**PROPOSED TECHNICAL ASPECTS**

This document provides a detailed overview of the technical components, architecture diagram, and system requirements for the Lost and Found application (LoboLocate). The goal of this app is to facilitate the reporting and matching of lost and found items, especially in the University of New Mexico area. This document covers APIs, libraries, models, and third-party services utilized in the app, along with prototype screenshots and an architecture diagram.

1. **Technical Components** 
   1. **Frontend (React Native)**

The front end of the app is built using React Native, to provide cross-platform compatibility (iOS and Android) and a fast development process. Key components of frontend are:

* Core Libraries:
  + React Navigation: For stack-based navigation across different screens. (For eg: Login->Welcome->ReportLostItem->Submit Report)
  + Expo: For easy access to device functionalities (camera, storage)
  + React Native Styles Components: For UI consistency and styling.
* State Management: Local component state is used for managing data changes like form input, image uploads, asynchronous calls to API.
  1. **Backend (Node.js/Express)**

The backend is hosted on Heroku and uses Node.js and Express to handle API requests. The backend handles the data flow between the client and MongoDB database. Key functionalities are:

* API Routes:
  + User Management: Login, Signup, user data fetching (/api/user/register, /api/user/Login, etc).
  + Item Reporting, and matching: Report of Lost/Found items, generating potential matches, and viewing users past reported items (/api/item/report, /api/item/potentialMatches, /api/item/history).
  + Real-Time Chat: Real-time chat functionality for communication between users and potential matches. Potential match also provides users with the finder’s name and email address so that they can contact via other methods.
  + Error Handling: Errors such as permission denial, failed network requests, and API key validation for Google Vision API are handled and logged.
* Database:
  + MongoDB: For storing user data, item details (when submitting the lost/found report), and chat history.
  + Mongoose: For schema definition and database management.
  1. **Cloud Storage and APIs**

This application integrates Google Vision API for item image recognition and description generation if the user decides to take a picture of the item or choose an image.

* Google Vision API: This is used to identify item categories (using object detection to find their distinguishing feature. For eg: bottle, pen, glasses), and label descriptions to enhance item details for matching. These fields are auto completed using Google Vision, which can be edited as per users’ preferences.
* Postman: Used Postman software for route testing and backend verification.
* Socket.IO: Used to enable real-time communication for chat functionalities, so that founder and owner can chat about where to meet to exchange the item.
  1. **Key Libraries and Packages**
* @react-navigation/native and @react-navigation/stack: Navigation and stack handling.
* @react-native-async-storage/async-storage: Local storage for managing user sessions and data.
* axios: HTTP client for API calls to backend services.
* expo-camera and expo-image-picker: Access to device camera and gallery for item photo submission.
* DateTimePicker: User-friendl date input for item reporting.
* react-native-popup-menu: UI component for in-app menu navigation.
* react-native-paper and styled-components: UI styling and element design.
  1. **Models**

Prototypes of the primary screens have clickable functionality have been designed and included with this submission. The submission also includes a runnable react-native code. Key screens for Figma include:

* Welcome Screen: Entry point after logging in/signing up. It shows navigation to main functionalities (reporting, viewing unclaimed items, potential matches, history).
* Report Lost/Found Item: Form where users enter item details, upload images, and specify lost/found status.
* Potential Matches: Displays list of possible matches for lost items, based on distinguishing features and item category. Potential matches is calculated at the backend.

1. **Prototype**

The prototype of this application was built using Figma. Here is the link:

[**https://www.figma.com/proto/XgpLrvdjt8neFJn8slB07u/LoboLocate-Figma-design?node-id=832-677&node-type=frame&t=m0RS7yj8Fb3oFDEj-0&scaling=min-zoom&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=777%3A253**](https://www.figma.com/proto/XgpLrvdjt8neFJn8slB07u/LoboLocate-Figma-design?node-id=832-677&node-type=frame&t=m0RS7yj8Fb3oFDEj-0&scaling=min-zoom&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=777%3A253)

Our team ourselves ran this code, and recorded a video which can be found at this link:

<https://www.youtube.com/watch?si=4XeyK-mAmmEyUd1Z&v=mtHUZE-uMTI&feature=youtu.be>

Github repository: <https://github.com/Aayush015/LoboLocate.git>

1. **Architecture Diagram**

**A diagram of a company

Description automatically generated with medium confidence**