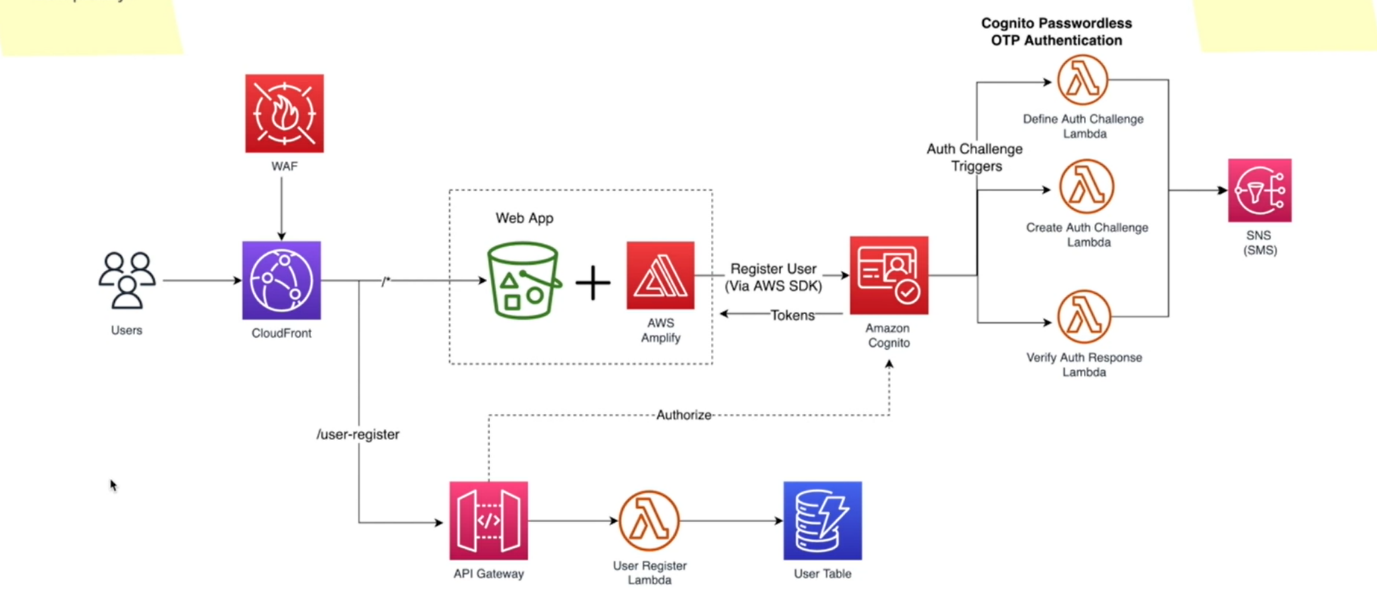
**7. Evaluating the User Registration Architecture**

**Architecture design**



how are we doing with our non-functional requirements. So, let's evaluate our architecture in terms of scalability, cost, security and complexity.

**Security**

* I can directly start with the security. As I already mentioned, we can define our security constructs such as WAF, web application firewall onto cloud front because this is the first point of contact for our application and we can block direct access to this S3 bucket using something called a cloud front OAI (origin access identity)
* So that users are not able to access this S3 bucket outside the cloud front. They always must go through cloud front when you are writing your code, the code in the lambda function must make sure that you do the input validation and follow other code in best practices.
* When you are writing the code now when one aws service call another aws service. In this case when aws amplify call Amazon Incognito. We use AWS SDK and it will use sic v for request signing and the token that we receive from cognitive user pool is also short lived here.
* Usually, the ID token and access token leaves for one hour and Amplify will use the refresh token to do the refreshing time to time and at the API gateway level, the request will be authorized with the Amazon incognito before calling the downstream services like use a Lambda Functions

**Scalability**

* So how about the scalability?
* Now, when it comes to scalability, I have taken a couple of approaches. most of these services are managed services with WAF, Cloud Front, S3, Cognito, SNS, Dynamo DB, API Gateway. Now most of these are serverless managed services, so they are inherently highly scalable and available that does not mean our architecture is also highly scalable and available.
* We must make sure when we are connecting these components or the managed services, we will do it in a way to ensure the scalability and availability in the whole architecture. We should not have any single point of failures.
* So as the first best practice, we use managed services. We are not going to create these services functionalities by ourselves and reinvent the wheel. We let AWS to worry about that and then we used most of these event driven triggers.
* aws Incognito has these event driven triggers. When the users are presented with the authentication challenge, Incognito will invoke this at the time they are required. So, we are not managing that.
* We are letting cognitive to invoke this service or the functions at the time that they are required. We only create the logic and add it to lambda functions that will be called by Amazon Incognito. Now have a look at this flow.
* When user information is going to be stored in a database table, we call it through an API gateway and it will synchronously invoke a user lambda function and it will synchronously call or store the data
* in a user table. So, one might wonder whether this is a good pattern. Well, obviously you might know that databases are inherently not as scalable as serverless functions. So as functions can scale up very easily, but compared to that, databases are not as scalable as it is.
* however, since we are using Dynamo DB and in the paper request mode, Dynamo DB can absorb quite several requests but however, we will be associating some concurrency limit to these lambda functions. We do this as a good practice because AWP is only allowing 1000 concurrent Lambda invocation per count.
* we can increase this limit by raising a support ticket. So here in this architecture, I would usually increase my account level concurrency, maybe up to 5000 or so. But still, again, I will add a cap or I will add a maximum concurrency, the lambda functions level.
* Here I would say, okay, this **user registered lambda function** can maximum have maybe let's say 100 or 200 concurrent lambda invocation. I will also define the same concurrency limits to other lambda functions as well because I do not want one of these lambda function to consume entire concurrency that is reserved for an account and that will suffocate all the other lambda functions in my architecture and all the other parts will start to suffer.
* So as a good practice, I will use this maximum concurrency. For that I will be using reserved concurrency for lambda.
* About our Web application I already mentioned that it is scalable and on top of that, we have cloud front and we will cash our front-end code at cloud front. So, when users try to load our application, it will not always go to S3 bucket and serve the website from the s3 bucket, it will be served from the cloud front cache.
* So here we have a reasonable scalability at user registration part.

**how about cost?**

* Now here we use serverless services.
* We have WAF, cloud front, AWS incognito, SNS, dynamo DB, API Gateway and so on. most of these services only charge for the usage, and if our application is not used by users, we almost do not have to pay.
* Unless a different WAF rules that charge you per hour, otherwise these services like Cloud Front, S3, Incognito, API Gateway and Dynamo DB. We do not really have to pay unless we are using it, including a sense.
* Now, towards the end of this series, I will calculate the cost for serving 1 million customers in this architecture.