Summer Fellowship Report

On

School Interface

Submitted by

Athmikha CDS

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

December 27, 2023

Acknowledgment

Under the mentor Mr. Sunil Shetye

I acknowledge the comprehensive report detailing the successful implementation of the school interface project. The document effectively captures the learning process, implementation strategies, and challenges faced during the integration of features such as message handling, user profile matching, and location-based matching. The clarity in explaining code optimization, API exploration, and test case development reflects a thorough understanding of the project's intricacies. The acknowledgment extends to the team's adept handling of conflicts encountered during integration, showcasing a commitment to ensuring a seamless and efficient final product. Well done on the meticulous documentation and successful project execution.

Contents

Chapter 1

Introduction

The presented report encapsulates a journey of comprehensive learning and adept implementation in the realm of school interface development. Delving into project setup, code implementation, and optimization strategies, the document meticulously chronicles the process of creating a robust model for handling messages and integrating it seamlessly within the existing framework. Through a demonstration project, the team showcases the efficacy of Django in managing user roles and messages, emphasizing both code reusability and system functionality. The report further delves into the intricacies of user profile matching, detailing the development and integration of a User Matching API with diverse matching algorithms. Exploring the nuances of location-based matching, the team addresses challenges faced during integration, demonstrating a commitment to resolving conflicts and ensuring a streamlined application. This report serves as a testament to the team's expertise in software development, API exploration, and meticulous documentation, marking a successful journey in enhancing the school interface system.

Chapter 2

Project set up Issue

2.1 Learning

Understanding how to efficiently implement a given flow chart on a model while efficiently utilizing existing code involves analyzing the flow chart, identifying relevant sections of the codebase, leveraging inheritance, following coding standards, modularizing the code, and thoroughly testing it.

2.2 Implementation

After going through the documentation and understanding the existing code, we created the ‘‘`Message`’’ model by leveraging inheritance from existing models. This optimized code reuse and efficiency by inheriting attributes like sender, recipient.

2.3 Github Url

https://github.com/Spoken-tutorial/school-Interface

Chapter 3

Demo project Implementation

During our demo project, we developed an API model using Django to create and manage users, teachers, parents, and messages. We implemented functionalities to add roles to existing users and delete roles from users. By utilizing the Django API, we effectively showcased the seamless interaction and functionality of these models, ensuring smooth user management within the system.

Chapter 4

Optimizing the code and Integration to actual project

4.1 Learning

Through my learning, I grasped the concept of abstract models as blueprints for specific models, facilitating code reuse. Inheritance allowed me to efficiently extend application functionality. Additionally, I learned to leverage foreign keys for establishing model relationships, enabling optimized data retrieval and improving overall application performance.

4.2 Implementation

A separate table was utilized to store the various if and else conditions, allowing for dynamic validation and easy modification of the business rules. Additionally, distinct models were created to handle different roles and message types, enabling efficient handling and processing of the message data based on their specific characteristics.

Chapter 5

Exploring API Building test cases

5.1 Learning

I acquired the skill of generalizing code initially designed for specific roles. I delved into the workings of various APIs used for sending OTPs, specifically focusing on the Indian context and understanding the process of obtaining shortcodes and sender IDs for SMS-based OTPs. Furthermore, I gained knowledge in writing test cases and implementing them in the test.py file, enabling comprehensive testing of the code's functionality.

5.2 Implementation

During the integration of the demo project with existing ones, we encountered the need to adapt the code to accommodate the pre-existing models. To optimize application speed, we modified the input mechanism from terminal-based to accepting JSON format via the body of the GET method when requesting the URL. We also extensively implemented diverse test cases for the message class, ensuring comprehensive test coverage.

Chapter 6

Working on UserCases

6.1 Documentation

The document details various attribute combinations and their associated matching criteria for user profiles. The goal was to establish robust match algorithms for identifying duplicate or matching user profiles based on specific attributes. Each attribute combination underwent meticulous assessment to determine its suitability for matching profiles. Factors such as name, email, phone, ID, date of birth (DOB), username, and location were evaluated for their matching potential. For instance, the Name Email combination exhibited a high matching certainty (100%) with a 90% fuzzy matching threshold, allowing minor variations. On the other hand, the Name Date of Birth (DOB) combination showed a moderate 60% matching criterion, considering potential duplicates arising from common names and shared birthdate.

6.2 Learning

The User Matching API provides a mechanism to compare user profiles based on several attributes like name, location, email, phone number, and username. It calculates similarity scores between user profiles using various matching algorithms.

Chapter 7

Implementation - Matching algorithms

7.1 Location Matching

7.1.1 Purpose

Compares the geographical locations of two users.

7.1.2 Process

Extracts the address, city, district, and state information of each user.Utilizes a token-based matching algorithm to assess the similarity between the addresses. Considers exact matches for city, district, and state names.

7.1.3 Outcome

Computes a similarity score based on these factors, contributing to an overall similarity assessment between user locations.

7.2 Username Matching

7.2.1 Purpose

Assesses the likeness between usernames.

7.2.2 Process

Compares the usernames of two users using a token-based matching algorithm.

7.2.3 Outcome

Produces a similarity score based on the comparison of usernames.

7.3 Email Matching

7.3.1 Purpose

Compares email addresses of user profiles.

7.3.2 Process

Employs a partial string matching algorithm to assess the resemblance between email addresses.

7.3.3 Outcome

Calculates a similarity score based on the comparison of email addresses.

7.4 Phone Number Matching

7.4.1 Purpose

Evaluates the similarity between phone numbers.

7.4.2 Process

Compares the phone numbers of two users using an exact matching approach.

7.4.3 Outcome

Generates a similarity score based on the exact match or mismatch between phone numbers.

Chapter 8

Final Outcome

Send a POST request to /match/ with user IDs to compare their profiles. The API responds with either "matched" or "not matched" based on the calculated similarity scores.

Profiles with similarity scores above the 0.5 threshold are considered a match.

Chapter 9

LocationMatchTestCases

This class is part of the test suite for the application's matching functionality and specifically tests the location matching feature within the User Matching API. Preparing Request Data: Constructs a POST request data containing the IDs of two users to be compared for location matching.

9.1 Sending Request

Utilizes the Django test client to send a POST request to the specified API endpoint ('/accounts/match/') with the formulated request data in JSON format.

9.2 Assessing Response

Evaluates the response received from the API call.

9.3 Assertions

Ensures the response status code is 200, indicating a successful API call. Verifies that the response contains the string 'matched', indicating a positive location match between the two users.

Chapter 10

Integration of Location Matching Feature

During the integration of the new location matching feature into the existing project, we aimed to enhance the matching functionality for user profiles.

- Adapting the codebase to seamlessly integrate with the existing models and frameworks.
- Optimizing the application's efficiency by transitioning from a terminal-based input mechanism to accepting JSON data via the body of the GET method for URL requests.
- Creating comprehensive test cases designed explicitly to validate the accuracy of location matching within user profiles.
- Implementing robust test cases specifically tailored for the location matching aspect of user profiles.

Conflicts Faced: During the integration process, we encountered several conflicts while merging the new location matching feature with the main project. These conflicts stemmed from discrepancies in code changes made simultaneously in different branches, leading to divergent versions that couldn't be automatically merged.

The issues mainly revolved around conflicting modifications in similar sections of code, which resulted in ambiguous references and overlapping changes. Resolving these conflicts required a careful manual review, identifying discrepancies, and making decisions on which changes to retain, modify, or discard.