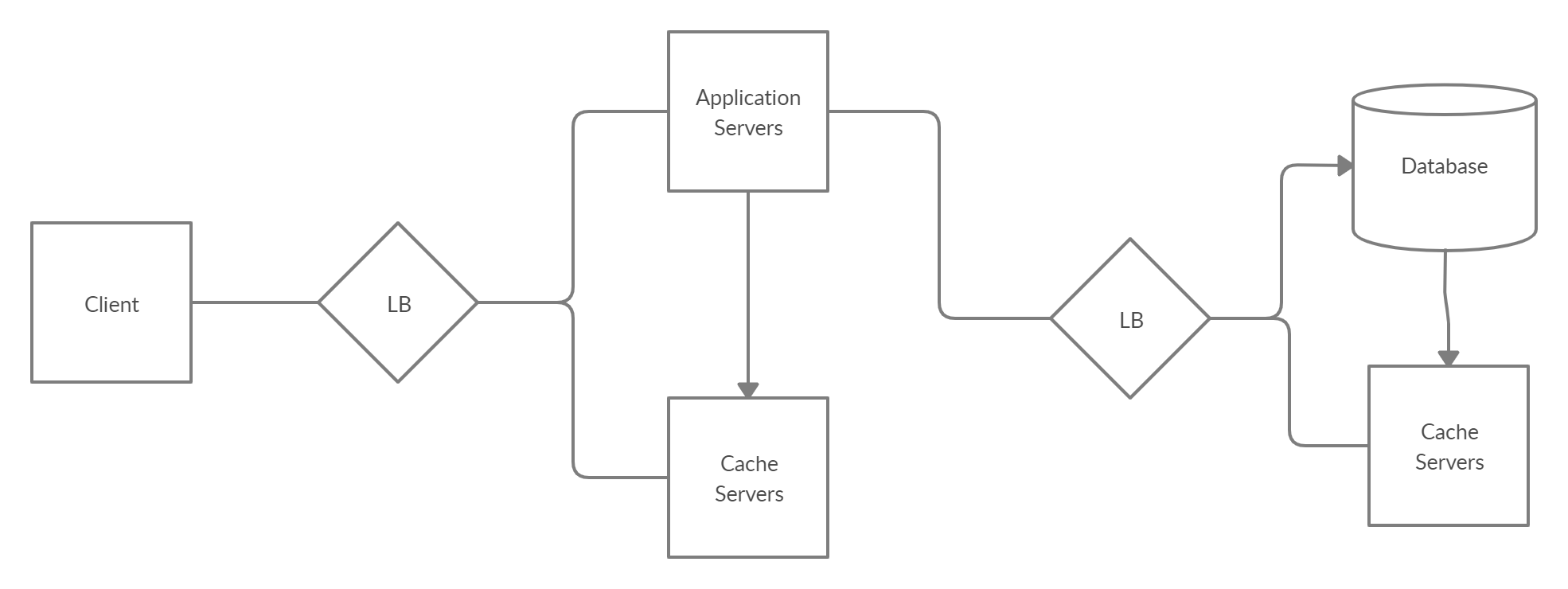
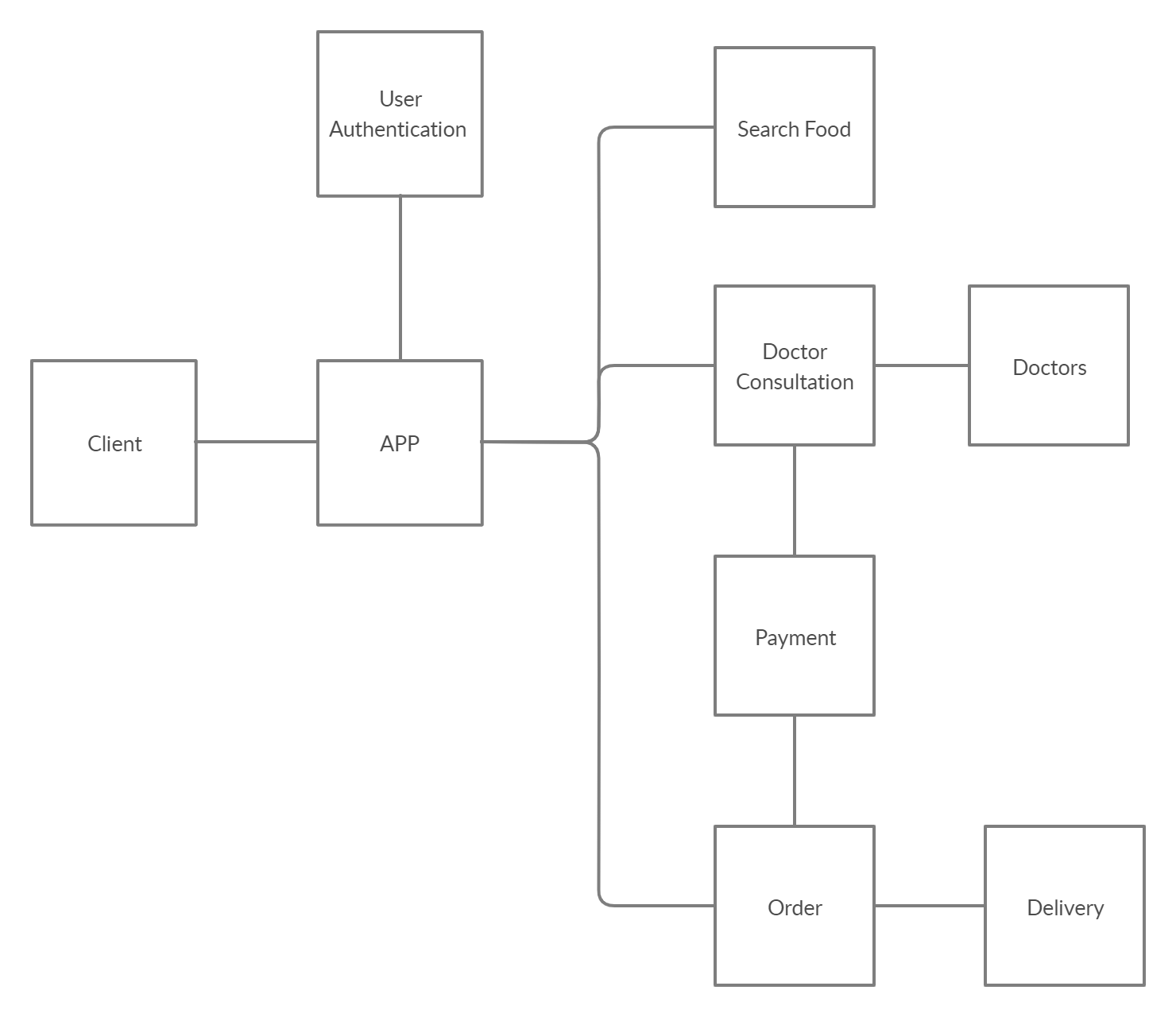
Design for any feature:



Microservice doc:



Design folder in creately: <https://app.creately.com/manage/project/uwuAgaOvuCl>

**User Authentication Service**

Responsibility:

Authenticate User

Data:

Phone no - primary key

Email

Name

Age

Gender

OTP

AuthToken

Rest APIs:

1. JsonString SignupUser(api\_key, name, email, age, gender, phoneNo)

Signs up user-

Return says whether there is an error or success

Return says why the error like duplicate no

If success invokes GenerateOTP

1. JsonString GenerateOTP(api\_key, phoneNo)

Generates OTP for a user

Return says whether there is an error or success

Return says why the error

Return says whether the client needs to retry

1. JsonString VerifyOTP(api\_key, phoneNo, OTP)

Verifies OTP for a user

Return says whether there is an error or success

Return says why the error

Return says whether the client needs to call GenerateOTP first

If success, return gives authToken

1. JsonString UpdateUser(api\_key, name, email, age, gender, phoneNo, authToken)

Verifies authToken and updates user

Return says whether there is an error or success

Return says why the error

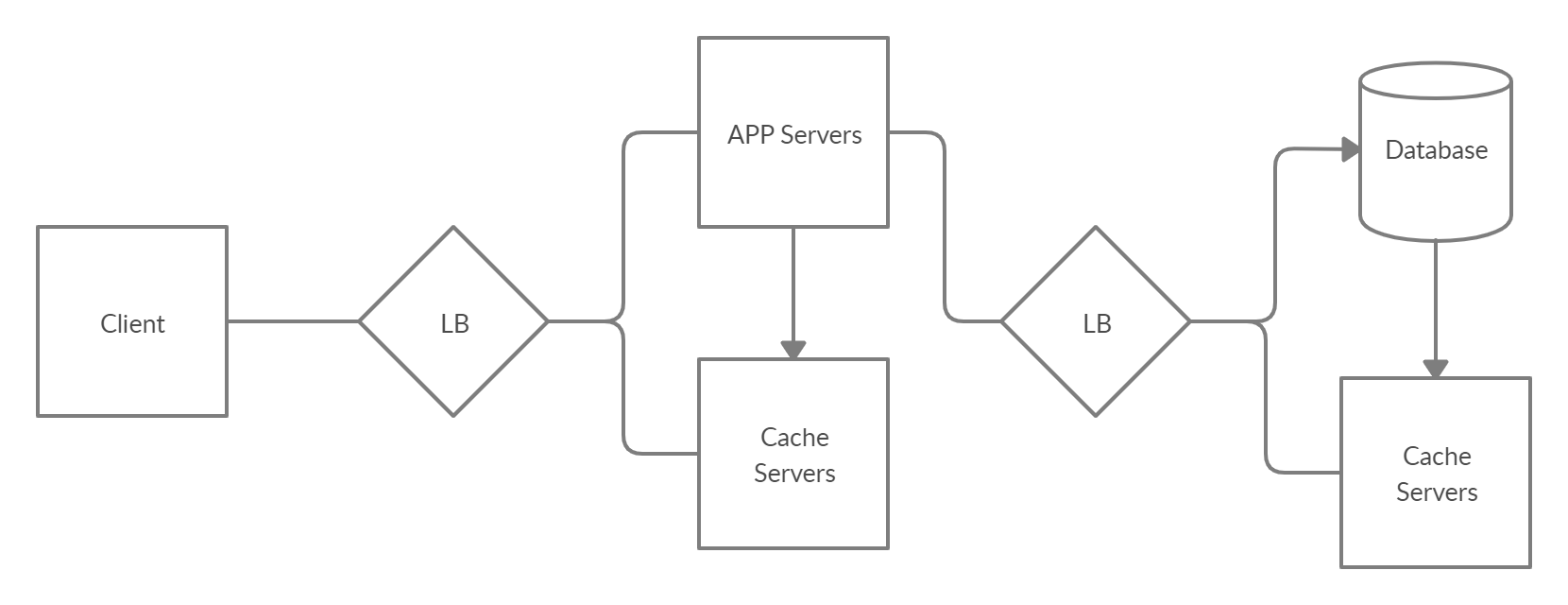
If success, return gives authToken

1. JsonString VerifyToken(api\_key, phoneNo, authToken)

Verifies authToken for a user

Return says whether there is a match or not

HLD:



Logic:

1. SignupUser stores user data into database. It also generates OTP and sends it to the user’s phone no. It also does some more things like checking if the no already exists and so on.
2. GenerateOTP checks if no exists. It generates OTP and sends it to the user’s phone no. It stores the OTP into the database
3. VerifyOTP verifies the given OTP and generates an authToken. It stored the token in DB and returns authToken
4. UpdateUser verifies the authToken and updates the user information and returns new auth token
5. VerifyToken verifies the given AuthToken

Capacity Estimations:

Obviously, the data will be less but we still will store it in multiple nodes

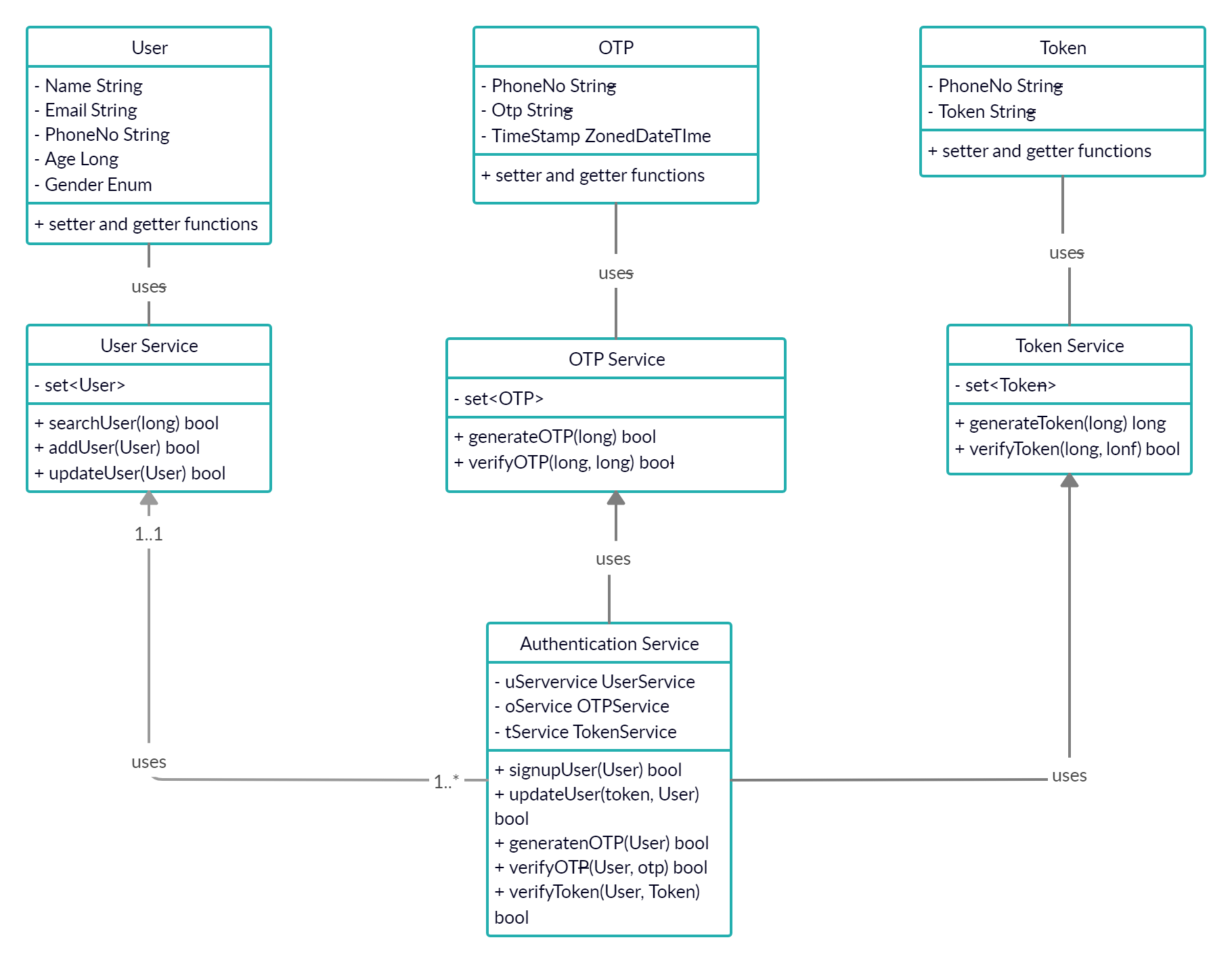
Database:

We will use Mongo as our database as we want to use NOSQL.

Sharding:

We will shard based on phone no

LLD:



**Food Service**

Responsibility:

Manages the food data and provides search APIs for User

Data:

Food and Diseases Mapping

Rest APIs:

Admin:

1. JsonString addFoodDiseaseMap(admin\_key, food, disease, correlationPercentage)

Authenticates Admin and add food, disease map

correlationPercentagetells how much is the mapping correlated

Return says whether there is an error or success

1. JsonString removeFoodDiseaseMap(admin\_key, food, disease)

Authenticates Admin and removes food, disease map

Return says whether there is an error or success

User:

1. JsonString searchFoods(api\_key, auth\_token, diseases)

Authenticates auth\_token

It takes diseases list as JsonString

Return says whether there is an error or success

If success, return gives a list of Food Object Ids and thumbnail paths

Front end can call their thumbnail images and show to user

One disease can have list of foods, so return list will be sorted based on their count and percentage in each disease

1. JsonString searchDiseases(api\_key, auth\_token, foods)

Authenticates auth\_token

It takes foods list as JsonString

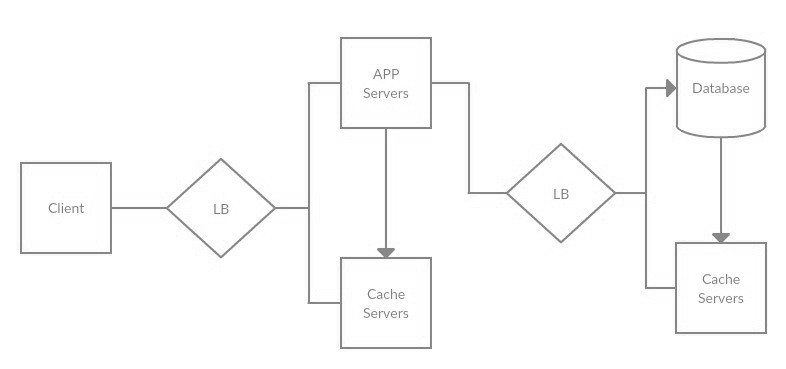
Return says whether there is an error or success

If success, return gives a list of Diseases Object Ids and thumbnail paths

Front end can call their thumbnail images and show to user

One food can have list of diseases, so return list will be sorted based on their count and [ercentage in each food

HLD:



Logic:

1. When the admin wants to add a map between food and disease, he can use this api.
2. When the admin wants to remove a map between food and disease, he can use this api.
3. We will run a SQL query for the foods having this diseases mapping. Once we get a result, we will sort them based on correlationPercetage
4. We will run a SQL query for the diseases having this foods mapping. Once we get a result, we will sort them based on correlationPercetage

Capacity Estimations:

Obviously, the data will be less but we still will store it in multiple nodes

Database:

We will use MYSQL as our database as food and disease are correlated.

Table will look like this:

|  |  |  |
| --- | --- | --- |
| Food | Disease | correlationPercentage |
| food1 | disease1 | 90 |
| food2 | disease1 | 80 |

Sharding:

As we are using MYSQL, we can only do vertical scaling. We will try to replicate information in multiple nodes as it is any way static data.

LLD:

