PROJECT REPORT

LFS02: Social Cohesion

Team: Name 404

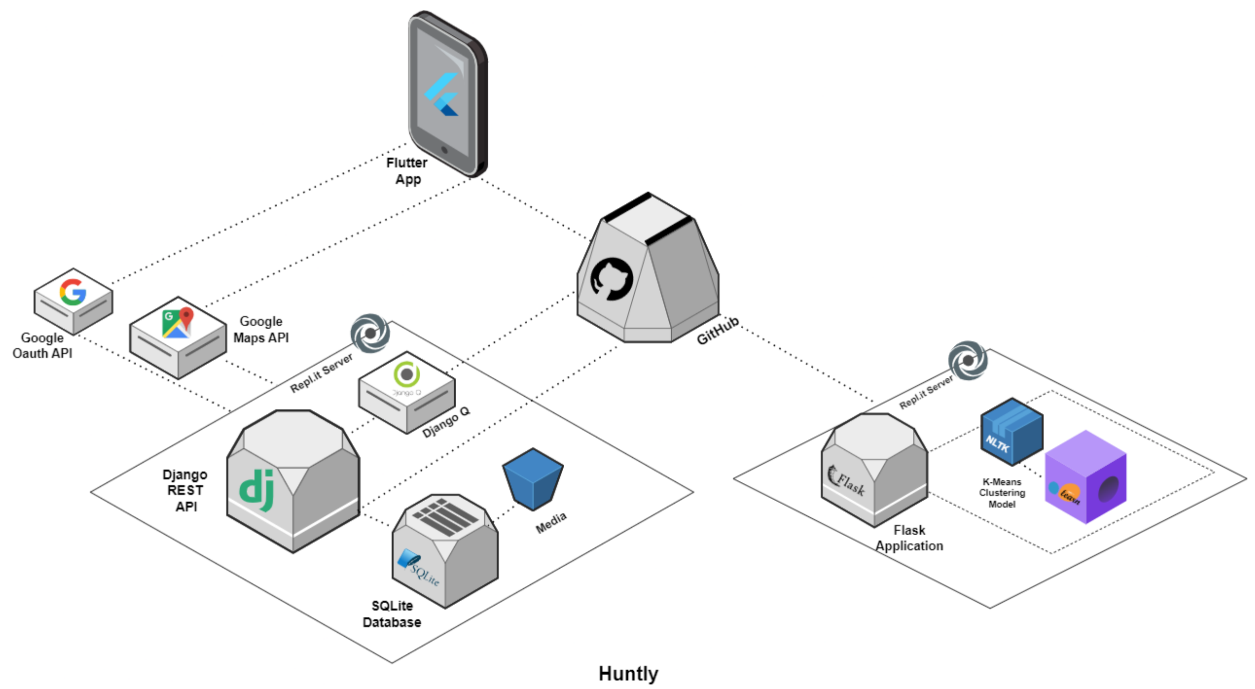
Parth Mittal
Pranav Agarwal
Mehul Todi
Abhiraj Mengade
Shashank SM

IDEA

Huntly

A cross-platform mobile application that brings people closer to the physical environment and forms meaningful connections by organising real-world Treasure Hunts for free and winning rewards. The app uses machine learning to match users and form teams of like-minded people. Users can choose from a wide variety of pre-set, themed clues or even create their own. Clues can be unlocked through QR code scanning or the answers are verified through location tracking. Users are encouraged to click pictures with their teammates and can view their pictures as memory threads.

NETWORK DIAGRAM



Huntly Network Diagram

.

SOFTWARES & TECHNOLOGIES

Frontend

Flutter (Dart)

DDD (Domain-driven design) Architecture

Where ? : The entire front-end is handled by flutter. It builds a cross-platform application (Android/IOS) of Huntly seamlessly.

Why ? : Flutter is Google's free, open-source software development kit (SDK) for cross-platform mobile application development, which caters rapid application development needs and allows easy UI development compared to other platforms such as Kotlin or React-native.

Packages :

1. flutter_bloc: for state management
2. google_sign_in: for client side google oauth, to get access_token
3. dio: for handling API requests
4. geo_locator: to get users current location in term of latitudes and longitudes
5. place_picker: provides a map interface to search and get longitudes and latitudes of any place on the map using google maps API.
6. qr_flutter: to generate a QR code for QR based clues
7. flutter_barcode_scanner: to scan a QR code and verify the QR based clues

Backend

Django REST Framework (Python)

MVC (Model–view–controller) Architecture

Where ? : We used the Django REST Framework to crest REST APIs and to query and manipulate the SQLite database connected to it.

Why ? : For creating Web APIs, the Django REST framework (DRF) offers a potent and adaptable toolset. Its biggest advantage is how much simpler serialization is as a result. Based on Django's class-based views, the REST framework is a great choice since we have

.

a Flutter based Android Application in the frontend.

Packages :

1. social-django: to add social auth, in our case Google OAuth2
2. rest-framework-swagger: for automatic documentation for APIs which makes the integration easier and less prone to errors.

Services

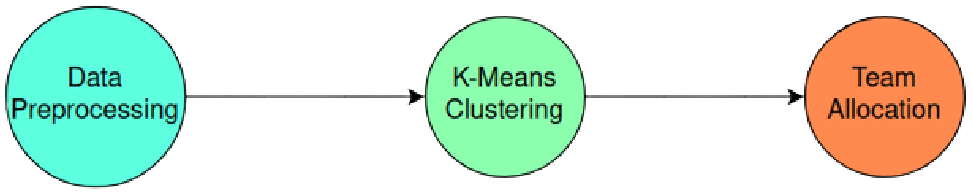
Flask, nltk, sklearn (Python)

Where ? : The ML model is used wher profile matching is needed for forming teams. Once users are registered for a hunt, our ML model assigns them teams through the use of K-means clustering algorithm based on their similarity of interests and bio,

Why ? : Flask is the best library for small scale API endpoint development. Deploying a server with a couple of endpoints is achieved speedily & efficiently with Flask.

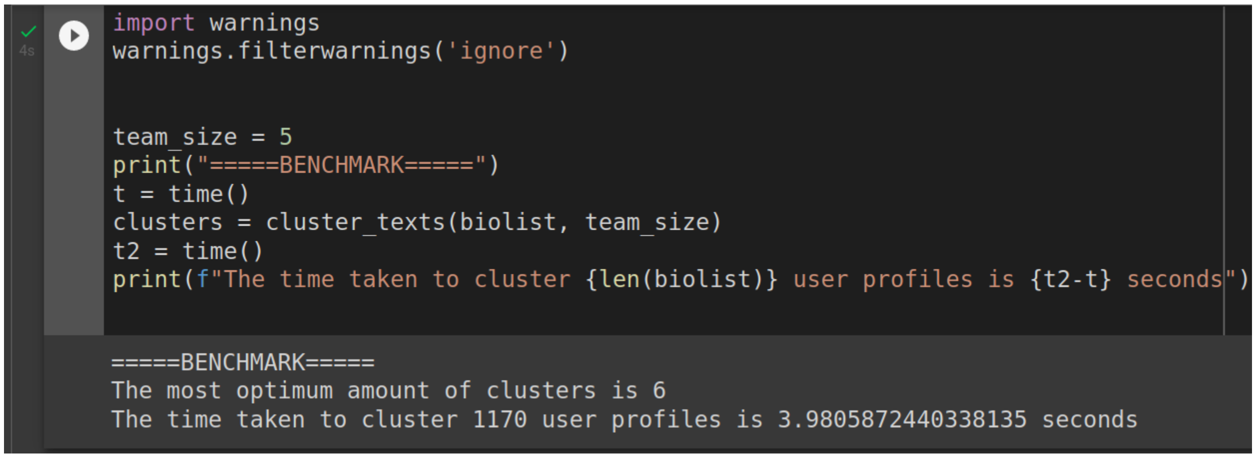
Why K-means clustering? K-means clustering guarantees convergence and quick adaptation. Hence, it proves the best clustering algorithm for our user profiles.

Model Workflow:



Benchmarking:

.



```
import warnings
warnings.filterwarnings('ignore')


team_size = 5
print("=====BENCHMARK=====")
t = time()
clusters = cluster_texts(biolist, team_size)
t2 = time()
print(f"The time taken to cluster {len(biolist)} user profiles is {t2-t} seconds")
```

```
=====BENCHMARK=====
The most optimum amount of clusters is 6
The time taken to cluster 1170 user profiles is 3.9805872440338135 seconds
```

Deployment & CI/CD

Replit

Where ? : We used replit for hosting and deploying our Django REST Framework backend and the ML API Endpoint on the web. Blazing fast CI/CD is also implemented through a replit.deploy.

Why ? : Compared to the other platforms, replit provides much ease-of-use and flexibility while deploying. Furthermore, replit offers a convenient pay-as-use pricing plan.

Scheduling

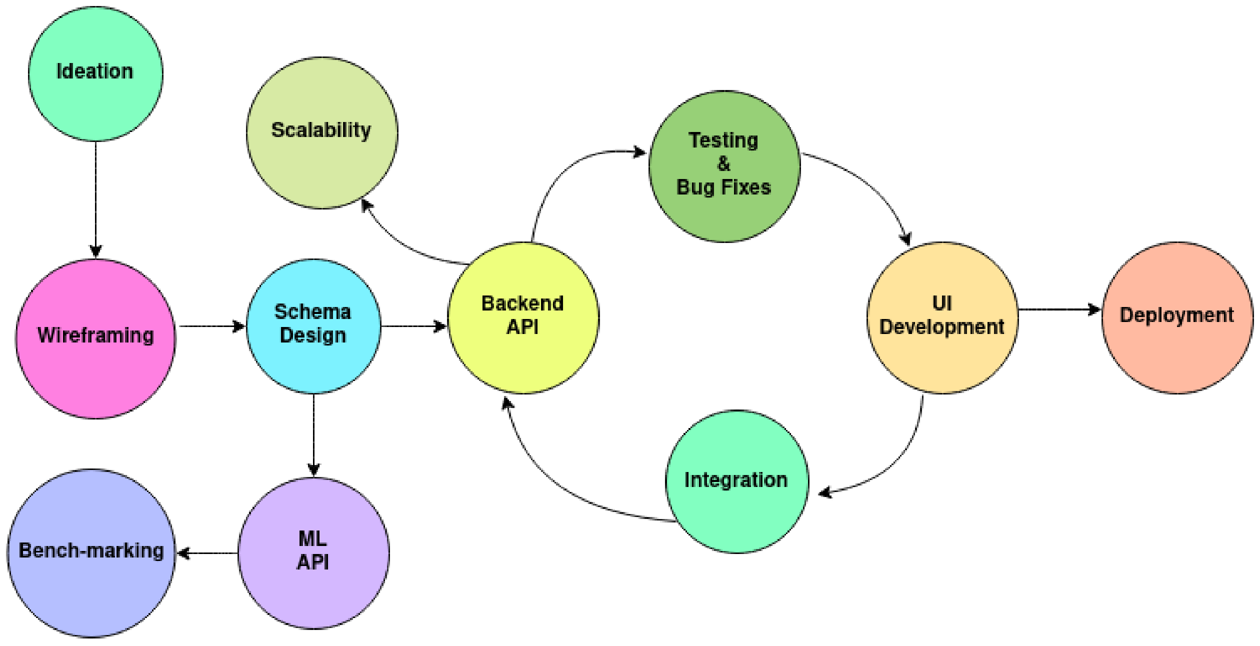
Django Q (Python)

Where ? : We use the django-q python package to schedule tasks in our DRF backend to perform periodic functions like creating teams before the treasure hut starts and distributing rewards after it is completed.

Why ? : It's simpler to set up and run in production compared to other alternatives like Celery. Django-Q can use just your existing database as a broker, which means you don't have to set up any new infrastructure.

PROCESS FLOW DIAGRAM

.



Process Flow Diagram

FEATURES

1. ### Authentication
   - Users can log in or sign up with Google OAuth.
   - The authentication is facilitated by django-social-auth.
2. ### Profile Builder
   - Users are then asked to enter some basic demographic details.
   - They are additionally asked for optional details regarding their tastes and preferred themes to help form teams of like-minded people using.

3. ### Search for Treasure Hunts
   - Users can search for already scheduled treasure hunts happening in their proximity.

4. ### Organize a Treasure Hunt

.

- Users can instead organize a treasure hunt and set up their preferences.
- Treasure hunts can be set as public or private:
  - Public - new and random users can join the treasure hunts and teams are chosen randomly.
  - Private - the treasure hunt is restricted to only some users that can be joined via an invite link. (yet to be implemented)
- They can set up geographical limits, the number of teams and the size of each team.
- Each treasure hunt can be organized in two ways:
  - Fully custom - the organizer has the ability to set up the theme, clues, the places where the clues will be set up and the rewards. Organizers can make their designed treasure hunt public which will then be added to our list of preset community hunts.
  - Preset - choose from a variety of already set up treasure hunts curated by our team for different themes and focus areas. (yet to be implemented)
- The clue verification can be done in two ways, the choice of which is left to the organizer's discretion.
  - QR code scanning: organizer needs to physically set up the QR codes in the required places that help verify the answer.
  - Location-based tracking: The answer to each clue is always a location. The teams need to reach the deciphered location and verify the same. Only upon successful verification, the next clue is unlocked, which is then displayed on the app. This is made possible using the Geolocator package by Flutter.

5. Gameplay

- Teams of like-minded people are formed, powered by a k-means clustering model. Participants can then view their team members' details, date, time, venue and theme of the treasure hunt.
- Each team member can view their clues on the app. Teams need to decipher the clues in order to unlock the next set of clues.
- The clue verification is done either through QR code scanning or location-based tracking, details of which have been outlined earlier.
- A leaderboard is maintained which is visible to all users at all times.

.

6. Rewards
   - The first team to decipher all the clues and reach the final destination will be declared the winning team. The treasure hunt is now declared finished.
   - All the winning team members are rewarded with discount coupons for popular brands.
   - Each organizer is also rewarded, with a higher reward for a custom hunt.

7. Memory threads
   - Users are encouraged to form new connections and click selfies with their friends.
   - These images are then stored in the app and can be viewed by the user as memory threads to capture their fun moments forever.

CHALLENGES FACED

1. Schedulers
   a. Motive: Form teams before the hunt starts with the help of our deployed ML service and distribute rewards after the hunt is completed.
   b. Problems: Had to trigger functions after a fixed period of time so we had to use schedulers. Tried to use the django-cron package but it was very buggy and deprecated
   c. Solution: Ended up replacing it with the django-q package which now runs a cronjob to form teams every 30 mins and another cronjob to distribute rewards every 10 mins.
2. Changing requirements
   a. Motive: Create schemas and APIs to fulfill all needs of the app based on the figma wireframe.
   b. Problems: Constantly changing requirements due to exploring new ideas
   c. Solution: Reiteration of schemas to cover all the requirements of the frontend dynamically
3. Breakdown of server due to CI/CD
   a. Motive: Implement CI/CD for the DRF backend deployed on Repl.it

.

   b. Problems: Backend crashed and repl got corrupted just before the second judging round.
   c. Solution: Quickly setup another repl and created dummy data and managed to complete the presentation successfully.

4. Docker and redis
   a. Motive: Add docker to improve scalability and redis to improve response time, reduce latency and optimize leaderboard.
   b. Problems: Dockerized the backend but while deploying realized that Repl.it does not support docker. Redis integration becomes harder as the process can no longer be automated in Docker.
   c. Solution: Not yet resolved.
5. Integration problems
   a. Motive: Seamless integration of flutter frontend with DRF APIs.
   b. Problems: Difficult to communicate all API details from backend team to frontend team.
   c. Solution: Added Swagger to DRF which generates automatic documentation due to class based views which eliminates any communication errors and simplifies the process.
6. Google OAuth
   a. Motive: Add Google Sign In in the Flutter App without using Firebase to reduce redundancy by storing user details in 2 places and reduce latency as the app only has to make calls to the DRF backend.
   b. Problems: Incomplete documentation and old blogs, so most of them didn't work.
   c. Solution: Compiled multiple resources and managed to implement Google sign in by hit and trial method.

FUTURE WORK

1. Optimize leaderboard using Redis
2. Option to delete / edit treasure clues
3. Improve QR code validation by sending a secret string from the backend rather than just generating qr code with clue id
4. Limit file types and size in memories and prevent inappropriate image uploads
5. Add filters and pagination for list treasure hunt page
6. Add option to download QR rather than just displaying QR code

.

7. Complete feature to create treasure hunt from presets

REFERENCES

1. https://medium.com/codex/google-sign-in-rest-api-with-python-social-auth-and-django-rest-framework-4d087cd6d47f (Django social auth and Google OAuth)
2. https://docs.djangoproject.com/en/4.1/ (Django)
3. https://mattsegal.dev/simple-scheduled-tasks.html (Django Q)
4. https://resocoder.com/2020/03/09/flutter-firebase-ddd-course-1-domain-driven-design-principles/ (DDD Architecture for Flutter)
5. https://pub.dev/packages/map_location_picker (Location picker)

.