Fachbereich Informatik und Medien

Technische Dokumentation
zum Semesterprojekt

Thema: „Erstellung einer Cloud-basierte Anwendung „Twitter-Nachrichtenanalyser“ unter Verwendung von CI/CD“

verfasst von:

Assil Bouhachem Matrikelnummer: 20203720

Darya Martyniuk Matrikelnummer: 20141525

Mohamed Ali Dridi Matrikelnummer: 20202242

Mohamed Sabiri Matrikelnummer:

vorgelegt am:

22.01.2018

Betreuer: Prof. Dr. Thomas Preuss, Dipl.-Inform. Lars Gentsch

Inhaltsverzeichnis

Die vorliegende Dokumentation ist im Rahmen der Lehrveranstaltung „Systemintegration“ im Wintersemester 17/18 entstanden. Sie soll dazu dienen, einen Überblick über die Architektur der entwickelten Web-Anwendung und die verwendeten Diensten zu verschaffen.

1. Aufgabenstellung

Ziel des Semesterprojekts war es, die Twitter Nachrichten zu analysieren und die Ergebnisse auf einer Web-Oberfläche zu präsentieren.
Die Analyse sollte mindestens folgende Metriken darstellen:

- häufigstes Hashtag

- häufigster Ursprungsort

Die Ergebnisse sollen in eine NoSQLDatenbank gespeichert werden. Die Webseite muss auf einer EC2Instanz oder Elastic Beanstalk bereitgestellt werden.

Im Projekt sollen zwei Testphasen mit min. einem Test pro Phase implementiert werden. Weitere Anforderung sind die Verwendung einer Versionskontrolle (Abk.: VSC), einen Build-Tool sowie einen CI-Server (Abk.: Continious Integration Server).

Als optionale Anforderung stand die Verpackung und Deployment der Anwendung per Docker.

2. Backend:

2.1. Architektur Backend:

- Ein Producer (in diesem Fall der Twitter-Feed) legt Daten in Amazon Kinesis-stream ab.
- Kinesis-stream puffert automatisch die Daten (in diesem Fall 5 MB oder 5 Minuten, je nachdem, welche Bedingung zuerst erfüllt wird) und liefert die Daten an Amazon DynamoDB.
- Eine Python-Lambda-Funktion wird ausgelöst, wenn eine neue Datei durch Kinesis-Stream in DynamoDB erstellt wird.
- ApiGateway löst eine andere Python-Lambda-Funktion mit einer (POST, GET Methoden) aus, die den Dynamodb scannt und holt einzeln die Hashtags und locations Items, parse sie in json-datei, legt sie in einer Array-Liste und sendet sie an APIGateway

- Eine andere entwickelte lokale Web-oberfläche erhält die Daten vom ApiGateway, und präsentiert die häufigsten Hashtags und Ursprungsorte auf einem Maps und in Diagrammen Form
- Die Webseite wird dann auf einer Ec2-Instanz bereitgestellt.

Abb. 1: Allgemeine Architektur Backend

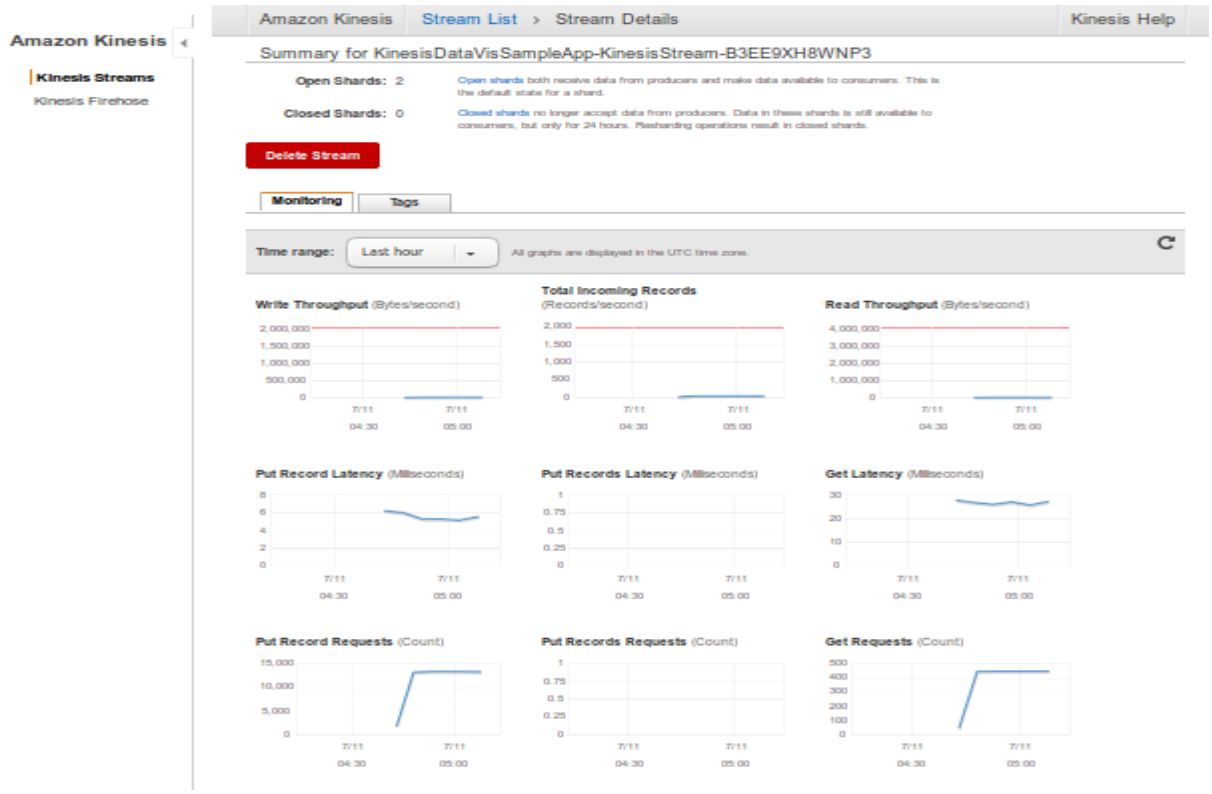
2.2. Vorraussetzung:

Um die Platform zu erstellen, benötigt man:

- Erstellen ein Konto auf Twitter Developer Platform
- Twitter credentials: um Aufruf zum öffentlichen API. Zu diesem Zweck ging man zu apps.twitter.com und erstellt eine neue App und füllt das Formular aus, um die einzigartigen credentials zu erhalten, um Anfragen von Twitter zur Daten zu erledigen, sobald man Twitter - credentials hat, legtet man sie in einer Konfigurationsdatei TwitterCredentials.py.
- Aws-konto(Amazon Web Services) zu erstellen.

2.3. Daten von Twitter-Streaming-Api auf Kinesis-Stream übertragen:

Um Daten von Twitter-Api in Kinesis-Stream zu bekommen, musste man Boto- und TwitterAPI- packages in Python verwenden zum Erstellen und Speichern von Datensätzen in Kinesis Steams.



Abbi.2: Daten-Übertragung von TwitterStreaming nach Kinesis

2.4. Integration von Amazon Kinesis-stream in AWS Lambda:

Durch die Integration von Amazon Kinesis Data Analytics-Anwendungen in AWS Lambda sind zusätzliche Szenarien möglich. Wenn man die Anwendungsausgabe dauerhaft in einem Kinesis-Stream speichert, kann AWS Lambda den Stream abfragen und eine Lambda-Funktion aufrufen. Ihre Lambda-Funktion kann die im Stream eintreffenden Datensätze dann verarbeiten und diese Datensätze beispielsweise in einem Ziel schreiben.

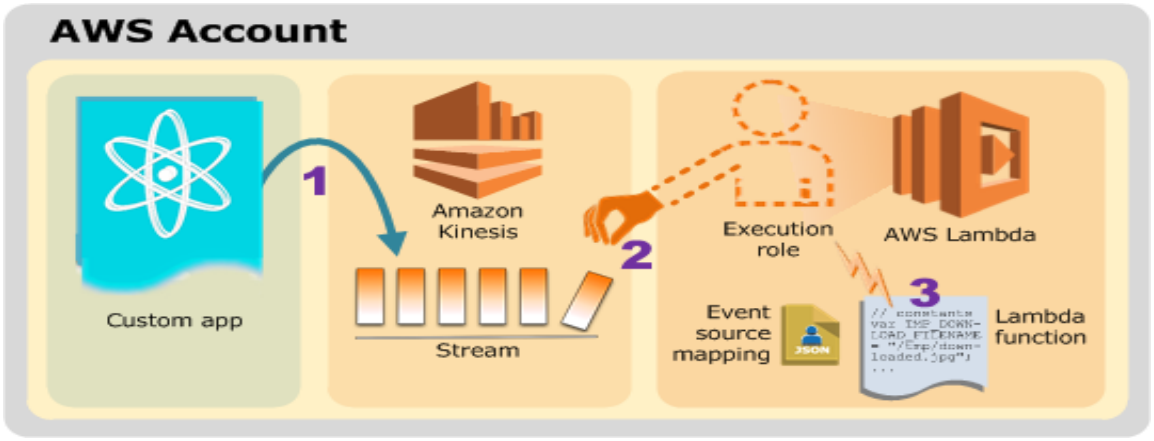


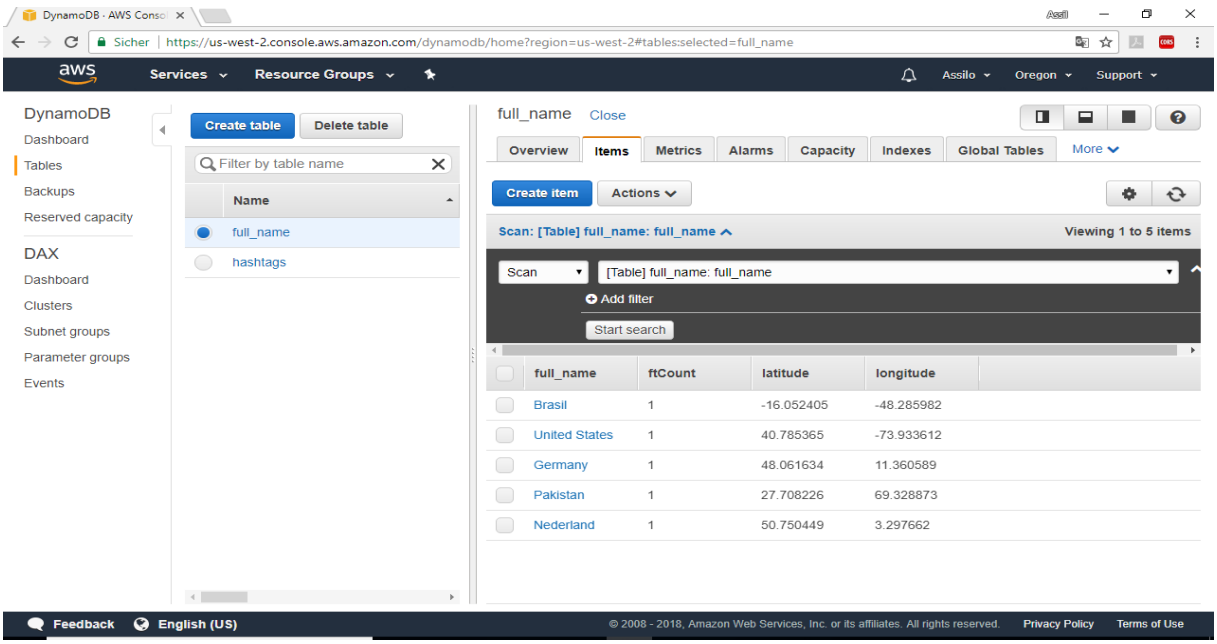
Abb.3: Integration Kinesis mit Lambda

2.5. DynamoDB Streams- und AWS Lambda-Auslöser:

Amazon DynamoDB ist in AWS Lambda integriert, indem man ein *Auslöser* erstellen kann. Hierbei handelt es sich um Codeelemente, die auf Ereignisse in DynamoDB Streams automatisch reagieren. Mit Auslösern kann man Anwendungen erstellen, die auf Datenänderungen in DynamoDB-Tabellen reagieren.

In diesem Zusammenhang speichert unser Lambda-function „ProcessKinesisRecords“ die Hashtags und das Vorkommen "htCount" in einer Tabelle "Hashtags" und speichert noch andere Attributen in der Tabelle „full_name“.

Ein Screenshot aus unserer Dynamodb-Datenbank zeigt die zwei Tabellen, die gefüllt werden.



Abbi4: Hashtags Table in DynamoDb

2.6.Amazon API Gateway:

man kann mit API Gateway eine RESTful-API erstellen, konfigurieren und hosten können, um Anwendungen den Zugriff auf die AWS Cloud zu ermöglichen.

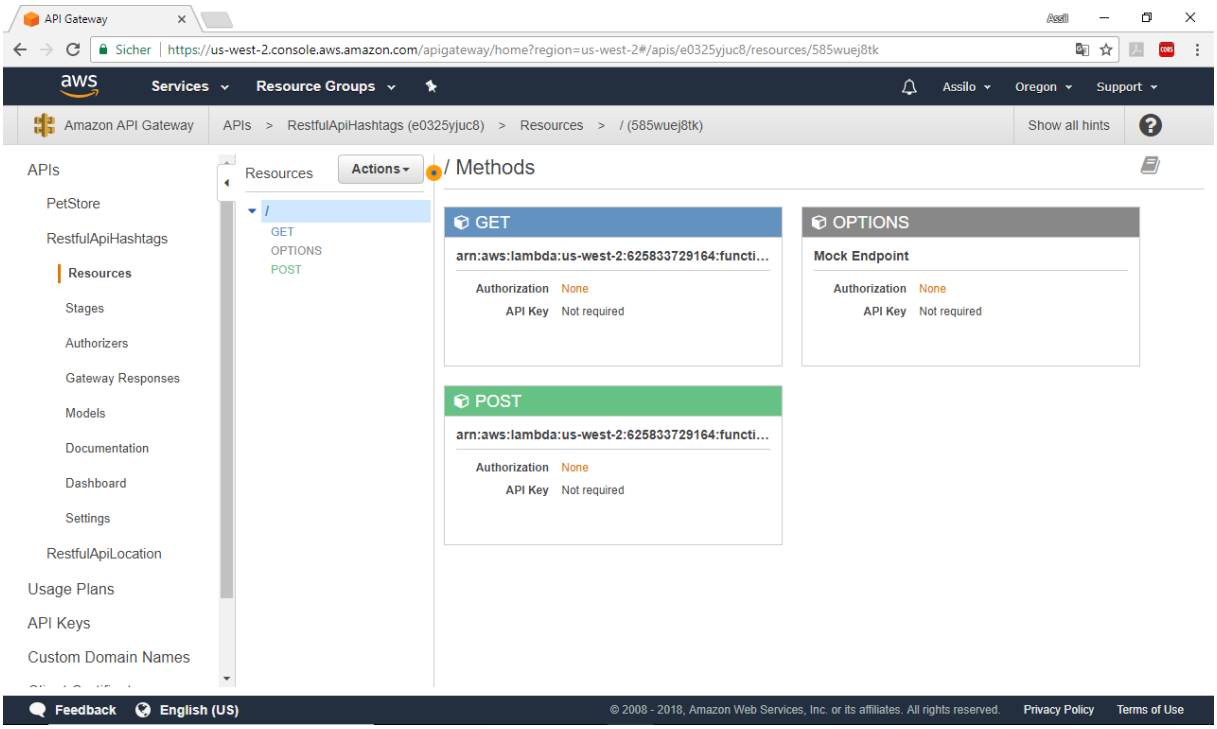


Abb.5: Beispiel Hashtags API

2.7.Technologien Backend:

- **Github:** version control system
- **VisualStudioCode:**
- **Virtualenv:** tool to create isolated Python enviremonts
- **Setup.py:** to setup our Python project after building package
- **Requirements_dev.txt:** to install the libraries that we needs for the project
- **CookieCutter:** Template to buid python package using Github repository. This Template make the structure of directories very sample.

➔**Features**

- Testing setup with unittest and python setup.py test or py.test
- Travis-CI: Ready for Travis Continuous Integration testing
- Tox testing: Setup to easily test for Python 2.6, 2.7, 3.3, 3.4, 3.5
- Sphinx docs: Documentation ready for generation with, for example, ReadTheDocs
- Bumpversion: Pre-configured version bumping with a single command

- Auto-release to PyPI when you push a new tag to master (optional)
- Command line interface using Click (optional)

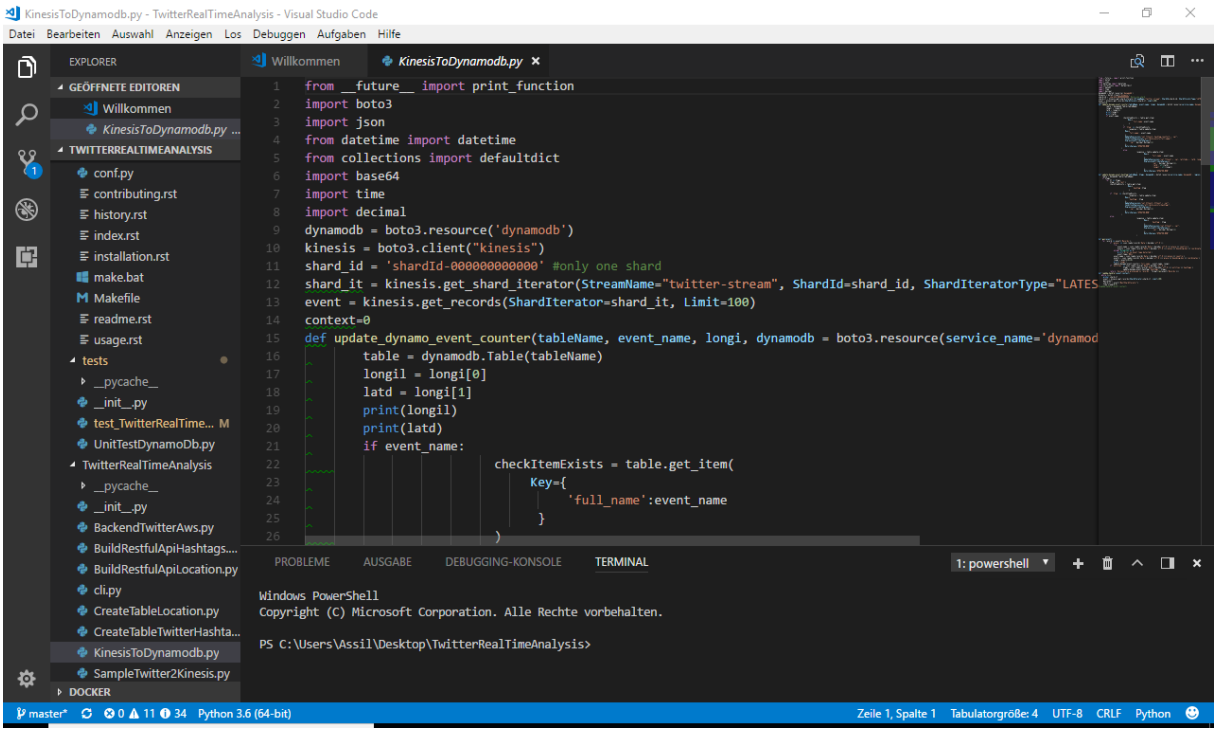


Abb.6: Struktur Backend

3.Front-End

Das Front-End für die Web Applikation wurde in Angular5, im Folgenden Angular genannt, entwickelt. Angular[1] ist ein Framework für die Erstellung von Client-Anwendungen in HTML und JavaScript bzw. TypeScript. Sie setzt auf *Node.js*[2] – eine Laufzeitumgebung zur Ausführung von JavaScript und *NPM*[3] *(Node Package Manager)*, die einen Zugriff auf die Node Module ermöglicht.

Für die Unterstützung der Front-End Entwicklern stellt Angular ein *Angular CLI* Tool zur Verfügung, das Vorlagen und Befehle für alle wiederkehrenden Aufgaben, vom Anlegen eines Projekts bis zum finalen Deployment bereitstellt (wie z.B. das Bauen des Projektes oder die

[1] https://angular.io/
[2] https://nodejs.org/de/
[3] https://www.npmjs.com/

Ausführung von Unit- und End2End Tests). Angular CLI[4] setzt auf *Webpack*[5], einem Modul Loader und Bundler, der dafür verantwortlich ist, die Teile des Projektes zu verpacken, bevor sie an den Client ausgeliefert werden. [1]

Die Tabelle 1 listet Versionsnummern der oben genannten Frameworks und Tools, welche in dem vorliegenden Projekt verwendet wurden.

Framework	**Version**
Angular	5.2.1
Angular CLI	1.5.5
Node.js	8.9.4
NPM	5.6.0
Bootstrap	3.3.7
Webpack	1.8.5

Tabelle 1. Versionsnummern der von Front-End eingezetzten Frameworks

Als Entwicklungsumgebung für das Front-End wurde Visual Studio Code sowie Google Chrome Browser (für das Debuggen) benutzt.

3.1 Architektur des Front-Ends:

Die Abbildung 2 stellt die Filestruktur des Front-Ends der Anwendung dar.

[4] https://cli.angular.io/
[5] https://webpack.js.org/

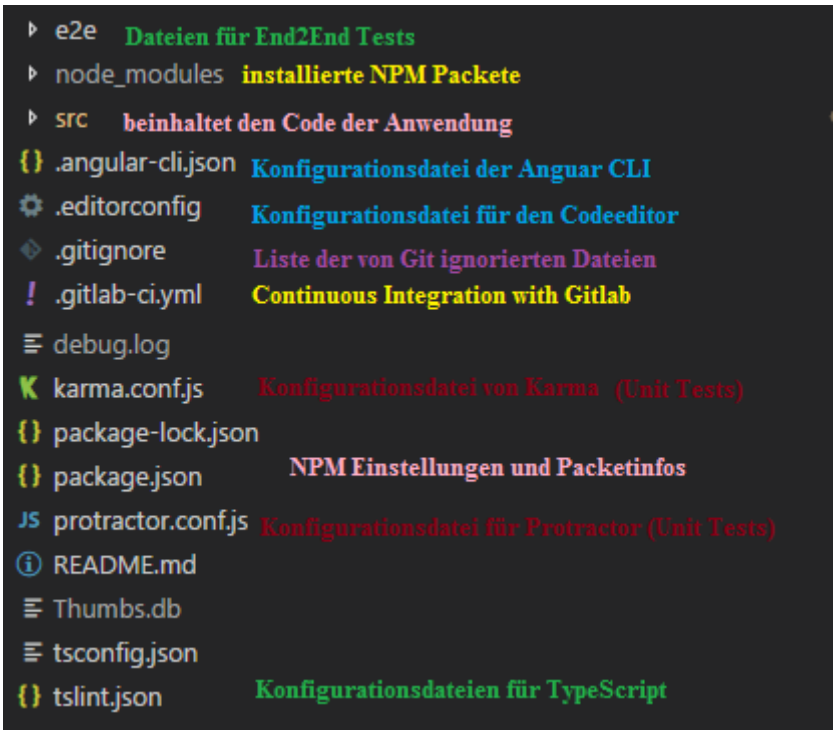


Abbildung 7: die Filestruktur des Front-Ends der Anwendung

Die vorliegende Angular Anwendung setzt sich aus sechs verschiedenen Komponenten zusammen, die jeweils einen kleinen Teil der Anwendung beschreiben. Zu jeder Komponente wird ein Template (eine HTML Datei), ein Stylesheet (eine css Datei) und eine Logik (component.ts Datei) zugeordnet. Die Hauptkomponente der Anwendung ist *AppComponent*, alle weiteren Komponenten werden diesem untergeordnet. Die Navigation zwischen den Komponenten wird mittels Angular Routing implementiert, die erlaubt es, in einer Single Page Applikation mehrere Seiten und so Navigationsstrukturen zu erstellen.

Es gelten folgenden Routing Regeln:

1) beim Laden der Webanwendung wird der User auf dem Template von *HomeComponent* (*http://..../home*) umgeleitet
2) In dem Navigationsmenu kann der User zwischen Tabs „Home“ und „Chart“ auswählen.
3) Ein Auswahl von „Charts“ in dem Navigationsmenu führt zu den Umleitung auf dem Template von *Statistic_TablesComponent* (*http://.../charts*)
4) Auf der „Charts“-Seite stehen dem User zwei Metriken zum Auswahl : eine Statistik über die am häufigsten verwendeten Hashtags (=> die Umleitung auf dem

HashtagComponent mit URL *http://..../charts/hashtag* innerhalb der „Charts“-Seite) oder eine Anzeige der Lokalisation von letzten Tweets (=> die Umleitung auf dem *LocationComponent* mit URL *http://..../charts/location* innerhalb der „Charts“-Seite)

Abb. 4 und Abb. 5 stellen die Architektur und Zusammenhang der Komponenten des Front-Ends dar.

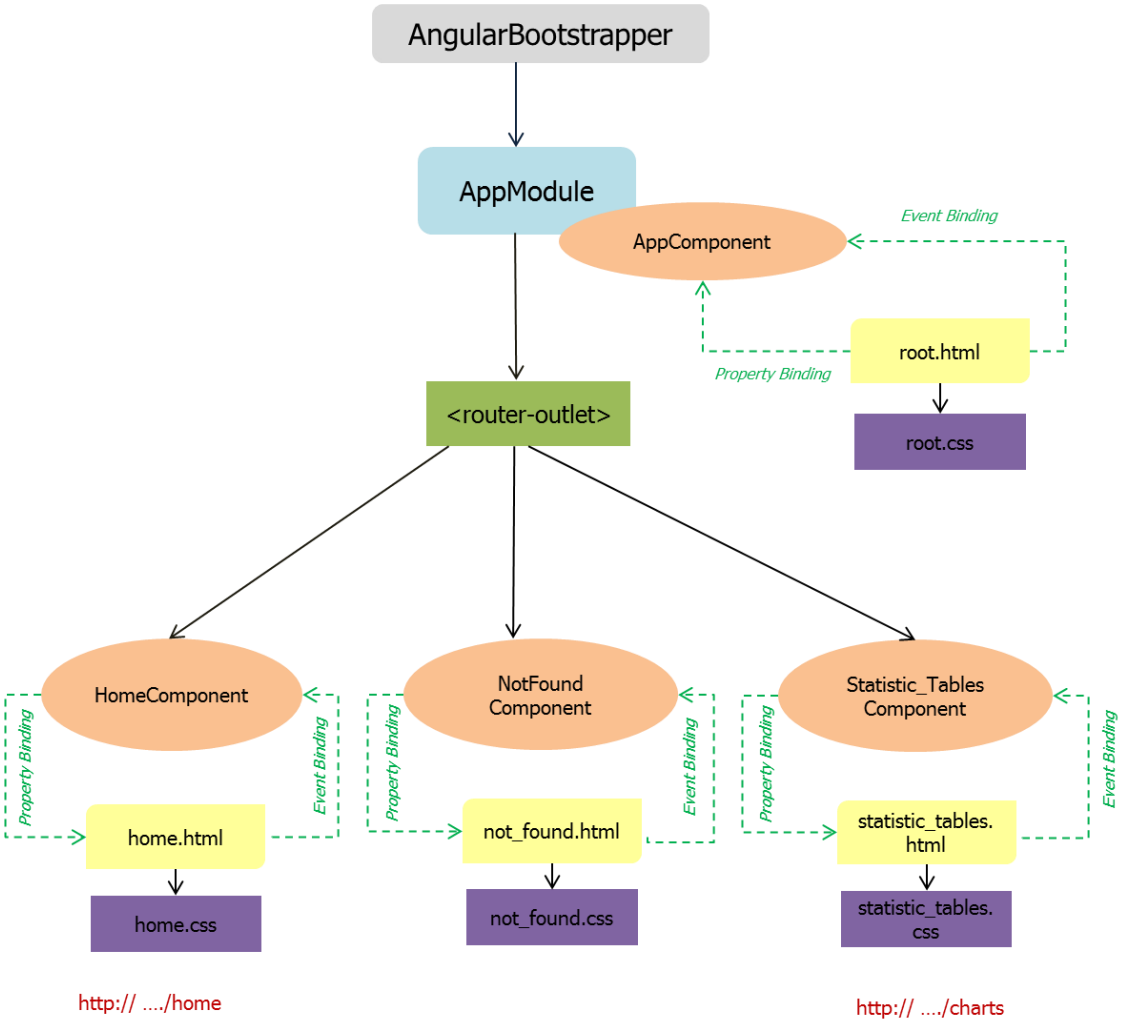


Abb.8: Architektur: primäre Routing in der Anwendung

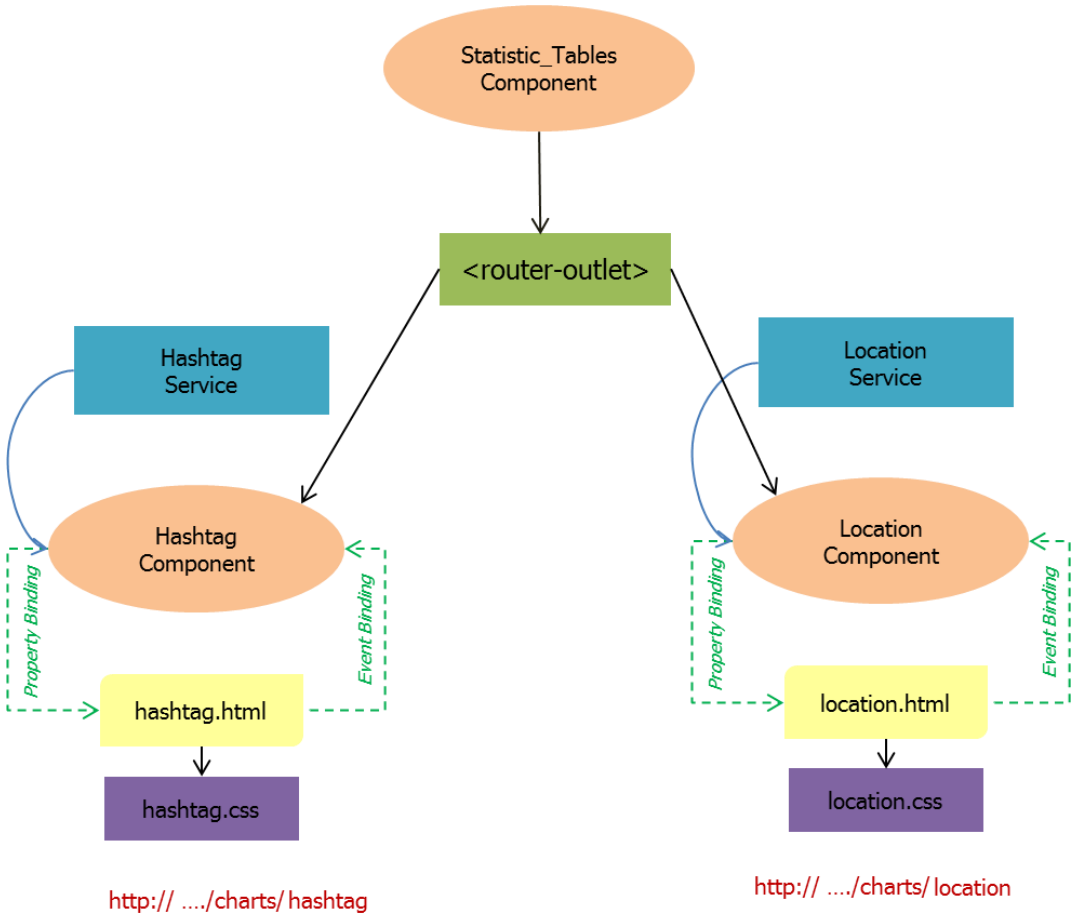


Abb. 9 : Sekundäre Routing in der Anwendung

3.2 Testing:

Das Testing des Front-Ends erfolgt in zwei Phasen (Abb.6):

a) *Unit Tests mit Jasmine[6] und Karma[7] Frameworks*

Hier werden der einzelnen Methoden mit Verwendung der gemockten Daten getestet.

Die Unit Tests können lokal mit dem Befehl

ng test

ausgeführt werden.

b) *E2E Tests mit Protractor [8]*

Das Testing der echte Applikation im Browser, Simulation der User Interaktionen. Das Ausführen der e2e Tests:

ng e2e

[6] https://jasmine.github.io/
[7] https://karma-runner.github.io/2.0/index.html
[8] http://www.protractortest.org/#/

Abb.6:Frameworks fürs Testing des Front-Ends

3.3 Build:

Es wird in der vorliegenden Anwendung zwischen zwei Enviroments unterschieden

a) *Stage Enviroment*, die für die Entwickler vorgesehen ist und

b) *Production Enviroment,* die für die Kunden bereitgestellt wird.

Das Deployment von Stage Enviroment läuft auf Surge[9] Storage , fürs Production Enviroment wurde einen Amazon S3 Bucket Storage[10] eingesetzt:

Stage Enviroment: http://systemintegration-stage.surge.sh

Production Enviroment: *http://twitter-analyser-production.s3-website.eu-central-1.amazonaws.com*

Für das Bauen der Anwendung wurde die von Angular CLI bereitgestellte Lösung benutzt.

Mit der Anweisung

ng build

kann die Anwendung für *die Stage Enviroment* gebaut werden.

[9] https://surge.sh/
[10] https://aws.amazon.com/de/s3/

Der zusätzliche Parameter --prod sorgt dafür, dass die Anwendung auf die *Production Evirome*nt gebaut wird:

ng build –prod.

In beiden Fällen erstellt Angular CLI einen */dist* Ordner und speichert dort das Build.

3.4 Continuous Integration/ Continuous Delivery (CI/CD):

Für die Versionskontrolle des Front-Ends wurde gitlab.com verwendet. Gitlab bietet eine Unterstützung für Continuous Integration/ Continuous Delivery.

Es sollten folgenden Schritten durchgeführt werden:

1. Das Konfigurieren von .gitlab-ci.yml, sodass die Unit Tests, e2e Tests und das Deployment auf die entsprechende Enviroment automatisch durchgeführt werden
2. Festlegen, wann und welche Enviroment aktualisirt werden muss:
    a) Stage Enviroment- nach jedem *git push*
    b) Production Enviroment – nach jedem *git push origin <tag>*.

Die Abbildung 7 zeitgt eine laufende Deployment auf die Stage Envioment der Anwendung.

```
Running with gitlab-runner 10.4.0 (857480b6)
  on docker-auto-scale (e11ae361)
Using Docker executor with image trion/ng-cli-karma ...
Using docker image sha256:df6af2881bdc78dbe588a5f5d6184c5b9f7a7ce526373f665533d74f2ec3a8f7 for predefined container...
Pulling docker image trion/ng-cli-karma ...
Using docker image trion/ng-cli-karma ID=sha256:e2983c7cee5f245dc6a55032a5115de86ea628a16f7905498c139cc97981361e for build container...
Running on runner-e11ae361-project-5133926-concurrent-0 via runner-e11ae361-srm-1516671205-5bae23ee...
Cloning repository...
Cloning into '/builds/summer_no_sad/Twitter_Analyser'...
Checking out cf6365b2 as master...
Skipping Git submodules setup
Checking cache for default:1...
Downloading cache.zip from http://runners-cache-5-internal.gitlab.com:444/runner/project/5133926/default%3A1
Successfully extracted cache
$ rm ./package-lock.json
$ npm install
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.3 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

up to date in 20.927s
$ ./node_modules/@angular/cli/bin/ng test --progress false --single-run=true --watch=false
23 01 2018 01:37:39.307:INFO [karma]: Karma v1.7.1 server started at http://0.0.0.0:9876/
23 01 2018 01:37:39.314:INFO [launcher]: Launching browser Chrome with unlimited concurrency
23 01 2018 01:37:39.330:INFO [launcher]: Starting browser Chrome
23 01 2018 01:38:00.170:INFO [Chrome 63.0.3239 (Linux 0.0.0)]: Connected on socket NMKXNvwncQ9xz1KMAAAA with id 36302259
Chrome 63.0.3239 (Linux 0.0.0): Executed 0 of 5 SUCCESS (0 secs / 0 secs)
Chrome 63.0.3239 (Linux 0.0.0): Executed 1 of 5 SUCCESS (0 secs / 0.436 secs)
Chrome 63.0.3239 (Linux 0.0.0): Executed 2 of 5 SUCCESS (0 secs / 0.743 secs)
Chrome 63.0.3239 (Linux 0.0.0): Executed 3 of 5 SUCCESS (0 secs / 0.996 secs)
LOG: 3
LOG: 3
Chrome 63.0.3239 (Linux 0.0.0): Executed 3 of 5 SUCCESS (0 secs / 0.996 secs)
LOG: 3
Chrome 63.0.3239 (Linux 0.0.0): Executed 4 of 5 SUCCESS (0 secs / 1.447 secs)
LOG: 110
LOG: 110
Chrome 63.0.3239 (Linux 0.0.0): Executed 4 of 5 SUCCESS (0 secs / 1.447 secs)
LOG: 110
Chrome 63.0.3239 (Linux 0.0.0): Executed 5 of 5 SUCCESS (0 secs / 1.635 secs)
```

Abb.10: Deployment auf die Stage Enviroment mit GitLab

3.5 Zusammenfassung Front-End:

Die Abb. 7 präsentiert die Screenshots des entwicker 1lten Front-Ends auf der production Enviroment.

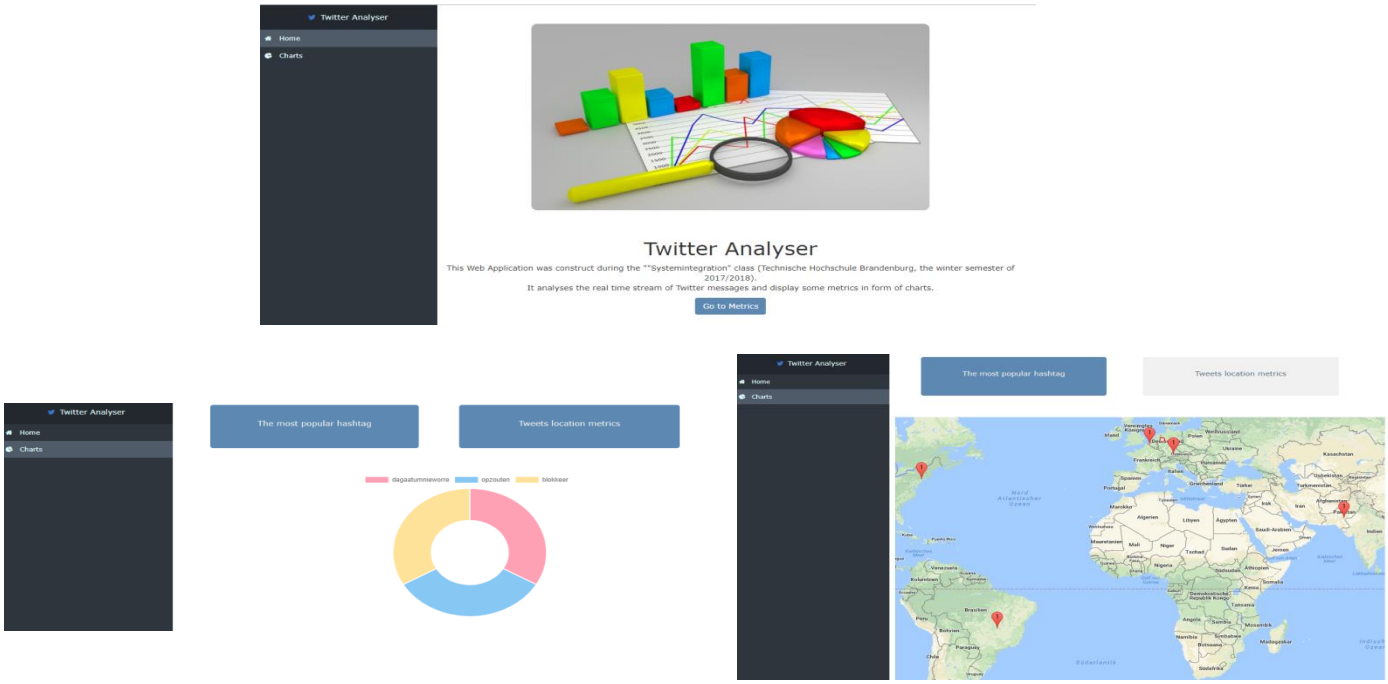


Abb.7: Screenshots des entwickelten Front-Ends auf der production Enviroment.

Repository: https://gitlab.com/summer_no_sad/Twitter_Analyser

Run Unit Tests: ng test

Run E2E Tests: ng e2e

Stage Enviroment: http://systemintegration-stage.surge.sh

Production Enviroment: http://twitter-analyser-production.s3-website.eu-central-1.amazonaws.com

CI/CD: .gitlab-ci.yml

Quellen:

[1]. Angular : https://angular.io/docs.

[2]. GitLab CI/CD: https://docs.gitlab.com/ee/ci/README.html