High-level Design of Uber

Learn how to design an Uber system.

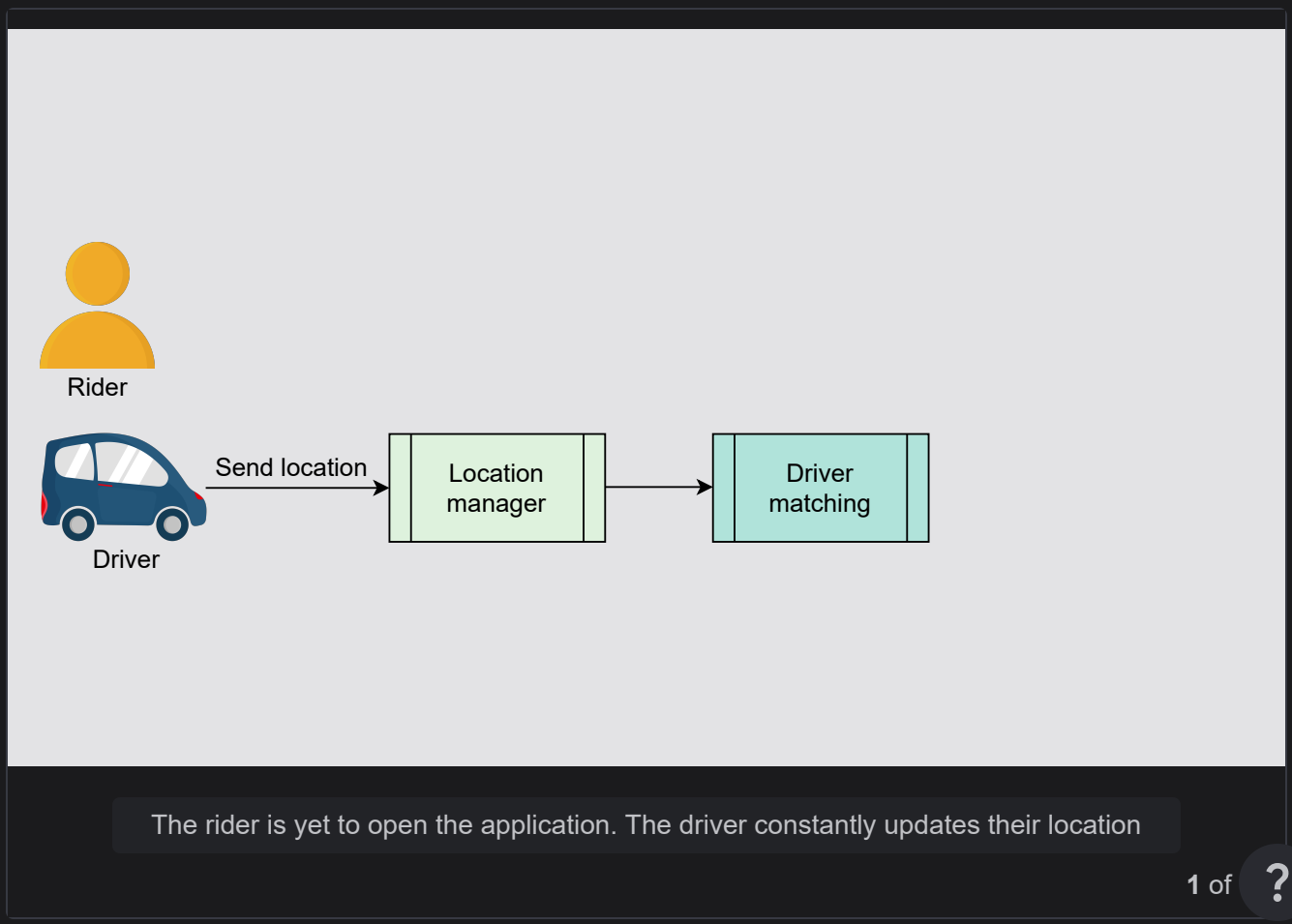
We'll cover the following

Workflow of our application

Before diving deep into the design, let’s understand how our application works. The following steps show the workflow of our application:

1. All the nearby drivers except those already serving rides can be seen when the rider starts our application.
2. The rider enters the drop-off location and requests a ride.
3. The application receives the request and finds a suitable driver.

4. Until a matching driver is found, the status will be “Waiting for the driver to respond.”
5. The drivers report their location every four seconds. The application finds the trip information and returns it to the driver.
6. The driver accepts or rejects the request:
    - The driver accepts the request, and status information is modified on both the rider’s and the driver’s applications. The rider finds that they have successfully matched and obtains the driver’s information.
    - The driver refuses the ride request. The rider restarts from step 2 and rematches to another driver.

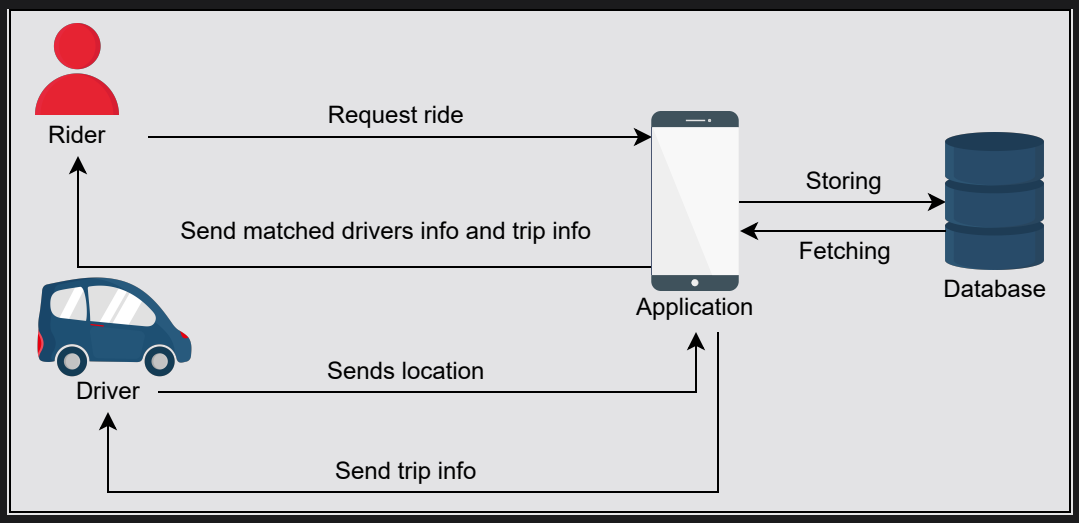


The rider is yet to open the application. The driver constantly updates their location

1 of ?

High-level design of Uber

At a high level, our system should be able to take requests for a ride from the rider and return the matched driver information and trip information to the rider. It also regularly takes the driver's location. Additionally, it returns the trip and rider information to the driver when the driver is matched to a rider.



High-level design

API design

Let's discuss the design of APIs according to the functionalities we provide. We'll design APIs to translate our feature set into technical specifications.

We won't repeat the description of repeating parameters in the following APIs.

Update driver location

```
updateDriverLocation(driverID, oldlat, oldlong, newlat, newlong )
```

Parameter	Description
`driverID`	The ID of the driver
`oldlat`	The previous latitude of the driver
`oldlong`	The previous longitude of the driver

`newlat`	The new latitude of the driver
`newlong`	The new longitude of the driver

The `updateDriverLocation` API is used to send the driver's coordinates to the driver location servers. This is where the location of the driver is updated and communicated to the riders. Sending `oldlat` and `oldlong` in the API call simplifies the server-side logic by reducing the need to fetch this data from the database. It also enables the service to track the actual route taken, even when the client could [illegible] low bandwidth, an alternate API without `oldlat` and `oldlong` can be used.

Find nearby drivers

```
findNearbyDrivers(riderID, lat, long)
```

Parameter	Description
`riderID`	The ID of the rider
`lat`	The latitude of the rider
`long`	The longitude of the rider

The `findNearbyDrivers` API is used to send the location of the rider for whom we want to find the nearby drivers.

Request a ride

```
requestRide(riderID, lat, long, dropOfflat, dropOfflong, typeOfVehicle)
```

Parameter	Description
`lat`	The current latitude of the rider
`long`	The current longitude of the rider
`dropOfflat`	The latitude of the rider's drop-off location
`dropOfflong`	The longitude of the rider's drop-off location
`typeOfVehicle`	The type of vehicle required by the rider—for example, busi and so on.

The `requestRide` API is used to send the location of the rider and the type of vehicle the rider needs.

Show driver ETA

```
showETA(driverID, eta)
```

Parameter	Description
`eta`	The estimated time of arrival of the driver

educative

Confirm pickup

```
confirmPickup(driverID, riderID, timestamp)
```

Parameter	Description

`timestamp`	The time at which the driver picked up the rid

The `confirmPickup` API is used to determine when the driver has picked up the rider.

Show trip updates

```
showTripUpdates(tripID, riderID, driverID, driverlat, driverlong, time_elapsed, ti
me_remaining)
```

Parameter	Description
`tripID`	The ID of the trip
`driverlat`	The latitude of the driver
`driverlong`	The longitude of the driver
`time_elapsed`	The total time of the trip
`time_remaining`	The time remaining (extract the current time from the ETA destination

The `showTripUpdates` API is used to show the updates of the trip, including the position of the driver and the time remaining to reach the destination.

End the trip

```
endTrip(tripID, riderID, driverID, time_elapsed, lat, long)
```

The `endTrip` API is used to end the trip.