***Details About The Assumptions:***

1. ***The application gets traffic from various regions***

***Implementation in the design :*** *CloudFront could offer a reliable solution to handle all the API requests of players without latency and high throughput thus results in smooth user experience.*

1. ***Building real-time leaderboards using a relational database would require a complex set of resource-intensive queries***

***Implementation in the design :*** *DynamoDB on demand (non-relational database). Provisioned DynamoDB might cost more when the traffic is unpredictable.*

1. ***Large stream of data at unpredictable rates***

***Implementation in the design :*** *AWS kinesis data streams to capture the data in real-time.*

***Pros :***

1. ***Serverless, quick to set up.***
2. ***Redis is very fast for lookups with low query latency.***
3. ***With DynamoDB on demand there is no need to specify read-write capacity - only pay for usage of dynamo table.***
4. ***With on demand provision, users don’t experience the throttled conditions which comes with provisioned capacity***
5. ***AWS kinesis data stream can easily scale from megabytes to terabytes per hour and from thousands to millions of records per second.***
6. ***With DynamoDB, we should know the access patterns beforehand unlike traditional SQL which is the key for efficient lookups.***
7. ***Cloud watch integration with AWS services (Kinesis, DynamoDB ) is very salient to keep records of logs, alarms and events.***

***Cons:***

1. ***Need to provision enough read/write capacity on your DynamoDB to provide balance across the table to avoid hot spots and throttled conditions.***
2. ***If there’s a need to reproduce code in another region / another AWS account / within the same when everything was deleted - Cloud Formation should be implemented to deploy code, create/update/delete our infrastructure.***
3. ***With on demand, the bill could be unexpectedly high ( if applications gets random spike)***
4. ***Traffic should not exceed table’s total provisioned capacity or the partition maxim capacity (Even with adaptive capacity enabled)***
5. ***When you get different access patterns than defined one, it is recommended to use secondary indexes for which AWS will charge.***

***When will the design break ?***

1. ***If the data exceeds the partitioned capacity of DynamoDB or limits of shards in Kinesis, throttled condition will be triggered which might probably result in loss of data and undesired results.***

***[solution : This condition can be avoided by implementing cloud watch with DynamoDB and Kinesis for continuous monitoring which will trigger the alarms though it might be expensive, users will have happy experience]***

***Under which conditions the design is an overkill ?***

1. ***When the application has to scale up and down according to the demands with unpredictable data streams , it’s crucial to have an auto scaling mechanism and at the same time it should prove to be cost effective. This design provides effective computation with the help of lambda where the lambda function can be easily triggered to changes in the request, DynamoDB.***
2. ***The websocket API gateway helps the application to be real time since it provides bi directional communication without the client having to poll for the request and provides event driven responses.***
3. ***This design works effectively for a gaming platform which demands interactive leaderboards, and to keep the users engaged where the application gets massive amounts of data to be managed & to analyze the data on demand.***