Creating Conceptual, Logical, and Physical diagrams for an online vehicle booking system like Uber involves several layers of abstraction. Here is how each of these diagrams can be approached in detail:

**1. Conceptual Diagram**

The conceptual diagram provides a high-level overview of the system and its key components. It focuses on major entities, relationships, and external systems without diving into technical details.

**Key Entities:**

* **Users:** Passengers and Drivers.
* **Trips:** Booking requests and trips being handled.
* **Vehicles:** Cars, bikes, etc.
* **Payments:** Payment gateway systems.
* **Maps API:** Third-party services like Google Maps or OpenStreetMap.
* **Notifications:** SMS or Push notifications for updates.
* **Admin:** System administrators who manage the platform.

**Conceptual Diagram Overview:**

* Users can either be **Passengers** or **Drivers**.
* Passengers **book trips** and provide details such as pickup and drop-off locations.
* The system matches **available drivers** based on proximity and vehicle type.
* **Payments** are processed through third-party systems.
* The **Map API** provides geolocation, routing, and traffic information.
* **Notifications** are sent to keep passengers and drivers updated.
* Admins manage the system, track trips, and handle disputes.

**Conceptual Diagram:**

lua

Copy code

+------------+ +-------------+ +-------------+

| Passengers | <---> | Trip/Booking| <--->| Drivers |

+------------+ +-------------+ +-------------+

|

|

+---------------------------------------------+

| |

+------------+ +-------------+ +-------------+

| Payments | | Maps API | | Notifications|

+------------+ +-------------+ +-------------+

|

|

+------------+

| Admins |

+------------+

**2. Logical Diagram**

The logical diagram adds more structure to the system by defining the major components, interactions, and data flow. This includes services, databases, and APIs without getting into the hardware details.

**Key Components:**

* **User Service:** Handles user authentication and profile management.
* **Booking Service:** Manages trip requests, status updates, and matching drivers.
* **Driver Service:** Manages driver availability, vehicle info, and location tracking.
* **Payment Service:** Integrates with third-party payment gateways.
* **Notification Service:** Manages communication between the system, passengers, and drivers.
* **Admin Service:** Provides admin-level access to the system for management tasks.
* **Maps Service:** Interacts with third-party maps APIs for routing and geolocation.

**Logical Diagram Overview:**

* **User Service** handles user sign-in, sign-up, and profile management.
* **Booking Service** takes booking requests, matches drivers, and tracks trip status.
* **Driver Service** updates the system with driver availability and real-time locations.
* **Payment Service** communicates with payment providers for processing.
* **Notification Service** sends trip updates to users.
* **Admin Service** allows monitoring and reporting functionalities.
* **Maps Service** is integrated with the Maps API for directions and ETA.

**Logical Diagram:**

sql

Copy code

+----------------------+

| User Service |

+----------------------+

|

v

+----------------------+

| Booking Service | <---> +------------------+

+----------------------+ | Maps Service |

| +------------------+

v

+----------------------+

| Driver Service |

+----------------------+

|

v

+----------------------+

| Payment Service | <---> +------------------+

+----------------------+ | Payment Provider |

| +------------------+

v

+----------------------+

| Notification Service |

+----------------------+

|

v

+----------------------+

| Admin Service |

+----------------------+

**3. Physical Diagram**

The physical diagram details the deployment architecture of the system, highlighting the servers, databases, networking, and third-party systems. This diagram shows how different components are physically distributed.

**Key Physical Elements:**

* **Web Servers:** Hosts the front-end and APIs for passengers, drivers, and admins.
* **Application Servers:** Hosts the backend logic (user management, booking, payments).
* **Database Servers:** Stores information about users, trips, payments, and drivers.
* **Load Balancers:** Distributes requests among multiple servers to handle high traffic.
* **Payment Gateway Integration:** Links to third-party payment providers.
* **Map API Integration:** Links to external map services.
* **Notification Gateway:** Handles SMS, emails, or push notifications.
* **External Services:** Includes map services, payment gateways, etc.

**Physical Diagram Overview:**

* The **load balancer** distributes traffic across multiple application servers.
* **Web servers** handle the user-facing front end.
* **Application servers** process business logic for users, trips, payments, and drivers.
* **Database servers** store persistent data such as user profiles, trip data, and payment logs.
* Third-party systems like **Payment Gateway** and **Maps API** are external services.
* **Notifications** are managed through an external SMS or email provider.

**Physical Diagram:**

lua

Copy code

+-----------------------------------+

| Load Balancer |

+-----------------------------------+

/ \

/ \

+------------------+ +------------------+

| Web Server | | Web Server |

+------------------+ +------------------+

| |

v v

+------------------+ +------------------+

| App Server | | App Server |

+------------------+ +------------------+

| |

+------------------+ +------------------+

| Database Server | | Database Server |

+------------------+ +------------------+

| |

+------------------+ +-------------------+

| Payment Gateway | | Maps API |

+------------------+ +-------------------+

|

v

+-------------------+

| Notification Server|

+-------------------+

**Summary:**

* **Conceptual Diagram** shows the high-level flow of entities and relationships.
* **Logical Diagram** adds components, services, and their relationships.
* **Physical Diagram** outlines how these components are deployed in a real-world infrastructure.