

CRESCENT Social Guardian (*contextual risk evaluation system
for child exploitation in networked text*)

Software Design Document

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

Date: 9/9/26

Software Design Document

TABLE OF CONTENTS

1.0	<i>INTRODUCTION.....</i>	<i>4</i>
1.1	<i>Purpose.....</i>	<i>4</i>
1.2	<i>Scope.....</i>	<i>4</i>
1.3	<i>Overview.....</i>	<i>4</i>
1.4	<i>Reference Material.....</i>	<i>4</i>
1.5	<i>Definitions and Acronyms.....</i>	<i>4</i>
2.0	<i>SYSTEM OVERVIEW.....</i>	<i>4</i>
3.0	<i>SYSTEM ARCHITECTURE.....</i>	<i>4</i>
3.1	<i>Architectural Design.....</i>	<i>4</i>
3.2	<i>Decomposition Description.....</i>	<i>5</i>
3.3	<i>Design Rationale.....</i>	<i>5</i>
4.0	<i>DATA DESIGN.....</i>	<i>5</i>
4.1	<i>Data Description.....</i>	<i>5</i>
4.2	<i>Data Dictionary.....</i>	<i>5</i>
5.0	<i>COMPONENT DESIGN.....</i>	<i>5</i>
6.0	<i>HUMAN INTERFACE DESIGN.....</i>	<i>5</i>
6.1	<i>Overview of User Interface.....</i>	<i>5</i>
6.2	<i>Screen Images.....</i>	<i>6</i>
6.3	<i>Screen Objects and Actions.....</i>	<i>6</i>
7.0	<i>REQUIREMENTS MATRIX.....</i>	<i>6</i>
8.0	<i>APPENDICES.....</i>	<i>6</i>

1.0 INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of CRESCENT Social Guardian. The intended audience is parents of minors that are actively online along with social media and gaming companies that have minors as a part of their user demographic. This document explains how CRESCENT Social Guardian aims to contribute to the safety of minors online.

1.2 Scope

Crescent is a software platform designed to help protect minors online by improving how parents, guardians, social-media companies, gaming companies, and moderation teams manage and respond to digital conversations. The system collects authorized communication data and analyzes both message content and contextual conversation patterns. Crescent creates a case when it detects possible grooming behavior or exploitation. Unlike systems that rely only on keyword matching, Crescent is intended to consider context, patterns, and the overall meaning of a conversation, which can reduce the need for parents to search manually through a child's phone or for platform owners to review large volumes of messages individually. Its scope includes secure information management, conversation analysis, risk flagging, dashboards, and reporting. Future scope may include integration with social-media, gaming, messaging, and other online platforms used by minors. The primary goal of Crescent is to contribute to the protection of children online by helping authorized adults and platform safety teams identify potential risks earlier and make more informed decisions. The project aims to help minors by contributing to safer online spaces by drawing greater attention to potential safety concerns in digital conversations. Parents benefit by saving time due to being provided clearer information about conversations that may require attention. Future scope can benefit social media and gaming companies by improving moderation efficiency, prioritizing higher-risk cases, reducing reliance on simple keyword filters, and creating a more organized, scalable digital safety system.

1.3 Overview

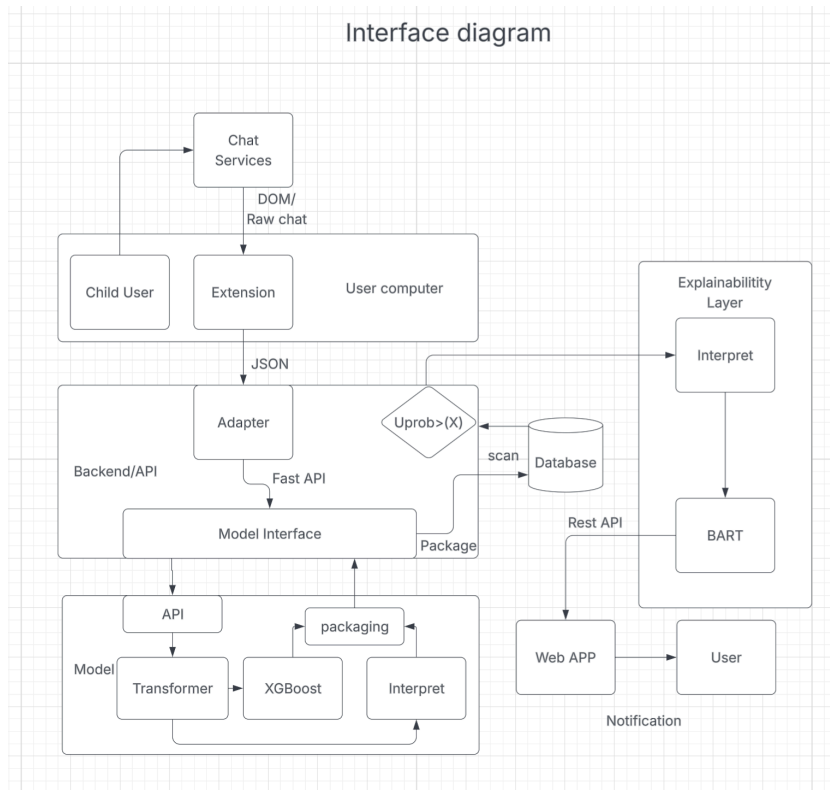
This document is organized to describe the design and implementation of CRESCENT Social Guardian. Section 2 provides an overview of the system and its overall functionality. Section 3 describes the system architecture, including the major system layers, their responsibilities, decomposition, and design rationale. Section 4 describes the system's data design, including how data is structured, processed, and stored. Section 5 provides the component-level design and describes the functions performed by the major components. Section 6 describes the human interface design, including the browser extension and web application used to interact with the system. Section 7 provides the requirements matrix, which maps the functional requirements from the Software Requirements Specification to the components and data structures that satisfy them.

2.0 SYSTEM OVERVIEW

Crescent is an online-safety software system designed to help parents and platform owners identify potential risks in minors' digital conversations without requiring constant manual review of chat logs. The project is intended for online environments where large volumes of communication make traditional moderating difficult and keyword-based filtering is deemed inefficient. Crescent's functionality begins with a browser extension that collects authorized chat-log data. An adapter component formats the conversation data into a structure compatible with the machine-learning model. The model performs preprocessing and analyzes windows of messages to evaluate the context, patterns, and meaning of the conversation rather than examining individual keywords alone. The resulting tensor representation is sent to a classifier head, which assigns readable risk categories based on the detected level and type of concern. Conversation details, analysis results, and risk classifications are then stored in a database. If a conversation reaches a defined threshold, the system sends a notification to the parent or authorized user. The flagged interaction is also added as a case within the Crescent web app, where the user can review the relevant conversation information and determine whether additional action is necessary. This design provides a scalable way to organize safety-related information, prioritize higher-risk conversations, and support more informed human review.

3.0 SYSTEM ARCHITECTURE

3.1 Architectural Design



Software Design Document

CRESCENT Social Guardian model is organized into separate, modular subsystems so that data collection, machine-learning classification, summarization, storage, and notifications can be developed and maintained independently. This decomposition keeps the browser extension lightweight, separates the machine learning components from the public-facing web application, and allows individual components to be updated without requiring a redesign of the entire system. The major subsystems are the user interface layer, backend layer, model layer, database layer, and web application layer.

Subsystem	Responsibilities
User Interface Layer (Extension)	<ul style="list-style-type: none">-Collects chat content from the minor's chat service through the DOM or raw chat data.- Converts collected chat information into JSON format.- Sends the collected content securely to the backend for analysis.
Backend Layer (Adaptor)	<ul style="list-style-type: none">- Accepts incoming chat data from the user interface layer.- Standardizes incoming data to fit the model's input requirements.- Connects the browser extension, database, and model interface through the FastAPI backend.- Sends data to the model interface and stores returned analysis packages in the database- Retrieves model output from database and sends it to the explainability layer upon UPROB triggering.
Model Layer (Classification Transformer Model)	<ul style="list-style-type: none">- Receives standardized chat text from the backend through the model interface/API to then be preprocessed- Uses a Transformer model to analyze the contextual meaning of chat messages.- Uses classifier head to assign a probability that content belongs to each

Software Design Document

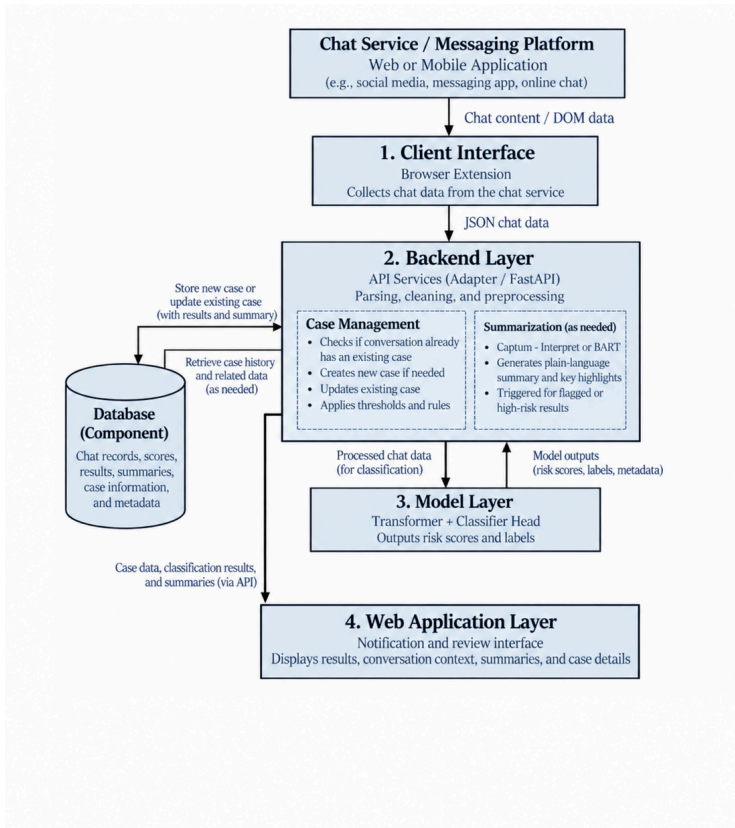
	<p>incoming window.</p> <ul style="list-style-type: none">- Packages classification results, scores, and relevant metadata for storage and later interpretation.- Returns the completed analysis package to the backend/database workflow.
Database Component	<ul style="list-style-type: none">- Stores chat-analysis packages, including messages, classification outcomes, probability scores, and related metadata.- Supports retrieval of records for later review, auditing, or analysis.- When a conversation record probability exceeds the established threshold, shown in the diagram as $uprob > (X)$.
Explainability Layer: Captum/Transformers Interpret/BART	<ul style="list-style-type: none">- Receives chat logs that contributed to the probability threshold being met.- Uses Captum within Transformers-Interpret to take in the messages and make raw math gradient calculations, locating specific “risky” words. Captum sends Pytorch floating point tensors to Transformers Interpret.- Uses Transformers Interpret to take in output from DistilBERT, processes and creates a list of Tuples (“Word”, 0.00). This list of tuples is passed into a middleware component that translates the tuples into raw strings that BART can use.- Uses BART as a summarizer, taking in flagged content and high-risk words to create a concise summary for the user to easily understand. The summary is sent to the user in the web application, which generates a notification for the user.
Web Application Layer	<ul style="list-style-type: none">- Allows users to log in to a viewable system to view triage alerts, details on chats, and see recommended actions.- It utilizes data from the database to fill the card system which is how the users will view triage reports.

- The system begins when the browser extension in the user interface layer collects chat content from the child user’s chat service
- The extension sends the chat data as JSON to the backend adapter, which receives, standardizes, and preprocesses the information before passing it to the model layer
- The model layer analyzes the tokenized chat content using the DistilBERT Transformer model and then generates a probability and a label using the XGBoost Classification model
- If a result exceeds the predetermined threshold, the stored model output is sent to the Explainability Layer, giving more information to the web application.
- The web application receives the information from the explainability layer, creates a case for the user to review, and sends a notification to the user.

3.2 Decomposition Description

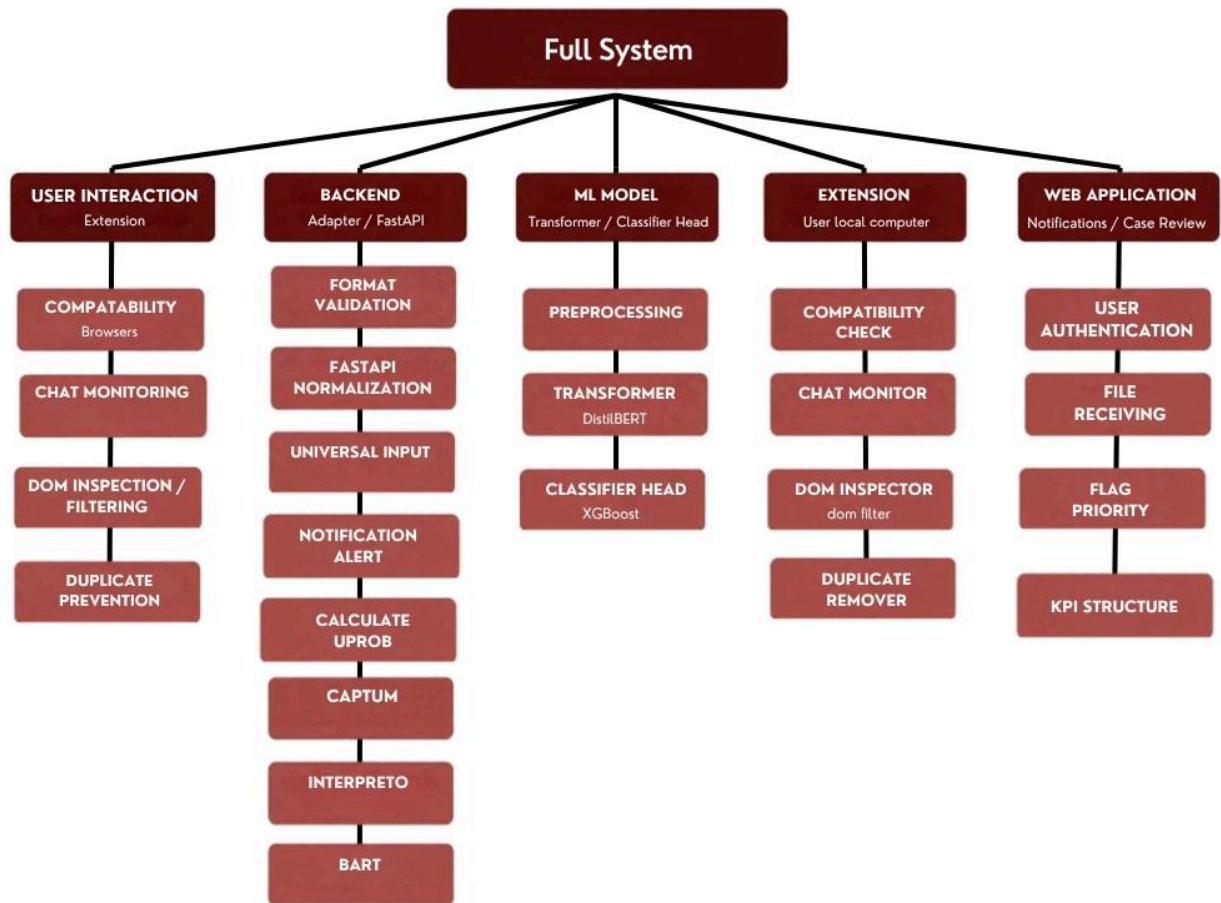
DFD:

Software Design Document



Decomposition Diagram

Software Design Document



3.3 Design Rationale

- Many BERT models have been used in this area of research previously, but DistilBERT is not one of them so we had a variety of tradeoffs to consider. We decided to use DistilBERT, because it is able to retain the majority of BERT's language understanding, while being smaller than BERT. Its smaller size lowers memory usage and enables faster results from the model.
- A critical issue is how we get data after we train/validate with Pan12. We decided that we can use a General Adversarial Network to test our model. The General Adversarial Network is another machine learning model that will produce data to try to trick our model into producing false outcomes.
- We chose to have an extension so we can easily pull chat screens from web-based chat services via DOM and mutationObserver. It will be downloaded to the local computer.
- We chose a web application for the user to log into a frontend service to view notifications and summarizations. A web application is very versatile and dynamically loaded from backend and database. It can protect data with user role-based authorization and authentications.
- We chose to use Captum for PyTorch's interpretability framework because it allows the model to remain unchanged as it computes gradient attribution tensors across the transformer. This choice was the best option of speed over other similar tools like LIME,

Software Design Document

which has random permutations.

- We chose to use Transformers-Interpret, as it works directly with Captum's tensors. Transformers-Interpret acts as a formatting connector for Captum's mathematical outputs. Transformers-Interpret handles transformer logic well and allows for a much less complex logic conversion, removing the need for manually aligning tokens from tensors.
- BART was chosen for concise text summarization. Another reason we chose to use BART was for user friendly presentation of information. There are many other summarizers to choose from, but they often simply take the exact phrasing with no true explanation. BART only provides information from flags to keep the majority of the original text unexposed for the minor's privacy.
- XG Boost was chosen due to its better performance and trainability versus the softmax function

4.0 DATA DESIGN

4.1 Data Description

After the adapter accepts the incoming data from the extension, it will transform the data into a normalized format. The format will be able to be fed into the model interface. From there, our preprocessing function will convert the data into a new data structure, which will be a PyTorch Tensor that contains the tokenized version of the messages at the 'input_ids', and an attention mask which specifies the amount of attention that a token will receive from the model. There will also be a special variable that stores the conversation ID of the conversation that this data belongs to. After the tensor has been fed into the model, this format will be deleted from memory. The output of the model will be another PyTorch tensor of scores. The backend will put the data into the format that the database will accept. The features of this format are: CONVERSATION_ID, AUTHOR_ID, CO_AUTHOR_ID, WINDOW_ID, MSG_TIMESTAMP, CHAT_MSGS, CHANNEL, WEIGHT, LABEL, SUMMARIZATION. From there, the backend will gather all chats with the same CONVERSATION ID to send for summarization. Once summarization is complete, the summarized chat and data will be entered into another section of the database for the web application to use for data display and user analysis.

4.2 Data Dictionary

Alphabetically list the system entities or major data along with their types and descriptions. If you provided a functional description in Section 3.2, list all the functions and function parameters. If you provided an OO description, list the objects and its attributes, methods and method parameters.

- **Conversation:** This will be a SQL table where each entry will represent a window consisting of the CONVERSATION_ID, AUTHOR_ID, CO_AUTHOR_ID, MSG_TIMESTAMP, WINDOW_ID, CHAT_MSG, CHAT PROVIDER, SCORES, LABEL.
- **JSON:** Package format for sending data from extension to the adapter.
- **Data Window Table:** This is an SQL entry within each conversation table that stores the messages sent, the author IDs, the timestamps for each message, a PyTorch tensor of the scores, and a label that is generated by the XGBoost model
- **Model Data:** This is a PyTorch tensor containing tokenized words and an attention mask that will be fed into the model
- **User-Metadata:** Sensitive information about the user, including addresses, names,

Software Design Document

etc. that will be flagged by our adaptor and a notification will be sent out to the parent/guardian of that child user. We will have a table within our SQL database that stores all of the author information including their name, their unique id, and a list of their conversation_ids.

5.0 COMPONENT DESIGN

Web Application:

- **Authentication and Authorization:**
 - AUTH_AND_AUTH(LOGIN, PASSWORD):
if login in AUTHOR_TABLE['logins']:
if HASH(PASSWORD) in AUTHOR_TABLE['passwords']:
return True
return False
- **Result and Verify KPI Cards:**
 - RESULT_VERIFY():
for card in CARDS_TABLE
for data in card:
if data == NULL:
alertBackend()
- **Display KPI Cards:**
 - KPI_Display():
for card in CARDS_TABLE:
display(card)

Backend:

- **Data Validation:**
 - VALIDATION(DATA):
REFERENCE conversation(SELECT DATA.conversation_id FROM
DATABASE['conversations']

if DATA.AUTHOR_ID == conversation.AUTHOR_ID and
DATA.CO_AUTHOR_ID == conversation.CO_AUTHOR_ID and DATA.PROVIDER ==
conversation.PROVIDER:
return True
else:
return False
- **Normalization:**
 - NORMALIZATION(DATA):
return DICTIONARY {
DATA.conversation_id: DATA.CHAT_MSG
}
- **Calculate Universal Probability:**
 - CALCULATE_UPROB(CONVERSATION_ID):

Software Design Document

```
REFERENCE convo (DATABASE[CONVERSATION_ID])
```

```
TOTAL_PROBS = 0
for scores in length(convo['SCORES']):
    TOTAL_PROBS += scores
return TOTAL_PROBS / length(convo['SCORES'])
```

- **Summarization:**

- SUMMARIZATION(CONVERSATION_ID):
REFERENCE convo (
(DATABASE[CONVERSATION_ID])

```
STRING conversation = ""
for MSGS in convo['CHAT_MSGS']:
    for MSG in MSGS:
        conversation += MSG
return BART(conversation)
```

- **Notification Alerts:**

- NOTIFICATION_ALERT():
sendToUSER("WARNING! PREDATORY BEHAVIOR DETECTED!")

- **Check for Duplicates:**

- DUP_CHECK(DATA):
REFERENCE conversation(DATABASE[data.CONVERSATION_ID])
for TIME in conversation['TIMESTAMPS']:
 if DATA.TIMESTAMPS[0] <= DATA.TIMESTAMPS[-1]
 return False
return True

Extension:

- **Compatibility Check:**

- COMPATIBILITY_CHECK(DATA):
if DATA.PROVIDER == APPROVED_PROVIDER:
 return TRUE
return False

- **Chat Monitor:**

- CHAT_MONITOR(URL):
THREAD:
 LAST_MSG = ""
 while (true):
 String Msg = getLastMessage(URL)
 if last_MSG != Msg:
 DOM_INSPECT(URL)
 last_MSG = Msg

- **DOM Inspect:**

- DOM_INSPECT(URL):
HTML_CODE = PULL_HTML(URL)

Software Design Document

```
MSGs = HTML_CODE['messages']
JSON data = {
    MESSAGES = [],
    AUTHOR_ID = HTML_CODE['author_id'],
    CO_AUTHOR_ID = HTML_CODE['co_author_id'],
    TIMESTAMPS = [],
    CONVERSATION_ID = HTML_CODE['conversation_id']
}
for (i = MSGs.length(); i >= MSGs.length() - WINDOW_SIZE; -i):
    data['MESSAGES'] += HTML_CODE[i]['Message']
    data['TIMESTAMPS'] += HTML_CODE[i]['TIMESTAMPS']
data['WINDOW_ID'] = assignWindowID();
DOM msg = DOM(data['Messages'])
msg.removeMetaData()
data['MESSAGES'] = msg
return DUP_PREV(data)
```

Model:

- **Preprocessor:**
 - PREPROCESS_DATA(data):
tokens = tokenizer(data['MSGs'], add_special_tokens = True, max_length = 128, padding = 'max_length', truncation = True, return_tensor = 'pt')
 - return tokens
- **DistilBERT Model:**
 - MODEL(tokens):
return DistilBERT(tokens)
- **XGBoost Mode:**
 - XGBoost(tensor):
return XGBoost(tensor)

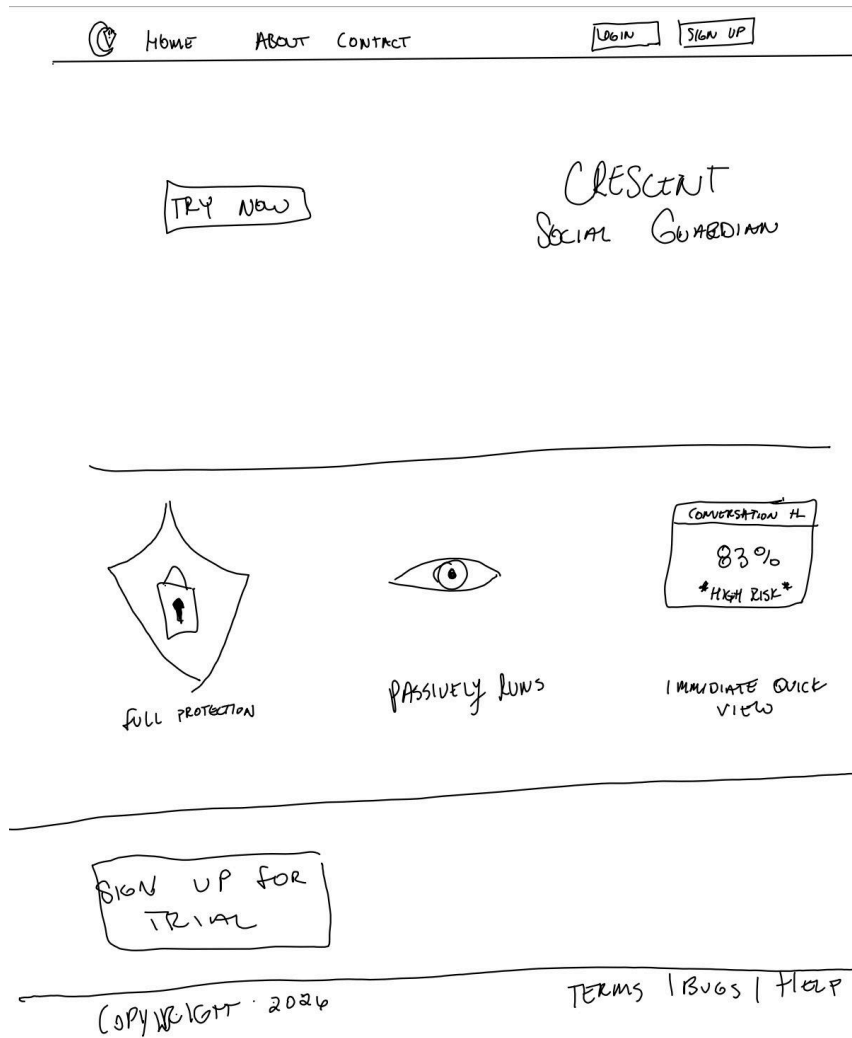
6.0 HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The parent will be offered a downloadable browser extension from our website once they agree to our terms* that can be downloaded on the intended device. Once the extension downloads and installs, it will run passively in the browser. Once the model finds a possible predatory chat, it will notify the parent via email/ text that there is an alert to review with a link to the web app. The parent will log into the web app. Upon verification of login credentials, the parent will be taken to a dashboard with the following notifications and information regarding alert(s) for each intended user and which service the chat is located in. The parent will be able to view a summarization of the conversation as well as an overall risk assessment of the conversation. There will be a specialization section to notify if the person were to share key important information regarding location.

6.2 Screen Images

Home Page:

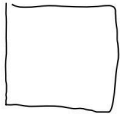


About us page:

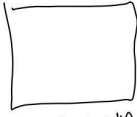
Software Design Document

[Home](#)[About](#)[Contact](#)[Login](#)[Sign Up](#)

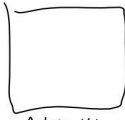
ABOUT US:



MICHAEL



ADRIAANA

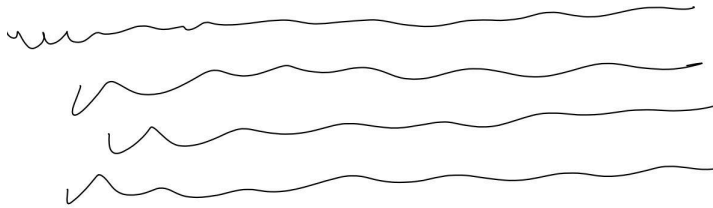


BHAVIK

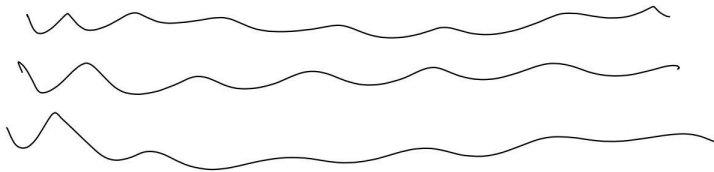


WICK

OUR MISSION:



HOW WE ARE DIFFERENT:



Contact us page:

Software Design Document

[HOME](#)[ABOUT](#)[CONTACT](#)[LOGIN](#)[SIGN UP](#)

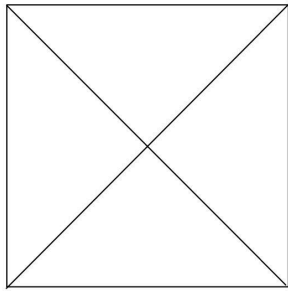
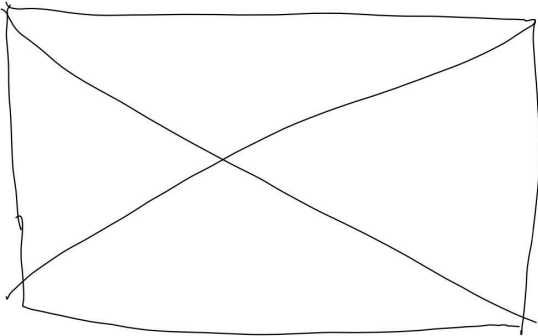
CONTACT US:

NAME

EMAIL

SUBJECT

MESSAGE



User Dashboard and Detailed Case page:

Software Design Document

PARENT

Crescent Social

ROLE: PARENT USER

2 HIGH

4 RISKS

FILTER: ☐ ☐ ☐ ☐

SEARCH: _____

CONVERSATION ID

87%
HIGH RISK

CONVERSATION ID

74%
HIGH RISK

CONVERSATION ID

37%
RISK DETECTED

CONVERSATION ID

48%
RISK DETECTED

SETTINGS

PARENT

Crescent Social

ROLE: PARENT USER

CONVERSATION ID: 101

PROBABILITY: 87%

KEYWORDS: "ADDRESS"

CHANNEL: DISCORD

TIME/DATE: _____

SUMMARY: _____

SETTINGS

6.3 Screen Objects and Actions

The web app will consist of 3 accessible pages and 2 pages requiring user login to view.

The 3 pages are :

- Home
- About us
- Contact

2 pages locked behind login credentials:

- User dashboard
- Detailed case view

Home: home page contains top bar containing the following buttons:

- Home
- About
- Contact
- Login
- Sign up

This page will explain what our software does and contain images and examples

About: the about us page contains the same top bar. This page displays pictures of us, our mission statement, and a paragraph on how we are different.

Contact: page contains the same top bar. This page allows personnel to contact us with a message attached.

User dashboard: This page is only accessible when the user logs in successfully. The top bar contains 2 kpi's, a filter, and a search bar. The side bar will contain different sections. The main screen will display the listed notification alert in card form for fast review / access.

Detailed case view: the user (upon clicking on a card) sees a detailed view of the card to include :

- Conversation id
- Probability
- Keywords
- Channel
- Time/date
- Summary

7.0 REQUIREMENTS MATRIX

Provide a crossreference that traces components and data structures to the requirements in your

Software Design Document

software requirements specification (SWRS) document.

Use a tabular format to show which system components satisfy each of the functional requirements from the SWRS. Refer to the functional requirements by the numbers/codes that you gave them in the SWRS.

SWRS Requirement	Requirement Summary	Components That Satisfy It	Data Structures / Stored Data	Traceability Explanation
FR-E1 Browser Compatibility	Ensure extension operates within supported Chromium-based web browsers and executes on supported web-based platforms	User Interface Layer	DOM Node Objects	The extension can only work should the incoming messages be coming in from an approved web browser, or else our extension will not be able to receive new messages.
FR-E2 Dynamic Chat Monitoring	Ensure extension detects newly rendered chat messages dynamically	User Interface Layer		We will have a chat-monitoring function that accepts data and feeds it into our pipeline. This will be a multi-threaded feature that will work concurrently across multiple conversations across multiple services.
FR-E3 DOM Inspection / Filtering	Ensure extension inspects DOM to identify and extract messages along with removing non-message interface content	User Interface Layer	DOM Object Node	Once the messages have been extracted from the HTML code, they are fed into a DOM Object Node to filter out any user metadata. From there, it will be turned into a JSON file to be sent into the backend.

Software Design Document

FR-E4 Duplicate Prevention	Ensure extension assigns/derives temporary identifiers to prevent duplicate extraction	Backend Layer	Database	Once the backend receives the data in JSON from the frontend, it will then ensure that there is not another window with the same conversation_id and an overlapping timestamp. If so, then a notification to the administrator, if not then the data will continue through the pipeline as usual.
FR-A1 Adapter Validation	Ensure adapter verifies format of chat logs	Backend Layer		The adapter must receive the minimum required data packages needed for the model to interpret and store properly.
FR-A2 FASTAPI	Ensure adapter normalizes data	Backend Layer		The incoming data will not be in an approved format that can be used by the preprocessing function to perform any operations, so the FASTAPI will fix this by converting the format into an approved format for the preprocessor.

Software Design Document

FR-T1 Preprocessing	Ensure the model accepts logs from adapter and properly formats the conversation text as required for model	Model Layer	Chat messages, input_ids, attention mask, conversation_id, PyTorch Tensor	The preprocessing component receives conversation data from the adapter (Backend Layer), tokenizes and formats the chat messages into the input structure required by the transformer, and associates the input with the appropriate conversation_id.
FR-T2 Transformer	Ensure the fine-tuned DistilBERT transformer accepts the input from the preprocessing component and generates contextual text representations for each conversation window.	Model Layer	input_ids, attention mask, PyTorch Tensor, contextual text representation	The DistilBERT transformer receives the tokenized conversation window and attention mask produced during preprocessing and generates a contextual representation of the conversation for use by the classifier head.
FR-T3 Classifier Head	Ensure XGBoost is configured for three-class probability output and Returns a NumPy array / classification label, along with applying the approved threshold to the returned probabilities	Model Layer	Probability scores, classification label, conversation_id, model-analysis package	The XGBoost classifier head receives the transformer output and produces probabilities for the three risk classifications. The approved threshold and priority rules are applied to determine the classification result, which is associated with the appropriate conversation_id and packaged for storage in the database.

Software Design Document

FR-EL1 Chat Record Receiving	The Explainability Layer will obtain chat logs once the uProb score from the softmax classifier head reaches a set threshold.	Backend Layer	Chat ID Chat Risk Score Threshold Score TokenizerOutput	The Explainability Layer can only work if the chat logs are properly ingested, taking the threshold score and accepting it and the chats as long as the threshold is met.
FR-EL2 Captum Conversion	Captum shall be used within Transformers-Interpret to generate floating-point tensors of the obtained chat logs.	Backend Layer	Raw Attribution Tensor Pytorch Tensors	The Captum Conversion within Transformers-Interpret can only be met once a chat record is properly received. These records are used to create floating-point tensors to be used by the second half of Transformers-Interpret.
FR-EL3 Tuple Conversion	Transformers-Interpret will obtain floating-point tensors and convert them into a list of tuples of string and floating-point values.	Backend Layer	Raw Attribution List: List[Tuple[str, float]]	Tuple Conversion can only be accessed when the floating-point tensors are created by Captum's math. The floating-point values are taken in and formatted into the raw attribution list, staging for the Tuple Translator.
FR-EL4 Tuple Translator	The tuples obtained from Transformers-Interpret will be formatted to be sent into the BART summarizer model.	Backend Layer	Raw Attribution List: List[Tuple[str, float]] Pydantic Schema for BART	The Tuple Translator is accessed once the raw attribution list is properly formatted and sent. This translator is the bridge between Transformers-Interpret and the BART

Software Design Document

				Summarizer. The attribution list properly formats the raw attribution list in a way for BART to understand by creating a natural language prompt that shows the high-risk tokens and chat IDs for BART to create a proper summary.
FR-EL5 BART Summarizer	BART will take in the formatted string from the Tuple Translator, generating summaries of the provided strings.	Backend Layer	Bart Summary	The BART Summarizer can only work under the condition that the Tuple Translator is working properly, sending formatted, natural language strings to BART. BART is able to create a summary result that is able to be sent to the database.
FR-W1 Card Receiving	Ensure web applications shall receive KPI card information from the backend and database with no loss.	Backend Layer	Database	If the card is missing key data, then this indicates a loss of data in the pipeline. A card should only be built after the data is verified to contain all required fields.

Software Design Document

FR-W2 Flag Priority	Ensure web application had flags triaged into a list by score	Web App Layer		We want the highest score to be seen first providing a faster response to more critical cases.
FR-W3 Notification Alert	Ensure web application sends out notifications when necessary	Web App Layer		Should the average of the window scores within a certain conversation meet a certain threshold, then a notification will be sent out via either text or email to the parent/guardian of the child.
FR-W4 KPI	Ensure web application utilizes KPI in card based structure	Web App Layer		A KPI card contains all the relevant information of a conversation including, the conversation_id, the risk percentage, and the overall classification of the conversation. These cards will be displayed on the web app. Parents can click on the cards and more relevant information will be displayed, like any keywords and a summary of the conversation.

Software Design Document

FR-W5 User Authentication / Authorization	Ensure web application only allows users to log in with specific name and password	Web App Layer	Database	Upon login, the name will be checked against the users table within the database to see if that name exists and the password will be hashed to see if it matches the hashed value stored in our database.
--	--	---------------	----------	---