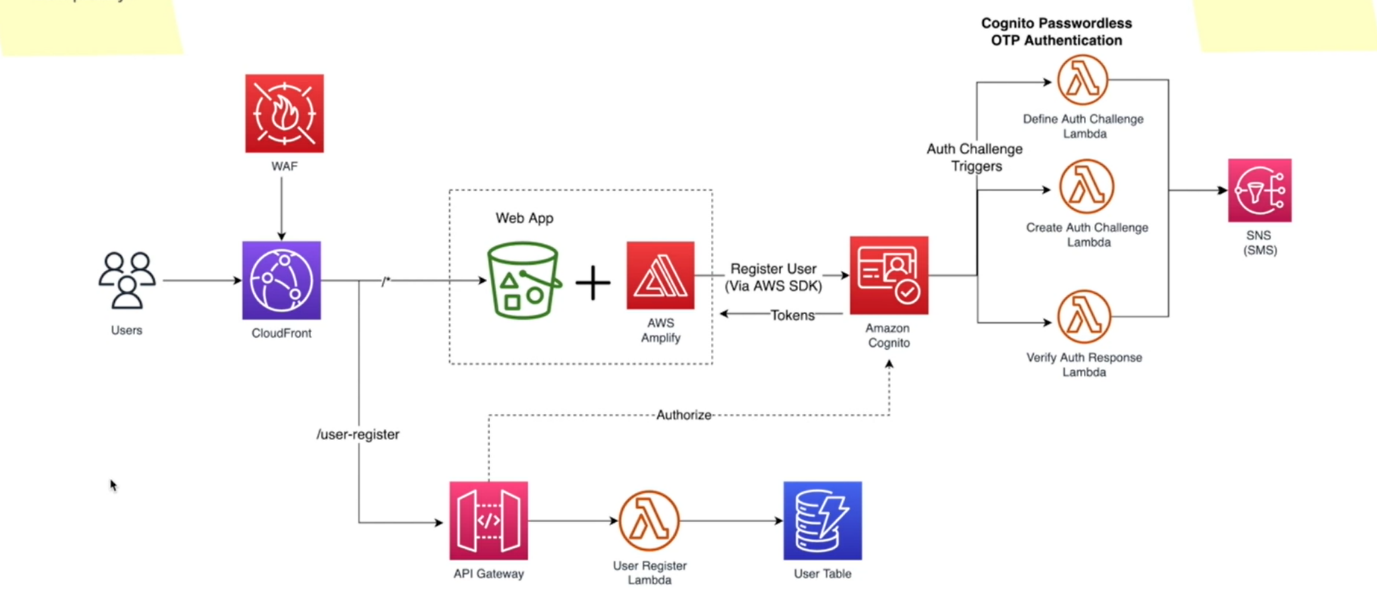
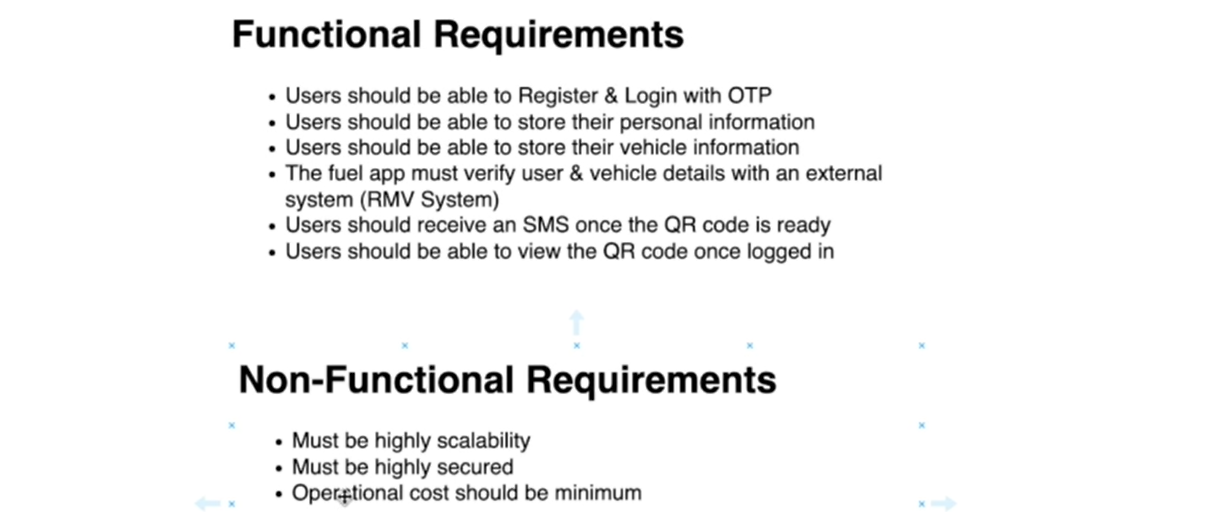
**5. Frontend Architecture Design**



**Non-functional Requirements**



* I will be using AWB services to build this architecture, now here I am starting with serverless because there is this non-functional requirement that our costs must be minimum. So, I do not want to pay for any idle time for this system because I understand at the time this system is launched, maybe on first few days there will be lots of traffic. But going forward, the traffic could be reduced.
* So, I always check whether we can use a serverless architecture so we do not have to pay for any idle time, but we will pay for only the usage and for some flaws, if the serverless architecture does not fit, I can always use a containerized application architecture, so it will use both serverless and containerized architecture kind of a hybrid mode.

**Architecture explanation**

So, I will explain the decisions as to why I use these services in this manner.

--- **S3**

* So, if I am architecting this application, I would put it into an S3 bucket as a static web application and serve it through cloud front.
* So here, the reason why I use S3 is its low cost, probably the lowest cost object storage that you can find and when we enable the static web hosting mode, it will be scaling automatically for you.
* So, I do not have to worry much about the scalability of our front-end application.

--- **cloudfront**

I will use the cloud front to serve this web application and cloud front access two things

1. One as an CDN content delivery network with caching support. So, when the users are requesting for the website, we will probably create this S3 bucket in Mumbai region because that is close to Sri Lanka and then the cloud front will start caching our static web application at the edge locations.
2. Another reason to use Cloud Frontier is to use it as a proxy. So, this accesses a reverse proxy, so it will proxy our web application as well as our backend API. So, all the requests that is coming from our users must go through cloud front. So, this gives us another opportunity for us to enable all the security related constructs at the cloud front level now that all the request must go through cloud front. If we enable services like WAF or Web Application Firewall, we can mitigate most of the web vulnerabilities.

**Functional requirement**

* **user should able to register & login with otp** - so let us have a look at our functional requirement. So, one of the functional requirements is that users should be able to register and login with OTP. So, this is basically the **Passwordless** authentication.
* **Amazon incognito** - So here I have used another managed service which is Amazon Incognito to handle our Passwordless authentication workflows.
* Amazon Incognito is a managed service and it will act as a user pool to your application. So, you did not have to manage your users in your application itself. You can decouple your users from your application and that is always a good thing because whenever you have so many users, you can scale those users separately so you do not have to scale your application
* when you have a big user base and it keeps growing and on top of that, Amazon Incognito is a managed service, so you do not have to worry about the scaling aspects of it and the charging is also based on the monthly active users.
* Now, Amazon Incognito supports these oath challenge triggers. For example, we can set up these different authentication challenges. In this case, users are supposed to receive an OTP through SMS, and then they are supposed to add the OTP and verify it. So, this is the authentication challenge process. An Amazon Incognito user pool particularly have this auth challenge triggers so we can define these different triggers such as create or challenge and verify auth response.
* So, we can essentially invoke lambda function in response to these triggers. So, when the users try to register, these lambda functions will take them through the OTP verification process. One will send the OTP and finally it will be verified as well.
* So once the user's authentication challenge is successful, they will be added into Amazon Incognito User Pool, and it will return tokens. Now these are the JWT tokens. These tokens essentially ID tokens and access tokens will be sent to the front-end application and in the front-end application we will be using AWS Amplify library.
* So, whenever these tokens are being received, this will be stored in local storage and AWS Amplify will handle the token refresh part as well.
* So, when a token is about to expire, amplify will handle the refreshing tokens using the refresh tokens behind the scene so you don't have to worry about that. Not only that, but also when amplify connects with these services, it will use signal for request Signing and that will ensure to call these Amazon services securely.
* **AWS SNS** - Now on the other hand, when the user is in the auth challenge workflow, users need to be notified with SMS. For example, in order to send the OTP, we must send an SMS to the user's mobile phone in order to handle all these SMS sending part. We will be using AWC SNS or simple notification service.
* Now this is also a managed service and SNS has this SMS notification type and that will take care of
* sending a SMS at scale.