HLD for Traveling Agency

Functional Requirement:(From Customer Perspective)

* Auth Login
* Search for Packages
* Package Details
* Enroll User
* Payment

Functional Requirement:(From Agent Perspective)

* Auth Login
* Create Packages
* Enrollment Details

Non Functional Requirement:

* Low latency
* Reliable
* Availability over Consistency
* Partition Tolerance

Capacity Estimation:

Assuming we have 100k users, out of which 1000 of them are actively browsing the App, the page size can go up from 1MB to 100MB(assuming we have images, gif and videos), so on average 1 person can take up to 50MB of bandwidth per page or package.

Considering each user browses around 3 packages daily.Therefore for 1000 users it will consume around 150GB of data per day. And if we convert by seconds it will be around

1.7MB/Sec.

Storage Estimation

We need to store 2 types of details

1. Users Details
2. Package Details

For User details we can opt for Relational Database(ex: mysql,postgres etc)

For Package details we can go for

* Package Meta Info: Document Store( ex: MongoDB) or Relational DBs
* Object Store for Videos and Images ( ex: S3,HDFS)

Assuming there are 1:30 active users for each package, Then the App must contain 34 packages of 50MB size. That comes to around 1.7GB of total data and by adding replication factor of 3 and compression factor of 80% , we might need a 4 GB hard disk.

API Design

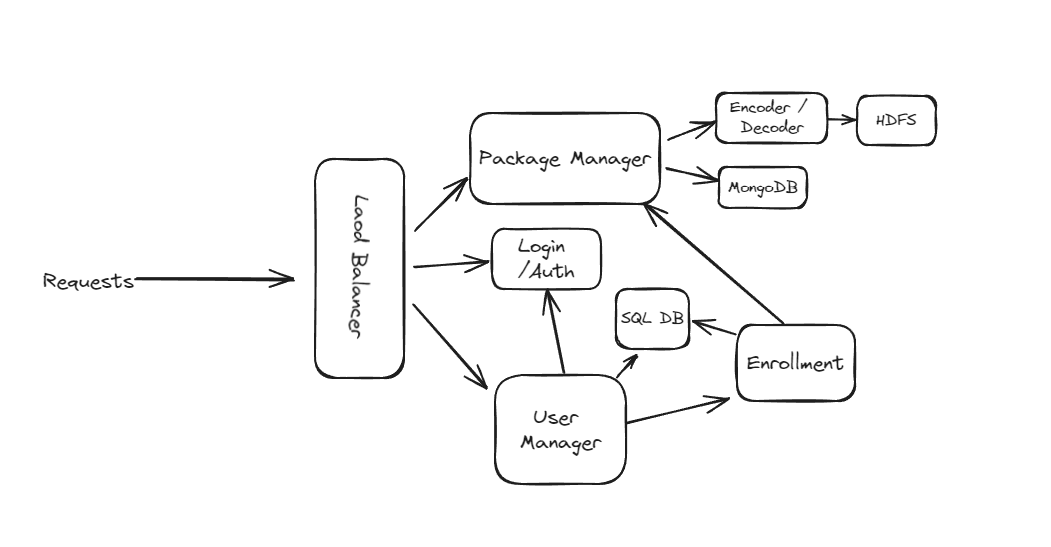
1. Create Package:

createPackage(packageName, Destinations, Cost, Capacity)

1. Enroll User:

enrollUser(userID,packageId,ItineraryOptedFor)

System Diagram



Scaling

We can scale all the system components Horizontally except Sequel databases, for the Sql DBs and replica sets we can scale them vertically.

Caching

We can implement a caching system for enrollment details and package details.

Message Queues

If in future we need to implement a notification module , we can send the notification asynchronously using a pub/sub system.

Pagination and Filtering

We can add filter params,offsets and limits to our APIs to reduce the data processing.

Rate Limiting and Load Shedding

We can rate limit the app based on IP/nodes/ request/sec.

Logging and Monitoring

We can add health check / heart beats for our system components and similarly alerts will be set up for any traffic breaches.

Resilience

* **Stateless**: We can set up containerized applications to help us in faster deployment and fault tolerance. we can rollback to the previous deployment. If something goes bad.
* **Stateful:** For any bad apple , we can remove the box from the cluster and debug the issue. And if it was a master box , we can replace the primary replica with the master.