Product Design

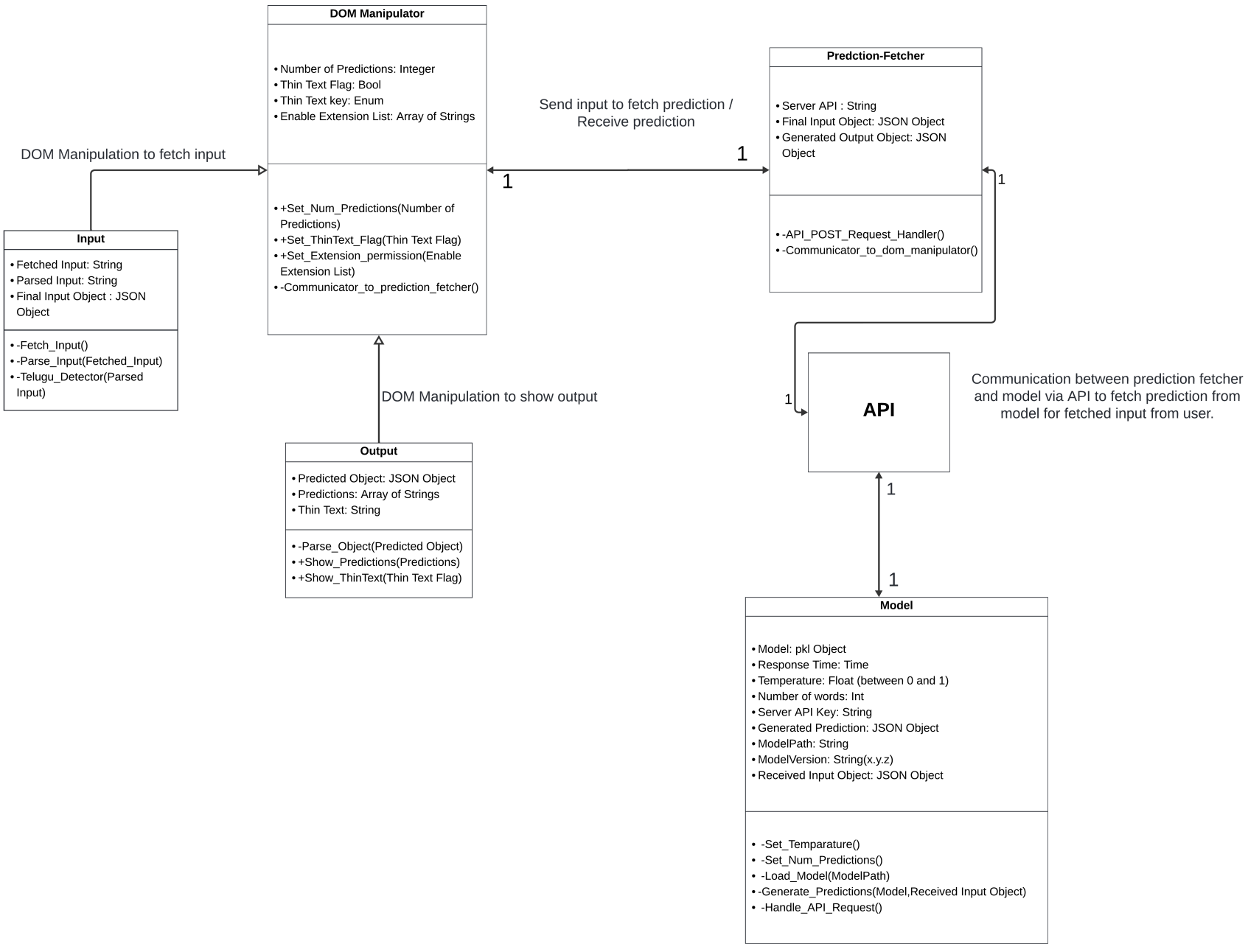
Team: 16

Team Members:

1. **Shreyansh (2022111002)**
2. **Siddharth Agarwal (2022101062)**
3. **Priet Ukani (2022111039)**
4. **Garvit Gupta (2022101113)**

Design Model

https://lucid.app/lucidchart/90ac76ef-0244-4028-8e79-1c372b01b657/edit?invitationId=inv_1455bfbb-ca3c-401b-ab15-16ff211dbc09&page=HWEp-vi-RSFO#



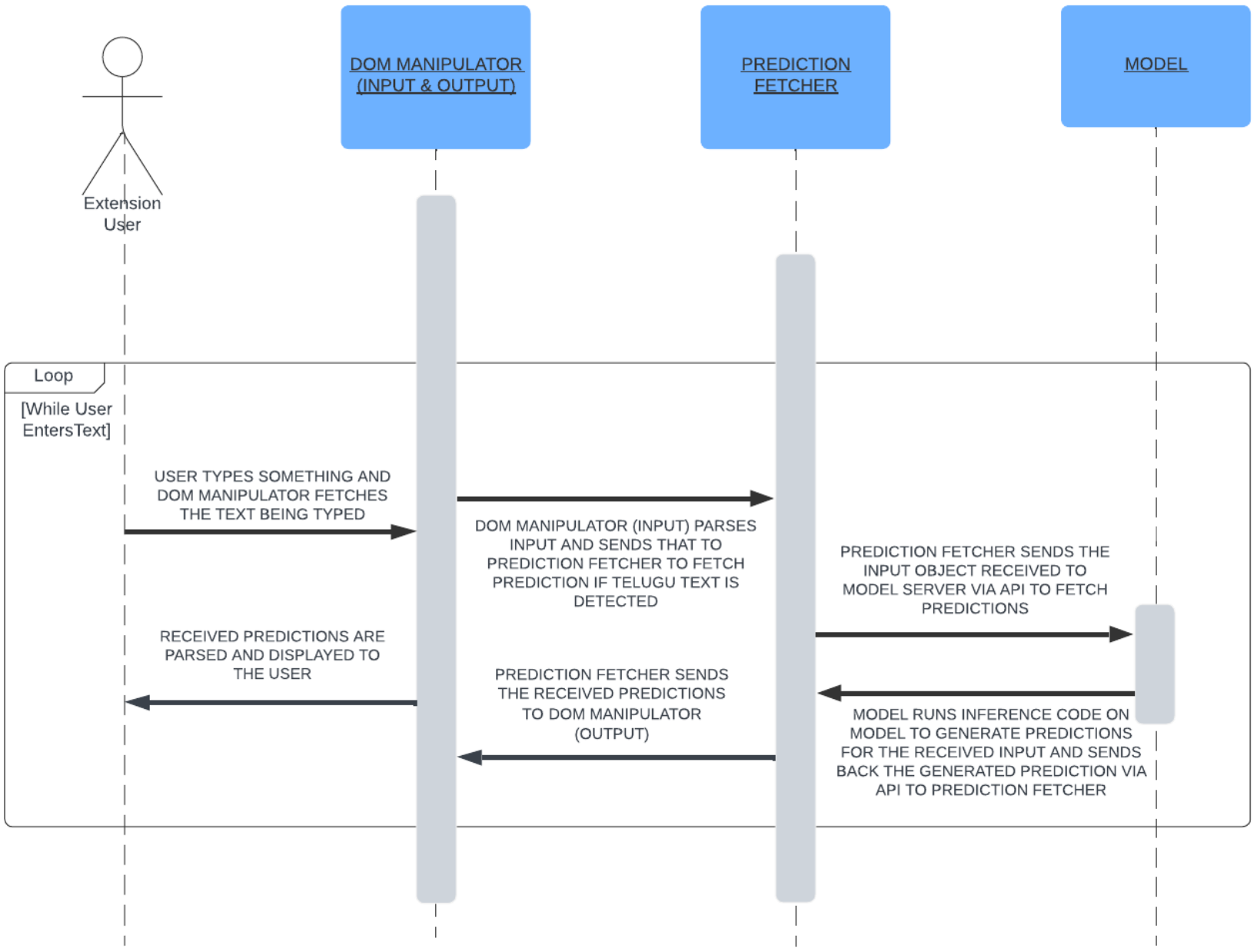
DOM Manipulator	**Class State:** This class manipulates DOM of live website to fetch input and to display output (It is generalisation of INPUT Manipulation and OUTPUT Manipulation) ***Number of Predictions***: Integer (User selects that how many predictions he/she wants to receive) ***Thin Text Flag***: Bool (User has choice to see predictions in thin text format or not) ***Thin Text key***: Enum (User can select that which key he/she wanna use to select the thin text prediction being displayed) **Disable Extension List**: Array of Strings (List of websites where extension has been disabled by user) **Class Behaviour:** ***+Set_Num_Predictions(Number of Predictions)***: Method to set Number of Predictions attribute according to user's choice ***+Set_ThinText_Flag(Thin Text Flag)*** : Method to set the Thin Text Flag attribute according to user's choice ***+Set_Extension_permission(Disable Extension List)***: Method to put or remove a particular website to or from disabled list ***-Communicator_to_prediction_fetcher()***: Method which communicates with prediction fetcher to send final input for prediction after Telugu detection and to receive the generated predictions.
Input	**Class State:** This class is specialisation of DOM Manipulation class. (It performs DOM Manipulation to fetch input from live website. It also parses the fetched input and checks if the fetched input is Telugu or not) ***Fetched Input***: String (This stores the raw input fetched from website) ***Parsed Input***: String (This stores the parsed input) ***Final Input Object*** : JSON Object (This is final JSONObject consisting of final parsed input along with Number of Predictions) **Class Behaviour:** ***-Fetch_Input()*:** This Method fetches input from live website ***-Parse_Input(Fetched Input)*:** This Method Parses the fetched raw input from website. ***-Telugu_Detector(Parsed Input)*:** This Method detects if parsed input is Telugu or not. If it is Telugu then the input is sent to prediction fetcher otherwise the sequence terminates and DOM Manipulator fetches another input.

Output	**Class State**: This class is also specialisation of DOM Manipulation class. (It performs DOM Manipulation to display the generated prediction. It also parses the generated prediction output to convert it into desirable form) ***Predicted Object***: JSON Object (This stores the received prediction object from prediction fetcher) ***Predictions***: Array of Strings (This is the final parsed output which will be displayed to user) ***Thin Text***: String (Best prediction out of all fetched predictions, which will be displayed in thin text format) **Class Behaviour:** ***-Parse_Object(Predicted Object)***: This Method parses the received predictions to convert it into desirable format. ***+Show_Predictions(Predictions)***: This Method displays the parsed final predictions to user ***+Show_ThinText(Thin Text Flag)***: This Method displays the best prediction in thin text format
Prediction Fetcher	**Class State:** This class handles the communication with Model server via API. It takes the final input from DOM Manipulator and sends it to model server to get predictions. After receiving generated predictions from the model server, it sends those output to DOM Manipulator to display predictions to the user. ***Server API :*** String (This stores the API key where request is sent for predictions) ***Final Input Object:*** JSON Object (This stores the final input to be sent for predictions) ***Generated Output Object:*** JSON Object (This stores the final generated predictions received from Model server) **Class Behaviour:** ***-API_POST_Request_Handler()***: This Method sends the POST request to Model server to generate predictions ***-Communicator_to_dom_manipulator()***: This Method communicates with DOM Manipulator to receive the final input object and to send the generated prediction object.

Model	**Class State:** This is the main Machine Learning prediction model server which generates predictions based on the user input. It receives the requested input via API, runs the inference code on model to generate predictions for the received input and then sends back the generated predictions via API to prediction fetcher. ***Model*:** pkl Object (This is pickel file of exported Model) **Processing *Time:*** Time (This is total time taken to process input) ***Temperature*:** Float (between 0 and 1) (This is the temperature for prediction generation) ***Number of words*:** Int (Number of words generated in each prediction) ***Server API Key*:** String (API Key to of server to receive request ***Generated Prediction*:** JSON Object (Generated prediction output) ***ModelPath:*** String (This stores the path of model to be loaded) ***ModelVersion:*** String(x.y.z) (This is the version of model being used for prediction) ***Received Input Object:*** JSON Object (This is the received input request for prediction) ***Class Behaviour:*** ***- Set_Temperature():*** *This Method sets the temperature for prediction generation* ***- Set_Num_Predictions():*** *This Method sets the number of predictions to be generated* ***- Load_Model():*** *This Method loads the model using the ModelPath* ***- Generate_Predictions(Model, Received Input Object):*** *This Method runs inference code on model to generate predictions* ***-Handle_API_Request():*** *This Method handles the communication with prediction fetcher to receive input and to send generated output.*

Sequence Diagram

https://lucid.app/lucidchart/50d8f73b-3f7d-4645-a7fc-6a69eb6abbaf/edit?invitationId=inv_c9dc3c8e-e947-4504-a80d-54fc4faa9ed3&page=0_0#



Design Rationale

Challenges Faced During Project Design:

- We had to decide on the architecture of the project. Which cloud service to use, what tools to include, cost of the solution, etc. This took a long time to research, explore different alternatives, and discuss with the client.
- Cloud service is yet to be provided from clients side, so temporarily we had to move to the local server alternative (Server running on a local system).

Past Alternatives considered for the design:

Backend Design (Model Prediction):

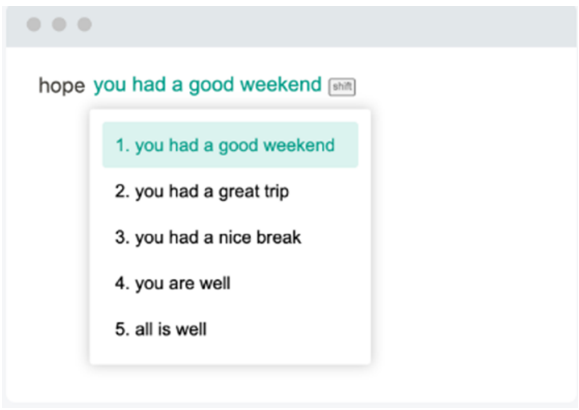
- Doing the model prediction locally on the user machine: This had a huge problem with the large download size of the model, and the computation is not possible on all machines as it requires higher system specifications. Then we decided to deploy the prediction model on some cloud services and started exploring them.
- Running the Model on Google Colab (Free Tier): The availability of resources was very uncertain; the model stopped running many times. Even Google Colab was not able to handle concurrent and multiple requests, so it was better to switch to other options.
- AWS: Amazon Web Services was the best alternative we found based on our research. It provides a variety of services and tools, like Amazon Sagemaker, the AWS Lambda function, and the S3 bucket, which collectively form an architecture for deploying an ML model online. The approximate cost was cheaper than the Google alternative for the same. AWS also provided a large range of tools in the free tier. Even though the market share and usability of AWS were greater than those of competitors, this made us choose AWS as the feasible solution.
- Google Cloud Platform (GCP): The service is almost similar to Amazon AWS but is more expensive and less reliable. GCP offers a limited number of functions per project as compared to AWS. The maximum execution time of GCP is approximately . 9 minutes, whereas for AWS it is 15 minutes for an instance.Supports a smaller number of programming languages than AWS. The cost per million requests for GCP is about $0.4, whereas for AWS it is $0.2.

- Microsoft Azure: The tools provided by Azure were quite limited compared to the other alternatives and very usable by applications, which was the reason we dropped the idea of considering Azure.

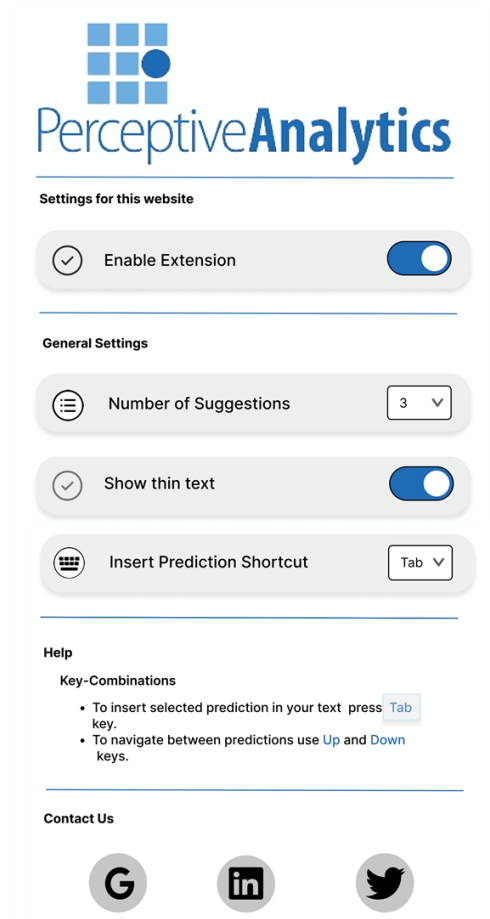
NOTE: AWS Lambda with S3 was the final cloud service preferred for the backend server deployment, but since clients didn't provide us the service so our current final design consists local backend server for now. This design will get updated if clients provide us AWS services in future.

Frontend Design(UI):

- We tried out various possible ways of showing the predictions inspired by various available extensions. The final decision was made based on the look and feel of the response. The final design chosen can be viewed below.



- The extension menu, which was finally chosen, is as follows:. It took multiple revisions based on the user inputs. We tried various alternatives based on the extension menus of other extensions like Grammerly.



All the work can be seen in the Architecture_Costing.pptx file in the same folder as this document.

https://iiitaphyd-my.sharepoint.com/:p:/g/personal/priet_ukani_research_iiit_ac_in/EdxgvFQbjyNBn-Tus1t8_DsBcMULqQpZfJgsgYTH_I3V7g?e=OGIfnf All the alternatives to the frontend as well as the backend design are mentioned here, along with the comparison and final solution.

Strengths and deficiencies of the final solution as compared to other solutions:

Right now our final solution consists local backend server which is due to inability of clients to provide AWS service. This solution is not as good as other cloud server alternatives in terms of computation and scalability, but if we focus on latency, then local server causes low latency in comparison to other remote server deployment.

Though in future if client provides AWS services, then we will change our design according to that, as scalability and computation are important aspects for this project.

Note: As this is a live document, this will be added in later versions once we switch to our chosen solution as the client provides us the functionality. Currently, we are running the prediction model locally and fetching responses from the server hosted by NGROK because we don’t have access to the cloud model.

Tradeoff: Cloud service is better for running models than locally on the user side, as less computation is required on the user side. Although it is difficult to deploy models on a cloud service and fetch predictions, it offers more latency than local deployment.