Senior AI Engineer Assignment

Pulsegen Technologies
Duration: 24 hours

Task

Take Google Play Store reviews from June 2024-till date for Swiggy/Zomato/Any Popular app of your choice. Daily data should be treated as a batch. Assume starting from June 1st, 2024, everyday you receive data with reviews.

Your AI agent should consume the daily data and create a trend analysis report for issues, requests and feedback in the reviews.

Output Requirements

The report should be a table where:

- **Rows:** Topics (related to issues, requests, feedback, etc.)
- **Columns:** Dates from T to T-30
- **Cells:** Frequency of topic occurrence on that date

The output report for date T should show a trend from T-30 to T.

Example Report Structure:

Topic	Jun 1	Jun 2	Jun 3	...	Jun 30
Delivery issue	12	8	15	...	23
Food stale	5	7	3	...	11
Delivery partner rude	8	12	6	...	9
Maps not working properly	2	4	7	...	5
Instamart should be open all night	1	0	3	...	4
Bring back 10 minute bolt delivery	0	2	1	...	6

Topic Requirements

Example topics for Swiggy:

- Delivery issue
- Food stale
- Delivery partner rude
- Maps not working properly
- Instamart should be open all night
- Bring back 10 minute bolt delivery

You can provide seed topics to start but ensure your agent identifies new evolving topics and creates new categories as needed.

Technical Constraints

Required: Use Agentic AI approaches. Traditional approaches like TopicBERT or LDA are not acceptable due to accuracy limitations.

Key Challenge: Similar but not identical topics being created as different categories, which creates inaccurate trends.

Example Problem:

- "Delivery guy was rude"
- "Delivery partner behaved badly"
- "Delivery person was impolite"

These should be consolidated into single topic: "Delivery partner rude"

Solutions: Implement high recall agents or use deduplication approaches. Consider taxonomy or ontology-based methods.

Input/Output

Input: App store link and target date
Output: Trend analysis report table as specified above

Deliverables

1. **Private GitHub Repository** - Share with `vatsal@pulsegen.io`
2. **Video Demonstration** - Upload to Google Drive
3. **Sample Output Reports** - Include in `/output/` folder

Evaluation Focus

The AI agent should have high recall to ensure relevant topics have valid trends. End users (product teams) will consume this report to identify most trending topics and new evolving issues.

Submission: 24 hours from assignment start

Video Upload, Sensitivity Processing, and Streaming Application Assignment

Overview

Build a comprehensive full-stack application that enables users to upload videos, processes them for content sensitivity analysis, and provides seamless video streaming capabilities with real-time progress tracking.

Project Objectives

Core Functionality

1. **Full-Stack Architecture**: Develop using Node.js + Express + MongoDB (backend) and React + Vite (frontend)
2. **Video Management**: Implement a complete video upload and secure storage system
3. **Content Analysis**: Process videos for sensitivity detection (safe/flagged classification)
4. **Real-Time Updates**: Display live processing progress to users
5. **Streaming Service**: Enable video playback using HTTP range requests
6. **Access Control**: Implement multi-tenant architecture with role-based permissions

Technical Requirements

Backend Implementation

- **RESTful API Design**: Create endpoints for video operations
    - Video upload with metadata handling
    - Video listing with filtering capabilities
    - Streaming service with range request support
- **Content Processing**: Implement a video sensitivity analysis mechanism
- **Real-Time Communication**: Use Socket.io for live progress updates
- **Database Management**: Store video metadata, processing status, and user data in MongoDB
- **Authentication & Authorisation**: Secure API endpoints with proper validation

Frontend Development

- **Upload Interface**: User-friendly video upload with progress indicators
- **Real-Time Dashboard**: Dynamic display of processing status and progress
- **Video Library**: Comprehensive list of uploaded videos with status indicators
- **Media Player**: Integrated video playback for processed content

- **Responsive Design**: Cross-platform compatibility and intuitive user experience

Advanced Features

Multi-Tenant Architecture

- **User Isolation**: Each user accesses only their own video content
- **Data Segregation**: Secure separation of user data and permissions
- **Scalable Design**: Support for multiple organisations or user groups

Role-Based Access Control (RBAC)

- **Viewer Role**: Read-only access to assigned videos
- **Editor Role**: Upload, edit, and manage video content
- **Admin Role**: Full system access, including user management and system settings

Video Processing Pipeline

1. **Upload Validation**: File type, size, and format verification
2. **Storage Management**: Secure file storage with proper naming conventions
3. **Sensitivity Analysis**: Automated content screening and classification
4. **Status Updates**: Real-time progress communication to the frontend
5. **Streaming Preparation**: Video optimisation for efficient streaming

Project Structure Requirements

Organization

- **Clear Separation**: Distinct backend and frontend directories
- **Modular Architecture**: Organised code structure with separation of concerns
- **Configuration Management**: Environment-specific settings and variables
- **Error Handling**: Comprehensive error management and user feedback

Development Standards

- **Clean Code**: Well-commented and maintainable codebase
- **Version Control**: Clear git commit history with meaningful messages
- **Documentation**: Comprehensive setup and usage instructions
- **Testing**: Basic testing implementation for critical functionalities

Stretch Goals (Optional Enhancements)

Advanced Filtering

- **Content-Based Filtering**: Filter videos by safety status (safe/flagged)

- **Metadata Filtering**: Search by upload date, file size, duration
- **Custom Categories**: User-defined video categorisation system

Performance Optimization

- **Video Compression**: Automatic optimisation for different quality levels
- **Caching Strategy**: Implement efficient caching for frequently accessed content
- **CDN Integration**: Content delivery network for improved streaming performance

Deliverables

Application Requirements

1. **Functional Demo**: Complete working application showcasing full workflow
   - Video upload process
   - Real-time processing updates
   - Content sensitivity analysis
   - Video streaming capability
2. **Code Quality**: Professional-grade implementation
   - Clean folder structure
   - Proper separation of concerns
   - Comprehensive error handling
   - Security best practices
3. **Documentation Package**:
   - Installation and setup guide
   - API documentation
   - User manual
   - Architecture overview
   - Assumptions and design decisions
4. **Deployment**:
   - Publicly accessible web application
   - Video demonstration of functionality
   - GitHub repository with complete source code

Workflow Demonstration

Complete User Journey

1. **User Registration/Login**: Secure authentication system
2. **Video Upload**: Intuitive upload interface with progress tracking
3. **Processing Phase**: Real-time updates on sensitivity analysis
4. **Content Review**: Status display (safe/flagged) with appropriate actions
5. **Video Streaming**: Seamless playback of processed videos
6. **Management Tools**: Video library with filtering and management options

Technical Specifications

Backend Technology Stack

- **Runtime**: Node.js (Latest LTS version)
- **Framework**: Express.js
- **Database**: MongoDB with Mongoose ODM
- **Real-Time**: Socket.io
- **Authentication**: JWT or similar secure token system
- **File Handling**: Multer or similar for video uploads

Frontend Technology Stack

- **Build Tool**: Vite
- **Framework**: React (Latest stable version)
- **State Management**: Context API or Redux
- **Styling**: CSS Modules, Styled Components, or Tailwind CSS
- **HTTP Client**: Axios or Fetch API
- **Real-Time**: Socket.io client

Infrastructure Requirements

- **File Storage**: Local storage or cloud storage (AWS S3, Google Cloud)
- **Video Processing**: FFmpeg or similar video processing library
- **Hosting**: Cloud platform (Heroku, Netlify, Vercel, or similar)
- **Database Hosting**: MongoDB Atlas or similar cloud database service

Success Criteria

Functional Requirements Met

- ✅ Complete video upload and storage system
- ✅ Real-time processing progress updates
- ✅ Video sensitivity analysis and classification
- ✅ Secure video streaming with range requests
- ✅ Multi-tenant user isolation
- ✅ Role-based access control implementation

Quality Standards Achieved

- ✅ Clean, maintainable code structure
- ✅ Comprehensive documentation
- ✅ Secure authentication and authorisation
- ✅ Responsive and intuitive user interface
- ✅ Proper error handling and user feedback
- ✅ Public deployment with demo video

Pulse - Module Extraction AI Agent

Overview

Create an AI-powered streamlit application that extracts structured information from documentation-based help websites. The goal is to identify key modules and submodules and generate detailed, accurate descriptions for each—entirely based on the content from the URLs provided.

Let me introduce you to **modules** and **submodules** of a product. Each product can be broken down into multiple modules which are managed by product managers. And further these modules can be broken down into submodules. Let's take an example of Instagram, instagram is a product used by billions of people. Instagram can be broken down into multiple modules that define instagram, like 'posting an image', 'reels', 'stories', 'manage account', etc. And all these modules have their own submodules that have some specific task/functionality to perform. For example, 'Manage Account' is a module that provides you the option to 'delete your account', 'activate/deactivate account'. Similarly, for 'reels' you can 'create reels', 'edit reels', 'share reels' and these all become the submodules of 'reels' module.

Requirements

Core Functionality:

1. Accept one or more help documentation URLs as input (e.g., https://help.instagram.com/)
2. Automatically crawl and process the content from all relevant pages linked within the URL(s)
3. Identify and extract:
   - Key modules representing major documentation categories
   - Submodules under each module
   - Detailed descriptions for both modules and submodules
4. Return a structured JSON output in the specified format

Technical Requirements

Input Handling

- Accept one or more URLs via command-line or function input
- Validate all input URLs
- Recursively crawl relevant documentation pages
- Handle edge cases like redirects, broken links, and unsupported formats

Content Processing

- Extract only meaningful documentation content, excluding headers, footers, and navigation
- Maintain content hierarchy based on document structure and layout
- Handle various HTML content formats such as text, lists, and tables

- Normalize content into a consistent internal representation

Module/Submodule Inference
- Infer top-level modules from sections or primary topics in the documentation
- Group logically connected subtopics under each module as submodules
- Automatically generate detailed and accurate descriptions for each using extracted content only

Output Format

```
{
    {
      "module": "Module_1",
      "Description": "Detailed description of the module",
      "Submodules": {
        "submodule_1": "Detailed description of submodule 1",
        "submodule_2": "Detailed description of submodule 2"
       }
    },
    {
      "module": "Module_2",
      "Description": "Detailed description of the module",
      "Submodules": {
        "submodule_1": "Detailed description of submodule 1",
        "submodule_2": "Detailed description of submodule 2"
       }
    }
}
```

Example Usage

```
# Start the module extractor
python module_extractor.py --urls https://help.instagram.com

# Sample Output
{
    {
      "module":"Account Settings",
      "Description": "Includes features and tools for managing Instagram account
preferences, privacy, and credentials.",
      "Submodules": {
        "Change Username": "Explains how to update your Instagram handle and display
name via account settings."
      }
    },
    {
      "module": "Content Sharing",
      "Description": "Covers tools and workflows for creating, editing, and
publishing content on Instagram.",
      "Submodules": {
        "Creating Reels": "Provides instructions for recording, editing, and sharing
short-form video content using Reels.",
        "Tagging Users": "Details how to tag individuals or businesses in posts and
stories for engagement."
      }
    }
}
```

Evaluation Criteria

Accuracy & Structure (40%)
- Correct identification of modules and submodules
- Logical grouping and consistency
- Quality and precision of generated descriptions

Technical Implementation (30%)
- Intelligent use of structure and content cues to extract hierarchy
- Resilient crawler and parser
- Efficient data processing pipeline

Code Quality (15%)
- Modular, maintainable architecture
- Clean code and consistent style
- Proper error handling and logging
- Clear documentation and comments

Visual Demonstration - Max. 5 mins (15%)
- Include screen recordings and screenshots showing:
  - Input URLs passed to the tool
  - Console or UI output with structured results
  - Visual confirmation that the tool successfully processes documentation

Bonus Points

1. **Advanced Features**
    - Support for multiple documentation sources
    - Answer caching mechanism
    - Support for different documentation formats
    - Confidence scores for answers
2. **Technical Improvements**
    - Docker containerization
    - API endpoint addition
    - Performance optimizations

Notes

1. You may use any programming language, but Python is preferred
2. Document any third-party libraries used
3. Include any assumptions made during implementation
4. Note any limitations of your approach

Submission Requirements

1. **GitHub Repository**
   - Private repo with code and a clear README
   - Include setup instructions, usage examples, design rationale, and known limitations
2. **Documentation**
   - Technical architecture description
   - Notes on approach, assumptions, and edge case handling
3. **Testing**
   - Sample inputs and expected outputs **(Run your code on atleast 4 different URLs)**

Here are some B2B companies help docs, you can test your code on any B2B company.

1. https://support.neo.space/hc/en-us
2. https://wordpress.org/documentation/
3. https://help.zluri.com/
4. https://www.chargebee.com/docs/2.0/

   - Handling of deep nesting or sparse content

4. **Visuals**

   - Upload a video explaining your approach (**Max. 5 mins**) to the repository or link externally (Google Drive, Loom, etc.)

Time Expectation

48 hours for a functional and clean implementation.

Pulse Coding Assignment

Objective:

Develop a script to scrape product reviews from G2 and Capterra for a specific company and time period. Your script should be able to accept a company name, start date, end date, and source as inputs, and output a JSON file containing the reviews. For bonus points, identify and integrate a third source for SaaS app reviews.

Task Details:

1. **Script Requirements:**
   - **Input Parameters:**
     - **Company Name:** The name of the company whose product reviews are to be scraped.
     - **Start Date:** The beginning of the time period for the reviews.
     - **End Date:** The end of the time period for the reviews.
     - **Source:** The source of the reviews, which can be either `G2` or `Capterra`.
   - **Output:**
     - The script should output a JSON file containing an array of reviews. Each review should include:
       - `title`: The title of the review.
       - `description/review`: The text content of the review.
       - `date`: The date the review was posted.
       - Additional information: Any other relevant details associated with the review (e.g., reviewer name, rating).
2. **Script Functionality:**
   - The script should be able to:
     - Scrape reviews from the specified source based on the company name and time period.
     - Parse the reviews and format them into the required JSON structure.
     - Handle pagination if necessary to collect all reviews within the specified time period.
     - Validate and handle errors gracefully (e.g., invalid company names, out-of-range dates).
3. **Bonus Points:**
   - Identify a third source that specializes in SaaS app reviews and integrate it into your script.
   - Ensure the script can handle this third source with the same functionality as for G2 and Capterra.

4. **Evaluation Criteria:**
    - **Time:** Efficient execution of the script.
    - **Code Quality:** Clean, well-commented, and maintainable code.
    - **Novelty:** Originality in approach or techniques used.
    - **Output:** Accuracy and completeness of the output data.

Submission Instructions:

Please submit your code along with any necessary instructions for running it, a sample JSON output, and your README file. If you integrated a third source for bonus points, include details on how to use it in your README.

Deadline:

Submit your assignment within 48 hours.

Good luck, and we look forward to reviewing your solution!