**Project Design Phase**

**Solution Architecture**

| | |
|---|---|
| Date | 20 June 2025 |
| Team ID | LTVIP2025TMID29572 |
| Project Name | Sustainble Smart City |
| Maximum Marks | 4 Marks |

## Solution Architecture:

The Sustainable Smart City system is designed as a modular, scalable AI-powered platform that enables citizens and authorities to engage with sustainability initiatives through an intuitive user interface and intelligent services. At a high level, the system is organized into five interconnected layers: the **User Dashboard**, **API Gateway**, **Microservices Layer**, **AI/ML Layer**, and the **Data Layer**. Each layer plays a critical role in delivering responsive, insightful, and real-time sustainability solutions.

### Frontend Layer

The system's user-facing interface is built using modern frontend technologies like **React** or **Vue.js**, ensuring a responsive and user-friendly experience across devices. It includes a dynamic dashboard that enables users to interact with different features such as the Recycle DIY tool, Village Comparator, Problem-Solution Finder, and Dream City Generator. JWT-based authentication ensures secure login and user management. For live updates and dynamic content, **WebSocket connections** are integrated. Visual elements like sustainability graphs and comparison metrics are displayed using visualization libraries such as **Chart.js** or **D3.js**.

### API Gateway and Load Balancer

To manage traffic efficiently and route requests, the platform uses an **API Gateway** such as Kong or AWS API Gateway. This layer handles **rate limiting**, **authentication**, and **logging**, making the APIs robust and secure. A **load balancer** like NGINX distributes incoming requests to different services, maintaining availability and preventing overload on any single service.

### Microservices Architecture

The platform is powered by a suite of independent microservices, each handling a specific feature of the system.

- The **Recycle DIY Service** processes user inputs describing waste materials. It utilizes **IBM Granite** models for material classification and recommends upcycled DIY solutions. Environmental impacts, such as carbon footprint reduction, are calculated based on a database of material properties and templates.
- The **Village Comparator Service** enables users to compare sustainability metrics across two regions. It gathers data from external APIs and internal sources, then leverages a **Retrieval-Augmented Generation (RAG)** pipeline backed by a vector

database. The **Google Flan model** is used to generate insights and recommendations for sustainable development.

- The **Problem-Solution Service** functions as an AI assistant for sustainability-related challenges. It uses natural language understanding to process user queries and performs a semantic search using RAG techniques to retrieve the most relevant solutions from a curated knowledge base.
- The **Dream City Generator** allows users to envision their ideal sustainable city. By extracting parameters from user input, the system uses generative AI models like **DALL·E** or **Stable Diffusion** to create city images. It also calculates city health metrics (e.g., pollution levels, renewable energy usage) and offers recommendations for improvement.

**AI/ML Layer**

This layer supports all AI-driven functionalities. Models such as **IBM Granite** and **Google Flan-T5** are hosted either on cloud platforms like IBM Watson or containerized within the infrastructure. An **image generation model** is integrated for the Dream City feature. For managing multiple model versions and tracking performance, **MLflow** is used. The **RAG engine** uses vector databases like Pinecone or Weaviate to store and search through embeddings generated by models like **Sentence-BERT**. Retrieval is based on **cosine similarity**, ensuring accurate and context-aware results.

**Data Layer**

Structured data such as user profiles, city metrics, and templates are stored in **PostgreSQL**, while unstructured content including AI-generated responses is managed in **MongoDB**. **Redis** serves as a caching layer to handle session data and frequently accessed information. The system draws data from a combination of sources including **government APIs**, **environmental datasets**, and **user-contributed content**. These sources keep the platform up-to-date with real-world sustainability metrics.

**Workflow and Feature Flow**

Each feature has a clearly defined workflow. For example, in the **Recycle DIY** flow, a user submits a material description, which is classified by AI. Templates and impact data are fetched from the database, and a personalized suggestion is returned. The **Village Comparator** involves comparing two regions by retrieving data, running a RAG process, and generating insights through AI analysis. The **Problem-Solution flow** starts with user queries, which are understood and matched against a semantic knowledge base, returning actionable solutions. For the **Dream City feature**, user preferences are translated into parameters, visualized using AI-generated imagery, and analyzed to produce city health scores and recommendations.

**Technology Stack**

On the backend, services are built with **Node.js (Express)** or **Python (FastAPI)** and deployed using **Docker containers** orchestrated by **Kubernetes**. Message queues like **Kafka** manage asynchronous processing, while **Prometheus** and **Grafana** provide real-time monitoring. AI model serving is handled by **TensorFlow Serving** or MLflow, with GPU support from **NVIDIA CUDA**. Large-scale data is processed in batches using **Apache Spark**, and **Kubeflow** automates the ML pipeline.

**Security and Compliance**

Security is enforced through **OAuth 2.0 authentication**, **RBAC-based authorization**, and strong data encryption protocols such as **TLS 1.3** and **AES-256**. The system includes protections against common web threats via input validation and vulnerability scanning. **GDPR compliance** is ensured by implementing user consent flows, anonymizing data, and logging access for audit purposes. Regular backups and penetration testing protect against data loss and breaches.

### Data Flow Architecture

Data flows through the system in three main pipelines. In the **real-time pipeline**, external APIs feed data into Kafka, processed in streams, and cached for instant dashboard updates. The **AI processing pipeline** takes user inputs through preprocessing, inference, post-processing, and caching for efficient response delivery. In the **batch processing pipeline**, large datasets undergo ETL operations and are stored in a data warehouse, with insights visualized in the analytics dashboard.

### Deployment Strategy

The system is deployed on cloud platforms like **AWS**, **Azure**, or **GCP**. It uses **Kubernetes clusters** for elastic compute, with cloud-based storage for data and media. **CDNs** like CloudFront or Azure CDN deliver static content efficiently. The **CI/CD pipeline** automates code integration, testing, containerization, and deployment, ensuring rapid iteration and continuous delivery.

### Scalability and Performance

The architecture supports **horizontal scaling** by independently scaling services and sharding databases. **Caching** at multiple levels (Redis, CDN, browser) improves response times, and load balancing distributes user requests efficiently. Optimization strategies include indexing database queries, GPU acceleration for models, and minimizing API latency below 200ms.

### Monitoring, Analytics, and Alerts

Operational health is tracked using **metrics dashboards** that monitor system performance, AI model accuracy, and user engagement. Alerts are triggered on performance drops, resource overuse, or AI model drift. Frontend performance is also monitored to ensure a seamless user experience.

## Solution Architecture Diagram:

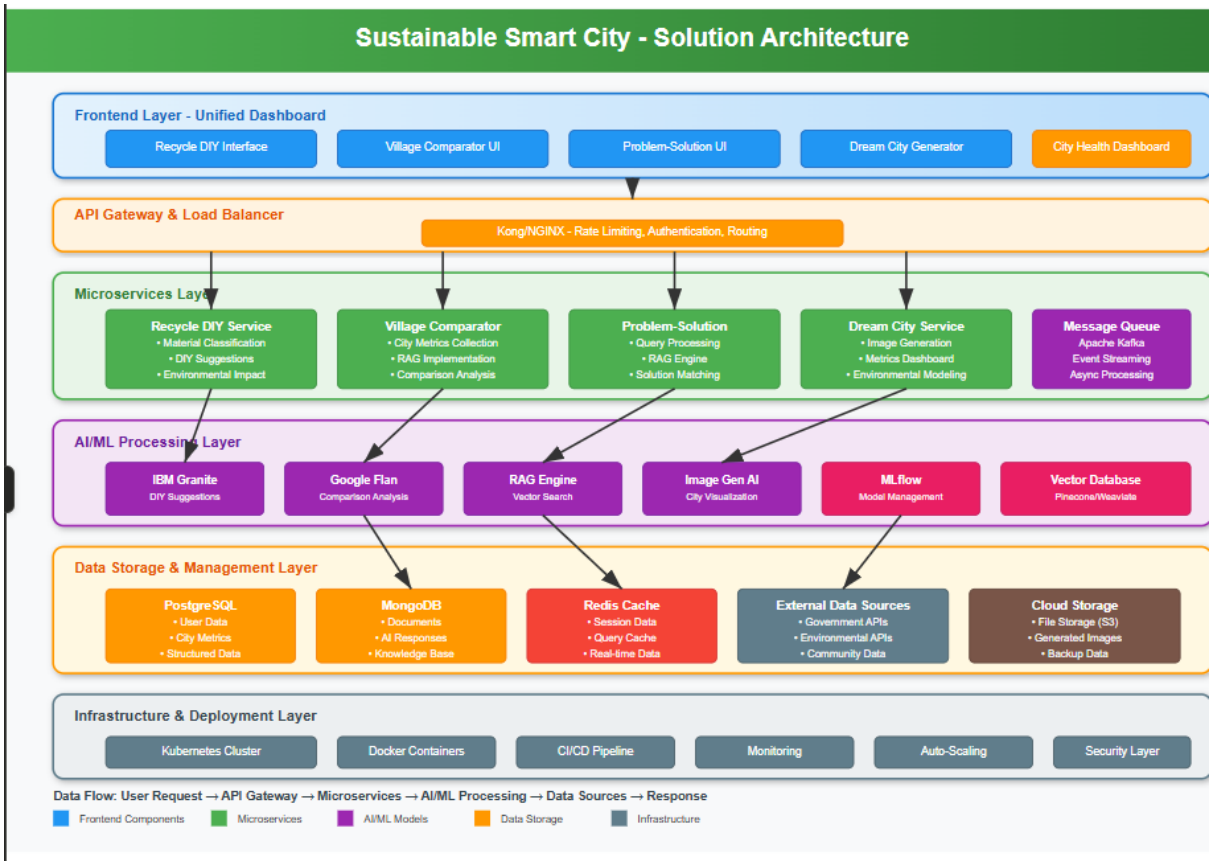


*Figure 1: Architecture and data flow of sustainable*