TURTIL INTERNSHIP

PROJECT: ENGAGEMENT INSIGHT ENGINE - An AI-based microservice.

INTERNSHIP DURATION 19TH MAY – 27TH JUNE.

SUBMITTED BY:

- YUVARAJ MADUGU,
  - Software Engineer - Intern
  - B.TECH - 3TH YEAR (UG).
  - Department of Computer Science Engineering - specializing in Artificial Intelligence and Machine learning.
  - Kommuri Pratap Reddy Institute of Technology, Ghatkesar, Telangana.

SUBMITTED TO:

- TURTIL - from TECHBRICKS PVT LTD.
  - Under the guidance of mentor@turtil.co
  - Mr.RAJ SHANMUGAM sir – HEAD MENTOR GROUP.

STATEMENT OF CERTIFICATE:

This is to certify that the project titled "ENGAGEMENT INSIGHT ENGINE" is a genuine and original work carried out by Mr. YUVARAJ MADUGU, under the guidance and supervision of Mr. RAJ SHANMUGAM – HEAD MENTOR, as part of the requirements of TURTIL - Software engineer Intern, during the period MAY 2025 – JULY 2025.

WORKMAIL: yuvarajm@turtilintern.com

PERSONAL MAIL: yuvarajmadugu@gmail.com

TABLE OF CONTENTS

ABSTRACT:

The Engagement Insight Engine is an intelligent, AI-driven microservice designed to deeply understand and improve user engagement on online learning and professional growth platforms. In modern upskilling ecosystems, many users sign up enthusiastically but gradually lose momentum, missing opportunities to showcase their skills, build connections, and achieve their goals. This engine addresses that problem by analyzing each user's profile and activity, comparing them to their peer group, and delivering smart, personalized nudges that encourage them to stay active and visible. By combining deterministic rules and machine learning, the system empowers platforms to guide users toward completing their profiles, uploading resumes, attending events, and actively engaging in projects — ultimately helping users "glow up" their presence.

At the core of the system is a hybrid logic engine that integrates rule-based and ML-based decision-making. The rule-based component, configured through config.json, enforces explicit engagement policies, such as suggesting a resume upload if a user's resume is missing while their batch's upload rate is high, or prompting event attendance if a user hasn't joined recent activities. This deterministic logic ensures critical nudges are always prioritized, providing a robust baseline of suggestions. When further nudges are needed, the ML layer — trained using logistic regression on synthetic engagement data — predicts additional actions that could maximize each user's success, like encouraging project participation or quiz attempts.

The entire system is architected as a FastAPI microservice that runs fully offline and is Docker-ready for seamless integration and deployment. The service accepts rich JSON inputs containing detailed user profile data, recent activity metrics, and peer snapshots, then returns a structured list of up to three high-priority nudges tailored to that user. These nudges are designed to be immediately actionable, each with a clear title, motivational message, and priority label to guide users step by step. The engine also provides a user-friendly Swagger UI interface for easy testing and API exploration, and logs every major request in real-time for transparency and analysis.

Beyond its technical features, the Engagement Insight Engine stands out for its reproducibility, scalability, and configurability. All data — including the 2,000 synthetic user profiles and peer snapshots — is generated offline to avoid leakage and ensure privacy. Model reports, feature importance plots, and detailed logs provide full visibility into system behavior. The modular design, customizable configuration, and seamless containerization make it easy for any team to adopt and adapt. By combining advanced analytics with human-centered design, the engine helps platforms transform passive users into proactive contributors, ultimately building stronger, more engaged learning communities.

INTRODUCTION:

In today's rapidly evolving digital learning and professional development world, user engagement is one of the most critical drivers of success. Despite enrolling in programs or joining learning platforms with high enthusiasm, many users struggle to stay consistently active, leading to incomplete profiles, missed events, and underutilized opportunities. Platforms often face challenges understanding which users need help, what actions to recommend, and how to motivate them effectively without overwhelming or annoying them.

The **Engagement Insight Engine** was designed to solve this gap by providing a powerful, intelligent microservice that analyzes individual user data, compares it against peer groups, and delivers smart, personalized nudges to guide users back on track. It does this through a hybrid approach that combines deterministic, rule-based logic with predictive machine learning models, ensuring a balance of reliability and adaptability. The system offers nudges such as encouraging resume uploads, prompting project submissions, motivating quiz attempts, or suggesting event participation, all prioritized to maximize user impact.

Technically, the engine is built using FastAPI and is designed to be fully offline and containerized with Docker, making it highly flexible and easy to integrate into existing platforms. It starts by ingesting rich user and peer data in JSON format and processes it using configurable rules defined in config.json, followed by machine learning predictions where needed. The final output is a set of up to three actionable, prioritized nudges presented in a clear and friendly format. With a clean API design, an interactive Swagger UI for testing, and thorough logging, the system is both developer-friendly and transparent.

Overall, the Engagement Insight Engine not only empowers platforms to drive higher engagement and profile completeness but also helps users build confidence, showcase their skills, and avoid missing out on valuable learning opportunities. By turning raw data into tailored, human-like guidance, this engine bridges the gap between passive participation and active, meaningful growth.

SYSTEM OVERVIEW:

The **Engagement Insight Engine** operates as an intelligent microservice that transforms raw user data into personalized, high-impact recommendations, helping users stay engaged and make meaningful progress on a platform. From data ingestion to delivering final nudges, the system follows a carefully designed, step-by-step process to ensure each suggestion is relevant and actionable.

The journey begins when a user's data is sent to the engine through a FastAPI endpoint. This data is structured as a JSON payload and includes three major parts: the user's profile (resume status, karma, goals, projects, quizzes, clubs, and buddy count), their recent activity metrics (login streaks, events attended, posts created, interactions with buddies), and a snapshot of their peer group's engagement behavior (batch-level resume upload rates, average projects, event attendance, and buddies participating in events). This rich context allows the engine to understand each user's position relative to their peers.

Once received, the data first flows through a **deterministic rule-based engine**, which applies explicit conditions defined in config.json. This layer checks for clear gaps, such as a missing resume when the peer group's upload rate is high, or low event attendance compared to active buddies, and immediately generates high-priority nudges. This ensures that obvious, critical recommendations are never missed and are always given priority.

If there's still room to provide more nudges (with a maximum of three per user), the system moves to the **machine learning engine**. Here, pre-trained logistic regression models analyze user behavior patterns to predict additional opportunities for improvement — for example, encouraging more project submissions or suggesting quiz participation. Once all nudges are identified, they are prioritized and compiled into a structured JSON response. Each nudge includes a type (like resume or event FOMO), a clear title, an actionable message, and a priority ranking to guide the user effectively.

Finally, the system logs every major step for transparency and debugging, and it returns the response in real time to the client platform. The entire engine runs fully offline and can be containerized using Docker, ensuring seamless deployment across environments. With its hybrid logic, configurable rules, and intelligent fallback predictions, the Engagement Insight Engine provides a powerful, reliable, and customizable solution to increase user motivation and drive meaningful platform engagement.

METHODOLOGY:

The **Engagement Insight Engine** follows a structured, hybrid methodology that combines rule-based logic and machine learning predictions to deliver personalized nudges. At the center of this approach is FastAPI, which seamlessly integrates with Pydantic models to handle data validation and transformation, ensuring robustness and clarity throughout the process.

When a user or client application interacts with the engine via **/analyze-engagement** endpoint in Swagger UI, they are presented with a predefined JSON schema. This schema is automatically generated from Pydantic models defined in **models.py**. These models specify the exact structure and data types required for each part of the request, including fields such as user_id, profile attributes (e.g., karma, resume status, goals), activity metrics (e.g., login streaks, last event attended), and peer_snapshot metrics (e.g., batch resume upload percentage, event attendance rates). When the "Try it out" button is used in Swagger UI, FastAPI internally uses these Pydantic models to parse the input, validate its correctness (types, required fields), and provide clear error messages if data is missing or malformed.

After validation, the parsed data enters the core logic layer of the system. First, the **rule-based logic** — configured via **config.json** — checks deterministic conditions. For example, if the user's resume has not been uploaded while 80% of peers have done so, or if the user hasn't attended an event in over a set number of days, immediate high-priority nudges are generated. These rules are carefully designed to address clear, easily identifiable engagement gaps without needing predictions.

If fewer than three nudges are generated from deterministic rules, the system then activates the **machine learning layer**, which uses pre-trained logistic regression models (stored as model_resume.pkl, model_event.pkl, etc.). This layer predicts additional areas of improvement for the user based on profile features and historical behavioral patterns, further enhancing personalization. The combination ensures that deterministic business priorities are always respected, while data-driven suggestions fill any remaining gaps.

Finally, all suggested nudges are compiled into a structured JSON response. Each nudge includes a type (such as resume, event, project, quiz), a user-friendly title, an actionable recommendation, and a priority level to clearly guide the user. The entire flow is logged for transparency, and the response is returned in real time, immediately visible in Swagger UI or to any client consuming the API. This hybrid approach guarantees that each user receives targeted, high-impact suggestions that are both reliable and dynamically personalized.

WORKFLOW:

- User fills JSON input in Swagger UI → validated by Pydantic models.
- Data passed to hybrid-based logic → high-priority deterministic nudges generated.
- ML fallback predictions fill up to 3 nudges total.
- Nudges structured and prioritized → returned as JSON response.
- Logs recorded → transparent and debuggable.

IMPLEMENTATION:

The **Engagement Insight Engine** was carefully designed in a modular, scalable, and fully offline-ready manner to ensure high flexibility, reproducibility, and performance. The system was built as a microservice using FastAPI, chosen for its simplicity, speed, and automatic generation of interactive API documentation (Swagger UI). Its microservice architecture supports easy integration into larger platforms, allowing individual components to be tested, maintained, and deployed independently. The entire workflow was split into clear tasks and structured phases, each focused on enabling smooth and transparent engagement analysis.

Technology Stack:

The backend service is implemented using **FastAPI** for API development, **scikit-learn** for building and training machine learning models, and **Pydantic** for request/response schema validation. Synthetic data generation and event scoring were handled using custom Python scripts. **Docker** was used for containerization to ensure portable, reproducible deployments, and **joblib** was used to persist trained models. Logging was implemented using Python's standard logging module to capture runtime insights and API interactions.

Modular Architecture:

The project follows a highly modular structure. Separate files manage data simulation (simulate_data.py), dataset preparation (generate_training_dataset.py), rule and score logic (event_fomo_score.py, config.json), model training and evaluation (train_model.py), and API schemas (models.py). Each module was designed to be reusable and easily extensible. The configuration file (config.json) centralizes thresholds and rule parameters, allowing quick adjustments without modifying the core logic.

ML Nudging Logic:

The core nudging logic consists of a hybrid approach combining deterministic rule-based checks and machine learning predictions. The rule-based layer enforces strict conditions, such as suggesting resume uploads if peer upload rates are high and the user hasn't uploaded, or detecting event FOMO based on the time since the last attended event. If fewer than three nudges are generated, the ML models trained on synthetic data predict additional personalized suggestions. The models were trained using logistic regression and evaluated through detailed classification reports and feature importance plots, ensuring strong interpretability and accuracy.

API Development:

RESTful endpoints (/analyze-engagement, /health, /version, /) were implemented using FastAPI. Input payloads are strictly validated with Pydantic models to prevent type mismatches or missing fields, and example schemas are embedded directly into the Swagger

UI for easy testing. The engine accepts a nested JSON payload with user profile data, activity, and peer snapshot, then returns a structured list of up to three prioritized nudges.

All major API requests are logged to improve monitoring and debugging during development and deployment.

Containerization with Docker:

A custom **Dockerfile** was created to containerize the entire service, ensuring platform-independent deployment. The resulting Docker image is lightweight and optimized for production environments, allowing the microservice to run seamlessly on local servers or cloud-based infrastructures. Requirements are specified in requirements.txt to guarantee reproducible builds.

Configuration & Logging:

All thresholds, rule parameters, and prioritization labels are managed centrally via config.json, enabling rapid tuning for different engagement strategies. Detailed logs are automatically created, capturing major API calls and logical decision points to assist with audits and debugging.

Deployment Readiness:

The engine is fully containerized, tested, and optimized for deployment on both local and cloud infrastructures. A simple Docker build and run process allows immediate deployment, and API versioning support ensures backward compatibility. Clear endpoint documentation and schema examples in Swagger UI make integration straightforward for client platforms. Together, these design choices make the Engagement Insight Engine highly reliable, transparent, and ready for real-world use cases where motivating user activity and community participation are key priorities.

TECHNOLOGIES USED:

The **Engagement Insight Engine** was developed using a thoughtfully chosen **set of modern technologies and tools**, organized to ensure clarity, maintainability, and smooth deployment. Each component was selected to balance high performance, easy integration, and offline capability.

Programming Language:

- **Python 3.9+**: Chosen as the primary language for backend development, data simulation, feature engineering, and integration of machine learning models.

Framework & API Development:

- **FastAPI**: A high-performance, modern Python framework used to build the RESTful API endpoints, enabling rapid development and automatic documentation through Swagger UI.
- **Uvicorn**: An ASGI server used to run the FastAPI application efficiently in both development and production environments.
- **Pydantic**: Employed for robust data validation and parsing. It ensures strict type checking for request payloads and helps define clear response schemas.

Machine Learning & Scoring:

- **scikit-learn**: Core library used for training machine learning models, such as logistic regression classifiers for nudging predictions, and for feature importance analysis.
- **joblib**: Used to serialize (save) and deserialize (load) trained ML models, making the deployment process faster and more reliable.

Supporting Libraries:

- **os**: Utilized from Python's standard library to manage file paths and environment-specific configurations across modules.
- **logging**: Integrated for capturing detailed runtime logs, monitoring API calls, and supporting debugging during both development and production.
- **json**: Used for reading and managing configurations in config.json as well as handling custom data formatting.
- **datetime**: Essential for processing time-based activity metrics, such as calculating days since the last event.

Containerization & Deployment:

- **Docker**: Employed to containerize the entire microservice along with its dependencies, ensuring a consistent and reproducible environment across local and cloud deployments.
- **Dockerfile**: Custom script defining build instructions for creating a lightweight, production-ready container image.

Testing & Validation:

- **Swagger UI (via FastAPI)**: Provides an interactive interface for live API testing, allowing developers and non-technical stakeholders to validate payloads and view real-time responses.
- **Manual test scenarios & logs**: Additional test flows and example inputs were created to validate the model behavior and rule logic under various user scenarios. All interactions are logged for transparency and traceability.

FILE STRUCTURE:

```
projectCode/
├── main.py
├── models.py
├── config.json
├── event_fomo_score.py
├── generate_training_dataset.py
├── processed_fomo_dataset.csv
├── simulate_data.py
├── simulated_profile.json, peer_snapshot.json
├── train_model.py
├── models/
│   └── model_resume.pkl
│   └── model_event.pkl
├── reports/
│   └── *.csv, *.png
├── requirements.txt
├── Dockerfile
└── README.md
```

KEY FILES:

- **main.py**
  This is the entry point for the FastAPI service. It handles endpoint definitions (/health, /version, /analyze-engagement), integrates rule-based logic and ML predictions, manages logging, and orchestrates the generation of user nudges.
- **models.py**
  Defines Pydantic data models for request and response schemas. Ensures strict validation of incoming JSON payloads and formats outgoing API responses clearly and consistently.
- **config.json**
  Central configuration file containing thresholds, rules for different engagement dimensions (resume, projects, events, quizzes), and priority labels. This allows the system to be easily tuned without changing code logic.
- **event_fomo_score.py**
  Contains functions to calculate event FOMO (fear of missing out) scores, analyze peer event attendance, and generate insights on user event participation gaps.
- **generate_training_dataset.py**
  Script to process raw simulated profiles and peer snapshots into a structured dataset (processed_fomo_dataset.csv) used for training machine learning models. Extracts relevant features and target labels.
- **processed_fomo_dataset.csv**
  The final dataset generated from simulated data, ready for training ML models. Includes user features and flags indicating nudging requirements.
- **simulate_data.py**
  Generates synthetic user profile data and peer snapshot data (simulated_profile.json, peer_snapshot.json). Supports fully offline data generation, avoiding leakage and ensuring reproducibility.
- **simulated_profile.json**, **peer_snapshot.json**
  Generated datasets representing user profiles and peer group snapshots. Used to simulate real-world engagement behavior for training and testing.
- **train_model.py**
  Handles training of machine learning models (e.g., logistic regression classifiers for resume and event nudges). Also produces evaluation reports and feature importance analyses stored in the reports/ directory.

Folders:

- **models/**
  Stores the serialized ML model files (model_resume.pkl, model_event.pkl) saved using joblib. These models are loaded during API runtime to generate fallback nudges.
- **reports/**
  Contains reports generated during model training, including classification metrics, feature importance CSVs, and visualization plots. Useful for debugging and improving model interpretability.

Other Files

- **requirements.txt**
  Lists all Python dependencies needed to run the project, ensuring consistency when installing packages locally or in Docker containers.
- **Dockerfile**
  Defines instructions to build a container image for the microservice. Supports easy, reproducible deployment across different environments.
- **README.md**
  The main documentation file summarizing project purpose, setup instructions, API usage guide, example payloads, and deployment commands.

PSEUDOCODE / ALGORITHM:

Main Flow:

Begin

1. Load pre-trained machine learning models (model_resume.pkl, model_event.pkl) and rule configurations from config.json.
2. Receive user engagement data via API (/analyze-engagement endpoint):
    - User profile data
    - Activity data
    - Peer snapshot data
3. Validate and parse input using Pydantic models.
4. Initialize an empty list of nudges.
5. Apply deterministic rule-based logic:
    - Check if resume upload is needed.
    - Check for project-related nudges.
    - Check for quiz-related nudges.
    - Calculate event FOMO score and evaluate event attendance.
    - Add nudges to the list if rule conditions are met.
6. If nudges < 3:
    - Prepare user feature vector.
    - Pass feature vector to pre-trained ML models.
    - Predict additional nudging probabilities.
    - Add highest priority ML-predicted nudges to fill up to 3.
7. Prioritize and sort final nudges.
8. Build JSON response containing nudges (type, title, action, priority) and status.
9. Return the response to the client.
10. Log request and results for monitoring and debugging.

End

Rule-Based Logic

Algorithm: apply_rule_based_nudges
Input: user_profile, user_activity, peer_snapshot, config
Output: nudges_list

1. If resume_uploaded is False and batch_resume_uploaded_pct > config threshold:
    - Add resume nudge to nudges_list.
2. If projects_added < config minimum projects and batch_avg_projects is high:
    - Add project nudge.
3. If last_event_attended is older than config days or buddies are attending events:
    - Calculate event FOMO score.
    - If score > threshold, add event FOMO nudge.
4. If quiz history is empty and relevant conditions met:
    - Add quiz nudge.
5. Return nudges_list.

Event FOMO Score Calculation

Algorithm: calculate_event_fomo_score
Input: user_activity, peer_snapshot, config
Output: fomo_score

1. Calculate days_since_last_event.
2. Evaluate buddies_attending_events count and batch event attendance.
3. Compute normalized score combining these factors.
4. Return fomo_score.

Machine Learning Predictions

Algorithm: predict_ml_nudges
Input: user_feature_vector, ml_models
Output: ml_nudges_list

1. Load pre-trained ML models for resume and event nudging.

2. Pass user_feature_vector to each model.
3. Obtain probabilities for needing each type of nudge.
4. If probability > configured threshold, add corresponding nudge to ml_nudges_list.
5. Return ml_nudges_list.

Final Response Generation

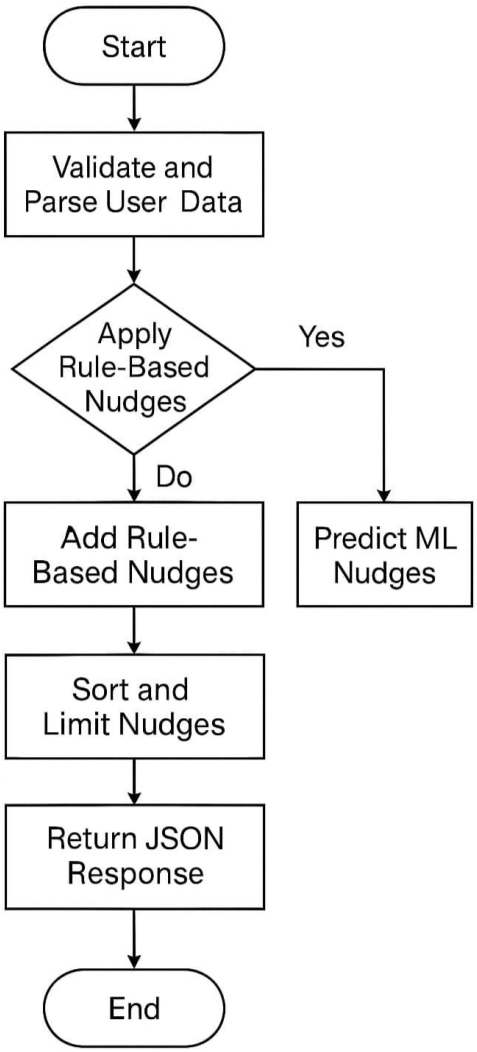
Algorithm: build_nudge_response
Input: nudges_list
Output: json_response

1. Sort nudges_list by priority.
2. Limit list to maximum of 3 nudges.
3. For each nudge, include:
    - Type (e.g., "resume", "event_fomo", "project", "quiz")
    - Title
    - Action message
    - Priority number
4. Create JSON object with status = "success" and nudges array.
5. Return json_response.

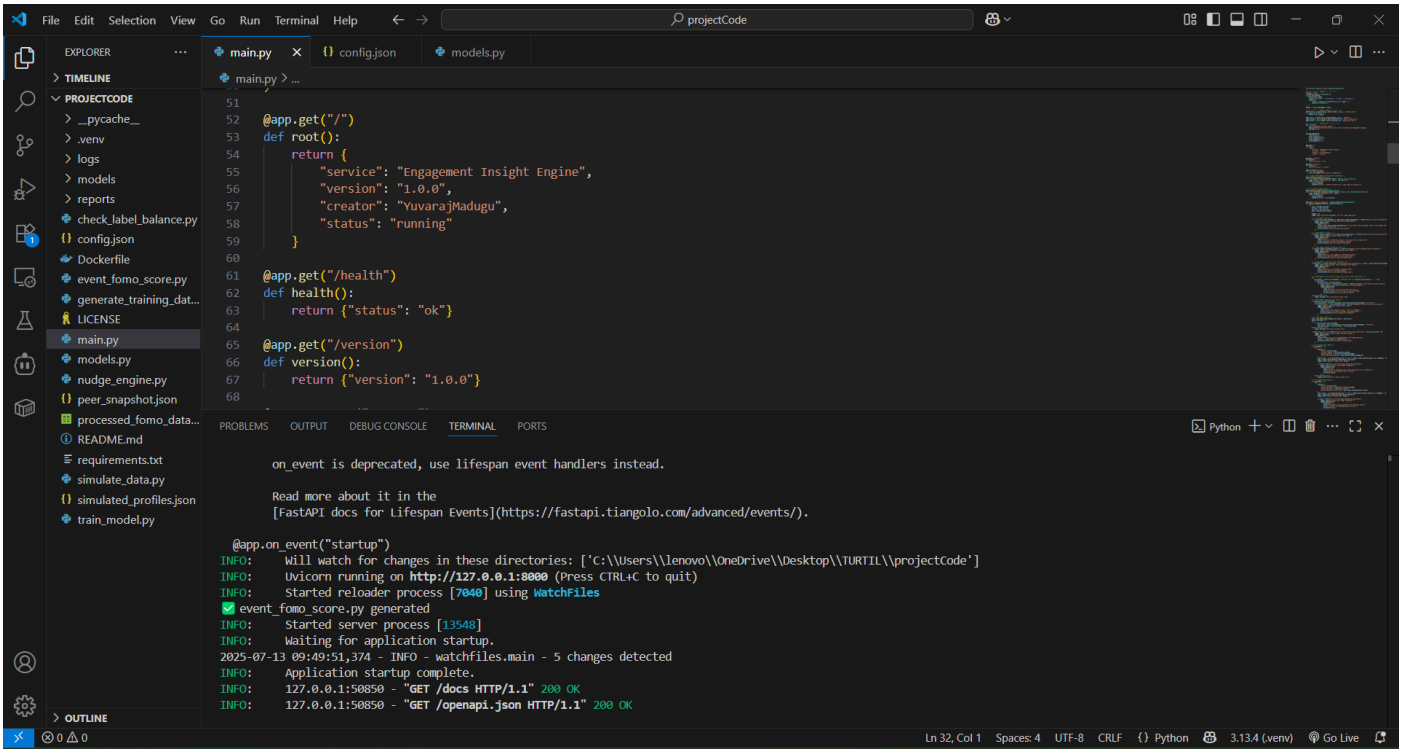
Summary of Flow:

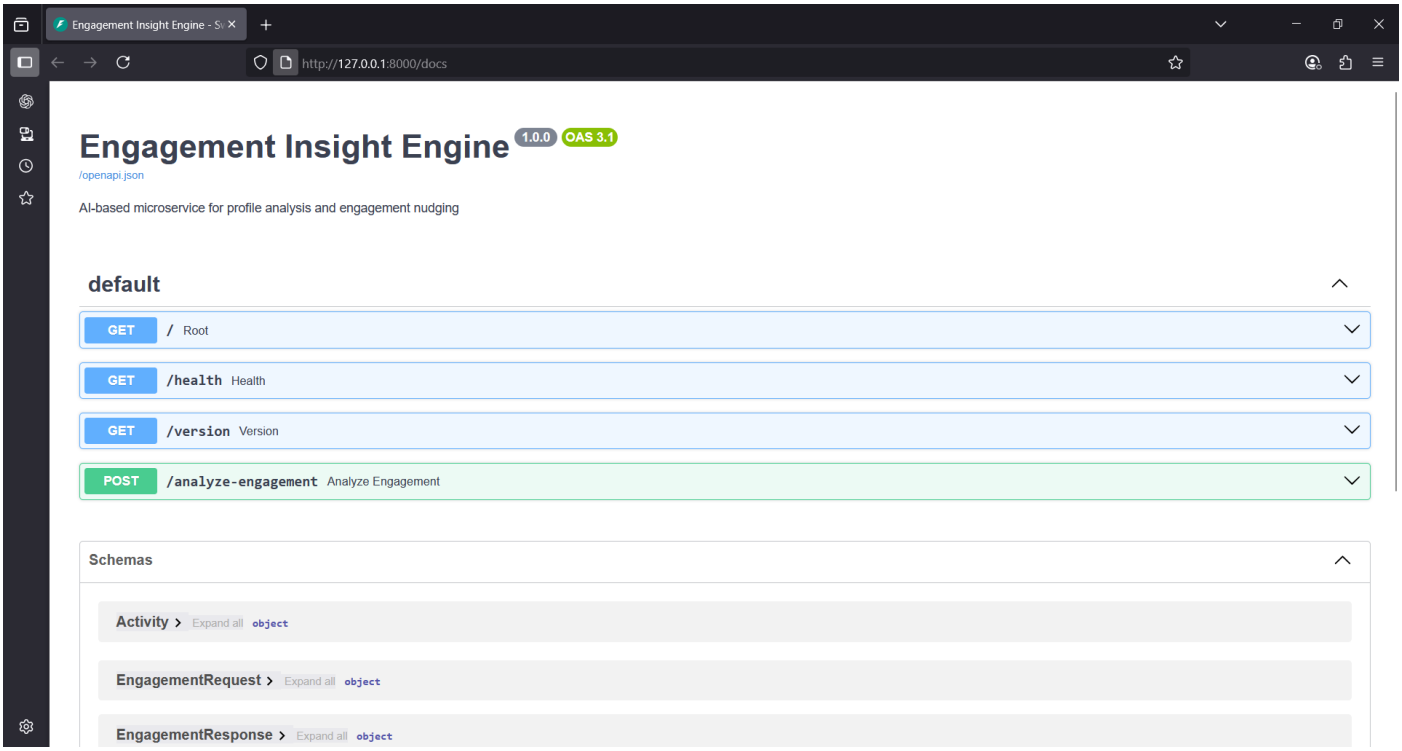
- Validate and parse user data.
- Apply deterministic rule-based nudges first.
- Fill gaps using ML predictions if needed.
- Assemble and sort final nudges.
- Return structured JSON response.

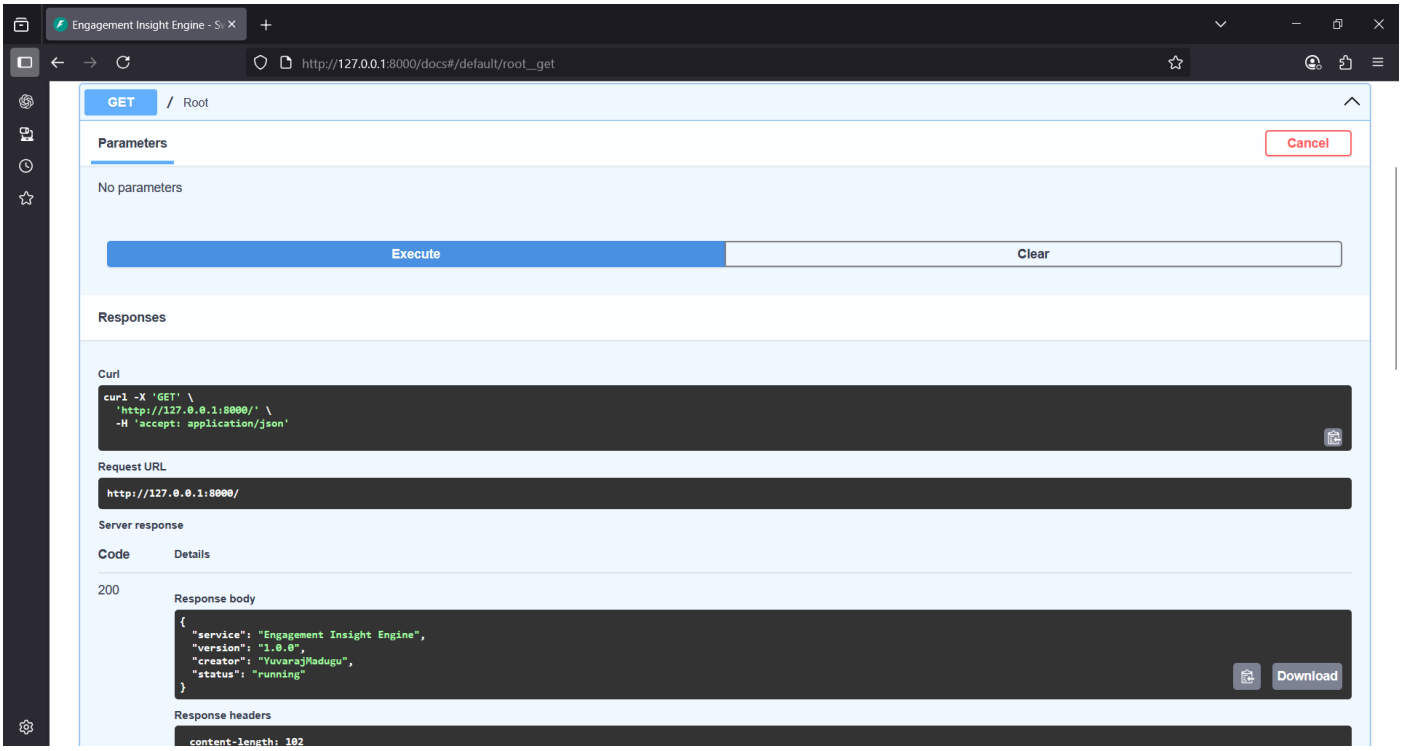
FLOWCHART DIAGRAM:

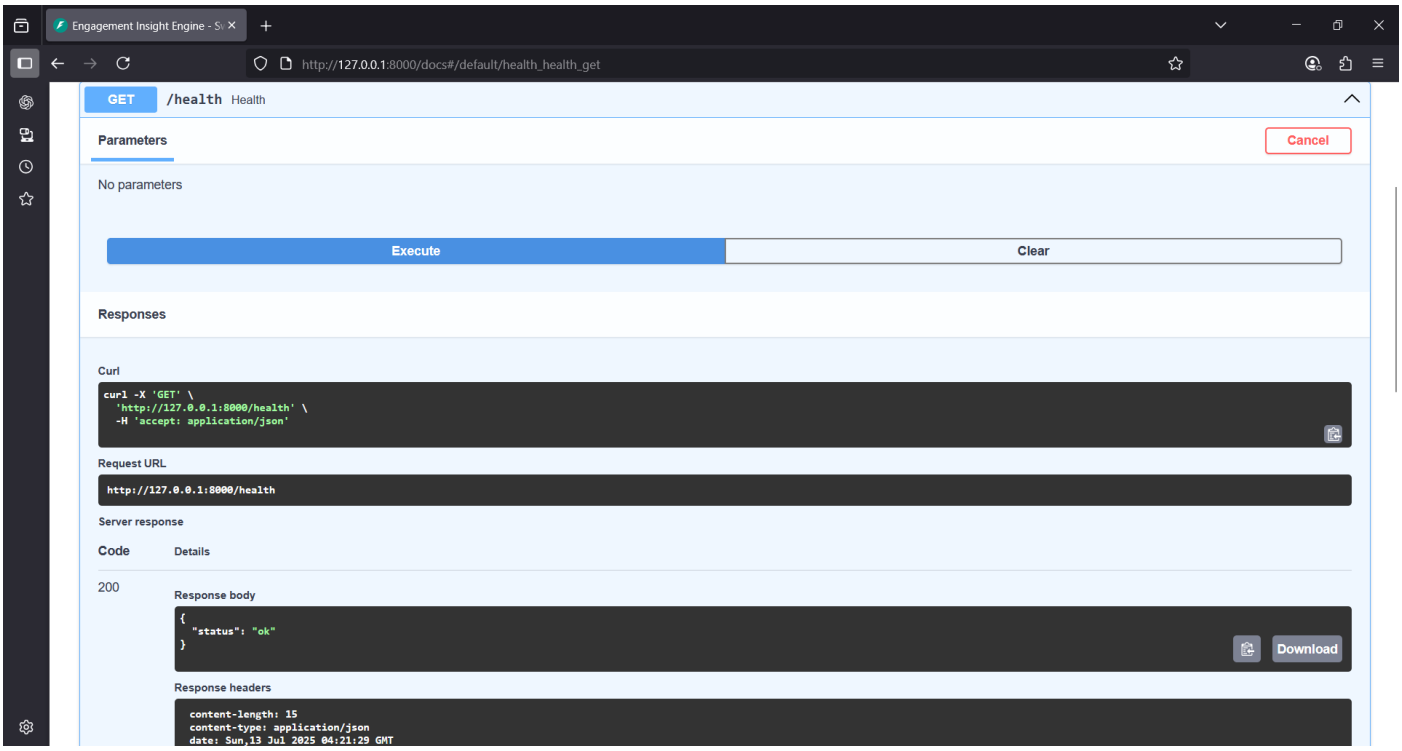


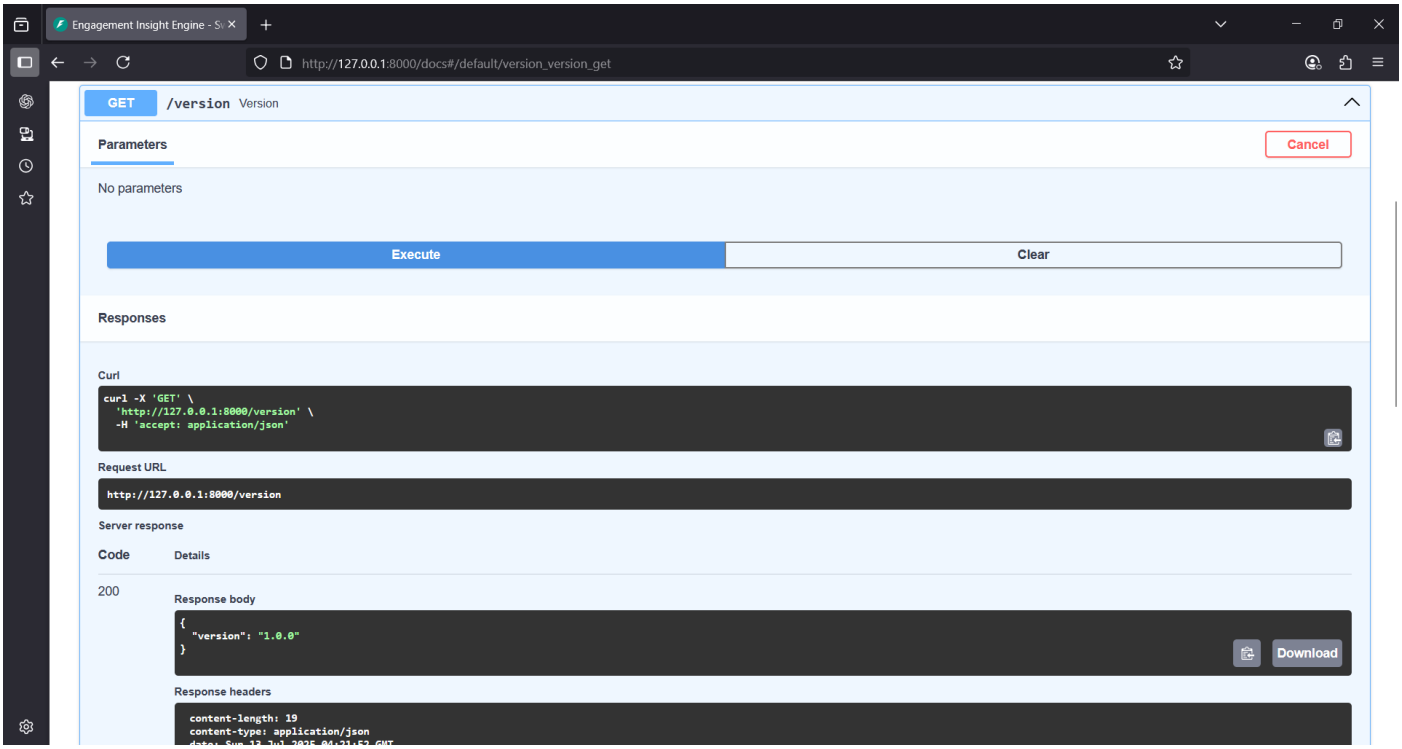
FASTAPI INTERFACE:

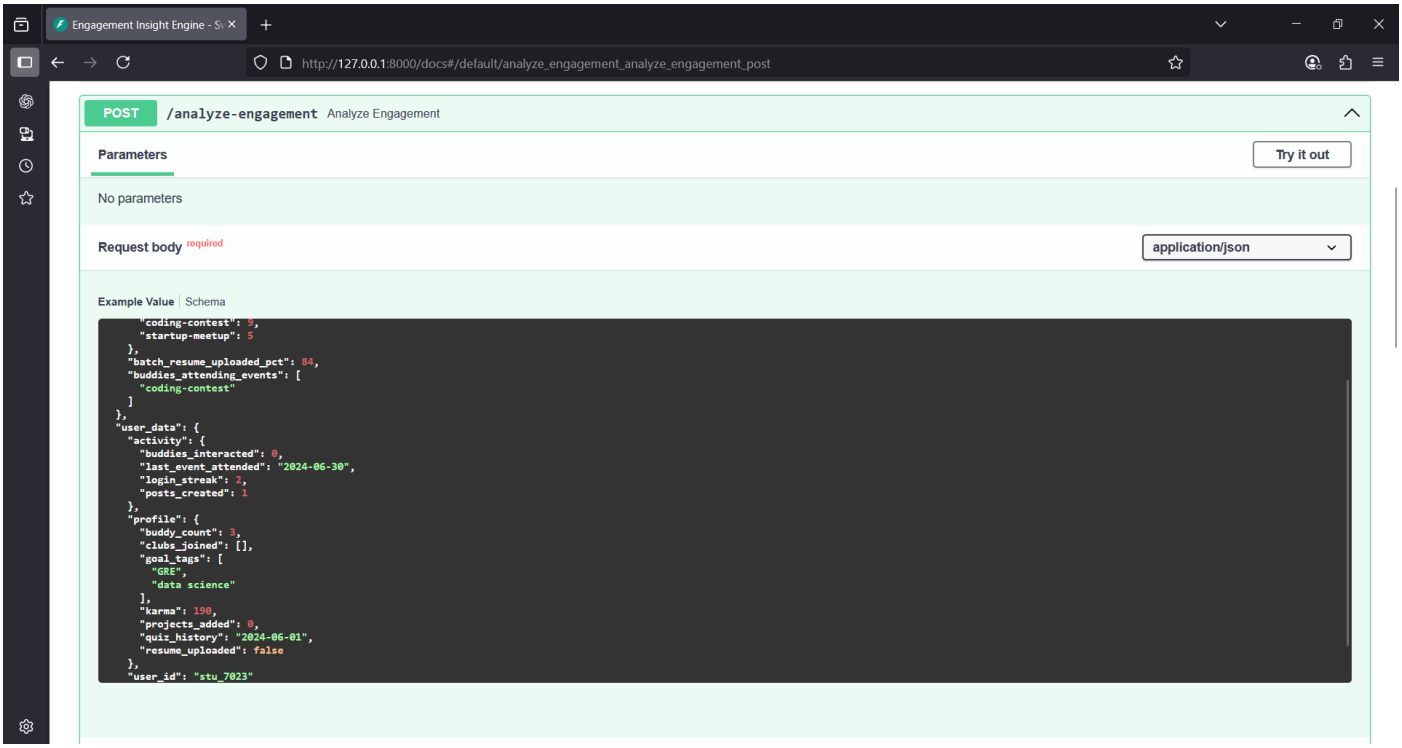


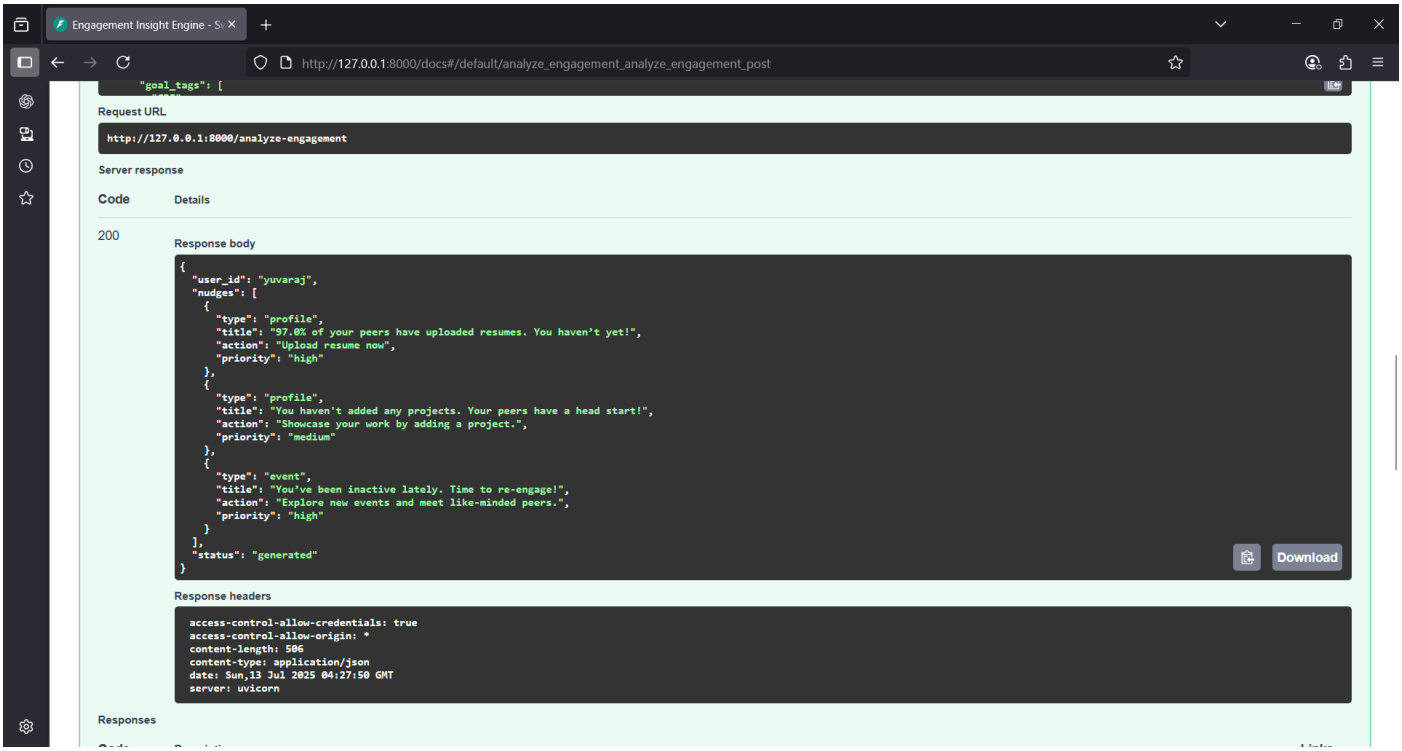


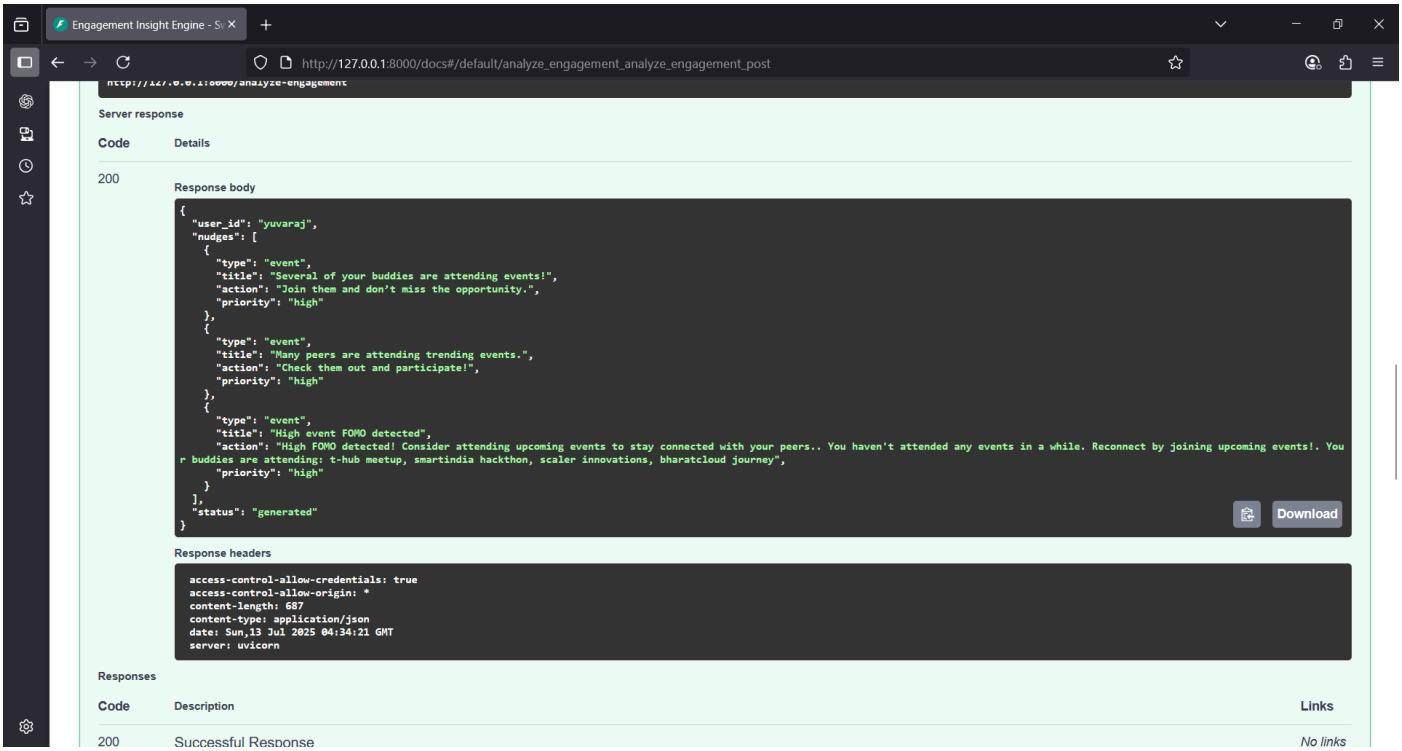

Engagement Insight Engine - S
http://127.0.0.1:8000/docs#/default/root_get
GET / Root
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/
Server response
Code Details
200
Response body
{
"service": "Engagement Insight Engine",
"version": "1.0.0",
"creator": "YuvarajMadugu",
"status": "running"
}
Download
Response headers
content-length: 102

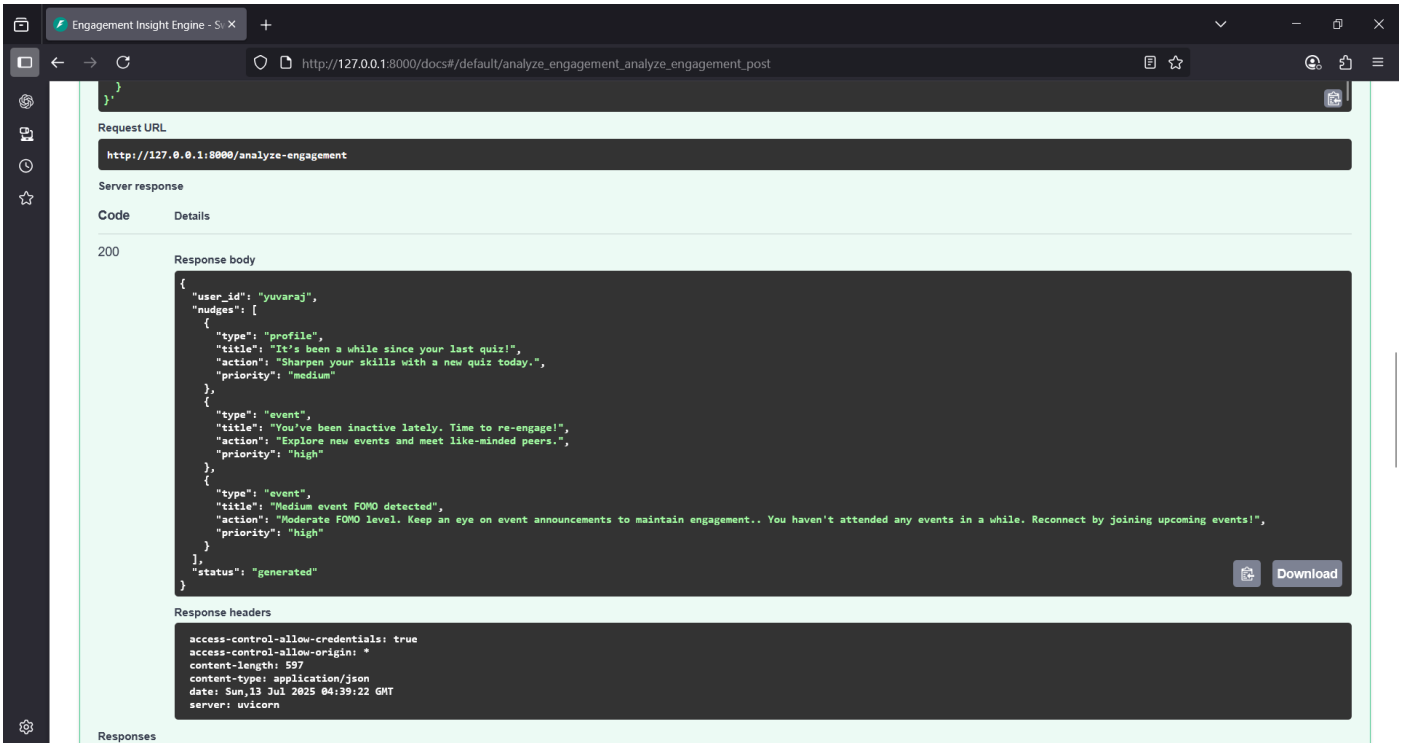

Engagement Insight Engine - S
http://127.0.0.1:8000/docs#/default/health_health_get
GET /health Health
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/health' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/health
Server response
Code Details
200
Response body
{
"status": "ok"
}
Download
Response headers
content-length: 15
content-type: application/json
date: Sun,13 Jul 2025 04:21:29 GMT

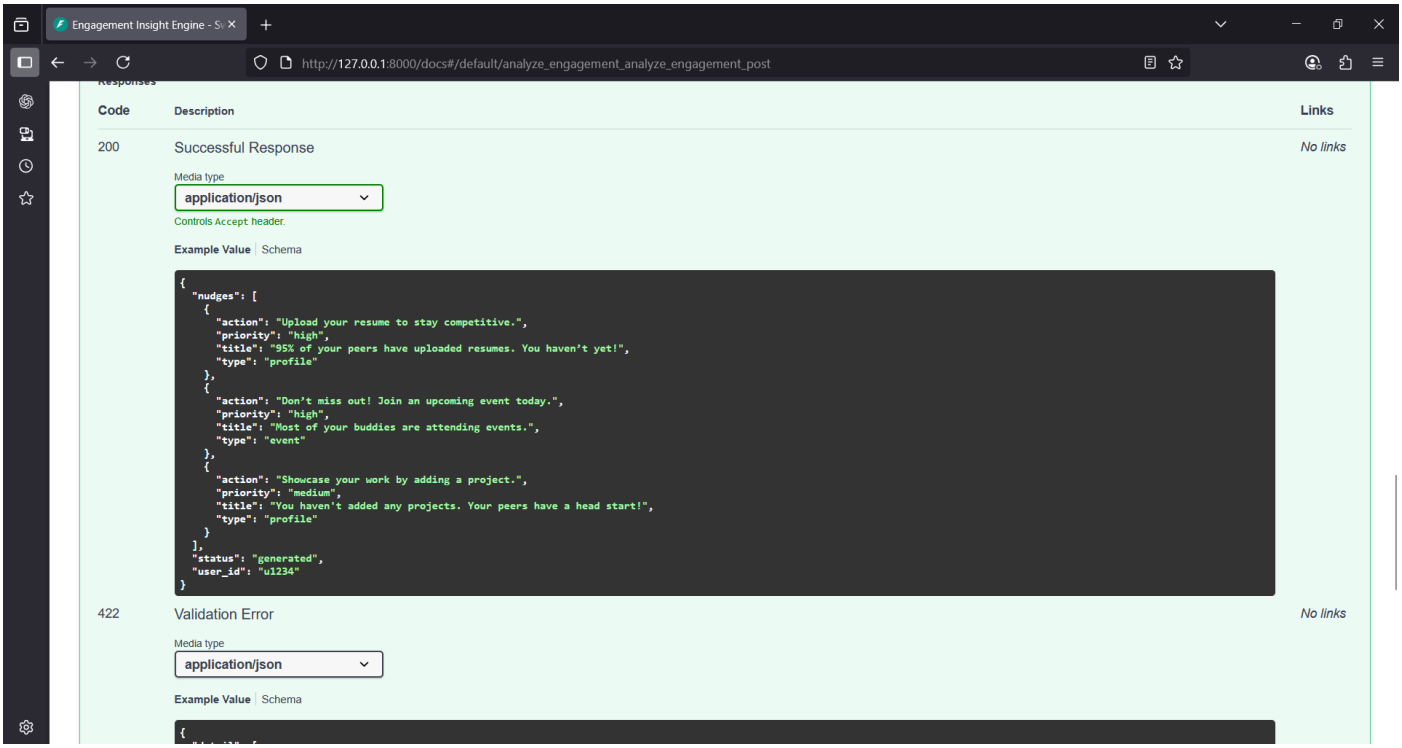

Engagement Insight Engine - Sv
http://127.0.0.1:8000/docs#/default/version_version_get
GET /version Version
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://127.0.0.1:8000/version' \
-H 'accept: application/json'
Request URL
http://127.0.0.1:8000/version
Server response
Code
Details
200
Response body
{
"version": "1.0.0"
}
Download
Response headers
content-length: 19
content-type: application/json


Engagement Insight Engine - Sv
http://127.0.0.1:8000/docs#/default/analyze_engagement_analyze_engagement_post
POST /analyze-engagement Analyze Engagement
Parameters
Try it out
No parameters
Request body required
application/json
Example Value
Schema
"coding-contest": 9,
"startup-meetup": 5
},
"batch_resume_uploaded_pct": 84,
"buddies_attending_events": [
"coding-contest"
]
},
"user_data": {
"activity": {
"buddies_interacted": 0,
"last_event_attended": "2024-06-30",
"login_streak": 2,
"posts_created": 1
},
"profile": {
"buddy_count": 3,
"clubs_joined": [],
"goal_tags": [
"GRE",
"data science"
],
"karma": 190,
"projects_added": 0,
"quiz_history": "2024-06-01",
"resume_uploaded": false
},
"user_id": "stu_7023"


Engagement Insight Engine - Sw
http://127.0.0.1:8000/docs#/default/analyze_engagement_analyze_engagement_post
"goal_tags": [
Request URL
http://127.0.0.1:8000/analyze-engagement
Server response
Code
Details
200
Response body
{
"user_id": "yuvaraj",
"nudges": [
{
"type": "profile",
"title": "97.0% of your peers have uploaded resumes. You haven't yet!",
"action": "Upload resume now",
"priority": "high"
},
{
"type": "profile",
"title": "You haven't added any projects. Your peers have a head start!",
"action": "Showcase your work by adding a project.",
"priority": "medium"
},
{
"type": "event",
"title": "You've been inactive lately. Time to re-engage!",
"action": "Explore new events and meet like-minded peers.",
"priority": "high"
}
],
"status": "generated"
}
Download
Response headers
access-control-allow-credentials: true
access-control-allow-origin: *
content-length: 506
content-type: application/json
date: Sun,13 Jul 2025 04:27:50 GMT
server: uvicorn
Responses


Engagement Insight Engine - Sw
http://127.0.0.1:8000/docs#/default/analyze_engagement_analyze_engagement_post
Server response
Code
Details
200
Response body
{
"user_id": "yuvaraj",
"nudges": [
{
"type": "event",
"title": "Several of your buddies are attending events!",
"action": "Join them and don't miss the opportunity.",
"priority": "high"
},
{
"type": "event",
"title": "Many peers are attending trending events.",
"action": "Check them out and participate!",
"priority": "high"
},
{
"type": "event",
"title": "High event FOMO detected",
"action": "High FOMO detected! Consider attending upcoming events to stay connected with your peers.. You haven't attended any events in a while. Reconnect by joining upcoming events!. Your buddies are attending: t-hub meetup, smartindia hackthon, scaler innovations, bharatcloud journey",
"priority": "high"
}
],
"status": "generated"
}
Download
Response headers
access-control-allow-credentials: true
access-control-allow-origin: *
content-length: 687
content-type: application/json
date: Sun,13 Jul 2025 04:34:21 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links


Engagement Insight Engine - S
http://127.0.0.1:8000/docs#/default/analyze_engagement_analyze_engagement_post
}
}'
Request URL
http://127.0.0.1:8000/analyze-engagement
Server response
Code
Details
200
Response body
{
"user_id": "yuvaraj",
"nudges": [
{
"type": "profile",
"title": "It's been a while since your last quiz!",
"action": "Sharpen your skills with a new quiz today.",
"priority": "medium"
},
{
"type": "event",
"title": "You've been inactive lately. Time to re-engage!",
"action": "Explore new events and meet like-minded peers.",
"priority": "high"
},
{
"type": "event",
"title": "Medium event FOMO detected",
"action": "Moderate FOMO level. Keep an eye on event announcements to maintain engagement.. You haven't attended any events in a while. Reconnect by joining upcoming events!",
"priority": "high"
}
],
"status": "generated"
}
Download
Response headers
access-control-allow-credentials: true
access-control-allow-origin: *
content-length: 597
content-type: application/json
date: Sun,13 Jul 2025 04:39:22 GMT
server: uvicorn
Responses


Engagement Insight Engine - S
http://127.0.0.1:8000/docs#/default/analyze_engagement_analyze_engagement_post
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
{
"nudges": [
{
"action": "Upload your resume to stay competitive.",
"priority": "high",
"title": "95% of your peers have uploaded resumes. You haven't yet!",
"type": "profile"
},
{
"action": "Don't miss out! Join an upcoming event today.",
"priority": "high",
"title": "Most of your buddies are attending events.",
"type": "event"
},
{
"action": "Showcase your work by adding a project.",
"priority": "medium",
"title": "You haven't added any projects. Your peers have a head start!",
"type": "profile"
}
],
"status": "generated",
"user_id": "u1234"
}
422
Validation Error
No links
Media type
application/json
Example Value | Schema

CONCLUSION:

The Engagement Insight Engine successfully addresses the growing need for personalized, data-driven engagement strategies within modern learning and professional communities. By combining deterministic, rule-based logic with predictive machine learning models, the system ensures that each user receives timely, relevant, and actionable nudges that encourage stronger participation and profile enrichment. The modular design and clearly defined configuration settings make it possible to easily adapt the engine to different organizations, user bases, or evolving engagement goals without significant code changes.

Through its fully offline and containerized architecture, the engine guarantees data privacy and operational independence, allowing deployments in secure environments without reliance on continuous internet access or third-party services. The use of FastAPI, Pydantic models, and robust logging provides a seamless developer experience, simplifying integration into broader ecosystems and ensuring the API remains transparent and easy to test via tools like Swagger UI. The incorporation of synthetic data simulation, feature engineering, and thorough model evaluation ensures that predictions remain reliable and interpretable.

Overall, this project demonstrates a balanced blend of modern machine learning techniques and practical rule-based design, offering an advanced yet user-centric solution to boosting user engagement. By guiding users to complete profiles, join activities, and align with community benchmarks, the engine fosters a healthier, more active, and supportive environment. With its strong foundation, clear structure, and scalable deployment options, the Engagement Insight Engine stands as a robust, future-ready microservice that can significantly enhance any community-driven platform.

REFERENCES:

- **ChatGPT (OpenAI)**: Used extensively for guidance, brainstorming implementation strategies, clarifying logic, and generating detailed documentation content whenever I encountered challenges during project development.
- **Claude AI (Anthropic)**: Referred to for resolving conceptual doubts, validating architectural decisions, and cross-checking rule-based and ML logic flows.
- **Krishna Naik — YouTube Channel**: Followed tutorials and practical walkthroughs on data science concepts, feature engineering, and model evaluation techniques, which helped shape the machine learning components of the project.
- **YouTube Tutorials:** Referred for Docker installation, container management, and image deployment workflows.
- **Docker Containerization Tutorials — YouTube (Various Creators)**: Learned best practices for containerizing Python applications, writing efficient Dockerfiles, and deploying microservices in isolated environments.