Architekturdokumentation – Projekt Quizify

Kernkonzepte und Methoden des Software Engineerings 2

WWI20B4 – Jonathan Janke

Amaan Ansari, Robin Bischoff, Emilia Wiesler, Jasmina Jäkle

Link zum GitHub Repository
https://github.com/AmaanAnsari/se_quizlet

Adresse
http://193.196.38.26:8000/static/

Architekturcharakteristiken & Begründung der gewählten Architektur

Für dieses Projekt konnten eine Vielzahl von Architekturcharakteristiken identifiziert werden.

Dadurch, dass es in Zukunft möglich sein soll, dass die Seite von vielen Nutzern gleichzeitig genutzt wird, muss die Architektur skalierbar sein. Außerdem wird es durch die Programmier-Wettbewerbe zukünftig zu Usage Spikes kommen, weshalb ebenfalls eine hohe Elastizität erforderlich ist. Eine Anforderung, die etwas vernachlässigt werden kann, ist die Performance und Responsiveness, da es für die User weniger wichtig ist, dass beispielsweise der Code schnell ausgewertet wird und sie ohne langes Warten das Ergebnis sehen können.

Es wurde eine Event-Driven Architektur verwendet. Dazu wurden verschiedene Services definiert, welche untereinander kommunizieren. Auf einen Broker wurde verzichtet. Es wird also Choreography und nicht Orchestration verwendet. Der Vorteil dabei ist, dass die Architektur deutlich skalierbarer ist, da der Orchestrator, der häufig zum Bottleneck wird, wegfällt.

Anfragen und zugehörige Antworten an den PythonCompiler werden allerdings über die Webapp geroutet. Gerne hätten wir die Anfragen vom PythonCompiler Services direkt beantwortet, um eine vollständige Cheography zu ermöglichen. Dies hat bei uns allerdings nicht funktioniert.

Aufbau des Systems

Das System besteht aus drei Services. Die Services laufen jeweils in einem eigenen Container, weshalb Docker Compose zum einfachen Erstellen, Konfigurieren und Verwalten verwendet wird.

Der erste Service ist ein Webservice, der die Webseite darstellt und HTTP API-Calls verarbeitet bzw. weiterleitet. Der zweite Service (pythoncompiler) kümmert sich um das Kompilieren des Codes und dessen Auswertung auf Basis der Tests. Das heißt, er nimmt die verschiedenen Testfälle entgegen und überprüft das Ergebnis des Compilers mit dem vorgesehenen Output und liefert dementsprechend bestanden oder nicht bestanden zurück. Als letzten Service gibt es den Authentifizierungsservice (db), welcher den User authentifiziert, das Userprofil speichert und bei Bedarf Userdaten zurückgibt.

Es wurde ein Netzwerk verwendet, welches die Kommunikation zwischen den Services erlaubt.

Nach einiger Überlegung wurde beschlossen den Authentifizierungsservice neben dem Webservice ebenfalls direkt von außen erreichbar zu machen, damit die Anfragen nicht unnötig über die Webapp geroutet werden müssen.

Die Container db und pythoncompiler haben zudem ein Volumen eingebunden, sodass Sie auf persistent gespeicherte Daten zugreifen können. Die DB verwendet das Volumen um die Datenbank (JSON Datei) zu speichern, während der pythoncompiler das Volumen benötigt, um Testfälle und Lösungen einzulesen.

Dockerfiles Aufbau

Für die Container wird Python 3.8 mit der Distribution Alpine verwendet. Innerhalb der Container werden benötigte Python Packages mithilfe von pip installiert. Zuvor wird dazu eine entsprechende Textdatei mit Requirements in den Container kopiert.

API Beschreibung

Eine Beschreibung der APIs erfolgt innerhalb des Quellcodes.

Beschreibung des Aufbaus und der Verwendung der Webseite

Die Webseite kann unter folgender Adresse aufgerufen werden: **http://193.196.38.26:8000/static/**

Die Webseite ist so aufgebaut, dass man zunächst auf der Main-View landet. Diese umfasst eine Kopfzeile, mit der zu bestimmten Abschnitten auf der Main-Page navigiert werden kann (About & Services) oder direkt zu der Quiz-Übersicht oder dem Login.

Damit die Ergebnisse gespeichert werden, ist es notwendig, dass mich sich einloggt. Dazu klickt man in der Kopfzeile auf den Eintrag Login und wird dann zu der entsprechenden Seite weitergeleitet. Wenn man noch keinen Account hat, dann muss man unterhalb des Logins auf „Create an account" klicken, um zur Seite für die Registrierung weitergeleitet zu werden. Dort kann man dann einen Usernamen oder eine E-Mail angeben und ein Passwort festlegen. Die beiden Felder dürfen nicht leer sein. Nach der Anmeldung wird dann ein Cookie gespeichert, der den Nutzernamen oder die E-Mail enthält.

Nach erfolgreicher Anmeldung wird man wieder zur Main-View weitergeleitet. In der Kopfzeile ist jetzt erkennbar, dass man angemeldet ist. Um auf die Quiz-Übersicht zu kommen kann jetzt entweder der entsprechende Punkt in der Kopfzeile oder der Button „TRY IT OUT!" im Abschnitt About ausgewählt werden.

In der Quiz Overview sind die fünf verschiedenen Aufgabenstellungen aufgelistet und man sieht, ob man angemeldet ist oder nicht. Nach Bearbeitung einer Aufgabe, wird hier auch der Fortschritt auf Basis der bestandenen Testfälle angezeigt.

Durch Klicken auf eine Aufgaben wird man zu dieser weitergeleitet. Auf dieser Seite befindet sich auf der linken Seite die Aufgabenstellungen, in der Mitte kann man den Code eingeben und auf der rechten Seite werden die Ergebnisse der Tests ausgegeben. Die Aufgabenstellung ist immer so aufgebaut, dass zunächst die Aufgabe erläutert wird, anschließend folgt ein Beispiel für den Input und den gewünschten Output und zum Abschluss gibt es eine Einschränkung der Größe des Inputs.

Wenn man jetzt die Aufgaben lösen möchte, dann kann man zunächst im Dropdown-Menü auswählen, in welcher Sprache man die Aufgabe lösen möchte. Anschließend schreibt man im mittleren Teil der Seite entsprechenden Feld den Code. Durch Klicken auf „Run Code" wird dieser dann auf Syntaxfehler überprüft. Das Ergebnis dieser Überprüfung wird dann im Feld unterhalb des Codes ausgegeben. Danach kann dann der Code anhand der definierten Testfälle überprüft werden. Dies geschieht durch Klicken auf „Submit Code". Im Feld ganz rechts wird dann das Ergebnis der einzelnen Tests ausgegeben. Wie genau dies funktioniert wird am Ende der Dokumentation erläutert.

Wenn man wieder zurück zur Quiz-Übersicht will, dann geht das über den entsprechenden Button auf der linken Seite, oder über den Eintrag in der Kopfzeile. Nachdem der Code erfolgreich submitted wurde, wird dies in der Datenbank für den entsprechenden User festgehalten. Sichtbar wird dies, indem in der Quiz-Übersicht für das erfolgreich gelöste Quiz der Kommentar „done, xx%" hinzugefügt wird. Hierbei entspricht die Prozentzahl dem Anteil der erfolgreich abgeschlossenen Tests.

Kommentar zur Bearbeitung der Aufgabe + Erläuterung der Lösung eines Quiz

Zunächst muss erwähnt werden, dass die Bearbeitung häufig in Form von Pair Programming in verschiedenen Konstellationen erfolgt ist. Deshalb lässt sich anhand der Commits nicht auf die Arbeitsleistung jedes Gruppenmitglieds schließen.

Außerdem ist der Code, aufgrund der Tatsache, dass ein Großteil der Gruppenmitglieder bereits wieder in der Praxisphase war, eher funktional geschrieben. Das heißt, es wurde kein Refactoring durchgeführt

und es gibt einige Dinge, die mit etwas mehr Zeit und freien Kapazitäten eleganter gelöst werden könnten. Insbesondere der Java Compiler konnte aufgrund eines Mangels an Zeit nicht implementiert werden. Konzeptionell würde dieser analog zum Python Compiler implementiert werden. D.h. man würde einen weiteren Service mit einem Java Image instanziieren, in welchem der Java Code dann ausgeführt werden würde.

Ebenfalls gestaltet sich das Schreiben von Code für das Lösen der Aufgaben schwieriger als gedacht, da für die Testfälle mit Input- und Output-Dateien gearbeitet wurde. Das heißt es muss vom Benutzer eine Input-Datei (input.txt) eingelesen werden, welche in jeder Zeile einen Testfall enthält. Anschließend muss der Benutzer das Ergebnis für jeden Testfall in jeweils eine neue Zeile der Output-Datei (output.txt) schreiben. Der Python Compiler Service prüft nach Programmausführung, ob die beiden Textdateien output.txt und solution.txt identisch sind. Letztere Datei ist bereits im System hinterlegt und beinhaltet die Lösungen zu den entsprechenden Inputs. Trotz der Schwierigkeiten wurden die Testfälle vollständig aufgestellt und auch implementiert. Dies kann überprüft werden, in dem beispielsweise für das Quiz Nummer 5 (Fibonacci Sequence) folgender Python-Code submitted wird.

```
with open('output.txt', 'w') as f:
    f.write('144\n5\n100\n1\n2\n2')
```

Dadurch wird das Schreiben der output.txt Datei simuliert, indem vier richtige und zwei falsche Ergebnisse zeilenweise in die Datei geschrieben werden. Wenn man anschließend auf die Quiz-Übersicht geht, dann sieht man das 67% aller Testfälle bestanden wurden. Würden die korrekten Werte in die Datei geschrieben werden, dann werden auch alle Tests bestanden und ein Ergebnis von 100% erreicht. Dies kann gerne durch Anpassung des obigen Codes simuliert werden. Dennoch können alle fünf Aufgaben durch Einlesen der input.txt-Datei und das Schreiben in die output.txt gelöst werden, auch wenn sich dies für Unerfahrenere im ersten Moment vielleicht etwas schwieriger gestaltet.