Functional description

Knowledge test to apply for the position of Backend Developer

Solution Description

To solve the problem raised, a completely serverless solution was implemented using AWS Lambda, developing the entire solution in Python 3.8 and Flask 2.0. The implementation is a Microservice that exposes a set of Endpoints that will provide a solution to the different problems raised.

It is important to point out that in addition to the requested functionalities, a set of additional functions were implemented, which seek to give the solution greater flexibility and help provide a more complete solution.
A continuación se describiran textualmente cada una de las funcionalidades implementadas.

Record log individually

This function was not requested to be built, since it was assumed that the log records were already stored by nature. However, in any service in charge of processing logs, you must have an entry point to be able to register them. Additionally, a mechanism is provided so that the team that will review the solution has the possibility of creating the data sets they want to test the different functionalities of the solution.

Name	Register log
Endpoint	/createSingleLog
Objectives	
1. Store a Log of an application in an S3 Bucket	
Business rules	
1. Application, category and description are required. In case of recording an error, the degree of severity of the error must be included. 2. The category value must be in a closed range of values. 3. The error severity value must be defined in a closed range of values. 4. No input value should contain the characters '-' or ':' 5. For each day a file is created to store logs. 6. If you are going to create a log and the file corresponding to the day has not yet been created, you must first create the file and then proceed to save the log. 7. The log must be saved in one line of the file and in the format specified in the description of the problem. 1. 8. Each log is stored with the time stamp recorded at the time of its creation.	
Inputs	
1. **application:** indicates the identifier of the application that generated the Log. 2. **category:** indicates the category of the log (SUCCESS, INFO, ERROR) 3. **levelError:** indicates the severity of the error (1,2,3,4,5) 4. **description:** general description of the blog	

Expected out
1. If the log is stored correctly, an HTTP Status 200 and a message indicating that the operation was successful is returned. 2. If there is a problem, an HTTP Status 400 is returned with a message indicating the problem.
Security considerations
1. Access to the method is done by authenticating the request using a JSON WEB TOKEN (JWT).

Statistics of logs loaded from a file

This function solves question number 1 of the questionnaire received.

Name	Logs stats in upload file
Endpoint	/statsLogsByFile
Objectives	
1. Read a file with a .txt extension, with the information stored in the same format as the logs are stored in S3. 2. From the reading of the file, a statistic must be returned that compares the logs by their category.	
Business rules	
1. The file must have a .txt extension 2. The information within it must have the same structure as the files where the logs are stored in S3	
Input	
1. **file**: text file must be processing	
Expected out	
1. If everything happens correctly, an object is returned indicating the number of SUCCESS type logs, ERROR type logs and the total number of logs stored in the file. This response is marked by an HTTP Status 200. 2. If an error occurs during the execution of this method, an HTTP Status 400 and a message indicating the problem occurred are returned.	
Security consideration	
1. To interact with this functionality, it is not necessary to implement any authentication mechanism.	

Log filtering.

This functionality directly answers questions 2 and 3.2 specified in the sent questionnaire. Both questions were built on a solution that was implemented using a Lambda Function, which is accessed by an Endpoint provided by API Gateway.

Name	Filter logs
Endpoint	/filterLogs

Objectives

1. Filter the logs stored in the last created log file based on various criteria.

Business rules

1. At least one of the filter parameters must be received.
2. When the limit or offset parameter is passed, both must be accompanied by each other, they cannot be passed individually.
3. The start and end dates must correspond to the format YYYY-MM-DDTHH:MM
4. The start date cannot be greater than the end date.
   If only the start date is passed, the logs generated from that date are searched, with no end date restriction.
5. If only the end date is passed, all the logs that were generated before that date are searched.
6. All variables to set the query can be combined.
7. All logs are loaded from the last log file created.

Inputs

1. **timestampBegin:** start date to search for logs
2. **timestampEnd:** end date to search logs
3. **application:** Identifier of the application that generated the log.
4. **category:** category of the log to be searched.
5. **severity:** degree of severity of the searched logs
6. **limit:** number of elements that can be loaded per page in each query.
7. **offset:** indicates the point from where the search segmentation will start.

Expected out

1. If there is no file that stores logs, the resulting empty list of logs is returned through an HTTP Status with value 200.
2. If the combination of input parameters does not return any results, an HTTP Status with value 200 is returned, indicating that no results were found for the specified search.
3. If limit and offset are not specified, an object with the list of resulting logs is returned through an HTTTP Status 200.
4. If limit and offset are specified, an object is returned under an HTTP Status with value 200 indicating: list of elements, limit with which the request was made and offset through which the request was made.

Security considerations

To interact with this functionality it is not necessary to implement any authentication mechanism.

Note: at the end of the document where the technical test is specified, it is asked how paging would be implemented, with the solution and implementation of this method, the concern presented is answered.

Create subscription mechanism

This method was not requested in the exam, but in order to generate the necessary data to complete the notification functionalities, it was decided to create this mechanism to store user subscriptions to application events.

Name	Create subscription
Endpoint	/createSubscription
Objectives	
1. Create a mechanism that allows users to subscribe to application events.	
Business Rules	
1. Subscriptions will be stored in a file located in S3.	
Input	
1. **email:** email of the registered user. 2. **application**: Identifier of the application to register to. 3. **category**: category of the log to which you want to subscribe.	
Expected out	
1. If everything happens correctly, the subscription is inserted and an HTTP Status is returned, with a value of 200 and a message indicating that the process was completed correctly. 2. If there is a problem, an HTTP Status with a value of 400 and a message indicating the problem that occurred is returned.	
Security consideration	
To interact with this functionality, it is not necessary to implement any authentication mechanism.	

Notify according to subscription

Function that is located after Endpoint, which is consumed by a Cron or scheduled task at the end of every day. The objective of the process is to send all the logs that match the subscriptions of the users.
This method solves question 3.1.

Name	Notify with logs
Endpoint	/notify
Objectives	
1. Create a notification mechanism that allows users who subscribe to the platform to receive daily logs generated corresponding to the subscription made.	
Business rules	

1. The logs will be grouped and an email will be sent to each subscribed user, indicating in the body of the same, the details of the stored logs that correspond to the subscription made by the user.
Inputs
1. There are no entries
Expected outs
1. Si todo ocurre correctamente, se devuele la lista de los mails enviados y se devuelve un HTTP Status, con valor 200 y un mensaje indicando que el proceso se completó correctamente. 2. Si existe un problema, se devuelve un HTTP Status con valor 400 y un mensaje indicando el problema ocurrido.
Security considerations
To interact with this functionality, it is not necessary to implement any authentication mechanism.

Generate security token

It allows to generate a security token to later use this in the calls to the functions that request said authentication. The authentication and token generation process is not something that was requested, but modern systems cannot be conceived that do not have these mechanisms to validate the request.

Name	Generate token
Endpoint	/getToken
Objectives	
1. Generate a token to be used in requests requesting authentication.	
Business rules	
1. A token must be generated from the symmetric encryption that is performed on a set of data received by parameters.	
Inputs	
1. **email**: email of the user who is requesting the token 2. **userId**: identifier of the user who is requesting the token	
Expected out	
1. If everything happens correctly, the generated token and an HTTP Status are returned, with a value of 200, as well as a message indicating that the process was completed correctly. 2. If there is a problem, an HTTP Status with a value of 400 and a message indicating the problem that occurred is returned.	
Security considerations	
To interact with this functionality, it is not necessary to implement any authentication mechanism.	