Final Project - Milestone 1

Christopher Roman — **cr469**
Mushaffar Khan — **mk2248**
Mubeen Sheeraz — **ms2689**

Due November 15, 2017

Camlator

1 System Description

1.1 Summary

We plan to create a messenger application similar to WhatsApp/Facebook Messenger. However, *Camlator* is different such that it will utilize Google's Translate API to translate messages between people who may converse in different languages.

1.2 Core Vision

With more than 6000+ languages in the world, it is sometimes difficult to get ones thoughts across to someone that does not speak the same language. That is why our main goal for the final project in 3110 is to build a chat box that is similar in vein to WhatsApp/Facebook Messenger such that the users can communicate with each other inspite of the language barrier. The application will ask each participant's for his/her language preference and they can send messages to any other participant in their preferred language without any sort of barrier. A good example is if person A's language preference is English and person B's language preference is where Spanish, they can converse with each other in their preferred language and *Camlator* will translate A's (English) messages to B's language preference (Spanish) and vice-versa for messages sent by B to A.

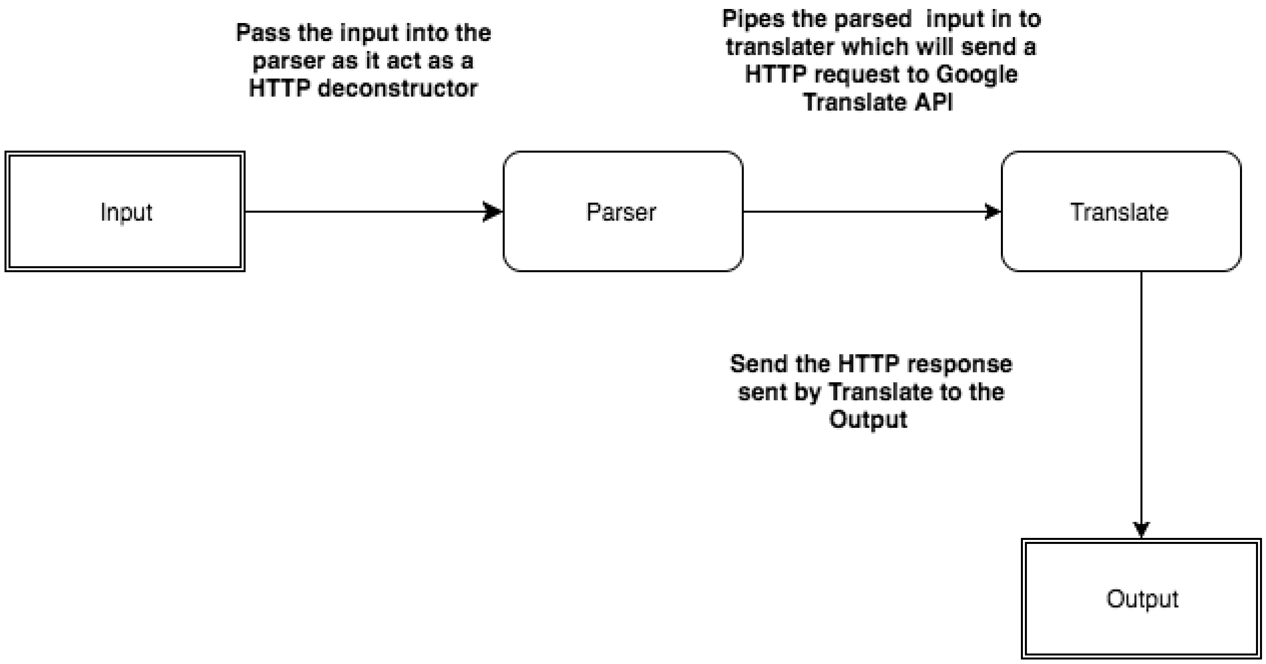
1.3 Key Features

Some of the key features of *Camlator* include:

- It will support general messaging between two users
- Message text will be translated in such a way that each user will see it printed onto their screen in a language of their choice, regardless of what language the other users prefer.
- Will support all UTF-8 characters rather than being limited to ASCII characters
- A front-end web page user interface in Bootstrap (CSS framework) that will display the messages as if it were in a messenger app

1.4 Top Level

We will be using Bootstrap (CSS framework) as our front end of *Camlator* and we will be using a HTTP server to host the two clients, which are the Inputs and Outputs in *a*, where they will be connected via sockets. As a client sends a message to the server, the server will receive the information (as a HTTP request) and parse it accordingly. It will then pass the information as a HTTP request to the Google Translate API and passes that response to the the other client(s) accordingly as they are all connected to the server via sockets. Figure *a* is a rough top level view of how we will be going about implementing *Camlator*.


Pass the input into the parser as it act as a HTTP deconstructor
Pipes the parsed input in to translater which will send a HTTP request to Google Translate API
Input
Parser
Translate
Send the HTTP response sent by Translate to the Output
Output

(a) Top Level of Camlator

2 System Design

1. **Client** — A Client represents an end user, and is split into two layers. First is the front-end layer, which handles dynamically updating the DOM to show new messages. The second is the back-end layer, which handles sending messages to the server to be sent to all other clients.

2. **Server** — A Server is the heart of the application. It's job is to receive all client's messages, translate them according to a client's language preference, and send the processed result to other clients. A Server keeps a log of every client's message history and can update a client's preferences. Servers must also deal with UTF-8 characters due to the plethora of supported languages.

3. **ClientData** — The ClientData is the data abstraction of a Client, meant to be used by the Server. ClientData holds all information about a particular Client, such as their language preference and name. This is important for when a Server needs to correctly translate incoming messages, update Client preferences, and send data back to Clients.

3 Data

3.1 Storage

Most of our data pertaining to this project will be the chat history amongst people, which will be stored as a JSON, in the server. The JSON will contain all the chat messages and the person that sent them as UTF-8 characters. Other information such as the profile of the participants in the chat room will be stored in the server as part of memory. Since we do not need the participant's information all the time, it is perfectly fine to have that data be stored in memory.

3.2 Communication

Our main form of communication will be through sockets. We will host the program on an HTTP server where all the clients will be connect to via sockets. Therefore, if one person that sends a message in the chat room, any other client connected the server will also receive the message to them (translated of course).

3.3 Data Structures

All of the information regarding each individual in the chat room will be stored as a record in memory. Another data structure we will most likely use are maps where we will map the user's ids to the messages they sent. We will also be using lists to go parse text history data associated with a particular user.

4 External Dependencies

Some of the modules we will be using are:

- **YoJson:** We will use YoJson to form and parse JSON files for messages
- **Batteries:** We will use the batteries module to be able to represent all possible languages in UTF-8 rather than ASCII characters.
- **cohttp:** To host our front end on a HTTP server
- **Bootstrap:** Use this CSS framework to display the messages amongst participants and to act as the front end for *Camlator*.
- **js_of_ocaml:** Will be used to update the client's webpage with the new information that was sent to him/her.

5 Testing Plan

We will be testing each function heavily as we are progressing along in the implementation. Also, as we begin to add more functionality to *Camlator* we will be testing to see if our old functions that we implemented were not affected by the changes. Moreover, once we have completed *Camlator* we will be testing to see how the overall functionality of *Camlator*, and how accurate it maybe in having a conversation with various languages. To make sure that we commit to our test plan as a team, not only will we test the functions we implement ourselves, we will also rigorously test each other's implementation to catch any nuances that one may have missed.