**Project Design Phase-II**
**Technology Stack (Architecture & Stack)**

| | |
|---|---|
| Date | 25 JUNE 2025 |
| Team ID | LTVIP2025TMID38998 |
| Project Name | TrafficTelligence: Advanced Traffic Volume Estimation with Machine Learning |
| Maximum Marks | 4 Marks |

---

## Technical Architecture:

**Architectural Diagram Description:**

The TRAFIICTELLIGENCE Project will leverage a modern, cloud-native, and microservices-based architecture to ensure scalability, reliability, and real-time processing capabilities. The diagram would depict a multi-tier structure with clear separation of concerns.

1. **User Interface Layer:** This layer, accessible via web and potentially mobile applications, acts as the primary interaction point for users (e.g., traffic authorities, urban planners, commuters). It would communicate with the backend application logic via RESTful APIs.
2. **API Gateway:** An API Gateway would sit in front of the application logic, handling request routing, authentication, rate limiting, and other cross-cutting concerns.
3. **Application Logic (Microservices) Layer:** This is the core of the system, composed of several independent microservices, each responsible for a specific domain or functionality (e.g., Data Ingestion Service, Real-time Analysis Service, Prediction Service, Reporting Service, Traffic Control Service). These services communicate with each other asynchronously, often using message queues, and interact with various databases and external APIs.
4. **Data Layer:** This layer comprises different types of databases tailored for specific data needs (e.g., relational for structured configuration data, NoSQL for high-volume real-time sensor data, data warehouses for analytical purposes). It also includes file storage for raw sensor data or processed reports.
5. **Machine Learning Layer:** Dedicated services or components housing trained Machine Learning models for tasks like traffic prediction, anomaly detection, or optimal routing. These models consume data from the data layer and feed insights back into the

application logic.

6. **External Services Integration:** This layer represents connections to various external APIs crucial for traffic intelligence, such as real-time traffic data providers, weather APIs, or mapping services.
7. **Infrastructure Layer:** The entire system would be deployed on a cloud platform, utilizing containerization and orchestration for efficient deployment, scaling, and management. This includes compute instances, networking components, and managed services for databases and messaging.

The flow for a typical request (e.g., fetching real-time traffic data) would be: User UI -> API Gateway -> Real-time Analysis Microservice -> NoSQL Database -> (potentially) Machine Learning Model -> Real-time Analysis Microservice -> API Gateway -> User UI. For data ingestion, it would involve external sensors/feeds sending data to an Ingestion Microservice -> Message Queue -> NoSQL Database/Data Lake.

---

## Table-1 : Components & Technologies:

| S.No | Component | Description | Technology |
|---|---|---|---|
| 1 | User Interface | Web UI for traffic monitoring dashboards, mobile app for commuter alerts. | React.js (Web), React Native (Mobile) |
| 2 | Application Logic-1 | Real-time Traffic Data Ingestion & Processing Service | Python (Flask/FastAPI), Apache Kafka (for streaming) |
| 3 | Application Logic-2 | Traffic Prediction & Anomaly Detection Service | Python (TensorFlow/PyTorch) |
| 4 | Application Logic-3 | Traffic Control & Optimization Service (e.g., signal timing) | Java (Spring Boot) |

| | | | |
|---|---|---|---|
| 5 | Database | Structured data storage for configuration, user profiles. | PostgreSQL |
| 6 | Cloud Database | High-volume, real-time sensor data storage (time-series data). | AWS Timestream / Google Cloud Firestore |
| 7 | File Storage | Storage for raw sensor data, historical logs, large reports. | AWS S3 / Google Cloud Storage |
| 8 | External API-1 | Real-time Traffic Data Feeds (e.g., from public transport authorities) | HERE Technologies Traffic API, TomTom Traffic API |
| 9 | External API-2 | Weather API for environmental impact on traffic | OpenWeatherMap API |
| 10 | Machine Learning Model | Models for traffic flow prediction, congestion detection, route optimization. | Custom models built with Scikit-learn, TensorFlow/Keras |
| 11 | Infrastructure | Application Deployment on Cloud | Kubernetes (for container orchestration), AWS EKS / Google Kubernetes Engine (GKE) |

---

## Table-2: Application Characteristics:

| S.No | Characteristics | Description | Technology |
|---|---|---|---|
| 1 | Open-Source Frameworks | List the open-source frameworks used | Python, Java, React.js, React Native, PostgreSQL, Apache Kafka, Kubernetes, Docker, Flask, Spring Boot, TensorFlow, PyTorch |
| 2 | Security Implementations | List all the security / access controls implemented, use of firewalls etc. | OAuth 2.0/OpenID Connect for authentication, JWT for authorization, HTTPS, Network Security Groups/Firewalls (Cloud provider), IAM Roles (Cloud provider), Regular security audits, OWASP Top 10 adherence |
| 3 | Scalable Architecture | Justify the scalability of architecture (3 – tier, Micro-services) | Microservices architecture allows independent scaling of components. Kubernetes for automated scaling (horizontal pod autoscaling). Cloud provider auto-scaling groups for underlying compute. |

| | | | |
|---|---|---|---|
| | | | Distributed databases (e.g., AWS Timestream) for data scalability. |
| 4 | Availability | Justify the availability of application (e.g. use of load balancers, distributed servers etc.) | Multi-Availability Zone (AZ) deployment, Load Balancers (e.g., AWS ELB/GCP Load Balancer) across microservices, Kubernetes self-healing (restarts failed pods), Database replication (Master-Slave/Multi-Master), Redundant network paths. |
| 5 | Performance | Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc. | Asynchronous communication via Kafka for high-throughput data processing. In-memory caching (Redis) for frequently accessed data. Content Delivery Networks (CDNs) for UI assets. Optimized database queries and indexing. Horizontal scaling of microservices to handle high RPS. |

---

This provides a comprehensive structure for your "TRAFIICTELLIGENCE PROJECT" design

phase. Remember to replace placeholder values like "[Your Team ID Here]" with your actual project details.