Identifier

Primary Test Case Series: FR-W5

2.2 Objective

Verify that only authenticated users can access protected web application functionality and that authenticated users can access only the functions permitted by their assigned role.

Authentication and authorization protect sensitive alert and conversation information from unauthorized access.

2.3 Inputs

Valid Authentication

Username: authorized_test_user

Password: valid_test_password

Invalid Authentication

Username: authorized_test_user

Password: incorrect_password

Test accounts shall also be assigned predetermined privilege levels.

Example roles may include:

Reviewer

Administrator

Read-Only User

The actual user roles shall match the finalized system design.

2.4 Outcomes

- Authentication succeeds.
- User is granted access to permitted areas of the application.
- Authentication fails.
- Access is denied.
- Protected application content is inaccessible.
- User is directed to authenticate or receives an access-denied response.
- User can perform functions assigned to the user's role.
- Restricted functionality is unavailable.
- Server-side authorization prevents the operation even if the user attempts to access the protected resource directly.

2.5 Environmental Needs

Hardware:

- Development workstation or laptop

Software:

- Web application
- Authentication service
- User database
- Supported browser
- Test user accounts

Other:

Test credentials shall be used rather than production credentials.

2.6 Special Procedural Requirements

Preconditions:

- Test users shall exist.
- User roles and privileges shall be configured.
- Authentication functionality shall be operational.

Passwords shall not be stored or logged in plain text during testing or after prod.

Postconditions:

- Confirm correct access level for each account.
- Confirm failed login attempts do not create authenticated sessions.
- Log out test users after execution.
- Remove temporary test accounts if required.

2.7 Intercase Dependencies

Authentication testing may be conducted independently of FR-W1 through FR-W4.

However, successful authentication may be required before executing user-interface tests for:

- FR-W2 – Flag Prioritization
- FR-W4 – KPI Card Display

3. General

3.1. Glossary

DistilBERT	Bidirectional Encoder Representations from Transformers, a pretrained language model that processes text using context from words before and after a given word. CRESCENT uses a fine-tuned BERT model to classify chat text
Captum	A PyTorch library used to generate interpretability information that helps show which parts of an input influenced a model prediction.
Data Adapter	The CRESCENT component normalizes the chat logs from any format into the standard that is required by the model to interpret.
Hugging Face	A software platform and library ecosystem that provides pretrained language models, tokenizers, and tools used to develop and run CRESCENT's DistilBERT-based model.
PyTorch	An open-source machine-learning framework used to train, evaluate, and run CRESCENT's DistilBERT detection model.
Extension	The component the scans the chat and inspects for DOM changes, packages data, and sends it to the Data Adapter.

3.2. Document change procedures and history

Changes to the CRESCENT Master Test Plan (MTP) will be managed through the project's shared Google Doc. Team members may propose changes by adding comments or using Suggesting mode and should provide a brief reason, such as a system update, testing issue, team feedback, or clarification. Proposed changes will be reviewed during project-team meetings, where the team will approve, revise, or deny them. Only approved changes will be incorporated by the document author, and the related comment will be resolved once the decision is recorded. Google Docs version history will provide an additional record of edits.

External References:

- <https://www.ftc.gov/legal-library/browse/rules/childrens-online-privacy-protection-rule-coppa>
- <https://docs.pytorch.org/tutorials/beginner/introyt/captumyt.html>
- https://www.researchgate.net/profile/Nancy-Agarwal/publication/358330008_Predatory_Conversation_Detection_Using_Transfer_Learning_Approach/links/620902fe7b05f82592e7926b/Predatory-Conversation-Detection-Using-Transfer-Learning-Approach.pdf
- <https://www.sciencedirect.com/science/article/pii/S0950705122011327>
- <https://github.com/cdpierse/transformers-interpret>
- <https://assets.super.so/7b61e5e8-a20b-49ec-9948-eb8a5300835a/files/82a90330-d914-4b5f-8055-a67fa9e72337.pdf>

Internal References:

- Requirements Document