Technical Specification - Labelling Tool

Last revision: 14 Jul 2022

Authors:

Tom Feltwell, Ian Johnson
Open Lab, Newcastle University
{tom.feltwell | ian.johnson2}@newcastle.ac.uk

Technical Review:

James Hodge
Open Lab, Newcastle University
j.hodge1@newcastle.ac.uk

Summary

This document describes the technical specification for the *Labelling Tool* that was designed as part of the Collective Crisis Intelligence project, which was run between October 2021 and June 2022. The project was run by Nesta, in collaboration with the IFRC and Open Lab, Newcastle University. The Labelling Tool was designed for use by the Cameroon Red Cross, to facilitate the labelling and analysis of mis-information and rumours around Covid-19. This document describes the design and a detailed proposed technical implementation of the Labelling Tool.

Intended Audience

This document is intended for developers and system architects who will be implementing the tool. This document details a proposed system architecture, detailed breakdown of all technical components and an estimate of tasks and associated time and financial costs to implement the system.

Acronyms/Glossary

Term	Definition
API	Application Programming Interface
CCI	Collective Crisis Intelligence
CRC	Cameroon Red Cross
IFRC	International Federation of Red Cross and Red Crescent Societies
RC	The Red Cross
Rumour	Piece of community feedback could be classified as misinformation/incorrect
SQL	Structure Query Language

System Overview

The Labelling Tool is a web application that is intended for use by volunteers and Red Cross staff. It provides a means for users to interact with the CCI model[1], by applying labels to rumours that are not recognised by the CCI model. They are also able to verify the labels that other users have applied, as well as inspect occurrence data through a set of different rankings.

This tool relies on the output of the CCI model, which will be running to support the Report & Respond tool (detailed separately).

This document focuses on the Labelling Tool, allowing volunteers and RC staff to engage with the model and wider labelling process, by performing manual labelling of rumours that the model is unable to recognise. There are three functions that users can do in the Labelling Tool:

1. Labelling - Users are shown a list of rumours the model does not recognise, and can suggest a label that should be applied to it. They can pick from a list of existing labels, or choose to suggest a new one. They can also fill in an optional rating of how harmful they think the rumour is.
2. Verifying - Users can see the label + rumour pairings that other users have suggested (in the Labelling activity, described above), and can indicate whether they agree with them. If not, they are able to provide some context for their disagreement, and an alternative label that they think is more suitable.
3. Rankings - Users can view usage data generated by volunteers using the model (via the Report & Respond system, a separate tool that uses the model). There are three different metrics to view the usage data, which are detailed in the implementation section.

The Labelling Tool is a web app and is not intended for use in the field, so we can assume it will be used somewhere where there is good internet connectivity. It is not bandwidth intensive and the proposed design should work OK on a low-bandwidth connection.

The CCI Machine Learning Model

The CCI model has already been developed as an independent part of the project, and the codebase is available in a public repository. This document describes the Labelling Tool, which will take input from the model to function. The full details of how the model functions can be found in the companion technical specification document for the Report & Respond tool [Github link].

The outputs of the model that are used in the Labelling Tool are:

[1] In this document we use “CCI model” to mean the integrated model pipeline developed for the CCI Cameroon project. This pipeline is made up of two models that are executed sequentially: 1) a classification algorithm that predicts rumour categories for incoming rumours and 2) a clustering algorithm that arranges unknown rumours into groups based on similarity.

- Unrecognised rumours: When a rumour is encountered by the model that is not recognised (it does not match any existing rumour), it will be placed into an unmatched[2] rumour group. This group is regularly output, and we propose that these unrecognised rumours are stored in the database (see implementation details below). These are then presented to users of the Labelling Tool for them to manually suggest a suitable label.
- Rumour occurrence: The Report & Respond tool records usage data each time a rumour is submitted by a volunteer, comprising: Rumour text, date and & time it was submitted. This data is then stored in the database, and processed into the rankings that can be viewed through the Labelling Tool.

French Language

Predominantly the Labelling Tool can function in both French and English. The only language that is fixed is the rumour data, which is input into the system in French language before it is available to the Labelling Tool. French was selected as the primary language of the tool as Cameroon is a majority francophone country and French is the preferred language of the Cameroon Red Cross.[3] It would be possible, and likely desirable, for the UI to be translated into English so the tool can be used in either language, but it is important to leave the rumour data untranslated to preserve the meaning, therefore it would remain in French.

Product overview

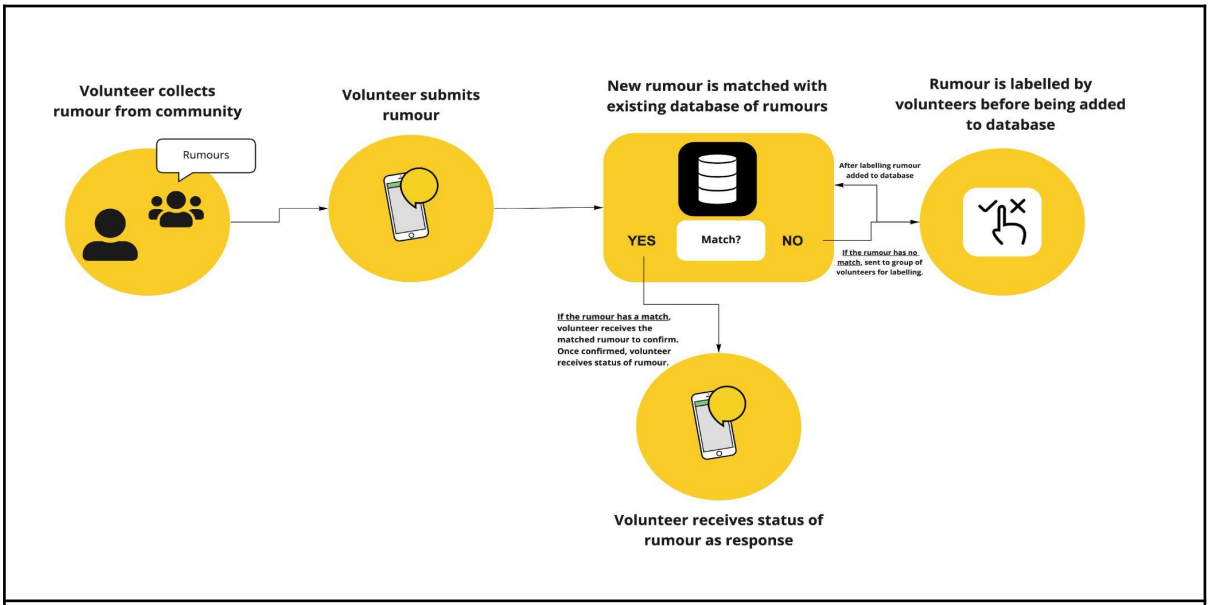

Figure 1. Broad product overview showing how volunteers engage with the model

[2] We use unrecognized and unmatched interchangeably throughout.
[3] Both French and English are official languages in Cameroon.

Expected usage characteristics

Use case	Description
1. Volunteer or RC staff member wants to engage with the model so uses the tool to apply labels to rumours that aren't recognised by the model.	Rumours that are not recognised by the model are stored in a pool. Using the Labelling Tool, a user can look at the rumours in this pool and then suggest a label that should be applied to the rumour. They can select a pre-existing label, or suggest a new label that should be created. This is the main function of the labelling tool. At the end of this process, they are also asked to (optionally) input how harmful they think the rumour is.
2. Volunteer or RC staff member wants to verify other users' label suggestions.	Once a label has been suggested for a rumour (see use case 1), these pairings are placed into a verification pool. A user can then view this pairing, and say whether they agree with the label suggested for the rumour. If not, they are asked to provide some information on their decision.
3. Volunteer or RC staff member wants to look at rumour metrics	The labelling tool provides three different metrics that rank rumours. A user can look at each of these rankings, and drill down into the rumours to see why they are ranked where they are (e.g. most commonly occurring in the past month). They can use this data to inform their intervention/response strategies.

Labelling Tool System Architecture

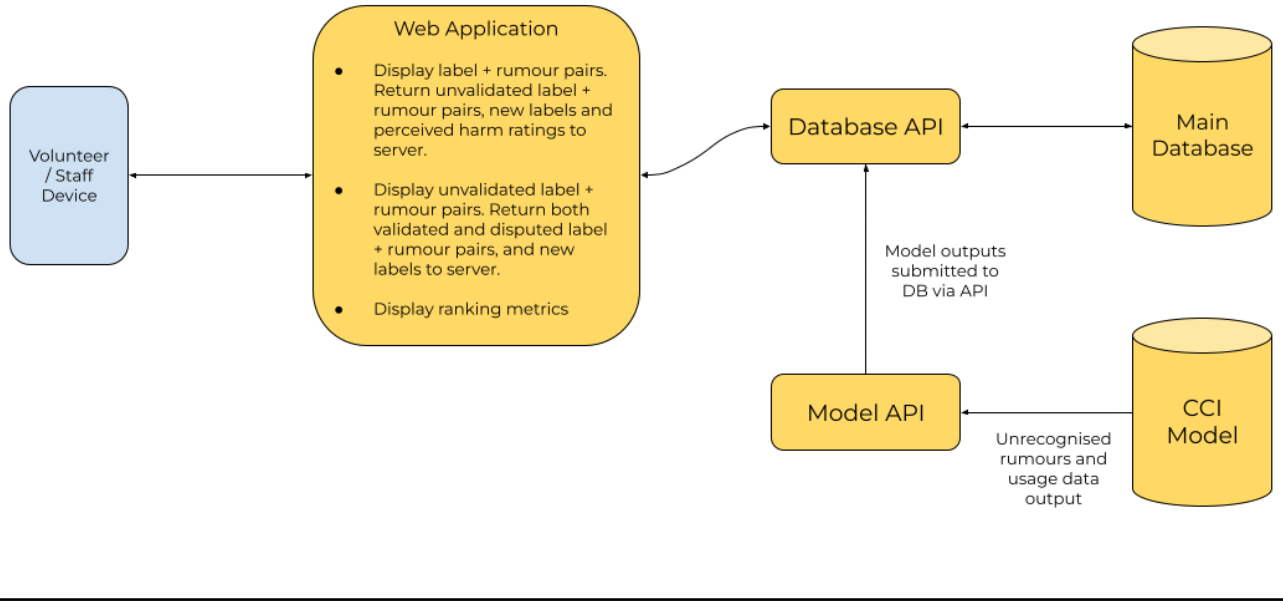

Figure 2. System architecture of the Labelling Tool showing how data is received from the model, and served from the database to the application.

The below table provides a breakdown of each component in this system architecture.

Component	**Details**
Web Application	Serves the web app Calls Database API to fetch data to be displayed. Calls Database API to put user generated data (input)
Database API	Interface for web server to put and fetch to the database.
Main Database	Store all of the data used by the application. Notably store data generated by the model (such as the unrecognised rumours for manual labelling), and store the label + rumour pairs generated by users of the Labelling Tool.
Model API	Invoked by the model to call Database API when producing outputs.
CCI Model	Trained machine learning model. In the context of this tool, the CCI model only produces outputs which are utilised in this tool. We assume the CCI model is already running to support the Report & Respond tool, and thus this document does not

	concern the implementation/hosting of the CCI model.

User Flows

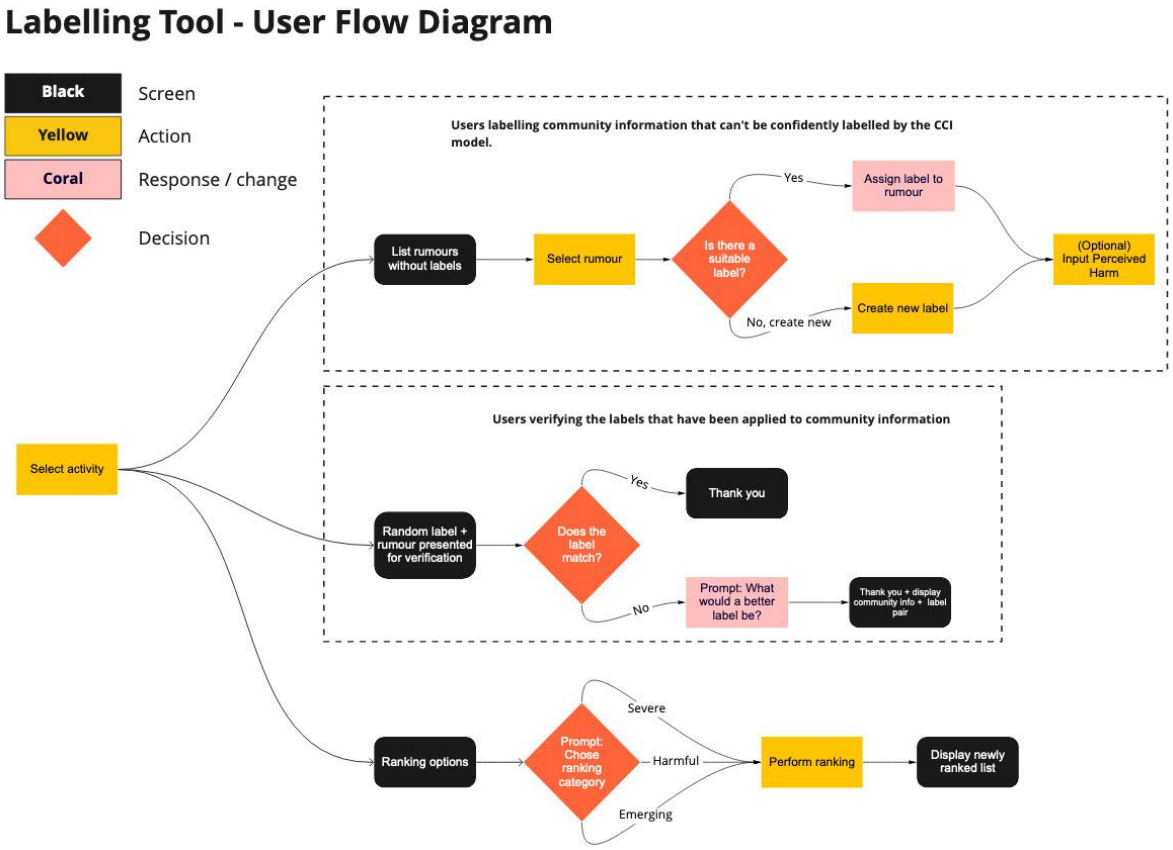

Figure 3. User flows through the labelling tool.

UI Prototypes

A prototype of the Labelling Tool was built in February 2022 with Adobe XD that shows the whole UI and all key interactions.

Links to prototype files	
XD share link (EN)	https://xd.adobe.com/view/bb4135f0-4e2f-44bc-b516-9d2b78442701-ee12/
XD share link (FR)	https://xd.adobe.com/view/52be0c34-c15d-48c4-ba00-a40fd97380bb-07ed/
XD File EN & FR	GitHub repository

Proposed implementation

We propose the Labelling Tool is a web application running on a modern Javascript framework to provide the frontend, that is hosted on a Node.Js server backend. An API is provided to allow the client to GET and POST data to an SQL database. In the backend, the SQL database will also receive data output from the model, which are used in the Labelling tool. Please see the system layers diagram below (Figure 4) for an overview of the data flow.

See the system architecture diagram in the introduction section. The system layers and data movement can be seen in the figure below:

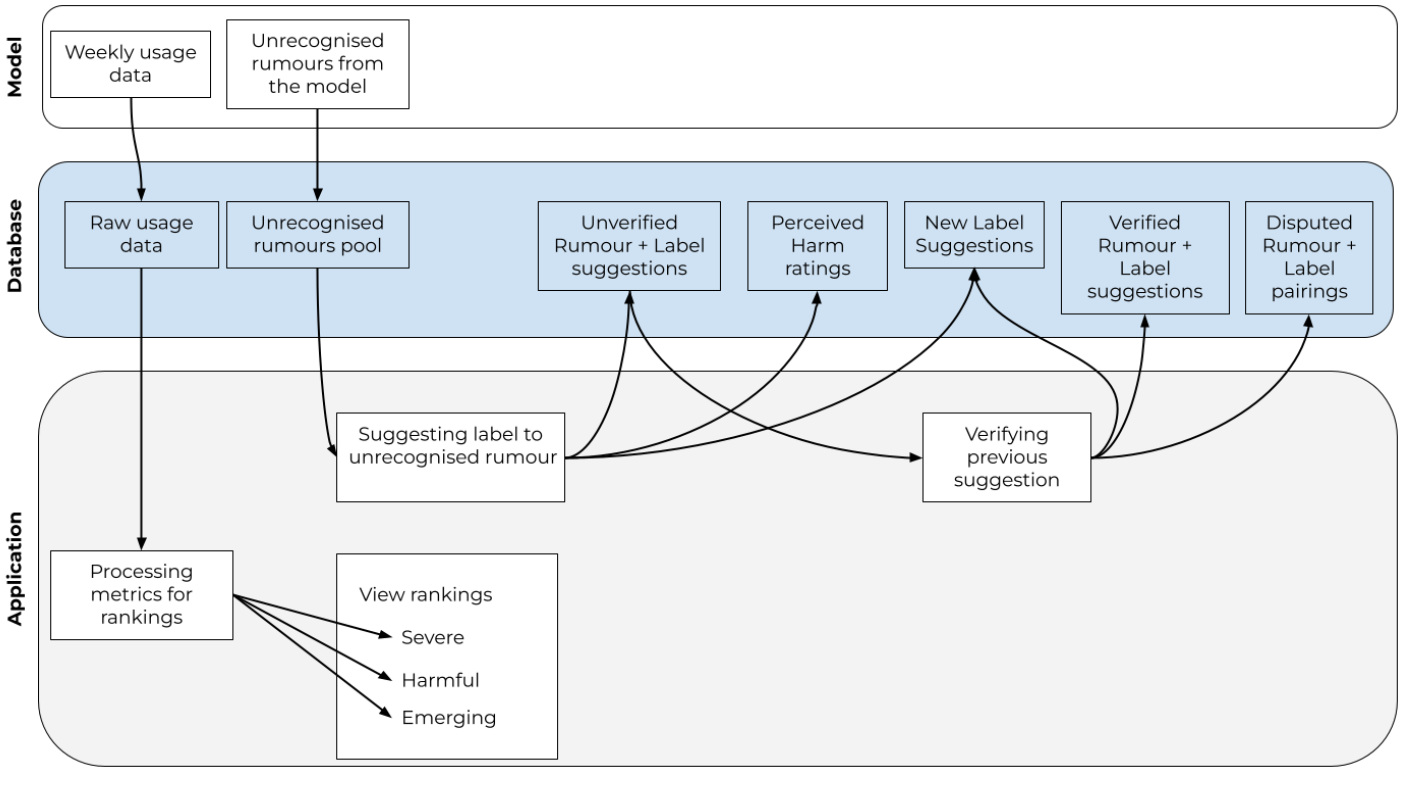

Figure 4. Systems layers showing data flows between layers.

System Component Details

The following sections provide detailed descriptions of how each component should function, including technology recommendations.

Web Application

We propose the main web application should use a modern Javascript framework to provide the client interface to the Labelling Tool. Due to the simple functionality of the web application (sending and receiving from an API, visualising receiving data, etc), we propose

the use of Vue.js. To support Vue, the server would run Node.js and the associated packages. See the flexible architecture boxout below for further details on the frontend implementation.

There are three main uses for the web application:

1. Applying labels to unrecognised rumours, and providing an optional Perceived Harm rating
2. Verifying label and rumour pairs produced by the above process. These can be either verified or disputed, with contextual information
3. Viewing usage metrics (rumours and labels) ranked by different criteria.

A full rundown of the user flow through the app can be seen on the previous pages (here), and an interactive prototype showing all the screens and the user flow can be seen here.

Flexible architecture point: The exact frontend framework used for the web app is flexible. The app itself is relatively simple so does not require complicated functionality. For developer speed we have suggested using a modern Javascript framework (Vue), which has good compatibility on devices. However further work is required to find out exactly which browsers are used by the volunteers and RC Staff, which will then guide the exact decision of the frontend framework.

1. Applying labels:

Unrecognised rumours are generated by the CCI model when the Report & Respond tool is used (beyond the scope of this document). If the model is not able to apply a label to an incoming rumour,it is unrecognised and output for manual analysis. Unrecognised rumours are stored in the database and used as the data source for the “Applying labels” user flow. A list of these unrecognised rumours is displayed to users, where they can pick one to apply a label to. They are asked to search for a suitable label. This search should be fuzzy, as there is often a large amount of labels (30+). If the user deems there is a suitable label they can apply it, generating an *unverified label + rumour pair*. If the user deems there is not a suitable label, they can create a new label suggestion. This new label suggestion is stored in the database, in its own table, for future review. This new label can then be applied to the rumour, generating an *unverified label + rumour pair*. We propose a theoretical limit on label length of 80 characters, but encourage this to be tested during Alpha and Beta testing.

Following the labelling process, the user is prompted to input (optionally) how harmful they perceived the rumour to be (Perceived Harm rating). This produces a numerical rating, and the user can also flag that the rumour is sensitive. The exact meaning of sensitive is still being discussed as of the time of writing, but is understood to be whether the rumour is discriminatory. When this is submitted it produces a *Perceived Harm rating.*

2. Verifying labels:

If there are unverified label + rumour pairs in the database, one will be pulled (at random) from the database and displayed to the user. They are then prompted whether they agree with the label that has been suggested for the rumour. If they agree, a *verified label + rumour pair* is submitted to the database. If they do not agree, they will be asked to provide a text description outlining their reasoning. They are then asked if they would like to suggest a more suitable label. This will generate a *disputed label + rumour pair*. If they answer “no”

they finish the process. If they answer “yes”, they will be prompted to input a label as per process #1 above. They can create a new label suggestion or use an existing label and apply it to the *disputed label + rumour pair.* At the end of this process, the *disputed label + rumour pair* is stored in the database for manual review by the Red Cross team. Description of this manual review process for disputed labels and rumours is beyond the scope of this document, and will need to be decided by the local organisation implementing the tool

By design, a user could sit and verify multiple label + rumour pairs, by going back into the verify process. A mechanism would need to be developed to ensure a user doesn’t receive the same label + rumour pair for verification twice, (e.g. storing the userId with each verification).

3. Producing metrics: The metrics are generated by combining the raw usage data that is received from the model/Report & Respond tool with the Perceived Harm ratings that are provided as part of the labelling process. We propose three rankings: Emerging, Harmful and Severe. How they are calculated is detailed in the below table.

Metric	**Details**
Emerging	Raw usage data, filtered for the past month. Ranked by frequency, so most commonly occurring rumour will be ranked #1.
Harmful	Rumours that have the highest Perceived Harm rating. The rumour with the highest Perceived Harm rating will be ranked #1.
Severe	Raw usage data filtered for the past month. Ranked by frequency (as with Emerging), and then sorted by highest Perceived Harm rating. This will show the most commonly occurring, most harmful rumour ranked #1.

Database API

The Labelling Tool is predominantly data driven so relies heavily on the database. The Database API handles requests for the database, primarily fetching data for the frontend, and handling input from the frontend and from the model. A proposed API specification is provided below:

Endpoint	**Method**	**Input query/params**	**Expected response**	**Onward processing**
/model/post/rumours	POST	1. Rumour text 2. Date time	HTTP code 200	Stored in Database
/model/post/usage	POST	1. Rumour text 2. Recognised? 3. If recognised, label 4. Date time	HTTP code 200	Stored in Database
/app/post/	POST	Unverif label + rumours	HTTP code	Stored in

			200	Database
/app/post/harm/	POST	Perceived harm ratings	HTTP code 200	Stored in Database
/app/post/label/	POST	New labels	HTTP code 200	Stored in Database
/app/post/verify/verified	POST	1. Label + rumour pair 2. Date time	HTTP code 200	Stored in Database
/app/post/verify/dispute	POST	1. Label + rumour pair 2. Contextual info 3. New label? 3a. New label if suggested 4. Date time	HTTP code 200	Stored in Database
/app/get/rumours/unverif	GET	-	1. Unverified rumours	-
/app/get/pairs/unverif	GET	-	1. Unverified label + rumour pairs	-
/app/get/rumours/usage	GET	-	1. Usage data	-

Main Database Structure

The following table is a proposed database structure to support the proposed system architecture. There is a small amount of relation between the tables (e.g. rumours being used across *unverified* and *verified* tables), so we suggest a relational SQL database be used. The functionality is simple, so any mainstream implementation of an SQL database would be suitable, such as PostgreSQL.

Table	**Details**
Unrecognised rumours	Rumours that could not be recognised by the CCI model (they are not similar to any existing label). Theoretically these will be produced regularly by the CCI model when it is operational.
Raw usage data	Usage data will be produced by the Report & Respond tool. This will consist of the rumour, the label, the date and time it was encountered.
Unverified Rumour + Label suggestions	A label and rumour pair, product of a user applying a label to a rumour. Also store the date & time when it was produced.

Verified Rumour + Label suggestions	Identical data structure unverified rumours, with extra fields denoting how many times it has been verified, and a list/array of the time & date when it has been verified.
Disputed Rumour + Label suggestions	When a user is verifying a rumour + label pair, it is possible to disagree. In this case, a new label will be proposed for the rumour. This label could be a new or existing label. The user may also submit contextual information to support their disagreement.
Perceived Harm ratings	Perceived harm ratings are produced by users at the end of the labelling process. This is an optional process. Propose the rumour is stored verbatim, with the numerical perceived harm rating, and a flag to mark whether the rumour is considered sensitive (e.g. discriminatory).
New label suggestions	New labels are produced in both the labelling and verifying parts of the Labelling Tool, by users who do not find a suitable label for a rumour. They are treated as suggestions, and the intention is they are manually analysed by the administrative users of the system.

Model API

Model will produce two outputs, unrecognised rumours and raw usage data. The output functionality from the model is not yet implemented, so the exact mechanism where this will be output from the model is not defined. Thus, we propose the following implementation:

The model is hosted on the same web server that the web application is running on, and an API is written specifically for the model (the *Model API*). The model can then call the Model API, which will process the data passed to it. It would have two endpoints:

1. Process unrecognised rumour
    a. Each time a rumour is not recognised when the model is run, it will invoke this endpoint. It will pass the rumour as text, and the date and time.
    b. This endpoint will then invoke the Database API to store the unrecognised rumour into the database.
2. Process usage data
    a. Each time a rumour is passed into the model (e.g. when it is being used by the companion Report & Respond tool), a record of usage is output. This will consist of: date time, rumour text, whether it was recognised, if recognised the label.
    b. This endpoint will then invoke the Database API to store the usage data into the database.

Flexible architecture point: This could be obsolete if the Database API is called directly from the CCI model. In this scenario the Database API endpoints would be hard coded within the model, and this may not scale depending on the back end architecture (e.g. if the model is hosted on a different machine, addresses would need updating etc). However, calling the Database API directly from the Model reduces the technical complexity.

Security considerations

The current implementation does not implement a security layer. Given the users of the Labelling Tool are able to view and provide suggestions about rumours to the RC staff, it is important to implement some access control to stop unauthorised usage. Being a web app, the Labelling Tool will be available to everyone with an internet connection. Security through obfuscation (e.g. not advertising the service, keeping the link secret, etc) will not be sufficient, as once an unauthorised user finds the service they can provide direct input to the system, thus polluting the data.

The simplest solution would be to implement a username and password system. This would require the username and password details to be input before use. This will stop any unauthorised usage, but would require some administrative work, either from the developers or from RC staff. For example someone in RC could be tasked with providing usernames and password to new volunteers/staff who wanted to use the tool.

Beyond access control, the APIs should implement SSL and therefore would be encrypted between the client and the server. There is no personal data used in the Labelling Tool, and the rumour data is not considered private (it is published on the IFRC Go platform). Therefore a simple username and password system should provide sufficient protection for the tool.

Projected Time & Costings

Assumptions: All development time is full-time, based on one developer working 9am - 5pm. Time estimates are conservative to factor in unforeseen technical challenges.
Financial cost assumed in GBP It should be noted tasks 1-3 may be conducted consecutively as there is interdependence between the tasks.

#	Task / Item	Details	Time cost	Financial cost
1	Developing web application, frontend	Writing frontend code, implementing UI designs, send/receive appropriate data for display.	6 weeks	Developer time x time cost
2	Developing web application, backend (inc APIs)	Configuring Node.Js server, writing database API, configuring to receive model input.	3 weeks	“ “
3	Configuring resources to host system	Configuring server, setting up databases, configuring send/receive of data between APIs.	2 weeks	“ “
4	Alpha Testing	Testing with small group of users (either external or internal to IFRC)	4 weeks	“ “
5	Iteration based on Alpha testing	Fixing bugs and making changes suggested from the alpha testing.	3 weeks	“ “
6	Beta Testing	Testing in Cameroon	4 weeks	“ “
7	Iteration based on Alpha testing	Fixing bugs and making changes suggested from the beta testing.	3 weeks	“ “
		Development Total	25 weeks	
	Recurring			
1	Technical support	Bug fixing, maintaining service	Variable	2 months ‘on call’ development time

		availability. Sensible to budget for at least 2 months technical support when the tool is launched to fix issues that arise in the field. Depending on scale of deployment and use of the tool, a service-level agreement should be discussed for expected uptime and maintenance response times.		once tool is released Service-level agreement TBC