- # Overview
    - An android app for sharing jokes and memes
    - Top trending memes/jokes based on user interests will appear once user opens the app
    - Users can give reactions to the meme/joke: rofl, like, dislike
    - Users will have option to upload meme/joke and mention the related topics
    - Many more features

Motivation for the project:
The project involves

- Various frontend features: Implementation of UI design in **Flutter**
- Backend: Implementation of API calls in **Node.js,** integration of **Cloud SQL** server with **App Engine** to deploy the application
- **ML** and Ranking algorithms
- Something to learn for each team member

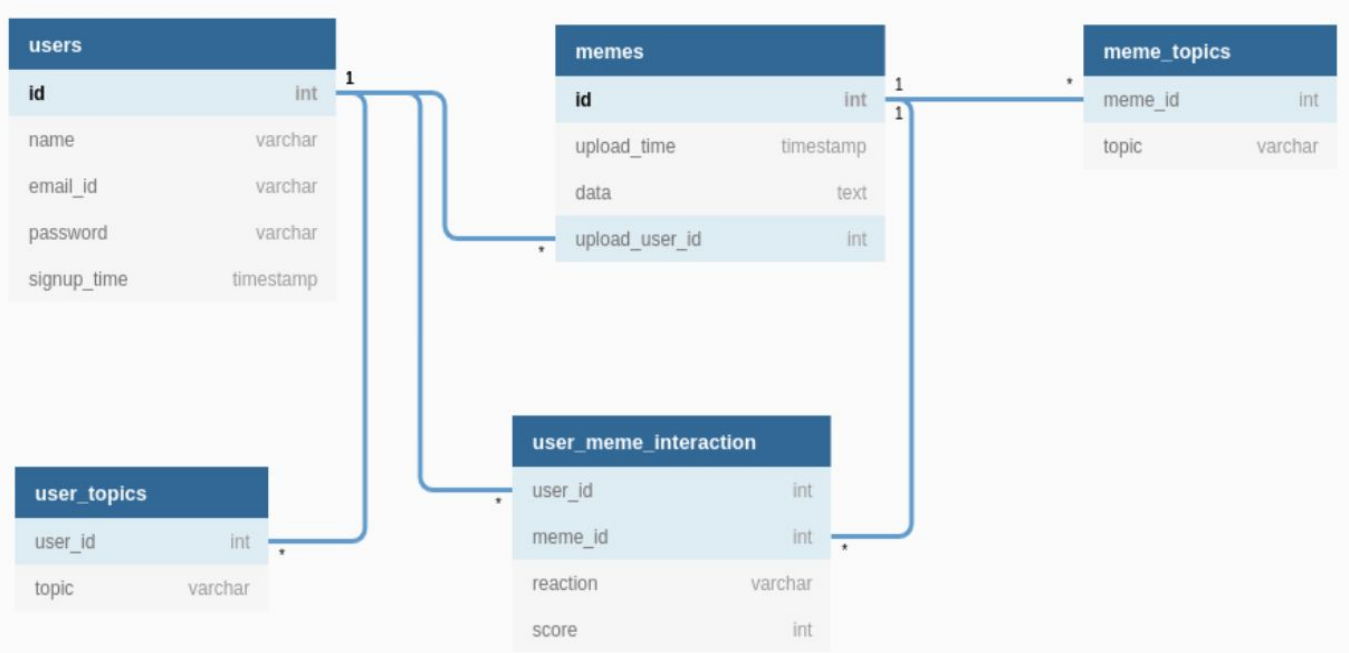
Overall a good learning experience and also fun to work with memes :p

- # UI
    - Frontend should meet 2 basic goals :-
        - Allows easy navigation between different pages of app and load pages smoothly
        - Encode user response as json and call Backend API using http package of flutter
    - Frontend has been implemented using flutter(dart language) but tested only on android. Main factors which drove us choose flutter were its same UI logics in all platforms and reduced code development time. It's fast reload proved very helpful as it reduced debugging time significantly.
    - Our app has these pages :-
        - Login page - Asks for user email and password
        - Signup page - Asks for all relevant details of user needed
        - Home page - User will be redirected to this page after login. One meme at a time will be shown to user and he/she can react to this meme.
        - Upload meme page - User can upload meme here. Large images will get compressed while uploading.

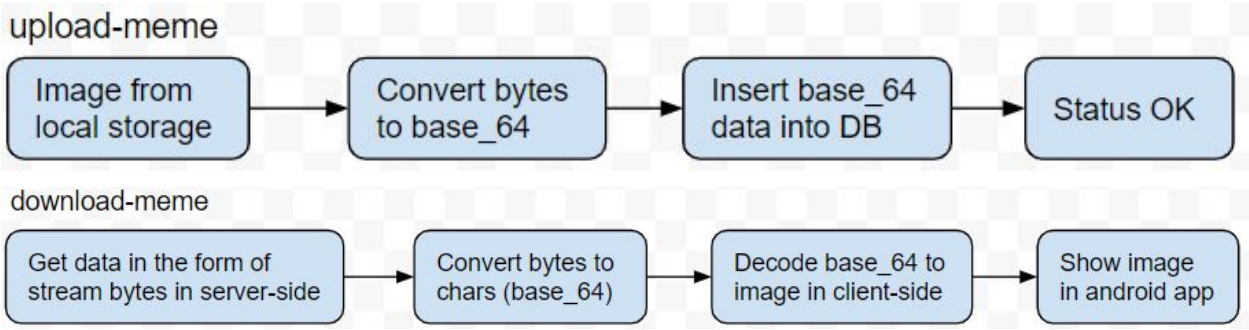
        - Profile page - Basic user info will be displayed on this page. User can also view and delete his own memes here.

- Web Infra/Backend
    - Backend should meet 2 basic goals :-
        - Serve client-side (Android app in our case) requests through REST API calls
        - Integrate Cloud SQL server with Google App Engine to deploy the application (both Node.js service for APIs and python service for ML)
    - Backend has been implemented in Node.js for its huge benefits including lightweight, scalability, library availability & efficient local testing and debugging
    - We have the following requests being served -
        - signup
        - signin
        - signout
        - upload-meme
        - download-meme
        - react-meme
        - delete-meme
        - delete-user
        - Download-my-meme

- Database schema



- Some key features :-
  - Password is stored in the form of hashed password not plaintext
  - Memes are stored in the database in the form of MEDIUMBLOB which allows us to store memes upto size 16MB
  - Memes are compressed to have fixed size for efficient storage in DB
  - Memes () are encoded in base_64 and timageshose data are stored in the DB. So, during, upload-meme and download meme, we have



- ML-Backend
  - Hot Ranking Algorithm

    Motivation:

    Need of ranking algorithm to give scores to jokes/memes based on user reactions, user topic interests, time of posting.

    The [Hot Ranking Algorithm used by Reddit] to display trending posts has been modified according to the following for this application.

    Initial Formula: `order(s) + sign(s) * seconds / 45000`

    ```
    where s=5*no_of_rofls+1*no_of_likes-1*no_of_dislikes,
          order(s)=log10(max(abs(s), 1))
          seconds = upload_date - start_date(when product was built)

** all weights would be modified according to performance
    ```

    Explanation of the formula:

    1. The first term calculates the score from the reactions on the joke/meme. As the function is logarithmic:
       a. The first reaction has the highest value
       b. The later reaction are getting less and less valuable
       c. For the older jokes/memes to stay on the trending, the bigger the number of reactions must be received
    2. The second term aims to give recent jokes/memes high scores

    Changes to incorporate user topic interests:

    - Suppose we have a user u interested in a set of topics t_u
    - Suppose we have a meme/joke m with topic tags set as t_m
    - Let h be the score we get from hot ranking algo
    - The number of topics tagged for meme/joke m: $n=|t_m|$
    - For meme m, the number of topics user u is interested: $n_i=|t_m \cap t_u|$
    - For meme m, the number of topics user u is not interested: $n_j=n-n_i$

    Final modified hot ranking algo score: $h*(n_i+0.1*n_j)/n$

- Collaborative Filtering

Motivation

Loads of recommender systems, like netflix, tinder etc use collaborative filtering to predict a user's expected interaction with a new item. So we used the user's past interactions with different memes to get a trained neural network model which predicts scores for each of the memes in the database.

Explanation

1. Made a dataframe entries corresponding to each of the user-meme interactions
2. Constructed a collaborative neural network model using embedding on this data.
3. Trained this model using the MSE loss b/w the actual and expected reaction as the loss function and ReLu as the activation function
4. Optimized the model parameters using the library torch.optim
5. Predicted scores for the given dataset and added them to the scores gotten using hot ranking (with appropriate weights) and updated them in the sql database

Important chunks of the code:

```
train_df=train_df[['user_id', 'meme_id', 'rating']]
train_dl = DataLoader(train_ds, batch_size=batch_size, shuffle=True)

num_users = len(train_df.user_id.unique())
num_items = len(train_df.meme_id.unique())

model = CollabFNet(num_users, num_items, emb_size=100)
model = nn.DataParallel(model)
model.to(device)

actual_y=train_df['rating'].values.astype(np.float32)
lrs = sltr(10, len(train_dl), eta_max=0.005)
final_scores= train(model, epochs=10, train_dl=train_dl, actual_y=actual_y, lrs=lrs)
```

Limitations and Future work

1. Model is trained for that particular instance of the database instead of incorporating just the new entries.

- Obstacles/Problems faced
  - Integrating the collaborative filtering code with the server's database due to some torch importing issues
  - Integrating ML with whole system and deploying in App Engine

- Future work
  - **Recommender systems (ML):**
    - Collaborative filtering: Model is trained for that particular instance of the database. Usually the database stays almost the same and only a few new entries of user_meme_interactions are made. So instead of the entire data being used for training, we can actually take care of just the new interactions.
    - Incorporate the time at which the meme gets uploaded. Score should decrease with time
  - Build classifier to classify memes instead of asking uploader to mention

- Contribution
  - Poorvi
    - Used collaborative filtering used by loads of recommender systems (eg: Netflix) to train on the basis of user-meme interactions
    - Constructed a nn model and optimized the trained model's parameters
    - Predicted scores for the given dataset and updated them in the database
    - Went through One Hot ranking algorithms too for integrating my code with that of Riya's
  - Riya
    - Found Hot Ranking algorithm used by Reddit and went through Collaborative Filtering Algorithm
    - Looked at various databases used for images to decide which one to use in this project
    - Created DB schema for the application
    - Wrote data generation script to generate random data for testing purpose
    - Wrote code for Modified Hot Ranking Algo
    - Wrote code to read from db and modify it according to the generated scores

        - Built a scheduler to update db with generated scores

    - Satya
        - Implemented backend part
        - Integrated Cloud SQL server and deployed Node.js service for API calls and python service for ML in App Engine

    - Shubham
        - Implemented frontend part
        - Collaborated with Satya to decide API designs (mainly handling errors and edge cases)

- Learnings
    - Learnt the way recommender systems using collab filtering work
    - Gained Experience working in gcloud platforms and pytorch implementations
    - Integration of code with that of others also taught us how to work in team projects.