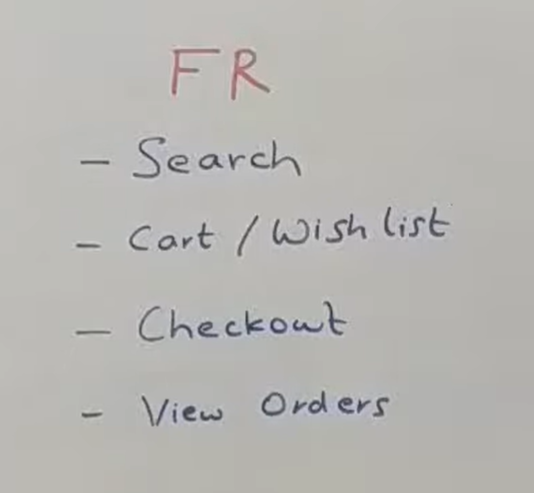
2.3 Amazon System Design

**Requirement** :

design an  e commerce application something very similar to amazon or flipcart

Let look some functional and non functional requirement that this platform should support

**FR**



**So the very first thing people should be able to search** what the product they want to buy and along with searching we should also able to tell them whether we can deliver them or not. So just say a particular user is in a very remote location where we cannot deliver a particular product then we should clearly call it out right on the search page.

Why let user has seen a hundred product and then they go to checkout flow add it into cart and then if we tell them that we can't deliver then it is very bad user experience

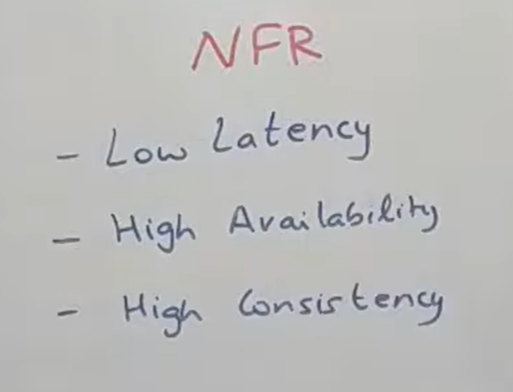
So at the page of search itself we should able to tell them we cannot deliver or if we are delivering then by when should we will able to deliver it to u.

**The next thing is there should be concept of cart or a whishlist**or something of that sort so that people can basically add a particular item into a cart. The next thing is people should be able to checkout  which is basically making a payment and completing the order will not look at how the payment exactly works like the integrating with payment gateways and all of that but we will broadly look at how the flow overall works.

The next thing is people should be able to view all of their past historical order as well.

The ones that have been delivered , the ones that have not been delivered everything.

**NFR**



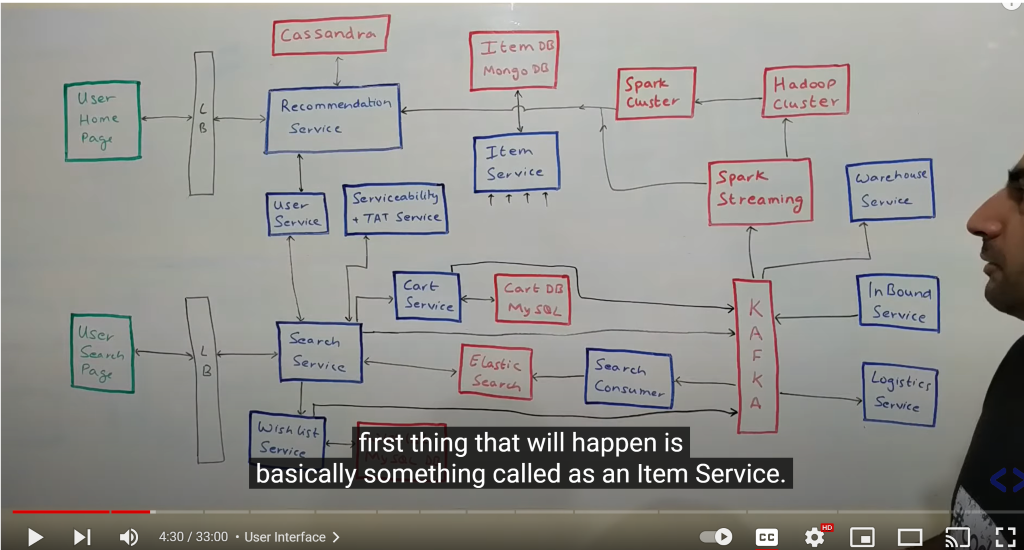
**From the non** functional side the system should have a very **low latency** why it will be a bad user experience if system is slow. It should be highly available and it should be highly consistent

So some of the product which are mainly dealing with payment and inventory counting they need to be highly consistent at the cost of availability. Certain components like search and all  they need to highly available may be at the cost of consistent at times.

And most of the user facing component should have a low latency.

Diagram

Description automatically generated



**Now lets start overall architecture of the whole system**

 We will do it into two parts. First we will look at the home screen and the search screen and then we will look at the whole checkout flows.

**Things in green** are basically user interfaces. It could be browser, mobile application anything. These two are not just **basically load balancer** but also reverse proxies and an authorization and authentication layer in between that will authenticate all the request that are coming from outside world into our system. All the things blue are basically the services that we have built. It could be kafka consumer , could be spark jobs could be anything. All the things in red are basically the databases or any cluster could be kafka cluster/ hadoop cluster or any public facing thing that we have to used.

Now lets look at the**user interface** what we have. So  we mainly have two user interface. One is the home screen which would be the first screen that user would come to it . In case of search page it would be a text box  where user put in the text and  will give the search result.

Will that let look how that data flow begin so a company like amazon would have various supplier that they would have integrated with now these supplier would be managed by various services on the supplier front. I am abstracting all of them something called as an **inbound service**s. What does it basically talk to various supplier system and get all of the data.

Now let say new  item has come or supplier is basically onboarding a new item. That information come through a lot of services and through**inbound  services** into kafka.

That is basically a supplier word coming into the whole search and user side of thing. Now there are multiple consumer on that kafka  which will process that particular piece of information to flow into the user world. Lets look at what that is so the very first thing that will happen is basically something called an **itemService so** basically  item service has lot of people talking to it so item service will be one of the imp thing that listen to this kafka topic and what it does it basically onboard a new item so it will provide various api to get an item by item id, add a new item, remove an item , update details of particular item and it will also have an imp api to bulk get lot of item. So get a get api with a lot of item id and it is in response it give details of all those item. Now item service sit on top of mongo db. Why do we need a mongo here so item info is fundamentally very non structured . Diff item types will have diff attributes and all of that

Need to be stored in a queriable format. Now if we try to model that in structured form into mysql kind of database that will not an optimal choice so thats why we use a mongo. To take an example let say if you look if you look at attribute of some product like shirt it would have a size  and colour attribute. If u look television some other attribute so it fairly a non structured data thats why use mongo db here.

Now coming to other consumer that will be sitting on kafka there is something called a search consumer. What it does is basically when a new item comes in  this **search consumer** is responsible for making sure that item is now avaliable for the users to query on.so what it does it all the items are coming in it basically reads through all of those item , it puts in the format that search system understand and it store it into its database. Now search uses a database called elastic search which is again a no sql

Database which is very efficient at doing text based queries.now this whole search will happen either on product name or product description and may be it will have filter on the product attributes . Fuzzy search is also supported by elastic search.

Now on top of elastic search there is something called **search service. This** search service is basically an interface that talks to the front end or any other component that want to search anything within the ecosystem.it provides various kind of api  to filter product to search by a particular string or anything of that type of sort. And the contract bn search service  and search consumer is fixed so both can understand what is the type of data that is stored within the elastic search. That is the reason consumer is able to write it and the search service is able for search  to it . Now once a user want to search something there are two main thing around it. One is basically trying to figure out the right set of items that need to be displayed but there is also an important aspect that should not show an item that we cannot deliver to the customer so for example if a customer is staying in very remote location and if we not able to deliver big item like ac to that pin code we should not show that search result to the user.

So search service talk to something called as  **serviceability and TAT (turn around time) service.** this basically does a lot of thing. First of all it try to figure it out where exactly the product is in what all warehouses. Now given one of those warehouse or some of those warehouses it tries to see do i have way to deliver product from this warehouse to the customer pincode and if i have what kind of product can i carry on

That route.now certain route can carry all kind of product but certain route might not able to carry thing like big product so all of these filtering stay in **serviceability and TAT.** now it also does one more thing that it tell in how much time i would able to deliver. Now if serviceability tell that i cannot deliver serach service will simply filter out those result and ignore that and  return the rest of the remaining thing . Now search might talk to **user service.**But user service is basically a service that is the source of true for the user data and it(search)  can query that to fetch(user service)  some attribute of user probably a default address basically which can be passed as an argument to serviceability service to check whether i can deliver it or not. Now search service return the response to the user which can be render and the people can see whatever they want to see.

Now each time a search happen an event is basically put into kafka. The reason behind this when somebody is searching for something they are telling you an intent to buy a product. That is very good source for building a **recommendation**. We will look how it works latter but this is the input into the recommendation engine and we will be using a kafka to pass it along so each search query goes into kafka saying this user id searched for this particular product . Now from the search screen user could be able to whishlist a particular product or add it to cart and buy it. All of those could be done using **whislist setvice and cart service.  wish**list is the repository all the wishlist in the ecosystem and the cart service is repository of all the carts in the ecosystem. Carts are basically shopping bag when people put item into it and then checkout. Now both these are built out in exactly the same way(almost). They provide api to add product into users's wishlist or cart. Get a user cart or wishlist or delete a particular item from wishlist.

And they would have a very similar data model and they are both sitting on own mysql database. Now from hardware stand of point i like to keep two separate h/w. Otherwise functionally both are totally same.now each time a person is putting product into wishlist or cart again they are giving signal. All of those can be put into kafka for a very similar kind of analytics. Now let look what those analytics would be . Now from this kafka there would something called**spark streaming consumer** . One of the very first thing that it does is kind of come up with some kind of report on what products people are buying right now . Those would be thing like coming up with report what was the most bought item in the last 30 min or what was the most wishlist item in last 30 min

So all of those would be inferred by spark streaming oher that that it also put all the data to **hadoop**saying this user has like this product , this user has searched for this product anything that happens. On top of it(hadoop) we could run various ML algorithm.

What product this user like and what other product like by other user so we can recommend product to the user all of those is calculated by **spark cluster on top of which** we can run various ML jobs to come up with the data. Once we calculate those recommendation this spark cluster basically talks to something called as **recommendation(r/m)  service** which is basically repository of all the r/m  and it has various kind of r/m .once is given a user id  if you store general r/m saying  what are the most recommended product for this user. And it will also store the same info for each category saying for electronics for this user these are the r/m. So when person is first on home page they will see general r/m and if they navigate into a particular category they will see the specific r/m we are specific to the category.

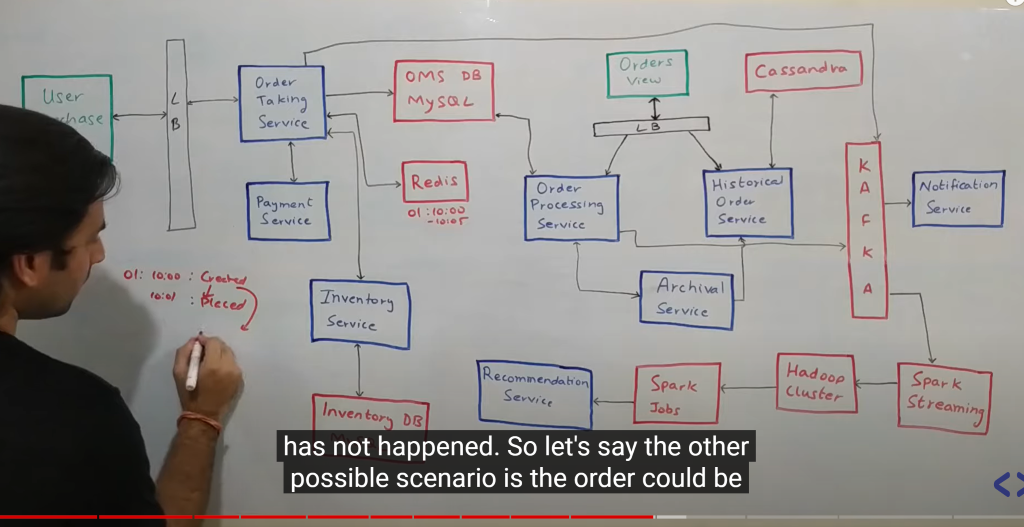
Now we have skipped a couple of component. **User service** is a very straight forward service. It provides various api to get details of a user , update details of user and all of that.it sit on top of mysql d/b and redis cache on top of it.**Now lets say search service want to get details of user** the way it work is first it will query redis to get details of the user if the user is present it will return from there but if the user is not present in redis it will query mysql d/b ( one of the slaves of that mysql cluster) get the user information stored  it in redis and return it back so that the user service will work

Now there are some other component that are here

One is l**ogistic service** and one is**warehouse service.** normally these two components comes in once the order is placed but in this case serviceablity might

Do query either of these two services not a runtime but before caching the information to fetch various attribute so for example it might query this warehouse service to get repository of all the item that are in warehouse or it might query logistic service to get details of all the pin that are existing so all of with those  info serviceability basically creates a graph kind of thing saying what is the shortest path to go from point a to point b and in how much time can i get there. Now we have not covered this service in details i have also implement google map this is very similar to that(serviceability). This doesnt do any calculation at runtime. It store all the info in cache and whenever anybody queries it will query the cache and return the result from the cache itself and no runtime calculation because those type of calculation are fairly slow.

**Now let look what happens when user try to place an order**



Now whenever user say i m ready to place an order take me to the payment screen

Which is the last piece in  the app flow then basically the request goes to something called **Order Taking service . Think** order taking service as part of order management system which takes the order. Now order management system sit on top of mysql d/b. Why mysql because if u look at order and table form of order . It will have a lot of tables some with order info some with customer info and there are lot of updates that happen on to the order database . Now we need to make sure that those updates are atomic and they can form a transaction so as to make sure that there are no partial updates happening . Now whenever order taking service first called the very **first thing** happen the record get created for an order and order id generated.**Now the next thing** we do put entry into redis saying this order id was created at some point in time and this record expires at some point.

Now the record goes into mysql d/b has initial status : created

The third thing happen is we basically called **inventory service. The ide**a of calling inventory service is we want to block the inventory. So let say there were 5 unit of television that a user want to purchase we will reduce the inventory count at this point of time and then send the user to payment. Now once the inventory is updated then user will go something called payment service but the interaction with payment can lead out 2 or 3 thing. One is payment is succesfull, payment failed or browser closed.

Now let see what can happen

**Payment successful** : order is placed and we update the status in mysql d/b. Now the work is done at that point of time an even is send into the kafka saying order has been placed with so and so details. Other scenario order could be cancelled : **payment failed**.Now if the payment failed we need to increment the inventory count again so thats a rollback transaction of thing will call inventory service for this. Now one more scenario user**close the browser**and payment service does not come back now what we will do. We cant keep the inventory blocked. Let there is one television and user close the browser so d/b says there is no television but it is physically present in warehose so now redis come into picture. Redis will have a expiry it wll have call back on top of redis which will basically be invoked saying that this particular report whatever you inserted it got expired. At that point of time order taking service will catch that event

And say that now this particular record has expired. I will follow the same that was followed for payment cancellation so at that point this order moved to cancelled state in mysql d/b and the inventory would get updated back again.

So everything is good but there are couple of scenario here

**1- what happens if your payment success event and expiry happens at the same time**

**So** there are 2 or 3 scenario in which this can happen

**Payment success at 10:01 and expiry happens at 10:05** so this is something that bound to happens for all orders. So one optimization for this each time payment success or failure we can delete the event from redis so as to make sure it expiry event does not come in.

**Other scenario is  expiry come first and then payment.**

Now whenever expiry occur 10:05 we need to make payment cancelled and decrease  count using inventory service but now the payment happened at 10:07 that we got to know. We can do 2 or three thing . We could either refund the money back to the customer  . Another way we already got payment so we could  make new order and change the status placed.

Now one thing i call it out this redis expiry is not very accurate we not might get call back exactly at 10:05. We might get callback 10:06 or sometime. It basically check at some point of time and if get some of key that are expired then it expire that.

Now once the payment happened or not happened we put all those event into kafka.

There can so many data in my sql but once the order is in terminal state like completed or cancelled we can move the order to cassendra so there is something called **archival service** which is a cron type of thing  which run every one day /12 hours  and it put the data from mysql into cassandra which reach in terminal state now how does it do that.

So **order processing service and historical order service** . These are two service are component of whole large order management system which do certain things of their own. So order processing service is the service which takes care of whole lifecycle of order . Once the order has been placed any changes to order happen through this.

Historical order service will provide api to fetch historical order.

Archived service will call order processing service and get all the terminal data and using historical order service it will get insert into cassandra and delete it from order processing service.let say there is some error then it will retry whole mechanism

Now user can go and see the past order that will be covered by **order view**

Let say there is order service which will talk to  **order processing service and historical order service and** get live data and historical data and merge it and display to the order view.

Why we use cassandra?

In cassandra we have finite number of query but large set of data.

Notification will be taken care by **notification service** like order got cancelled or delivered like sms , email.

While user placing all the order all the event goes into kafka on which we run spark streaming consumer . It does a lot of thing. One of the first thing it can publish an report what item have been ordered most in last one hour. It also put whole of the data into hadoop cluster. Spark jobs run on hadoop cluster which will give recommendation to user. So like if u order a white board u will also order a marker so all those of things taken care