Implementierung

4. April 2017

Inhaltsverzeichnis

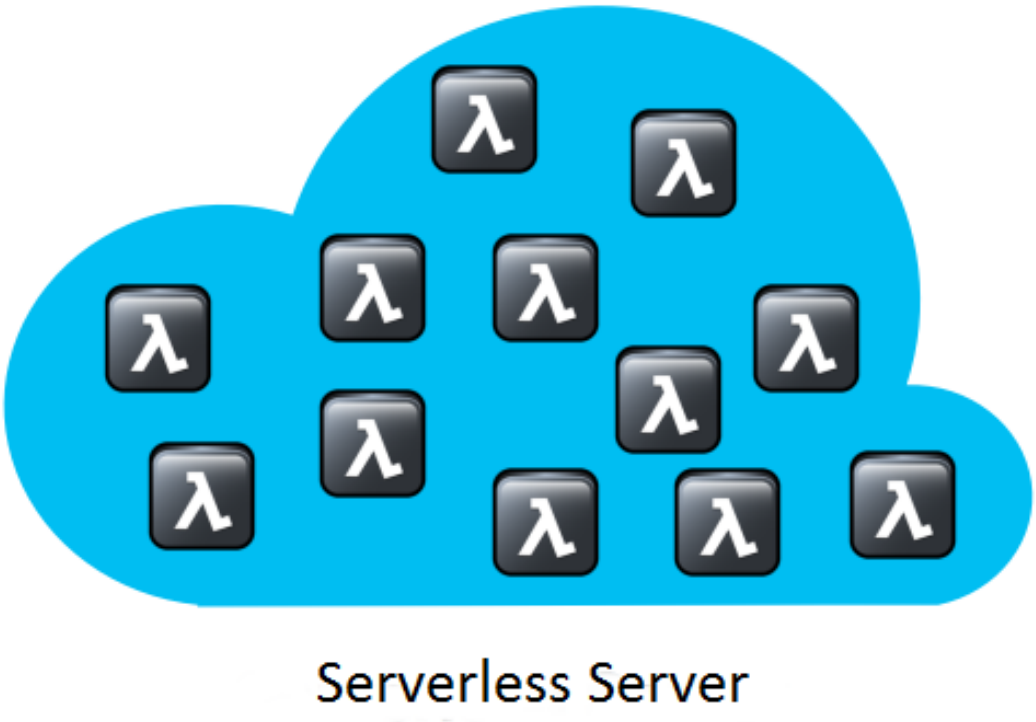
Kapitel 1

Einleitung

In diesem Dokument wird die erste Implementierungsphase des Serverless Server Projekts beschrieben.

Zuerst werden die Probleme und Änderungen im Vergleich zum Entwurf angezeigt.

Im nächsten Teil werden die Testfälle dargestellt. Sie testen die Klassen, welche die Hauptfunktionalität des Projekts implementieren.



Kapitel 2

Probleme und Änderungen am Entwurf

In diesem Kapitel werden die Änderungen dokumentiert, die während der ersten Implemetierungsphase gemacht werden.

2.1 Schichtenmodell

Zum verbesserten Verständnis des Systemaufbaus, steht zu Beginn eine Grafik, die das Schichtenmodell beschreibt.

2.2 ApiController

2.2.1 RestApiController

- Der Name der Klasse ApiController wurde in RestApiController umbenannt.
- Fast alle Methoden wurden zur besseren Verständlichkeit umbenannt.
  - uploadLambda(UploadLambdaRequest): ResponseEntity<UploadLambdaResponse> wurde in postLambda(UploadLambdaRequest): ResponseEntity<UploadLambdaResponse> umbenannt.

  - executeLambda(nameOfLambda: String, config ExecuteLambdaRequest): ResponseEntity<ExecuteLambdaResponse> wurde in postLambda(nameOfLambda: String, config ExecuteLambdaRequest): ResponseEntity<ExecuteLambdaResponse> umbenannt.

  - updateLambda(name: String, config: UploadLambdaRequest): ResponseEntity<Void> wurde in putLambda(name: String, config: UploadLambdaRequest): ResponseEntity<Void> umbenannt.

  - showLambda(name: String): ResponseEntity<UploadLambdaRequest> wurde in getLambda(name: String): ResponseEntity<UploadLambdaRequest> umbenannt.

  - generateSubtoken(name: String, expiryDate: String): ResponseEntity<String> wurde in getSubtoken(name: String, expiryDate: String): ResponseEntity<String> umbenannt.

- Das Attribut tokenCreator: TokenCreator wurde durch authFacade: AuthFacade umgesetzt. Es folgt aus den Änderungen des Auth Teils.

- Der Rückgabetyp der Methode putLambda(name: String, config: UploadLambdaRequest): ResponseEntity<Void> wurde zu ResponseEntity<UploadLambdaResponse> verändert.Der Grund ist: token verändert sich nach update, deshalb soll es zurückgegeben sein.

2.2.2 Exception

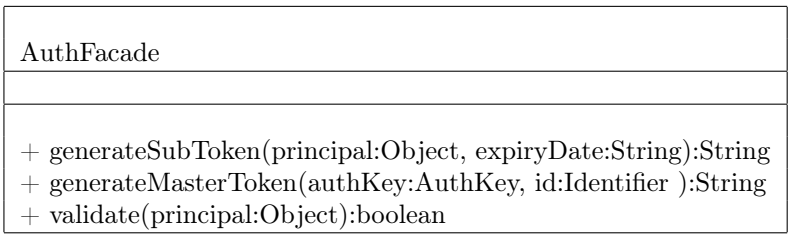
ExceptionsHandler

Neue Methoden hinzugefügt, welche Exceptions bearbeiten.

2.3 Auth

AuthFacade

Hinzugefügt, um die Schnittstelle zur Authentifizierung und Tokengenerierung fest und einheitlich zu machen.



Beschreibung:
Interface zur Definition der Fassade für die Authentifizierung.

Attribute:

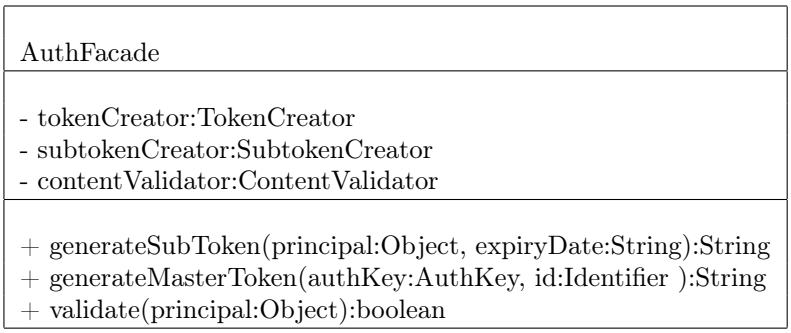
- keine

Methoden:

- generateSubToken(principal:Object, expiryDate:String):String
  Deklaration einer Methode zur Generierung von Subtokens (Tokens mit einem expiryDate)
- generateMasterToken(authKey:AuthKey, id:Identifier):String
  Deklaration einer Methode zur Generierung von MasterTokens (Tokens ohne expiryDate)
- validate(principal:Object):boolean
  Deklaration einer Methode zur Validierung des AuthKeys.

AuthFacadeImpl

Hinzugefügt als Implementierung der AuthFacade.



Beschreibung:
Implementierung der AuthFacade.

Attribute:

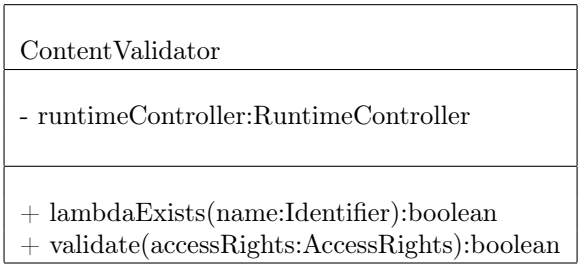
- tokenCreator:TokenCreator
  Verbindung zum TokenCreator zur Generierung von Tokens
- subtokenCreator:SubtokenCreator
  Verbindung zum SubtokenCreator zur Generierung von Subtokens
- contentValidator:ContentValidator
  Verbindung zum ContentValidator zur Validierung von Tokens

Methoden:

- generateSubToken(principal:Object, expiryDate:String):String
  Generiert einen Subtoken
- generateMasterToken(authKey:AuthKey, id:Identifier):String
  Generiert einen Mastertoken
- validate(principal:Object):boolean
  Validiert einen Token

ContentValidator

Hinzugefügt, um Logik aus der AuthFacadeImpl in eine seperate, austauschbare Klasse auszulagern und um eine alleinige Klasse für die Kommunikation mit der Runtime zu schaffen.



Beschreibung:
Klasse zur Validierung von Tokens.

Attribute:

- runtimeController:RuntimeController
  Verbindung zum RuntimeController zum Vergleich von AuthKeys.

Methoden:

- lambdaExists(name:Identifier):boolean
  Ruft RuntimeController auf, um die Existenz von einem Lambda mit Identifier zu prüfen.
- validate(accessRights:AccessRights):boolean
  Ruft RuntimeController auf, um die Existenz von einem Lambda mit Identifier zu prüfen. Prüft, ob der AuthKey aus dem Token mit dem AuthKey aus der Runtime übereinstimmt.

SubtokenCreator

Hinzugefügt, um den speziellen Vorgang zum parsen von Daten für den expiryDate auszulagern.

Beschreibung:
Klasse zur Erstellung von Subtokens (Tokens mit einem Ablaufdatum)

Attribute:

- tokenCreator:TokenCreator
  Verbindung zum TokenCreator zur Erstellung von Tokens.

Methoden:

- generateSubToken(accessRights:AccessRights, expiryDate:String):String
  Generiert einen Token mit einem Ablaufdatum.

JwtAuthTokenFilter

keine Änderungen.

AccessRights

keine Änderungen.

JwtAuthEntryPoint

keine Änderungen.

JwtAuthProvider

keine Änderungen.

JwtAuthSuccessHandler

keine Änderungen.

JwtAuthTokenStringWrapper

keine Änderungen.

TokenCreator

keine Änderungen.

AuthenticationConfig

keine Änderungen.

2.4 Model

2.4.1 Converter

RequestServerConverter

- Alle Methoden wurden umbenannt zur besseren Verständlichkeit.
  - uploadToLambda(uploadRequest: UploadLambdaRequest): Lambda wurde in uploadRequestToLambda(uploadRequest: UploadLambdaRequest): Lambda umbenannt.
  - executeToConfig(executeRequest: ExecuteLambdaRequest): ExecuteConfig wurde in executeRequestToExecuteConfig(executeRequest: ExecuteLambdaRequest): ExecuteConfig umbenannt.
  - lambdaToUpload(lambda: Lambda): UploadLambdaRequest wurde in lambdaToUploadRequest(lambda: Lambda): UploadLambdaRequest umbenannt.
- Die Methode runtimeAttributesRequestToRuntimeAttributes(runtimeAttributesRequest: RuntimeAttributesRequest):RuntimeAttributes wurde hinzugefügt, um von RuntimeAttributesRequest Objekt ein RuntimeAttributes Objekt zu machen.

- Die Methode runtimeAttributesToRuntimeAttributesRequest(runtimeAttributes: RuntimeAttributes): RuntimeAttributesRequest wurde hinzugefügt, um von RuntimeAttributes Objekt ein RuntimeAttributesRequest Objekt zu machen.

2.4.2 Exception.Lambda

LambdaDuplicatedNameException

keine Änderungen.

LambdaNotFoundException

keine Änderungen.

2.4.3 Exception.Messages

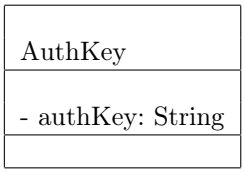
SemanticRequestException

Hinzugefügt um semantische Fehler in JSON Anfrage zu bearbeiten.

2.4.4 Lambda

AuthKey

Hinzugefügt, um den AuthKey des Lambdas aus der Runtime in ein Objekt zu verpacken.



Beschreibung:
Klasse zur Verpackung des AuthKeys der Lambda-Funktion.

Attribute:

- code: String
  Der AuthKey der Funktion als String.

Methoden:

- keine

Code

keine Änderungen.

ExecuteConfig

keine Änderungen.

Identifier

keine Änderungen.

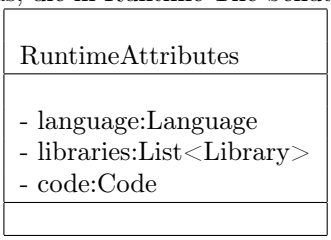
Lambda

- Das Attribut lambdaImage wurde gelöscht.

- Das Attribut runtimeAttributes:RuntimeAttributes wurde hinzugefügt.Die Attribute language:Language, libraries:List<Library>, code:Code wurden in runtimeAttributes:RuntimeAttributes übertragen, um bessere Verschachtelung zu realisieren.

RuntimeAttributes

Hinzugefügt, um Attribute des Lambdas, die in Runtime Tile benutzt wurden, in ein Objekt zu verpacken.



Beschreibung:
Klasse zur Verpackung der Attribute der Lambda-Funktion, welche Runtime Teil benutzt.

Attribute:

- language:Language
  Die Sprache der Funktion .
- libraries:List<Library>
  Die Biblioteken der Funktion .
- code:Code
  Der Code der Funktion .

Methoden:

- keine

Language

keine Änderungen.

Library

keine Änderungen.

Parameter

keine Änderungen.

RunCycles

keine Änderungen.

2.4.5 Messages

ExecuteLambdaRequest

keine Änderungen.

ExecuteLambdaResponse

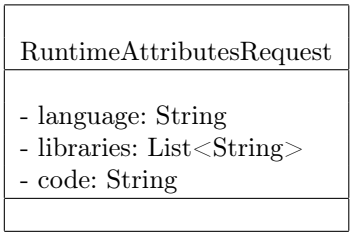
keine Änderungen.

RestErrorInfo

Das Attribut detail:String wurde gelöscht.

RuntimeAttributesRequest

Hinzugefügt, um JSON Datei verständlicher zu machen.



Beschreibung:
Enthält die in Java Attribute geparste RuntimeAttributesRequest Objekt aus JSON Datei, die die Anfrage zum Hochladen enthält.

Attribute:

- language: String
  Die Sprache der Funktion als String.
- libraries: List<String>
  Die Biblioteken der Funktion .
- code: String
  Der Code der Funktion als String.

Methoden:

- keine

UploadLambdaRequest

Die Attribute language:String, libraries:List<String>, code:String wurden in runtimeAttributes:RuntimeAttributesRequest übertragen, weil die Struktur von Lambda Klasse verändert wurde.

UploadLambdaResponse

keine Änderungen.

2.4.6 ServiceLayer: Service

LambdaManagerFacade

- Der Name der Schnittstelle LambdaFacade wurde in LambdaManagerFacade zur besseren Verständlichkeit umbenannt.

- Die Methode authenticate(principal:Object):boolean wurde gelöscht. Dies folgt aus den Änderungen im Auth Teil.

- Der Rückgabetyp der Methode deleteLambda(name: String): String wurde zu void verändert. Der Grund ist: deleteLambda Methode soll nichts zurückgeben, falls alles gut ist. In dem Fall, wenn was schief läuft, wurde passende Exception geworfen.

LambdaManagerFacadeImpl

- Der Name der Klasse LambdaFacadeImpl wurde in LambdaManagerFacadeImpl zur besseren Verständlichkeit umbenannt.

- Manche Methoden wurden zur besseren Verständlichkeit umbenannt. Es wurde noch die Lambda und ExecuteConfig Objekterstellung in RestApiController verlegt, um Anfragen allein dort zu behandeln. Deswegen wurden die Anfragen als Parameter in alle Methoden der LambdaManagerFacadeImpl Klasse durch den Lambda und ExecuteConfig Objekten umgesetzt.

  - upload(config: UploadLambdaRequest): String wurde in addLambda(Lambda lambda): String umbenannt.

  - update(name: String, config: UploadLambdaRequest): String wurde in updateLambda(String name, Lambda lambda): String umbenannt.

  - execute(name: String, config: ExecuteLambdaRequest): String wurde in executeLambda(String name, ExecuteConfig executeConfig): String umbenannt.

- Die Attribute path: String, maxTimes: int, timeout: long wurden gelöscht. Es wurde wegen der Änderungen in LambdaRuntime Teil gemacht.

- Das Attribut authHandler: AuthHandler wurde durch authenticatior: AuthFacade umgesetzt. Es wurde wegen der Änderungen in Auth Teil gemacht.

- Die Methode authenticate(principal:Object):boolean wurde gelöscht.

- Der Rückgabetyp der Methode deleteLambda(name: String): String wurde zu void verändert.

2.4.7 ServiceLayer: Lambdaruntime

Allgemeine Änderungen: Die Klassen wurden zur besseren Übersichtlichkeit und Struktur auf die Sub-Pakete images, execution und communication verteilt.

RuntimeController

- Die Erzeugung der Images wurde in eine die Klasse ImageManager verschoben (mit den zugehörigen Methoden). Damit ist RuntimeController nun eine reine Fassade.
- Die Rückgabe- und Parametertypen der Methoden buildImage(), rebuildImage() und deleteImage() wurden angepasst.
- Neue Methoden:
  - lambdaExists(id: Identifier): boolean
    Methode leitet die Anfrage and den ImageManager weiter
  - getAuthKey(id: Identifier): AuthKey
    Methode leitet die Anfrage an den ImageManager weiter
  - getLambda(id: Identifier): Lambda
    Methode leitet die Anfrage an den ImageManager weiter

InstanceManager

Die Klasse wurde ins Paket execution verschoben, sonst keine Änderungen.

LambdaInstance

Die Klasse wurde ins Paket execution verschoben, sonst keine Änderungen.

AbstractLambdaFactory

Die Klasse wurde ins Paket images verschoben, sonst keine Änderungen.

LambdaImage

- Die Klasse wurde ins Paket images verschoben.
- Neues Attribut: authKey: AuthKey

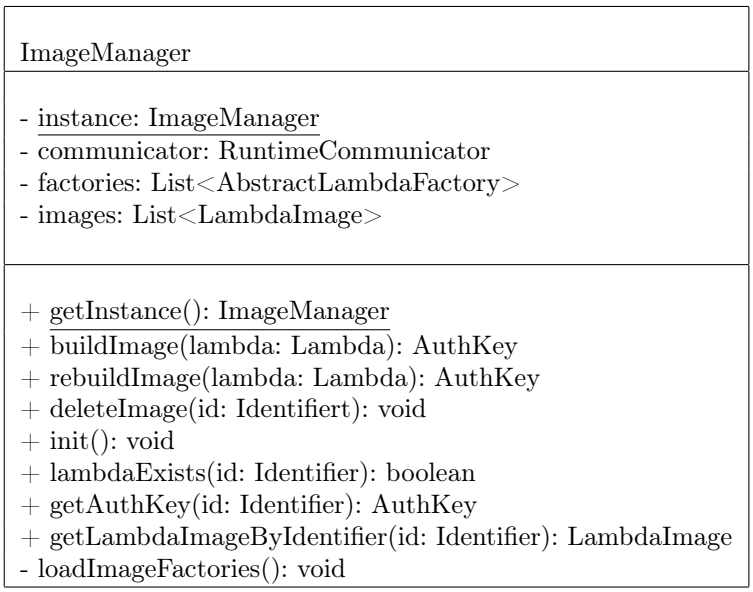
Python3LambdaFactory und Python3LambdaImage

Die Klassen wurden ins Paket images verschoben, sonst keine Änderungen.

OAuthHandler

Die Klasse wurde entfernt. Ihre Funktionalität wurde in die Klasse ImageManager integriert.

ImageManager



Beschreibung:
Eine neue Klasse zur Verwaltung von Images, die Erzeugung von Images in RuntimeController und die Authentifizierung in OAuthHandler wurden in diese Klasse verschoben.

Attribute:

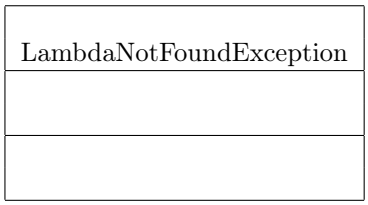
- instance: ImageManager
  Instanz für das Singleton-Pattern
- communicator: RuntimeCommunicator
  Referenz auf den Kommunikator
- factories: List<AbstractLambdaFactory>
  Die Fabriken zur Produktion der Images
- images: List<LambdaImage>
  Liste der vorhandenen Images

Methoden:

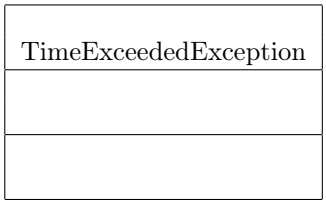
- getInstance(): ImageManager
  gibt die einzige Instanz von ImageManager zurück
- buildImage(lambda: Lambda): AuthKey
  erzeugt ein Image
- rebuildImage(lambda: Lambda): AuthKey
  baut ein Image neu
- deleteImage(id: Identifier): void
  löscht ein Image
- init(): void
  initialisiert den ImageManager

- lambdaExists(id: Identifier): boolean
  gibt zurück, ob das Image vorhanden ist
- getAuthKey(id: Identifier): AuthKey
  gibt den zu Identifier gehörigen AuthKey zurück
- getLambdaImageByIdentifier(id: Identifier): LambdaImage
  gibt das zum Identifier gehörige LambdaImage zurück
- loadImageFactories(): void
  lädt die Fabriken zur Erzeugung der LambdaImages

LambdaNotFoundException



TimeExceededException



Beschreibung: Exception-Klasse für den Fall, das auf nicht vorhandene Lambdas zugegriffen wird.

CommandType

Die Klasse wurde ins Paket communication verschoben, sonst keine Änderungen.

RuntimeCommand

Die Klasse wurde ins Paket communication verschoben, sonst keine Änderungen.

RuntimeConnectException

Die Klasse wurde ins Paket communication verschoben, sonst keine Änderungen.

RuntimeCommunicator

Die Klasse wurde ins Paket communication verschoben, sonst keine Änderungen.

2.5 Application

keine Änderungen.

Kapitel 3

Testfälle

3.1 Auth

Die Authentifizierung besteht aus zwei Teilen. Das Kapitel " Spring Security Tests" beschreibt durch Tests das Zusammenspiel mit Spring Security, welches vor dem Eintritt in die LambdaManagerFassade passiert. Hier wird das übergebene JSON Webtoken auf Syntax und zeitliche Gültigkeit geprüft.
Der zweite Teil hinter der Fassade 'Validation Tests" geschieht mit Zusammenspiel der Runtime und der Fassadenklasse. Diese prüfen die Werte aus dem Token (z.B. AuthKey) mit den zugehörigen Images und erlauben Zutritt bei korrekten Werten.
"Token Generation Tests" widmet sich allein der Generierung von Tokens.

3.1.1 Spring Security Tests

Die " Spring Security Tests" werden mithilfe von JUnit in einem separaten Projekt durchgeführt, nachdem der Server hochgefahren ist. Grund dafür ist die komplexe Spring Testing Umgebung, die lange, intensive Einarbeitung benötigt, welche nicht gegeben ist.

3.1.2 Validation Tests

Die "Validation Tests" werden mithilfe von JUnit durchgeführt.

Test: validateMasterToken

"validateMasterToken" testet Validierung einen gültigen Master-Token.

- Vorbedingung
  Ein gültiger Master-JWT wird als ein Objekt AccessRights von dem aufrufenden Modul übergeben. Die Lambda mit dem entsprechenden AccessKey existiert.
- Beschreibung
  Die Methode validate wird mit dem JWT als Parameter aufgerufen, und es wird geprüft, ob sie "trueßurückgibt.
- Nachbedingung
  Der Master-JWT wird validiert und "true"wird zurückgegeben. Der JWT bleibt unverändert.

Test: validateSubtoken

"validateSubtoken" testet Validierung einen gültigen Subtoken.

- Vorbedingung
  Ein nicht abgelaufener Sub-JWT wird als ein Objekt AccessRights von dem aufrufenden Modul übergeben. Die Lambda mit dem entsprechenden AccessKey existiert.
- Beschreibung
  Die Methode validate wird mit dem JWT als Parameter aufgerufen, und es wird geprüft, ob sie "trueßurückgibt.
- Nachbedingung
  Der Sub-JWT wird validiert und "true"wird zurückgegeben. Der JWT bleibt unverändert.

Test: **validateInvalidSubtoken**

"validateInvalidSubtoken" testet Validierung einen ungültigen Subtoken.

- Vorbedingung
  Ein nicht abgelaufener Sub-JWT wird als ein Objekt AccessRights von dem aufrufenden Modul übergeben. Die Lambda mit dem entsprechenden AccessKey existiert nicht.
- Beschreibung
  Die Methode validate wird mit dem JWT als Parameter aufgerufen, und es wird geprüft, ob sie eine Exception zurückgibt.
- Nachbedingung
  Eine Exception wird zurückgegeben.

Test: **validateSubtokenPastDate**

"validateSubtokenPastDate" testet Validierung einen abgelaufenen Subtoken.

- Vorbedingung
  Ein abgelaufener Sub-JWT wird als ein Objekt AccessRights von dem aufrufenden Modul übergeben. Die Lambda mit dem entsprechenden AccessKey existiert.
- Beschreibung
  Die Methode validate wird mit dem JWT als Parameter aufgerufen, und es wird geprüft, ob sie "falseßurückgibt.
- Nachbedingung
  "False"wird zurückgegeben.

3.1.3 Token Generation Tests

Die "Validation Tests" werden mithilfe von JUnit durchgeführt.

Test: **getMasterToken**

"getMasterToken" testet die korrekte Erstellung von MasterToken mit korrekten Eingaben.

- Vorbedingung
  Gültige Parameter AuthKey und Identifier werden von dem aufrufenden Modul gegeben. Ein gültiger AuthKey ist ein Objekt AuthKey mit Inhalt Runtime ID als String von der erstellten Lambda-Funktion. Ein gültiger Identifier ist ein Objekt Identifier mit Inhalt Name der zu erstellenden Lambda-Funktion als String.

- Beschreibung
  Der resultierende JWT als String wird nach dem Aufruf der AuthFassadeImpl mit Methode generateMasterToken(auth, id) (mit vorher erstellten Objekten) mit einem extern erstellten JWT mit denselben Parametern verglichen.

- Nachbedingung
  Ein gültiger JWT (ohne Auslaufdatum) wurde erstellt und zurückgegeben. Die Parameter vom Anfang bleiben unverändert.

Test: **getSubToken**

"getSubToken" testet die korrekte Erstellung von JWT mit korrekten Eingaben.

- Vorbedingung
  Ein gültiges AccessRights und ein gültiges, in der Zukunft liegendes Datum im Format yyyy-mm-dd hh-mm-ss werden übergeben.

- Beschreibung
  Ein AccessRights Objekt mit gültigen Parametern wird erstellt. Dann wird generateSubToken(accessRights, expiryDate) (expiryDate ist ein gültiger String) aufgerufen. Der zurückgegebene wird dann mit einem extern erstellten JWT verglichen.

- Nachbedingung
  Ein gültiger JWT mit Auslaufdatum wurde erstellt und zurückgegeben. Die Parameter vom Anfang bleiben unverändert.

3.2 RestApiController

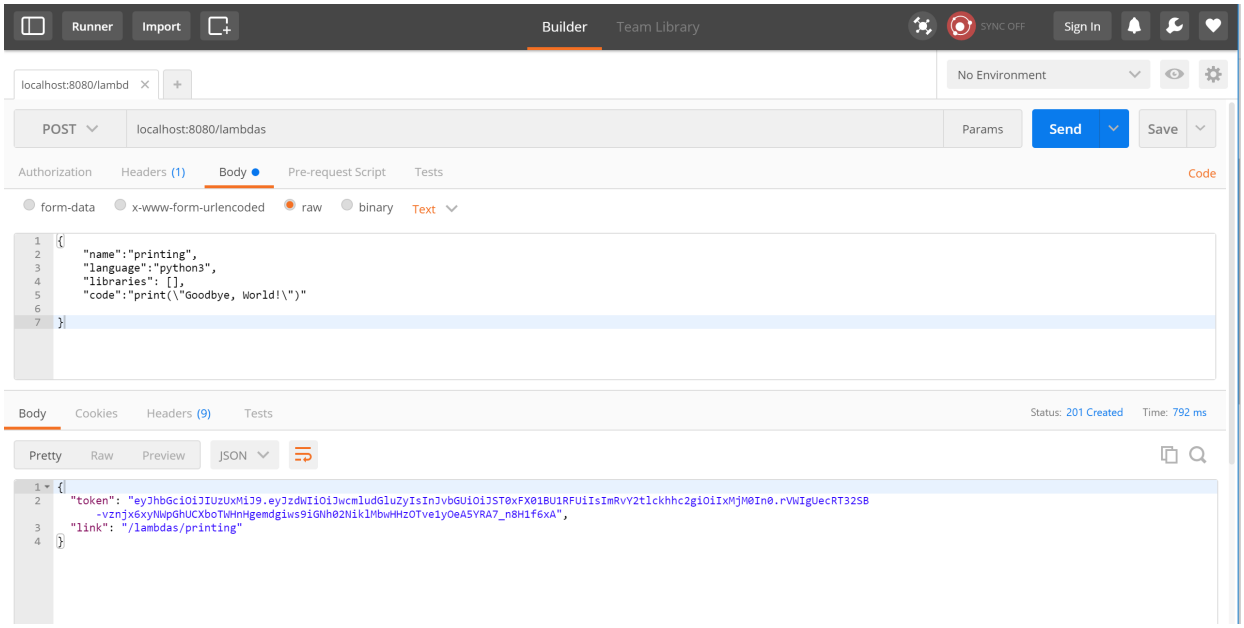
Hier werden Tests dargestellt, die Spezifikation der RestApiController-Klasse prüfen. Alle Tests werden mithilfe von JUnit durchgeführt.

3.2.1 Test: postValidLambda

Hier ist Hochladen von Lambda mit korrekter Anfrage eines Nutzers geprüft.

- Vorbedingung
  Lambda, die ein Nutzer hochladen will, muss nicht im System sein, Anfrage muss korrekt sein.
- Beschreibung
  Zuerst wird Existenz von Lambda im System geprüft. Dannach wird entsprechenden postLambda-Befehl von RestApiController aufgeruft. Endlich werden Nachbedingungen mithilfe von entsprechenden Methoden von RestApiController geprüft.
- Nachbedingung
  Lambda muss schon im System sein und kann aktualisiert, gezeigt, gelöscht, ausgeführt werden und Subtoken kann erstellt werden, damit andere Nutzer, die diesen Subtoken besitzen, entsprechende Lambda benutzen können.

Hier wird gezeigt, was bei der Aufruf der postLambda-Methode zurückgegeben wird.



3.2.2 Test: postExistedLambda

Hier ist Hochladen von Lambda geprüft, die schon im System existiert.

- Vorbedingung
  Lambda muss schon im System sein.
- Beschreibung Zuerst wird Existenz von Lambda im System geprüft. Dannach wird entsprechenden postLambda-Befehl von RestApiController aufgeruft. Endlich werden Nachbedingungen mithilfe von entsprechenden Methoden von RestApiController geprüft.

- Nachbedingung Lambda, die Nutzer hochladen will, muss nicht im System sein. RestApiController muss Antwort mit Status "BAD REQUEST" schicken.

3.2.3 Test: postInvalidLambda

Hier ist Hochladen von Lambda geprüft, wenn Nutzer unkorrekte Anfrage schickt (z.B. unkorrekte Name von Lambda).

- Vorbedingung
  Lambda, die ein Nutzer hochladen will, muss nicht im System sein, Anfrage muss korrekt sein.
- Beschreibung
  Zuerst wird Existenz von Lambda im System geprüft. Dannach wird entsprechenden postLambda-Befehl von RestApiController aufgeruft. Endlich werden Nachbedingungen mithilfe von entsprechenden Methoden von RestApiController geprüft.
- Nachbedingung
  Lambda muss nicht im System sein, d.h. Lambda kann nicht aktualisiert, gezeigt, gelöscht, ausgeführt werden und Subtoken kann nicht erstellt werden, damit andere Nutzer, die diesen Subtoken besitzen, entsprechende Lambda benutzen können.

3.2.4 Test: postLambdaValidExecute

Hier ist Ausführung von Lambda geprüft, wenn Nutzer korrekte Anfrage schickt.

- Vorbedingung
  Lambda, die ein Nutzer ausführen will, muss im System sein, Anfrage muss korrekt sein.
- Beschreibung
  Zuerst wird Existenz von Lambda im System geprüft. Dannach wird entsprechenden postLambda-Befehl von RestApiController aufgeruft. Endlich werden Nachbedingungen mithilfe von entsprechenden Methoden von RestApiController geprüft.
- Nachbedingung
  Lambda muss im System bleiben, d.h. Lambda kann aktualisiert, gezeigt, gelöscht, noch mal ausgeführt werden und Subtoken kann erstellt werden, damit andere Nutzer, die diesen Subtoken besitzen, entsprechende Lambda benutzen können.

3.2.5 Test: postLambdaInValidExecute

Hier ist Ausführung von Lambda geprüft, wenn Nutzer unkorrekte Anfrage schickt (z.B. Eingabe, dessen Typ keine Signatur von Lambda passt).

- Vorbedingung
  Lambda, die ein Nutzer ausführen will, muss im System sein, Anfrage muss unkorrekt sein.
- Beschreibung
  Zuerst wird Existenz von Lambda im System geprüft. Dannach wird entsprechenden postLambda-Befehl von RestApiController aufgeruft. Endlich werden Nachbedingungen mithilfe von entsprechenden Methoden von RestApiController geprüft.
- Nachbedingung
  Lambda muss im System bleiben, d.h. Lambda kann aktualisiert, gezeigt, gelöscht, noch mal ausgeführt werden und Subtoken kann erstellt werden, damit andere Nutzer, die diesen Subtoken besitzen, entsprechende Lambda benutzen können.

3.3 LambdaManagerFacadeImpl

Hier werden Tests dargestellt, die Spezifikation der LambdaManagerFacadeImpl-Klasse prüfen. Alle Tests werden mithilfe von JUnit durchgeführt.

3.3.1 Test: addValidLambda

Hier ist Ergänzung von Lambda ins System geprüft, wenn Lambda-Obkekt korrekte Attribute besitzt (z.B. korrekter Name, Sprache).

- Vorbedingung
  Lambda muss nicht im System sein, Lambda-Objekt muss korrekte Attributen besitzen.
- Beschreibung
  Zuerst wird Existenz von Lambda im System geprüft. Dannach wird addLambda-Befehl von LambdaManagerFacadeImpl aufgeruft. Endlich werden Nachbedingungen mithilfe von entsprechenden Methoden von LambdaManagerFacadeImpl geprüft.
- Nachbedingung
  Lambda muss im System sein, d.h. Lambda kann aktualisiert, gezeigt, gelöscht, ausgeführt werden.

3.3.2 Test: addExistedLambda

Hier ist Ergänzung von Lambda ins System geprüft, wenn Lambda schon im System existiert.

- Vorbedingung
  Lambda muss im System sein.
- Beschreibung
  Zuerst wird Existenz von Lambda im System geprüft. Dannach wird addLambda-Befehl von LambdaManagerFacadeImpl aufgeruft. Endlich wartet man auf entspreche Ausnahme.
- Nachbedingung
  Es muss eine Ausnahme geworfen werden.

3.3.3 Test: executeValidLambda

Hier ist Aufführung von Lambda ins System geprüft, wenn Lambda im System existiert.

- Vorbedingung
  Lambda muss im System sein.
- Beschreibung
  Zuerst wird Existenz von Lambda im System geprüft. Dannach wird executeLambda-Befehl von LambdaManagerFacadeImpl aufgeruft. Endlich werden Nachbedingungen mithilfe von entsprechenden Methoden von LambdaManagerFacadeImpl geprüft.
- Nachbedingung
  Lambda muss im System bleiben, d.h. Lambda kann aktualisiert, gezeigt, gelöscht, noch mal ausgeführt werden.

3.3.4 Test: executeInvalidLambda

Hier ist Aufführung von Lambda ins System geprüft, wenn Lambda im System nicht existiert.

- Vorbedingung
  Lambda muss nicht im System sein.
- Beschreibung
  Zuerst wird Existenz von Lambda im System geprüft. Dannach wird executeLambda-Befehl von LambdaManagerFacadeImpl aufgeruft. Endlich wartet man auf entspreche Ausnahme.
- Nachbedingung
  Es muss eine Ausnahme geworfen werden.

3.4 LambdaRuntime

3.4.1 Test: testUpload

"testUpload"testet das Hochladen von einer Test-Lambda und das Bauen von ihrem Image.

- Vorbedingung
  Die Lambda existiert noch nicht im System.
- Beschreibung
  Erstmal wird geprüft, ob die Lambda im System existiert, und falls sie existiert, gibt der Test "falseßurück. Dann wird ein Image für diese Lambda gebaut, und wird geprüft, ob die Lambda jetzt existiert. Anschließend wird geprüft, ob die Lambda aufrufbar ist, indem sie aufgerufen wird.
- Nachbedingung
  Die Lambda existiert im System und ist aufrufbar.

3.4.2 Test: testUpdate

"testUpdate"testet das Aktualisieren von einer existierenden Test-Lambda und das Umbauen von ihrem Image.

- Vorbedingung
  Die Lambda existiert bereits im System.
- Beschreibung
  Erstmal wird ein Image für die Test-Lambda gebaut. Dann wird das Image für diese Lambda umgebaut (rebuildImage), und wird geprüft, ob die Lambda jetzt immer noch existiert. Anschließend wird geprüft, ob die Lambda aufrufbar ist, indem sie aufgerufen wird.
- Nachbedingung
  Die Lambda existiert immer noch im System und ist aufrufbar.

3.4.3 Test: testDelete

"testDelete"testet das Löschen von einer existierenden Test-Lambda.

- Vorbedingung
  Die Lambda existiert bereits im System.
- Beschreibung
  Erstmal wird ein Image für die Test-Lambda gebaut. Dann wird das Image für diese Lambda gelöscht, und wird geprüft, ob die Lambda jetzt nicht mehr existiert. Anschließend wird geprüft, ob die Lambda aufrufbar ist, indem sie aufgerufen wird (eine Exception ist an der Stelle erwartet).
- Nachbedingung
  Die Lambda existiert nicht mehr im System und ist nicht mehr aufrufbar.

3.4.4 Test: testRun

"testRun"testet das Ausführen von einer existierenden Test-Lambda.

- Vorbedingung
  Die Lambda existiert bereits im System.

- Beschreibung
  Erstmal wird ein Image für die Test-Lambda gebaut. Dann wird getestet, ob das Image für diese Lambda aufgerufen werden kann. Anschließend wird geprüft, ob die Lambda immer noch existiert und aufrufbar ist, indem sie noch mal aufgerufen wird.

- Nachbedingung
  Die Lambda existiert immer noch im System und kann noch mal aufgerufen werden.

3.4.5 Test: testRunNotExisting

"testRunNotExisting"testet das Ausführen von einer nicht-existierenden Test-Lambda.

- Vorbedingung
  Die Lambda existiert nicht im System.

- Beschreibung
  Ein Image wird für die Lambda NICHT gebaut (sie existiert also nicht im System), und es wird getestet, ob das Image für diese Lambda aufgerufen werden kann (eine Exception ist erwartet). Anschließend wird geprüft, ob die Lambda immer noch nicht existiert.

- Nachbedingung
  Die Lambda existiert immer noch nicht im System.

3.4.6 Test: testStatelessness

"testStatelessness"prüft, ob eine Test-Lambda, die mehrmals ausgeführt wird, immer das gleiche Ergebnis zurückliefert.

- Vorbedingung
  Die Lambda existiert im System.

- Beschreibung
  Eine Test-Lambda wird angelegt, die versucht es, eine Datei anzulegen, und wenn die Datei noch nicht existiert, gibt die Lambda "createdßurück, sonst älready exists". In dem Testfall wird es in einer for-Schleife 20-Mal geprüft, ob die Test-Lambda jedes Mal "createdßurückgibt.

- Nachbedingung
  Die Lambda gibt in jeder Ausführung das gleiche Ergebnis zurück.

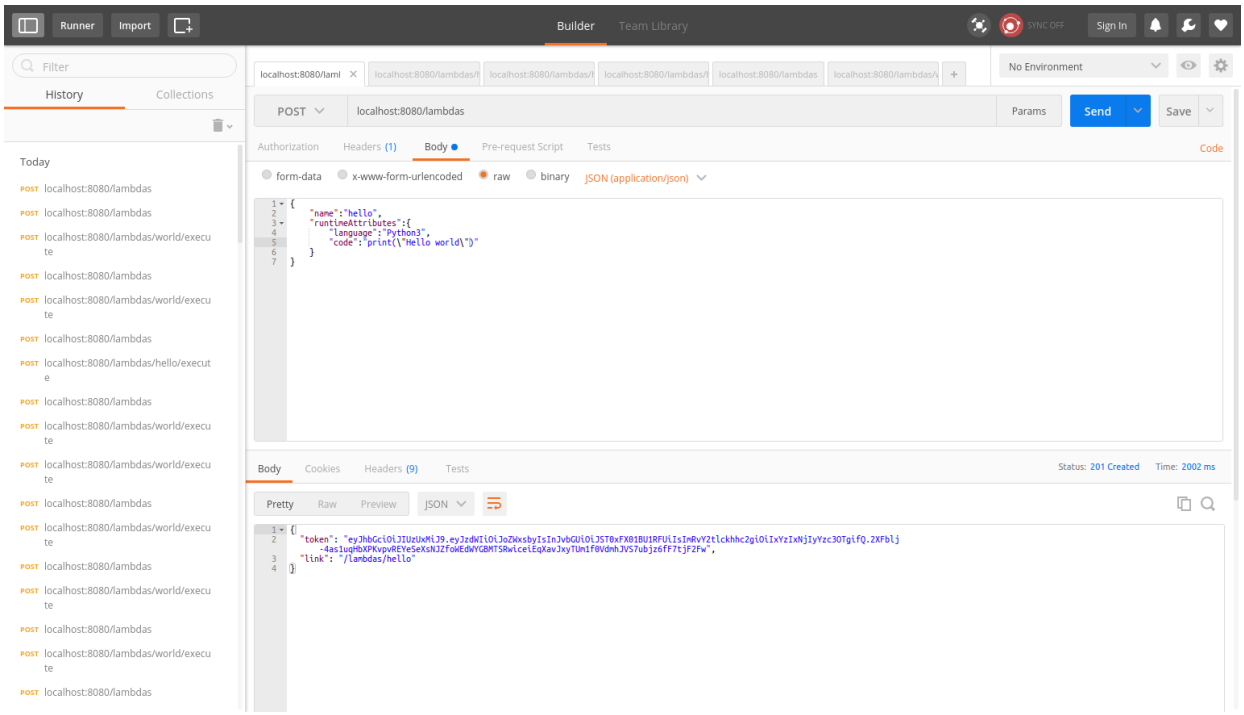
Kapitel 4

Bedienungsanleitung des Programms

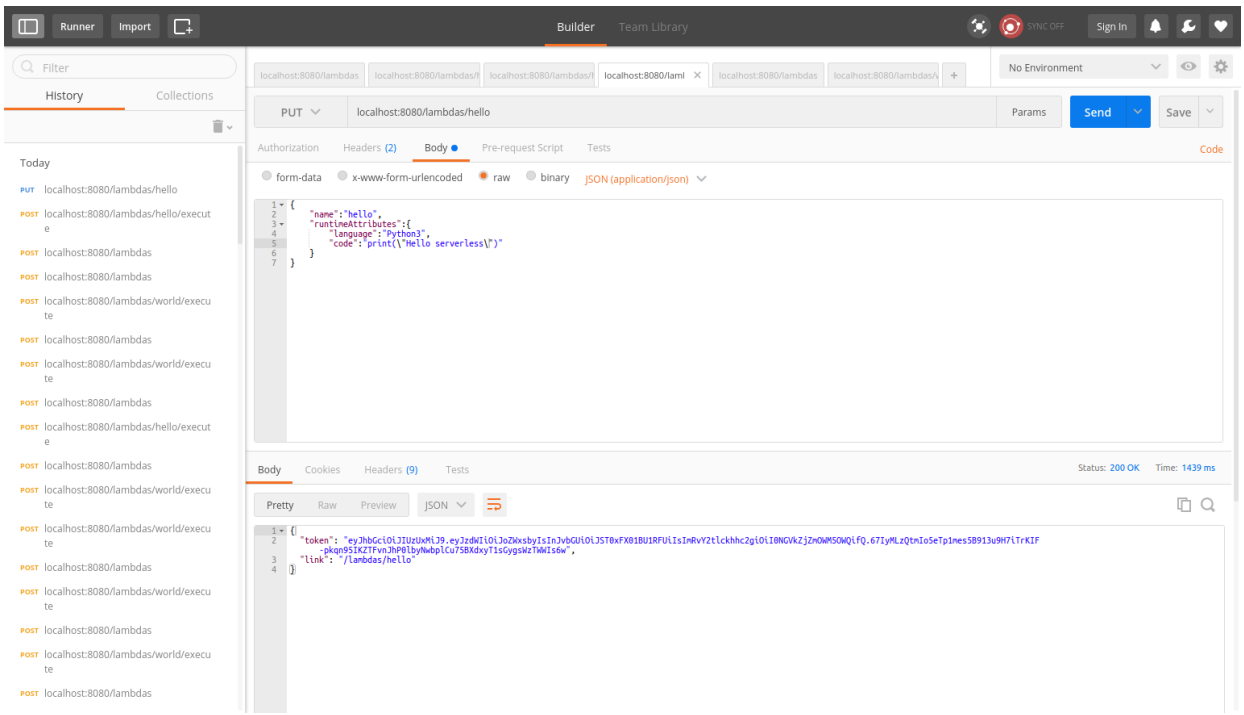
4.1 Programm starten

Das Programm muss bereits auf einem Server laufen, um Befehle empfangen und ausführen zu können. Die REST-konforme Befehle werden direkt auf diesen Server in curl-Format geschickt, mit einer JSON-Datei als body. Als Antwort wird ebenfalls eine JSON-Datei erhalten.

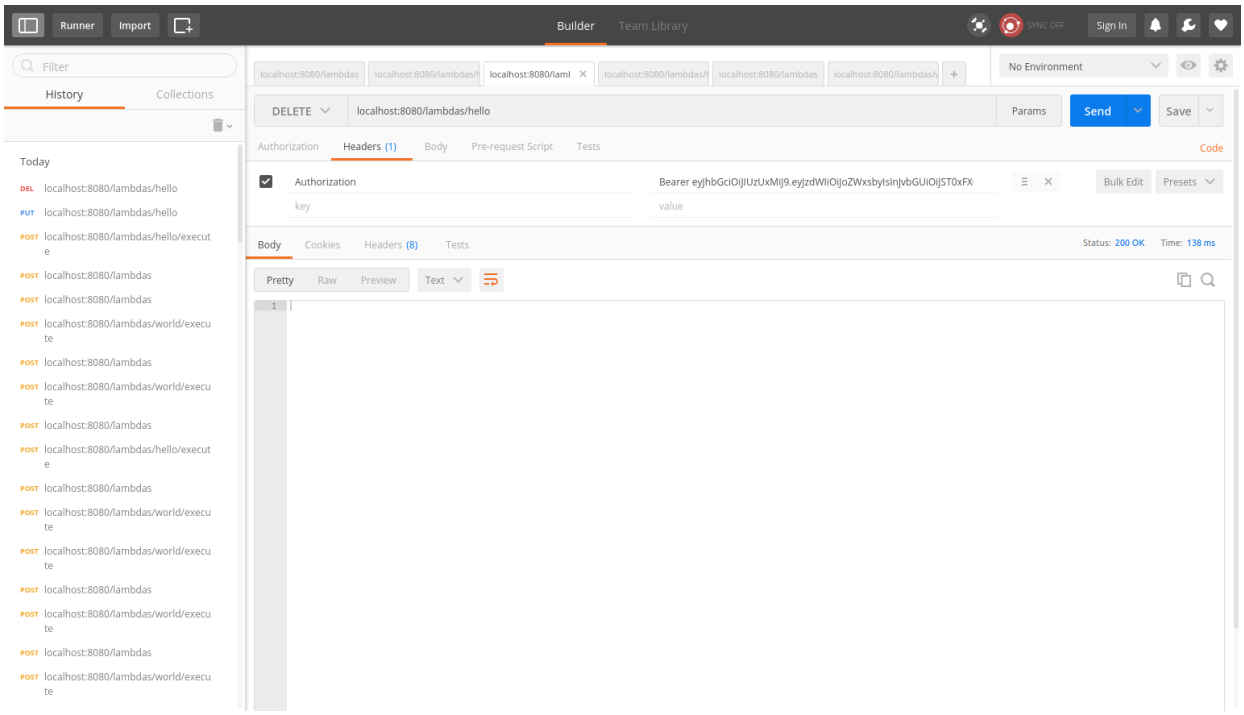
4.2 Lambda hochladen



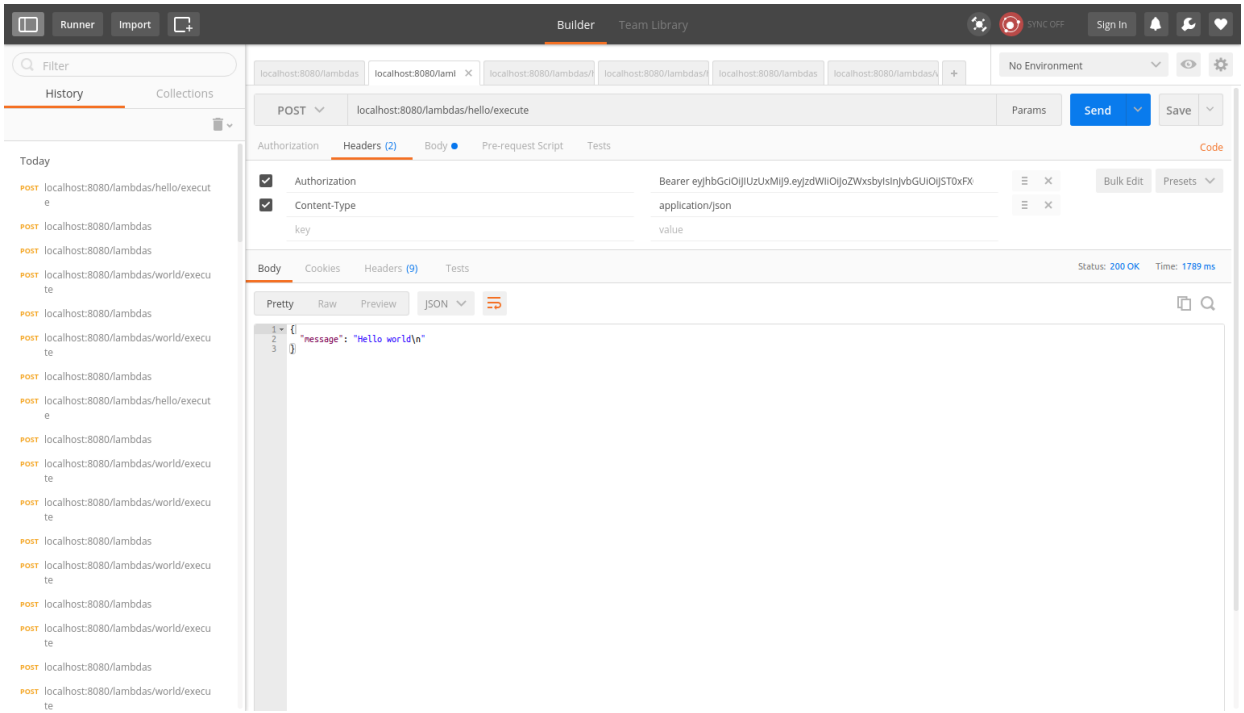
4.3 Lambda aktualisieren



4.4 Lambda löschen



4.5 Lambda ausführen



Kapitel 5

Vergleich der implementierten Funktionen mit der Definition aus dem Pflichtenheft

5.1 Erfüllung der Musskriterien

Musskriterien	funktionaler Art
√	Der Lambda-Entwickler soll die Lambda-Funktion mithilfe einer einfachen REST-API hochladen können.
√	Der Lambda-Entwickler soll die Lambda-Funktion mithilfe einer einfachen REST-API ausführen können.
√	Der Lambda-Entwickler soll die Lambda-Funktion mithilfe einer einfachen REST-API löschen können.
√	Der Lambda-Entwickler soll die Lambda-Funktion mithilfe einer einfachen REST-API ändern können.
√	Der Lambda-Entwickler soll die Lambda-Funktion mithilfe einer einfachen REST-API benennen können.
√	Die Konfigurationen (z.B. Signatur, Funktionsparameter, Ausführungsanzahl) der Lambda-Funktion sollen spezifiziert werden können.
√	Der Lambda-Entwickler sollen mithilfe eines Tokens authentifiziert werden können.
√	Lambda-Verwender sollen mithilfe eines Tokens, der vom Lambda-Entwickler ausgegeben wird authentifiziert werden können und mit diesem Token Lambda-Funktionen ausführen.
-	Laufende Instanzen gelöschter Lambda-Funktionen sollen gestoppt werden.
√	Lambda-Funktionen sollen zustandslos sein.
√	Lambda-Funktionen sollen gekapselt werden.
√	Lambda-Funktionen dürfen nur eine vom Server spezifizierte Zeit lang laufen.

Musskriterien nichtfunktionaler Art

√ Das System soll in Rechner- und Speicherressourcen skalierbar sein.

√ Das System soll um Sprachen für Lambda-Funktionen erweiterbar sein. //Dynamisches Laden von Sprachen wurde implementiert.

5.2 Erfüllung der Wunschkriterien

Wunschkriterien funktionaler Art + erweiterte Funktionen

- Lambda-Entwickler sollen eine Höchst-Laufzeit spezifizieren können, die nicht die maximale Laufzeit überschreitet.
- Zu jedem Token wird ein Zähler erstellt, der für statistische Zwecke ausgelesen werden kann.
- Der Lambda-Entwickler können ein Repository angeben und von dort die Lambda-Funktion auf den Server laden.
- Versionsverwaltung für hochgeladene Lambda-Funktionen.
- der Server kann Firewall-Einstellungen für Lambda-Funktionen definieren.
- Der Lambda-Entwickler können Firewall-Einstellungen für Lambda-Funktionen definieren, sofern sie nicht den Server-Einstellungen widersprechen.
- Sofern die Ausführungszeit einer die Lambda-Funktion ein Limit überschreitet, wird die HTTP-Verbindung abgebrochen und bei Beendigung der Lambda-Funktion wird das Ergebnis bereitgestellt.

? Prepaid-Bezahlsystem für Tokens. //ein Bitcoin-System wird implementiert.

- spätere Bereitstellung des Ergebnisses für Lambda-Funktion.
- Der Serverbetreiber kann Statistiken einsehen.

Wunschkriterien nichtfunktionaler Art

√ Die Ausführung einer stark ressourcenaufwändigen die Lambda-Funktion beeinträchtigt nicht die Ausführung anderer Lambda-Funktionen auf dem Server.

5.3 Erfüllung der Abgrenzungskriterien

- Entwicklung einer graphischen Nutzeroberfläche.
- Entwicklung einer Nutzerverwaltung mit Datenbank. //nicht notwendig, da alle Nutzerinformationen in den Tokens gespeichert werden.