Assignment 4

1. **Describe the concept of throttling in APIs**

Before explaining the concept of throttling in **APIs**, it is important to define what an API is and how they work. First of all API stands for “Application Programming Interface”. It is an interface that developers use to **establish the interaction with software components or external resources from their own code**, in other words, it is a software component that connects to other products and services. This usually applies to engineers that develop an already existing project, but applications that have internal APIs are known to be very well designed. Some advantages that this brings, both for the programmer and for the code itself are:

- Programmers don’t really need to know how the components they’re using are implemented, they just simply use it.
- Reduces development time, therefore helps save money.
- Helps connect an infrastructure through cloud-native app development.
    - *Cloud-native application:* when an microservices application architecture is connected through APIs and it is a way to increase development speed. This is donde by companies in order to stay competitive by their rapid development and the innovation of their services.
- Allows you to share data with costumers and also external users.
- Flexibility to connect with partners (businesses can collaborate with IT teams), which provide opportunities for innovation.

Now, having define an API we can explain what **throttling API** is within the software development. It is the process of limiting the number of requests to an API that a user can make in a certain period of time. It is often used by organizations to asure as much as posible the security, performance, scalability, authentication and availability of their service. To have a better understanding, lets explain how it works:

- First the client or user calls an API that connects with a web service or application. Then the API throttling logic comes in when the recent request is over the limit of API calls, if not, everything performs as usual and the task wanted by the user is executed. In the contrary, an error message return to the user letting known that the request exceed the limit and will have to wait for a pre-agreed time period or pay a certain amount of money depending on how many more API calls it needs.

Why is this one of the biggest assets of organizations? It not only benefits users but also the website itself. One of the primary problems it resolves, is the unfair use of APIs. When there are multiple users connected to an application or a web page, signs of performance degradation starts to show, therefore users who have better connection have a better experience, so API throttling mechanisms are used.

2. **Describe the concept of pagination in APIs.**

Pagination is a process used to divide a big dataset into smaller and more digestible pieces, known as pages.

Page limitations

All Square API endpoints that support this practice also also support a *limit* field. This is used to indicate the amount of responses we want in return (page size). If this field is not specified, default number will apply. Fields such as *default* and *maximum* will vary depending on the endpoint, each one should provide page size details in the technical reference.

To achieve pagination goal, there are different methods on how to do it, depending on your APIs needs. Some are:

- *Offset Pagination:* it uses the *limit* and *offset* commands that are already present in the SQL library. *Offset* command tells the server the amount of items that should be skipped and the *limit* command indicates the amount of items that should be returned.

```
GET /<parameter>?offset = 0&limit = 10
```

In this example, with *offset* we indicate that are 10 paintings being searched at a time and with the *limit* parameter we indicate that it should skip any item.

- *Keyset Pagination:* it takes a cue from SQL database searching, by passing a parameter with a timestamp to be able to paginate through an API in a very meticulous way. It does this with the filter of the previous page, using a key to determine how it wants to be sorted and split it into pages.
    - For example, the most recent request is:

```
GET /items?limit = 20
```

    Here, after accessing the next page (first to second page), query finds the first 20 results (indicated with the *limit* command). Then, it can be used to filter the next page as the code below, indicating that it should paginate until the last page is reached.

```
GET /items?limit = 20 &created:lte:2019-01-20T00:00:00
```

- *Seek Pagination:* primary characteristics is consistency, as it orders persistently even when new items are added. *After_id* and *start_id* are newly added parameters.
    - Following the first context of the previous example (`GET /items?limit=20). After making this request and having the returned results, client now makes a second query using the *after_id* command:

```
GET /items?limit = 20 &after_id = 20
```

    Upon clicking the next page, user now finds the last id of 40 (indicated with *after_id)* from the results obtained before and makes a third query now using the same command as the starting id:

```
GET /items?limit = 20 &after_id = 40
```

3. **Describe the concept of callback function.**

When we talk about a callback function, we make reference to a function that is passed as an argument into another function, which later is invoked by the outer function. Expectation is that the supplied function will be executed, meaning it will perform the wanted action whenever a specific condition happens.

It is commonly used to make a separation of an application specific logic from the code that could be generic, in other words, separating event completions and/or operations from libraries, communication stack, etc.

```
function greeting(name) {
alert('Hello, ${name}');
}
function userInput(callback){
const name = prompt("Enter your name");
callback(name);
}

userInput(greeting)
```

Callbacks can be separated into two categories:

- *Synchronous:* it must be invoked before a function returns and it is used in the same thread context as the function that is calling it.
- *Asynchronous:* on the contrary with synchronous callbacks, *asynchronous* are executed after the invoking function returns, which is common to happened in another thread of execution.

There are multiple components in callbacks:

- *Storage for the callback*: usually a variable that holds only a single value, static array with a *max_size* or a dynamic list.
- Function which makes the invocation to the registered callback at the proper time
- Mechanism/function that registers the callback. Can also be a constructor or initialization function parameter or configure it with *struct.*
- Callback function should have concrete implementation, meaning it is application-specific and supplies the behavior that is not specified.
- Concrete callback implementation should be registered within the proper module.

4. **Describe the concept of cold start in AWS Lambda.**

Firstly let's define what AWS Lambda is. **AWS Lambda** is a compute service provided by Amazon, that allows code to be executed without managing any servers. It is bases on the response of events in AWS, meaning that code is on a high-availability infrastructure and performs the administrations of the compute resources provided, server is included, the maintenance of the systems, automatic scaling, capacity provisioning and logging. Quite popular, since it allows you to focus only on your software of your product and the business aspect of it, instead of managing access controls, operating system, etc.

- There is no need to worry on how you should launch AWS resources or how to manage them. You just need to supply the code in one of the languages that Lambda supports and run it.
- **NOTE**: It can only be used to execute background tasks.

Cold Start happens when the first request is made to AWS Lambda and it makes reference to the number of times it takes AWS Lambda to "warm up" containers before they can be used and functions are operational. Containers used by Lambda to execute the code have a predefined runtime environment, therefore the first function being called takes longer to process. AWS demands a supply of containers for spinning up, it means that functions should be kept warm for a limited time after executing, so that the container is ready for any new request.

This issue is primary relevant to applications that need to be executed in real-time or those that rely on split-second timing.

Best practices to reduce/avoid AWS Lambda cold start:

- Avoid *HTTP/HTTPS calls:* they cause cold starts and increases the invoking time of a function. This is because when an HTTPS call happens, it not only runs this call, at the same time there are SSL and multiple security related calls are ocurring.
    - It is recommended to only use the HTTP client available in Amazon SDK.
        - Amazon SDK provides with multiples HTTP client libraries that allows you to make and SDK call.

- *Reducing Setup Variables:* setting up static variables within a serverless container takes a while, the more static variables you have, the more cold start latency will occur.
    - It depends on the size of the first function and its complexity.
- *Preload dependencies:* highly recommended to uses handlers between the invoking service and the function, to load all dependencies before the invocation is made. It tries to resolve the problem of importing the dependencies after invoking functions, which will consist on waiting a lot of time.
- *Avoid huge functions*: even though the serverless ecosystem breaks the monolithic structure into various granular services, having large functions imply more set up time, since you still need ro analyze the architecture, determine how to separate concerns and complexity coming as factor.

5. **Describe each HTTP methods.**

HTTP stands for “Hypertext Transfer Protocol” and is used to load web pages via hypertext links. Application layer protocol designed to be able to transfer information between various networked devices. Is the foundation of the World Wide Web. The way the internet establishes communication between users and web browsers is with **HTTP requests**.

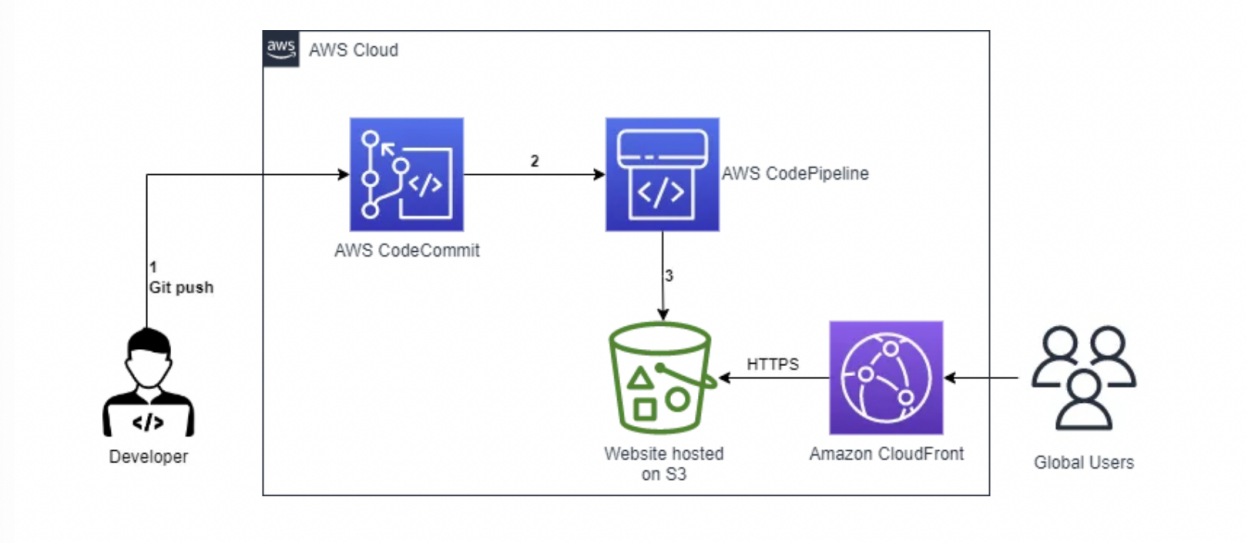
HTTP methods are ofter referred to as “ HTTP verbs”. There are multiple methods already defined by HTTP to indicate a desired action to be performed. Although methods share common features, each of them have different semantic to be applied.

- *GET:* used to obtain information from the specified resource. It requests a representation, which means it is read-only.
    - It retrieves data by specifying parameters in the request.
- HEAD: does the same as ‘GET’, but it transfers the status line and header section, not the body.
    - It sends the header, and no further data is sent after that.
- POST: it submits/sends data to the specified resource. It causes a change in state or some side effects on the server. (Create operation)

- PUT: it replaces all current representations of the target resource by uploading the content to the body in the server. (Update/replace operation)
- DELETE: it removes all current representations of the specified resource.
- CONNECT: it establishes a tunnel to the server that is identified by the target resource.
    - Client establishes a network connection to a web server over HTTP.
- OPTIONS: it describes all communication options for the target resource.
    - When the client want to know the HTTP methods and other options supported by a web server.
    - The client specifies with an URL the target server or can use an asterisk *, which refers to the entire server.
- TRACE: it performs a message loop-back test through the path to the target resource.
    - Used to echo the contents of a request back to the requester. Often used for debugging at the time of development.
- PATCH: it applies partial modifications (updates) to a specifies resource. (Update operation)

6. **Describe how you can automate a deployment of a static website to S3.**

One way to do this automation is using **AWS CodePipeline.** It is a continuous delivery service used to automate the process of building, testing and deploying a website/application into a testing or production environment. Defined by CI/CD pipeline, which stands for “Continuous Integration and Continuous Delivery).



When making a push of an .html file to a remote repository in **AWS CodeCommit**, AWS CodePipeline should be trigger to automatically deploy the file to an S3 bucket. Then with a **AWS CloudFront** distribution we make sure that any traffic is redirected to HTTPS, since it will deliver the content to global users.

What is AWS CloudFront?

It is a web service provided by Amazon, that delivers static and dynamic web content through a global network using edge locations, allowing access to users globally. User requests are routed to an edge location that provides the lowest latency.

What is AWS CodeCommit?

Is a secure, highly scalable and managed resource control service that facilitates teams to collaborate on code. It eliminates the need to operate your own code source control system or the scaling of infrastructure.

- Can be used to store anything (code, binary files, etc) and works seamlessly with existing Git tools.

Having defined all important conceptos, this are the steps that should be followed:

1. First, a new AWS CodeCommit repository should be created and then cloned. An HTTPS link should be provided when creating the repo, with it you can clone it.

2. Using the command line, the websites file (webpage.html) should be added to staging with the following command:

```
git add website.html
```

   - Make the initial commit with:

```
git commit -m "initial commit"
```

   - Then push the changes to the AWS CodeCommit remote repo with:

```
git push
```

3. Create an S3 bucket to host a static website. Here is important to edit the buckets policy, by adding a JSON that grants public read access to the website. An examples is as follows:

4. Create a CI/CD pipeline, it can be done by running this command:

```
aws codepipeline create-pipeline --pipeline
```

- A source stage should be added. This is where all input artifacts for your pipeline will be stored. (AWS CodeCommit)
- A deploy stage should be determine. Select Amazon S3.

5. Verify if the pipeline was created successfully by making a change in the file, added and commit it to the repo before pushing. Access the bucket website endpoint and confirm if the change was reflected.
6. Create a the CloudFront distribution. It can be done by running the following command:

```
aws cloudfront create-distribution --origin-domain-name fi
rstpage.s3.us-east-1.amazonaws.com
```

   - You must specify the origin domain which is your s3 bucket. Once created a .net link will be provided, that is your distribution domain name and can be used to access the website.

This way we have now successfully created an Amazon CloudFront distribution that allows caching of the static webpage. Now the website is reachable with the CloudFront CDN and users request are redirected to HTTPS, controlling traffic.

7. **Read the Real-world Engineering Challenges #8: Breaking up a Monolith article and write a summary and opinions about it.**

Summary

The article dives in into the challenge that Khan Academy phased in 2019, when they needed to migrate their project of one million lines of Python code to Go and splitting them across over 40 services. The process of the migration to a different language lasted 3.5 years and involved 100 software engineers.

Despite already having experimented in the past to change their tech stack for this project, the thing that make them take immediate action was Python 2’s end-of-life announcement. But this not only meant to migrate, they need to:

- Get off Python 2 and delay was not an option because of Python 2 end of life.
- Replace the REST API with GraphQL, which they experimented with it before.

- Break the Python monolith into services.

Project was divided intro two phases:

- Phase 1 - Minimum Viable Experience: In this stage they had to decide which were the key features of the product, which if removed would alter the primary functionality and Khan Academy's identity. Features in the MVE were mostly about content publishing, content delivery, progress tracking, etc.
    - This phase took approximately 2 years to complete (August 2021).
    - 95% of the website traffic was navigating through the new code.
    - 32 services were built.
- Phase 2 - Endgame: In this stage the remaining things were migrated, there was still a lot to do. Rewriting the remainder of the Python monolith, all internal tools needed to be rebuilt from Python to Go.
    - This phase took 1.5 years to complete.
    - New features were built.
    - 5 more services were created, bringing the total amount to over 40.
    - Non-MVE behaviors were added to existing services.

For migration, the team needed to build the basic infrastructure of the new service, which meant putting GraphQL federation service in place.

- "filed by field" was the chosen approach.
- Majority of traffic was flowed through the existing monolith.
- Side-by-side testing of GraphQL services.
- Test coverage was increased because of new tests in Go.

Opinion

There is no doubt that they were forced to do the migration because of Python 2 end of life, it wasn't a decision made because of a previous research on how to improve their product. I think it's important to always keep track on the life-time of the tech stack being

used and evaluate if there are any that have more benefits that the current one and start working on migration with time and with a well established plan.

Viewing the positive things, is that in a sense migration was “easier”, because they already had a product, which is why they decided for MVE instead of a MVP. This way it helped developers focus more on the development of new services or how to improve existing ones to be more “user friendly”. Also, testers crated much more tests in Go than what they had in Python, assuring the functionality by increasing the coverage.

Definitely it wasn’t an ideal situation for developers, because of how hasty it was and mainly because they have to adapt to a new language and a learning curve starts, which is normal, but in this case developers had to learn while also having to deliver results.