**Project Design Phase-II**

**Technology Stack (Architecture & Stack)**

|  |  |
| --- | --- |
| Date | 27 june |
| Team ID | LTVIP2025TMID53139 |
| Project Name | House Hunt: Finding your Perfect Rental Home |
| Maximum Marks | 2 marks |

**Technology Stack (Architecture & Stack)**

**Architectural Style: Microservices Architecture**

* **Reasoning:** This approach offers high scalability, fault isolation, independent deployments, and technology flexibility. It's well-suited for complex applications with evolving features like a House-Hunt platform.
* **Communication:** RESTful APIs (synchronous) and Message Queues/Event Streaming (asynchronous).

**Core Technology Stack:**

| Category | Technology / Tool | Purpose |
| --- | --- | --- |
| **Frontend** | React / Next.js | Modern JavaScript library for building SPAs/SSG (for SEO). |
|  | HTML5, CSS3, JavaScript | Core web technologies. |
|  | SASS/LESS (CSS Preprocessor) | Enhanced CSS styling. |
|  | Axios / Fetch API | HTTP client for API communication. |
| **Mobile (Hybrid)** | React Native / Flutter | Cross-platform mobile application development. |
| **Backend (Microservices)** | Node.js (Express/NestJS) / Python (FastAPI/Django) | Primary language/framework for services (API development). |
|  | Java (Spring Boot) / Go (Gin) | Alternative for specific services if expertise exists. |
| **Database (Relational)** | PostgreSQL with PostGIS | Primary data store for structured data (properties, users) and geo-spatial queries. |
| **Database (NoSQL - Optional)** | MongoDB / Redis | For flexible user profiles, caching, session management. |
| **Search Engine** | Elasticsearch | Powerful, scalable search and analytics engine for property indexing and complex queries. |
| **Caching** | Redis / Memcached | In-memory data store for caching frequently accessed data. |
| **Message Broker** | Apache Kafka / AWS SQS | Asynchronous communication between services, event streaming. |
| **Containerization** | Docker | Packaging applications and their dependencies. |
| **Container Orchestration** | Kubernetes (EKS/GKE/AKS) | Managing, scaling, and deploying containerized applications. |
| **API Gateway** | NGINX (as a reverse proxy) / AWS API Gateway / Kong | Centralized entry point for APIs, routing, security. |
| **Cloud Provider** | AWS / Google Cloud Platform (GCP) / Azure | Infrastructure hosting, managed services. |
| **CI/CD** | GitLab CI/CD / GitHub Actions / Jenkins | Automation of build, test, and deployment processes. |
| **Monitoring & Logging** | Prometheus & Grafana / ELK Stack (Elasticsearch, Logstash, Kibana) / AWS CloudWatch | Application performance monitoring, centralized logging. |
| **Infrastructure as Code (IaC)** | Terraform / AWS CloudFormation | Provisioning and managing infrastructure through code. |
| **Version Control** | Git (GitHub / GitLab / Bitbucket) | Source code management. |

Export to Sheets

**5. Technical Architecture (Detailed Explanation)**

Building on the conceptual diagram, here's a more detailed breakdown:

1. **Client Layer:**
   * **Web Application:** A React (or Next.js for SSR/SSG benefits) Single Page Application (SPA) serves as the primary user interface. It communicates with the backend via RESTful APIs.
   * **Mobile Applications:** Developed using React Native or Flutter for cross-platform compatibility, also communicating via RESTful APIs.
   * **Admin Dashboard:** A separate web application (could be React/Vue/Angular) or a dedicated module within the main web app, providing tools for property management.
   * **CDN (Content Delivery Network):** For static assets (images, CSS, JS bundles) to ensure fast content delivery globally.
2. **API Gateway Layer:**
   * All client requests first hit an **API Gateway** (e.g., NGINX, AWS API Gateway).
   * It handles authentication enforcement, request routing to appropriate microservices, rate limiting, and API versioning.
3. **Authentication & Authorization Service:**
   * A dedicated microservice (or managed service like AWS Cognito, Auth0) handles user registration, login, token issuance (JWT), and authorization checks.
   * Integrates with the User Database.
4. **Microservices Layer (Backend):**
   * **User Service:** Manages user profiles, saved properties, and communication preferences. Interacts with the User Database.
   * **Property Service:** Handles CRUD operations for property listings, including details, amenities, and image references. Interacts with the Property Database.
   * **Search Service:** Powers all search capabilities. It takes search queries from clients, queries Elasticsearch, and returns relevant property IDs/summaries. It might also periodically sync data from the Property Database to Elasticsearch.
   * **Image Service:** Handles image uploads (to S3-like storage), processing (resizing, optimization), and serving.
   * **Notification Service:** Sends emails (via SendGrid/SES) and SMS (via Twilio/SNS) for inquiries, alerts, or system messages.
   * **Inquiry Service:** Processes incoming property inquiries, stores them, and triggers notifications.
   * **Admin Service:** Provides APIs specifically for admin operations (e.g., moderating listings, managing users).
   * **Messaging Queue (Kafka/SQS):** Used for asynchronous communication between services (e.g., when a new property is added, publish an event to a queue, and the Search Service consumes it to update Elasticsearch).
5. **Data Layer:**
   * **PostgreSQL (with PostGIS):** The primary relational database for structured, transactional data like user accounts, property master data, and relationships. PostGIS enables efficient geo-spatial queries.
   * **Elasticsearch:** The dedicated search index for fast, full-text, and filtered search queries on property listings.
   * **Redis:** Used as a distributed cache for frequently accessed data (e.g., popular listings, session data) to reduce database load.
   * **AWS S3 (or similar Blob Storage):** For cost-effective and scalable storage of property images and other media files.
6. **DevOps & Infrastructure:**
   * **Docker:** All microservices are containerized.
   * **Kubernetes:** Manages the deployment, scaling, and self-healing of these Docker containers across cloud infrastructure.
   * **Terraform/CloudFormation:** Infrastructure is provisioned and managed as code for consistency and repeatability.
   * **CI/CD Pipelines:** Automated pipelines (e.g., GitLab CI/CD) handle code commits, testing, building Docker images, and deploying to Kubernetes.
   * **Monitoring (Prometheus/Grafana, CloudWatch):** Collects metrics from services and infrastructure to provide dashboards and alerts.
   * **Logging (ELK Stack):** Centralized log aggregation and analysis for debugging and operational insights.

**6. Components & Technologies Table**

| Component Category | Key Components | Technologies/Tools Chosen (Examples) | Purpose |
| --- | --- | --- | --- |
| **Client Side** | Web Application | React, Next.js, HTML, CSS, JavaScript, Axios | User-facing web interface for property search and interaction. |
|  | Mobile Applications | React Native / Flutter | Native-like experience on iOS and Android devices. |
|  | Admin Dashboard (Web) | React / Vue.js / HTML, CSS, JavaScript | Interface for administrators to manage content. |
| **Network & Gateway** | Content Delivery Network | AWS CloudFront / Cloudflare | Caching static assets, reducing latency. |
|  | API Gateway / Load Balancer | NGINX / AWS API Gateway / Kubernetes Ingress Controller | Single entry point, request routing, security, load distribution. |
| **Backend Services** | User Service | Node.js (NestJS) / Python (FastAPI) | User authentication, profile, saved items. |
|  | Property Service | Node.js (NestJS) / Python (FastAPI) / Java (Spring Boot) | Property listings CRUD, details, amenities. |
|  | Search Service | Node.js / Python | Integrates with Elasticsearch for complex search queries. |
|  | Image Service | Node.js / Python | Handles image uploads, processing, and serving. |
|  | Notification Service | Node.js / Python (integrates with SendGrid, Twilio) | Sending email/SMS notifications. |
|  | Inquiry Service | Node.js / Python | Manages property inquiries. |
|  | Admin Service | Node.js / Python | APIs for admin-specific operations. |
| **Data Stores** | Primary Database | PostgreSQL (with PostGIS) | Relational data for users, properties (master data), transactions. |
|  | Search Database | Elasticsearch | Fast full-text search, complex query capabilities. |
|  | Caching Layer | Redis | In-memory cache for frequently accessed data. |
|  | Object Storage | AWS S3 / Google Cloud Storage | Scalable storage for images and large files. |
| **Messaging** | Message Broker | Apache Kafka / AWS SQS | Asynchronous communication, event streaming, decoupling services. |
| **DevOps & Operations** | Container Runtime | Docker | Packaging applications into portable containers. |
|  | Orchestration | Kubernetes (EKS/GKE/AKS) | Automating deployment, scaling, and management of containers. |
|  | CI/CD Pipeline | GitLab CI/CD / GitHub Actions / Jenkins | Automating build, test, and deployment processes. |
|  | Infrastructure as Code | Terraform / AWS CloudFormation | Defining and provisioning infrastructure via code. |
|  | Monitoring | Prometheus & Grafana / AWS CloudWatch | Collecting and visualizing metrics, setting alerts. |
|  | Logging | ELK Stack (Elasticsearch, Logstash, Kibana) / AWS CloudWatch Logs | Centralized log collection, analysis, and visualization. |
| **Development Tools** | Version Control | Git (GitHub / GitLab) | Collaborative source code management. |
|  | API Testing | Postman / Insomnia | Testing REST APIs. |

Export to Sheets

**7. Application Characteristics Table**

This table summarizes key non-functional attributes and design goals for the House-Hunt application.

| Characteristic | Description | How it's Achieved (Architectural/Tech Choice) |
| --- | --- | --- |
| **Scalability** | Ability to handle increasing user load and data volume without significant performance degradation. | Microservices, Horizontal scaling of services (Kubernetes), Database sharding/read replicas, Caching (Redis), Elasticsearch. |
| **Performance** | Fast response times for user interactions, especially search queries and page loads. | CDN, Caching (Redis), Optimized database queries, Elasticsearch for search, Efficient API design. |
| **Reliability** | System's ability to perform its required functions under stated conditions for a specified period. | Redundant deployments (Kubernetes), Load balancing, Database backups, Fault-tolerant microservices, Monitoring & Alerting. |
| **Availability** | The percentage of time the system is operational and accessible to users. | Cloud provider's high availability features (e.g., AWS AZs), Kubernetes self-healing, Redundant services. |
| **Security** | Protection against unauthorized access, data breaches, and vulnerabilities. | HTTPS, JWT for authentication, OAuth2, Password hashing, Input validation, Regular security audits, Principle of Least Privilege. |
| **Maintainability** | Ease with which the system can be modified, updated, and bug-fixed. | Microservices architecture (smaller, independent codebases), Clear API contracts, CI/CD, Code standards, Documentation. |
| **Extensibility** | Ability to add new features or integrate with external systems with minimal changes to existing code. | Microservices architecture, Well-defined APIs (OpenAPI), Event-driven architecture (Kafka/SQS). |
| **Usability** | Ease of use and learnability for the target audience. | Intuitive UI/UX design, Mobile responsiveness (React Native/Flutter), User testing. |
| **Observability** | Ability to understand the internal state of the system from its external outputs. | Centralized logging (ELK Stack), Distributed tracing, Comprehensive monitoring (Prometheus/Grafana). |
| **Cost-Effectiveness** | Optimizing resource utilization and leveraging managed cloud services to manage operational costs. | Cloud-native services (pay-as-you-go), Containerization efficiency, Auto-scaling. |