Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Entwicklung eines Portal zum Tennisspielersuche unter der Verwendung von Microservices

Exposee zur Bachelorarbeit

angestrebter Abschluss

Bachelor of Science

an der

Hochschule für Technik und Wirtschaft (HTW) Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang Angewandte Informatik

1. Gutachter: Prof. Dr. Christian Herta
2. Gutachter: Lehrbeauftragter Tobias Dumke

Eingereicht von:

Sakhr Nabil Al-absi

Datum: 19.10.2021

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Die Webentwicklung für Unternehmer ist heutzutage unerlässlich. Die Entwicklung einer Website ist der Schlüssel zur Verwirklichung dieser Ziele. Wenn das Ziel eine erfolgreiche Entwicklung einer Softwareanwendung ist, gibt es einen Moment in dem Anwendungslebenszyklus, der geändert werden muss.

Mit dem Wachstum von Handygerätenanzahl und das Cloud-Infrastruktur, Backend, Data muss immer für eine weitstehende Anzahl von Geräten verfügbar bleiben. Bei einem monolithischen Architekturstil zieht eine kleine Änderung im Code eine weitere Änderung in der gesamten Applikation mit sich. Dadurch wird der Neuentstand neuer Bugs in der Applikation zu einer gesammten Änderung der Applikation führen. Skalierbarkeit ist bei diesem Fall sehr stark beeinträchtigt und hier ist wo das Microservice-Architektur ins Spiel kommt.

Microservices-Arhchetiktur weist eine hohe Skalierbarkeit auf und kann komplizierte Entwicklungsprojekte in kleineren Subprojekte zerlegen. Diese Subprojekte können unabhängig von einer Programmiersprache miteinander kommunizieren bzw. Daten teilen. Laut einer aktuellen Nginx-Umfrage (NGINX-Community 2015) nutzen derzeit 36% der befragten Unternehmen Microservices, weitere 26% befinden sich in der Forschungsphase.

Die Motivation ist ein Anwendungsbeispiel (die Suche nach einem Tennisspielpartner) mit Microservices so aufzubauen, sodass die Anwendung in losen gekoppelten Diensten unterteilt ist. Die Dienste sollen fein abgestuft und die Protokollen leichtgewichtig sein. Microservices bietet nämlich eine größere Modularität an, wodurch die Anwendungen einfacher zu entwickeln, zu testen, bereitzustellen und vor allem einfacher zu ändern und zu warten ist.

Der Artikel "Microservice Trade-Offs "9 von Martin Fowler (Folwer 2015) fasst die Vorteile auf drei Punkte zusammen:

1. Es hat starke Module-Grenzen,
2. unabhängiger Bereitstellung (Deployment),
3. Technologievielfalt.

Nach Ansicht von Fowler liegen die Hauptvorteile dieses Punktes nicht nur in der offensichtlicheren Erleichterung bei der "Auswahl eines geeigneten Tools für die Aufgabe" Es ist offensichtlich, dass man in einer Microservices-Umgebung leichter eine größere Vielfalt an Programmiersprachen, Frameworks und Datenspeichern einführen und damit experimentieren kann.

Außerdem wird die Entwicklung über Python und pythonbasierten Frameworks (wie Django und Flask) erfolgen. Die große Unterstützung durch die Community und die Verfügbarkeit von Bibliotheken von Drittanbietern machen Python zur bevorzugten Sprache für die meisten Menschen, die häufig kleine Aufgaben automatisieren müssen.

1.2 Zielsetzung

Im Rahmen dieser Arbeit wird ein Beispiel für den Aufbau eines Portals zur Verbindung von Tennisspielern untereinander durchgeführt, wobei der Schwerpunkt auf der Implementierung unter Verwendung einer Microservices-Architektur liegt. Am Ende des Projekts sollte die Anwendung leicht skalierbar, und die Entwickler sollten in der Lage sein, neue Funktionen mit der besten Programmiersprache je nach Bedarf zu implementieren. Neue Ideen lassen sich leichter umsetzen, einführen, ausprobieren und möglicherweise wieder verwerfen, sowohl auf der Ebene der Entwickler als auch bei der betrieblichen Unterstützung.

Kapitel 2

Methodik

Nachdem alle Anforderungen bekannt und verfeinert werden, wird eine Recherche nach geeigneten Software-Entwurfsmustern, UML-Diagrammen usw. durchgeführt. Sobald eine Vorstellung gebildet ist, welche Diagramme für dieses spezifische Projekt geeignet sind. Die Klassen und Methoden werden mit diesen Entwurfsmustern und Diagrammen vorbereitet. Nachdem der Erstellung eines klaren Diagramms bzw. von passenden Entity-Relationship-Modelle (ERMs), wie das Programm aussehen wird, erfolgt der zweite Schritt, nämlich die Implementierung. Im Implementierungsteil werden alle Variablen, Funktionen und Feature, die in den Diagrammen und Anforderungen vordefiniert wurden, nach und nach implementiert. Am Ende werden alle Klassen und Funktionen getestet, um zu prüfen, ob alles ohne Probleme funktioniert. Wenn eine Grundlage geschaffen ist, dass die Anwendung ohne Fehler funktioniert, wird sie für die Bereitstellung (Deployment) vorbereitet, wobei jeder Dienst überwacht wird und für Continuous Integration (CI) und Continuous Delivery (CD) geeignet sein sollte.

2.1 Anforderungen

Es wird nach einer kompletten Entwicklung eines Tennisspieler-Suchportal angestrebt. Diese wird mit einer sequenziellen Planung von der Idee zur Implementierung erreicht. Am Anfang werden die Anforderungen durchgelistet, um eine Grundidee des Portals zu verschaffen. Manche diese Anforderungen sehen so weit wie folgendes:

1. Benutzer können sich registrieren
2. Benutzer können sich Ein- und Ausloggen
3. Registrierte Benutzer können ihre Verfügbarkeit in das Kalender hinzufügen, bearbeiten, oder löschen.
4. Benutzer können die Verfügbarkeit anderen Benutzern überprüfen
5. Benutzer können anderen verfügbaren Benutzer nach Region und Stärke ausfiltern.

6. Benutzer können spezifischen Tennisspielern in ihre Favoritenliste hinzufügen.
7. Benutzer können anderen Tennisspielern anfragen bzw. buchen.
8. Tennisspieler hat dann die Wahl den Termin zu stornieren.
9. Benutzer können anderen Tennisspieler bewerten und ein Kommentar hinterlassen
   (a) Die Bewertung wird freigeschaltet, nur wenn bereits ein erfolgreiches Spiel zwischen den Spielern stattgefunden hat.
      i. gebucht und ist nicht zum Spiel gekommen
      ii. gebucht und ist zum Spiel gekommen -¿ bewerten
   (b) Die Bewertung wird auf 2 wesentliche Punkte beruht:
      i. Stärke des Gegners
      ii. Kommentar
10. Neben der Suche nach verfügbaren Spielern soll auch eine Funktion zum Hinzufügen die eigene Anfrage. Das man nach einem Spielpartner, um eine gewisse Zeit sucht. Anschließend wird eine Übersicht anhand einer Skizze mit Adobe XD vorbereitet, um eine deutliche Visualisierung des Endprodukts vorzuzeigen. Daraus wird abgeleitet, welche wichtigen Komponenten für die weiteren Vorgänge noch nötig sind.

Mit dem Klassen-Diagramm werden die Eigenschaften von den Benutzern repräsentiert und mit dem ERM werden die Beziehungen zwischen Benutzern, Kalender, Anfragen, Verfügbarkeiten, usw. vorgestellt.

2.2 Softwaredesign

Es wird im Detail erklärt, wie APIs gestaltet werden sollten. Es wird definiert, was APIs sind und es wird beschrieben, welche Eigenschaften eine gute API haben sollte. Dann wird erklärt, wie man APIs entwirft, indem eine Reihe von APIs aus unserer Tennis-player-finder Beispielanwendung, die Auflistung der einzelnen Anwendungsfälle und die Beschreibung des Denkprozesses bei der Definition von Eingaben und Ausgaben für jede API.

API (Application Program Interface) ist ein Satz von Routinen, Protokollen und Werkzeugen zur Erstellung von Softwareanwendungen. Die API legt fest, wie Softwarekomponenten zusammenwirken sollen, und APIs werden bei der Programmierung von Komponenten der grafischen Komponenten der grafischen Benutzeroberfläche (GUI) verwendet

Weiterhin werden UML-Diagramme, Design Patterns, Entity-Relationship-Modelle gründlich untersucht, um die beste Anpassung an unsere Microservice-Architektur bei der Implementierung unserer Anwendung zu finden.

2.3 Implementation

Bei diesem Schritt wird den Programmcode so implementiert, so dass es die Vorplanung, die Methoden, und die Entwurfsmustern den vorherigen Schritten entsprechen. In dieser Phase wird mittels Internetrecherchen neuer Programmcode implementiert und Erfahrungen von Dozenten und Studierenden bei Schwierigkeiten einbezogen.

Jede Funktion und jedes Feature wird auch iterativ implementiert und je nach den Anforderungen der Anwendung ergänzt.

2.4 Testing

Der implementierte Programmcode wird im letzten Schritt auf Fehler und Bugs überprüft. Die Möglichkeit zur Wiederverwendung des Tests soll im Vordergrund stehen. Die Testcases müssen nicht automatisiert, allerdings ohne großen Aufwand und Vorkenntnisse manuell durchgeführt werden können. Auf eine bestimmte Testing-Strategie, die mit dieser Architektur passt, wird noch untersucht.

2.5 Deployment

2.5.1 Continuously Integrating / CI Microservice

Eine unverzichtbare Maßnahme zur Automatisierung der Softwareentwicklungsumgebung für die Microservice Entwicklung ist der Einsatz von Continuous Integration (CI). Der Einsatz von KI-Systemen kann einige sehr wertvolle Vorteile für den Entwicklungsprozess bringen: Wie Verkürzte Feedback-Schleife und Rückverfolgbarkeit

„Ein weiterer Punkt zur kontinuierlichen Integration und ihrer Beziehung zu Microservices-Architekturen ist, dass Sie Ihre CI-Systeme so einrichten sollten, dass sie jeden Microservice einzeln behandeln. Viele der Vorteile des Einsatzes von Microservices liegen in der Tatsache begründet, dass sie es den Anwendern ermöglichen, Entscheidungen auf einer sehr feinkörnigen Ebene zu treffen. In diesem Sinne empfehlen wir nicht weniger als ein Code-Repository pro Service. Ein solches Setup macht Dinge wie den Wechsel der Programmiersprache, die Optimierung in Bezug auf Caching- und Datenspeichertechnologien, das Erfassen des Zwecks des Repositorys und die Übertragung der Eigentumsrechte viel einfacher, als wenn mehrere diskrete Dienste in derselben Code-Revisionseinheit zusammengefasst werden.“ (Jr. und Schmelmer 2016)

2.5.2 Continous Code Delivery / CD

Das Konzept der kontinuierlichen Bereitstellung (Continuous Delivery, CD) greift die Konzepte der kontinuierlichen Integration auf und wendet sie auf den gesamten Software-Release-Zyklus an. Dabei handelt es sich um eine Praxis, bei der das Entwicklungsteam jeden Schritt, den ein Softwareprodukt durchläuft, vom Einchecken des Codes bis zur Freigabe der Software für die Produktion, einsehen kann und die Integration der Werkzeuge automatisiert wird. bis hin zur Freigabe der Software für die Produktion.

2.5.3 DigitalOcean

Für die Deployment gibt es ein sehr guten Cloud-Infrastruktur-Anbieter. In der DigitalOcean-Hauptseite wird die Möglichkeit von der Infrastruktur wie folgendes beschrieben:

1. Hosten von Websiten mit der einfachsten Cloud-Hosting-Plattform.
2. Erstellung von Webanwendungen oder API-Backends auf einer robusten Infrastruktur.
3. Bereitstellung von Container-basierte Anwendungen mit verwaltetem Kubernetes.
4. Beschleunigung der Entwicklung mit intuitiven API, Entwickler-Tools und CI/CD-Add-ons. (Digital Ocean — Welcome to the Developer Cloud 2021)

Kapitel 3

Erwartete Ergebnisse

Das Anwendungsbeispiel von Tennisspielersuchende wird in verschiedenen Frameworks je Anhand Effizienz und Bedarf implementiert. Manche Funktionen und Dienste werden innerhalb des Django- oder Flask Frameworks bereitgestellt. Zur selben Zeit wird auch einen anderen Dienst für das Frontend bereitgestellt. Diese Dienste sollten unabhängig voneinander verfügbar sein und zugleich miteinander kommunizieren, in dem immer nach notwendigen Ressourcen aufgerufen werden kann.

Die untenstehende Abbildung zeigt eine grobe Vorstellung wie es am Ende aussehen sollte:

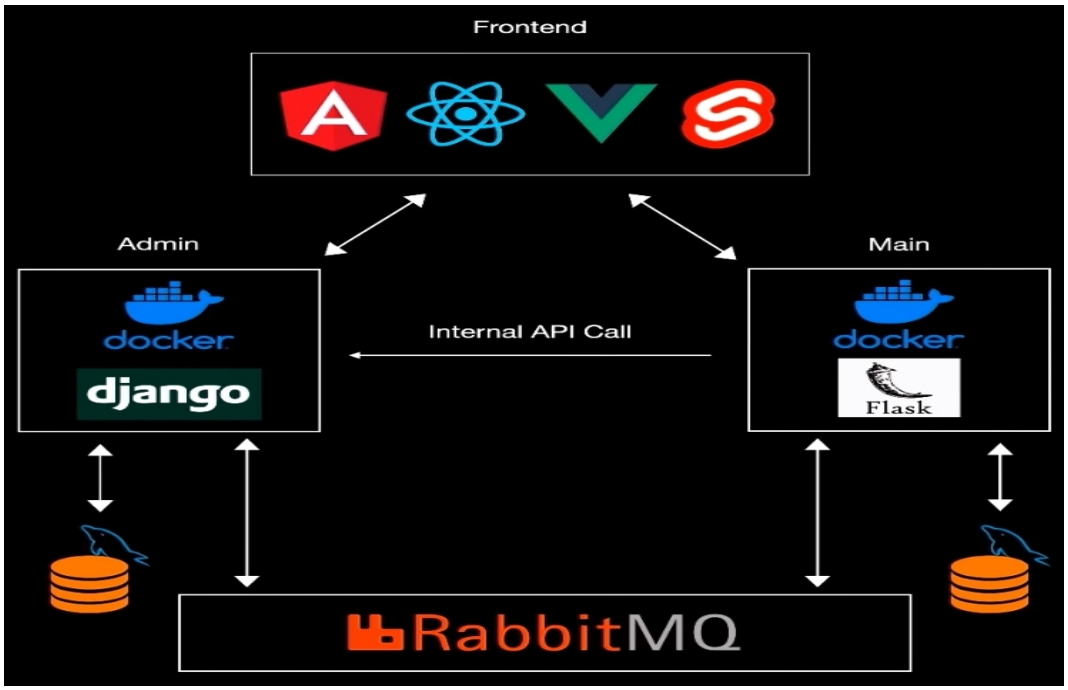

Abbildung 3.1: Django, Flask und ReactJS mit Microservices

Die Einführung einer Microservices-Architektur bedeutet, dass die Produktionsumgebung über eine große Anzahl von einsatzfähigen Einheiten verfügt, für die letztlich der Entwickler verantwortlich ist, sicherzustellen, dass sie entsprechend den Anforderungen wie erwartet funktionieren. Wie bei jeder Art von Software muss man wissen, wann Probleme auftreten, und zwar idealerweise, bevor sie sich auf die Anwendungsmetriken auswirken, z. B. auf Benutzerbesuche, Anmeldungen oder die Anzahl der Termine. In manchen Fällen kann es Stunden - oder Tage - dauern, bis man herausfindet, dass ein kleines Stück Software nicht funktioniert, und es ist nicht gut, wenn man Produktionsprobleme erst entdeckt, nachdem die Benutzer sie entdeckt haben. Es ist äußerst wichtig, ein gut abgestimmtes Überwachungs-Toolset einzurichten, um die Entwickler zu alarmieren, wenn ein Dienst nicht richtig funktioniert.

Jeder einzelne Dienst in der Umgebung muss ausnahmslos überwacht werden. Verfügbare Dienste, auch "up and running"genannt, sind gefragt.

3.1 Monitoring von Services

Als erstes muss sichergestellt werden, dass die Dienste tatsächlich von ihren Clients innerhalb oder außerhalb des Netzes erreichbar sind. Eine einfache Möglichkeit, die Verfügbarkeit eines Dienstes zu überprüfen, besteht darin, einen Health-Check-Endpunkt anzusprechen, der in der Regel dafür zuständig ist, den von ihm verwendeten Datenspeicher anzusprechen und eine einfache Antwort zu geben. Die einzigen Dinge, die wir überwachen können, sind die, die wir messen. Im Allgemeinen ist dies ein Bereich, in dem Sie mehr Messdaten sammeln sollten, als Sie tatsächlich nutzen.

Einige Tools, die wir für die Überwachung verwenden können, sind StatsD, Graphite und New Relic APM. (Jr. und Schmelmer 2016)

3.2 Code Dokumentationen

Hier werden alle Schritte zur Installation des Projekts beschrieben und klar dokumentiert. Diese sind Sachen wie Abhängigkeiten (Dependencies), Moduls, damit das Endprojekt ohne Herausforderungen auf dem eigenen Rechner ausgeführt werden kann. Das heißt, das Projekt wird in GitHub gepusht und ein ReadMe.md Datei erstellt, wo alle diese wichtigen Informationen zur Erfolgreichen Ausführung verfasst.

Daneben wird, jeden Schritt, den beim Schreiben des Codes unternommen wird, durch die Commits in gitHub nachverfolgbar sein. Außerdem werden Funktionen und Features in Iterationen implementiert und veröffentlicht, sobald sie fertig sind.

Kapitel 4

Zusammenfassung

4.1 Zeitplan

Erste Woche: Klarer Definition des gewünschten Ziels

Zweite Woche: Entwurf mit Adobe XD

Dritte Woche:

- Entwurf mit UML-Diagrammen und ERM's
- Einführung zu Microservices und Anfang der Einrichtung von
  - Django
  - Flask
  - Frontend: vlt ReactJS

Vierte Woche: Einrichtung der Frameworks (Django & Flasks) und Anfang der Implementation

Fünfte Woche: Implementation

- Django: Benutzer zu DB persistieren. richtige REST API bilden (CRUD)
  - Benutzer kann:
    1. Name, Region, Stärke und Alter hinzufügen bzw. bearbeiten
    2. Foto hinzufügen
- Postman: Testen, ob Speichern/Bearbeitung/Löschen von Benutzern richtig funktioniert.

Sechste Woche:

- Eine simple Frontend-Seite für
  - Registrieren /register,
  - Anmelden /anmelden,
  - Homepage /home
  - Edit Profile /profile
  - User /user/<id>
  - About /about

Siebte Woche:

- Implementation der eigenen Kalender und Verfügbarkeitsfunktion
- Der Kalender kann von den angemeldeten Benutzern bearbeitet werden.
  - Ein Kalender hat nur ein Owner (Besitzer)

Achte Woche: Implementation der Suchfunktion & Entwicklung der Testkonzepte

- Suche nach:
  - Alter, Region, Stärke, oder alle

Neunte Woche: Entwicklung der Testkonzepte & Deployment

Zehnte Woche: Deployment & Zusammenfassung der Abschlussarbeit

4.2 zu lesenden Literaturen

Bücher, die zum Lesen geplant sind:

1. Architecture Patterns with Python by Bob Gregory and Harry Percival
2. Microservices Development Cookbook Design and build independently deployable, modular services by Paul Osman
3. Scrum the art of doing twice the work in half the time by Jeff Sutherland
4. Software Design creating solutions for ill-structured problems by David Budgen
5. Microservice from Day One by Cloves Carneiro Jr. and Tim Schmelmer
6. Docker The Complete Beginners Guide To Starting With Docker by Austin Spencer
7. Building Microservices - Designing Fine-Grained Systems by Sam Newman

8. Software Design - Creating Solutions for ill-Structured Problems by David Budgen

9. Wissenschaftliches Arbeiten - Methodenwissen für das Bachelor-, Master- und Promotionsstudium by Bern Heesen

Literatur

Folwer, Martin (Juli 2015). „Microservice Trade-Offs“. In: URL: https://martinfowler.com/articles/microservice-trade-offs.html.

Jr., Cloves Carneiro und Tim Schmelmer (2016). *Microsercices from Day One: Build robust and scalable software from the start*. 1. Aufl. Apress.

NGINX-Community (2015). *The Future of Application Development and Delivery Is Now Containers and Microservices Are Hitting the Mainstream*. URL: https://www.nginx.com/resources/library/app-dev-survey/ (besucht am 19. 10. 2021).