Google Developer Student Club

Inductions 2025

Deadline: 7th February 2025 (Friday) 11:59 PM IST

Instructions:

- ***You must choose only a single task from the options provided.***
- ***This is an individual task; collaboration leads to immediate disqualification.***
- ***Create a GitHub account here.***
- ***Good Coding Practices: Naming Conventions, Meaningful Variable and File Names, Appropriate Folder Structure, Code Readability, and Comments are encouraged.***
- ***Each task is self-contained; interpretations may vary, and there is no one "answer" to a single problem.***
- ***Make sure to add a README file in your GitHub repository consisting of:***
  - ***Brief task description***
  - ***Related screenshots***
  - ***Instructions to run the project***
  - ***If hosted, its URL***
- ***Submit the code even if the task is incomplete.***
- ***Create a public repository for your task and submit the link in the form.***
- ***Task prioritization underscores the emphasis on challenging assignments, with evaluation leaning towards recognizing and rewarding diligent efforts in handling more intricate tasks over simpler ones.***

For any queries, contact:

Vasudha (+91 75177 18822)	**Varad (+91 99677 21357)**	**Prince (+91 99739 38636)**
Web Development	**AI/ML**	**Web Development**

Web Development

Task-1: Front-End (Beginner)

LEETCODE SOLUTION VIEWER

Problem Statement:

Develop a web application that allows users to browse solutions for LeetCode problems by integrating with the GitHub API and fetching solutions from the haoel/leetcode GitHub repository.

Key Features:

- Search Functionality: Enable users to search for solutions using problem names or numbers (e.g., "Two Sum" or "1").
- Solution Display: Fetch and display available solutions from GitHub, categorized by programming language (Python, Java, C++).
- Code Snippet Viewer: Present syntax-highlighted code snippets and allow users to toggle between different programming languages.
- Responsive Design: Ensure a seamless user experience across desktop and mobile devices.

Useful Resources: To fetch the README

Task-2: Front-End (Advanced)

HABIT STREAK COUNTER

Problem Statement:
Create an application that helps users track daily habits and monitor their progress over time.

Key Features:

- Habit List: Users can maintain a list of habits and mark them as completed.
- Streak Tracker: A visual tracker (e.g., progress bar or calendar heatmap) to indicate habit consistency.
- Reset Mechanism: Automatically reset streaks when a habit is missed.
- Reminders (Optional): Provide notifications or reminders to encourage consistency.

Useful Resources:

1. Basics - Learn HTML & CSS – Full Course for Beginners
2. Frontend Development Guide: React Course - Beginner's Tutorial for React JavaScript Library [2022]
3. React full course Playlist - React playlist

Task-3: Back-End (Intermediate)

MARKDOWN TO HTML CONVERTER API/CLI

Problem Statement:
Create a service that converts Markdown files into well-formatted HTML output.

Key Features:

- File Upload: Accept Markdown files via POST requests.
- Conversion Engine: Parse and return clean, formatted HTML.
- Enhancements (Optional):
  - Inline styling support (light/dark themes).
  - Caching mechanism using Redis.

Tech Stack:

- Backend: Node.js, Express (or Python with Flask)
- Libraries: marked (for Markdown parsing)

Useful Resource: NodeJs tutorial **HTML TO MD(python)** **Node js Lib**

Task 4: Back-End (Advanced)

WEATHER AND NEWS AGGREGATOR API

Problem Statement:

Develop an API that retrieves real-time weather data and the latest news for a given city, consolidating information from multiple sources.

Key Features:

- User Query Handling: Accepts city name as input.
- Data Fetching: Retrieves
  - Weather Information from OpenWeatherMap/Weatherstack.
  - Top News Articles from NewsAPI/New York Times.
- Optimizations (Optional):
  - Data caching with Redis to minimize API calls.
  - User authentication via JWT/OAuth for personalized preferences.

Tech Stack:

- Backend: Node.js, Express.js
- APIs: OpenWeatherMap, NewsAPI
- Database (Optional): MongoDB (for storing user preferences and cache data

Useful Resources:

1. Learn to build APIs

Task-5: Full-Stack (Beginner)

BLOG APPLICATION

Problem Statement:

Develop a blog application supporting two user roles—Authors and Viewers—with role-based access control.

Key Features:

- Authentication System: Viewers can register/login to become authors.
- Blog Management:
  - Authors can create, edit, and manage blogs.
  - Viewers can browse and read published blogs.
- Secure API Endpoints: Implement authentication using JWT/OAuth.
- Deployment (Bonus): Deploy the application on a free hosting platform (e.g., GitHub Pages, Vercel, Heroku).

Tech Stacks:

- Frontend: React, or traditional HTML/CSS/JavaScript
- Backend: Node.js (Express), Flask, FastAPI or Django
- Database: MongoDB or PostgreSQL

Resources:

- Django: Django Blog Project | Part 1
- MERN: MERN Stack

Task-6: Full-Stack (Advanced)

CONTRIBUTION TO OPEN-SOURCE PROJECT - SummaRise

Problem Statement:

Enhance an existing GitHub project ‘SummaRise’ which summarizes users' emails using Google OAuth, Gmail API, and Google Gemini. It summarizes a user's emails by fetching them through Google OAuth & Gmail API and then processes the content using Google Gemini. The task involves solving issues like creating a frontend or developing additional REST API endpoints to enhance the application’s functionality.

Key Contributions:

- Frontend Enhancements:
    - Design an intuitive UI for displaying email summaries.
    - Develop authentication status, error handling, and settings pages.
    - Start with dummy email summaries for initial development.

- Backend Enhancements:
    - Develop REST API endpoints to retrieve summarized emails.
    - Implement secure authentication (JWT, OAuth).
    - Optimize email fetching and summarization.

- Integration & Deployment:
    - Ensure smooth frontend-backend communication with REST APIs.

    - Address CORS issues and ensure seamless cross-platform compatibility.
    - Follow open-source contribution best practices (forking, branching, pull requests).

Tech Stacks:

- Frontend: React, HTML, CSS, JavaScript
- Backend: Flask (Python), Node.js
- Authentication: Google OAuth, JWT

Resources:

- GitHub Repository: https://github.com/Princekumarofficial/SummaRise
- Video Tutorial: Untitled design (1).mp4.
- Flask Tutorial: Flask Tutorials - YouTube
- REST API and full stack tutorial : Python + JavaScript - Full Stack App Tutorial
- Frontend Development Guide:
    - React Course - Beginner's Tutorial for React JavaScript Library [2022]
    - React Tutorial

Note:

This task is moderately challenging and encourages you to showcase your full-stack skills. Try to solve as many issues as possible, and submit a pull request with detailed notes about your contributions.

Machine Learning

Task-1: (Level-1)

RECOMMENDER SYSTEM WITH VARIOUS FILTERING TECHNIQUES

Problem Statement:

Design and implement a recommender system using different recommendation algorithms such as **Collaborative Filtering**, **Content-Based Filtering** and **Hybrid Filtering**. The recommender system should be able to suggest relevant products, services, or content to users based on their preferences and behaviours.

Key Features:

- Collaborative Filtering (CF): Implement user-item collaborative filtering using techniques like Matrix Factorization (e.g., SVD) or K-Nearest Neighbors (KNN).
- Content-Based Filtering: Use item features (e.g., genres, categories, descriptions) to recommend items that are similar to those the user has shown interest in.
- Hybrid Filtering: Combine collaborative and content-based methods to enhance the recommendation accuracy.
- Evaluation Metrics: Evaluate the performance of the recommender using metrics like **Precision, Recall, F1-Score, and Root Mean Squared Error (RMSE)**.
- Deployment: Deploy the recommender system on a website.

Learning Objectives:

- Understanding the different filtering techniques and their advantages.
- Working with real-world datasets to implement and test recommender systems.
- Fine-tuning models and optimizing for better recommendation quality.
- Bonus points for implementing chainlink VRF for randomness generation and selecting a winner.

NOTE: You can choose any domain/field for building the recommender system

Task-2: (Level-2)

CHATBOT WITH RAG + VOICE FEATURES

Problem Statement:

Create an intelligent chatbot that integrates Retrieval-Augmented Generation (RAG) for answering queries and Voice Interaction for a hands-free experience. The chatbot will retrieve relevant information from an external knowledge base and then generate responses based on the retrieved context.

Key Features:

- RAG (Retrieval-Augmented Generation): Integrate a search or retrieval model that pulls in relevant documents or data from a knowledge base and then uses a language model to generate human-like responses.
- Voice Interface: Implement speech-to-text and text-to-speech for voice interaction, allowing the user to speak to the chatbot and get audio responses.
- Deployment: Deploy the chatbot on a website.

Learning Objectives:

- Deep understanding of RAG and its role in improving chatbot performance.
- Integration of speech recognition and synthesis.
- Working with LLMs and deploying AI solutions in real-world applications.

Resources:

- **Streamlit Documentation**

Google Developer Student Club, IIT Indore