Universität Stuttgart

Advanced Software Engineering

Übungsblatt 2

Maximilian Peresunchak (st152466@stud.uni-stuttgart.de)
Nico Reng (st188620@stud.uni-stuttgart.de)
Viorel Tsigos (st188085@stud.uni-stuttgart.de)
Philip Reimann (st182312@stud.uni-stuttgart.de)
Christian Keller (st166512@stud.uni-stuttgart.de)
Johannes Heugel (st183360@stud.uni-stuttgart.de)
Benedikt Wachmer (st177118@stud.uni-stuttgart.de)
Miles Holl (st180549@stud.uni-stuttgart.de)

Wintersemester 2025

15. November 2025

Aufgabe 1:

a) Die gegebenen Features lassen sich wie folgt den Klassen des Kano-Modells zuordnen:

Klassifikation	**Feature Nr.**	**Begründung**
Basismerkmal	6	Wenn Support benötigt wird und nicht verfügbar ist, ist die Nutzerfahrung massiv beeinträchtigt. Wird kein Support benötigt, bietet dieses Feature keine Vorteile für einen Nutzer.
Leistungsmerkmal	2	Desto besser Routen für Nutzeranfragen berechnet werden, desto mehr steigt auch die Nutzerzufriedenheit. Die Performanz des Systems steht also im direkten Zusammenhang mit der Kundenzufriedenheit.
	4	Ist nicht zwingend Voraussetzung für die App, kann jedoch deutlich die Nutzerzufriedenheit beeinflussen, da das Feature die Nutzbarkeit unter realen Bedingungen spürbar verbessert bzw. bei Abwesenheit verschlechtert.
Begeisterungsmerkmal	1	Zwar hat die Angabe des eingesparten CO_2 einen direkten positiven Einfluss auf die Nutzererfahrung, insbesondere da GreenRide direkt auf umweltbewusste Nutzer abzielt. Für das zentrale Ziel der App Menschen nachhaltig von A nach B zu bringen, ist dieses Feature jedoch nicht zwangsläufig notwendig und wirkt sich bei Abwesenheit nur bedingt auf die Nutzerfahrung aus, da auch ohne eine Anzeige CO_2 reduziert wird.
	5	Ist als unerwartetes Zusatzfeature für Poweruser zu verstehen. Für die reguläre Nutzung nicht notwendig, erhöht jedoch den Komfort.
	3	Für den Nutzer unerwartetes Feature, das die Motivation die App zu nutzen und im Allgemeinen die Nutzerbindung fördert. Resultiert demnach in eine höhere Nutzerzufriedenheit, wird jedoch bei Fehlen nicht vermisst.

b) Wenn GreenRide seine Zielgruppe um Gelegenheitsnutzer im ländlichen Raum erweitert, verschieben sich die Gewichtungen der Features zu Einfachheit und Zuverlässigkeit: Begeisterungsmerkmale wie Gamification, Smartwatch-Integration oder CO_2-Anzeige verlieren an Bedeutung, da Gelegenheitsnutzer sich bereits mit Basis-Features zufrieden stellen und die App nur als "Mittel zum Zweck" nutzen. Im Gegenzug gewinnt, aufgrund der geringen Dichte von Mobilitätsangeboten im ländlichen Raum, der Faktor der Zuverlässigkeit an Bedeutung. Zur Zuverlässigkeit zählt dabei neben der Korrektheit der Informationen, auch die Robustheit der App unter realen Bedingungen (z.B. Funklöcher) und die dauerhafte Erreichbarkeit eines Notfall-Supports. Darüber hinaus sind Gelegenheitsnutzer weniger geübt mit der Bedienung der App (oder des ganzen Geräts), wodurch die Usability an Bedeutung gewinnt.

Im Hinblick auf die oben diskutierten neuen Schwerpunkte von GreenRide ergeben sich folgende potenzielle Anforderungen:

- **Anzeige der Verfügbarkeit der Mobilitätsangebote**
  Dem Nutzer werden zuverlässige und präzise Informationen zur Verfügbarkeit der verschiedenen Fahrzeuge zur Verfügung gestellt, sodass dieser z.B. nicht auf einen entfallen Bus wartet oder zu einer Ladestation geht an der kein Mietwagen zur Verfügung steht.

  Leistungsmerkmal: Die Kundenzufriedenheit steht im direkten Zusammenhang mit den angezeigten Informationen, steigt/sinkt also mit deren Zuverlässigkeit und Genauigkeit.

- **Reduzierte Benutzeroberfläche mit einfacher Benutzerführung**
  Mit Hinblick auf die Gelegenheitsnutzer sollte eine simple Benutzeroberfläche mit Schwerpunkt auf Usability vorliegen. Die Nutzung soll intuitiv und unkompliziert sein.

  Basismerkmal: Voraussetzung für die erfolgreiche Nutzung der App durch Gelegenheitsnutzer. Die beste Benutzeroberfläche ist diejenige, die so intuitiv gestaltet ist, dass der Nutzer gar nicht an diese denkt.

c) **CO_2-Einsparungsanzeige**

- **Funktionale Frage:**
  „Was würden Sie davon halten, wenn die App Ihnen anzeigt, wie viel CO_2 Sie durch Ihre Fahrten eingespart haben?"
- **Dysfunktionale Frage:** „Was würden Sie davon halten, wenn die App Ihnen KEINE Informationen zur eingesparten CO_2-Menge anzeigt?"

Notfall-Support

- **Funktionale Frage:**
  „Was würden Sie davon halten, wenn die App bei technischen Problemen (z. B. defekter Scooter) einen schnellen Notfall-Support über einen SOS-Button bietet?"
- **Dysfunktionale Frage:** „Was würden Sie davon halten, wenn die App bei technischen Problemen KEINEN schnellen Notfall-Support über einen SOS-Button bietet?"

d) Die Grenzen des Kano-Modells in einem agilen Entwicklungsumfeld mit iterativen Releases liegt vor allem darin, dass eine solche Analyse lediglich eine Momentaufnahme der Erwartungen der Kunden und der Zielgruppe ist, die sich mit der Zeit ändern können: Während der Kunde zunächst eine Applikation mit Funktion x fordert, wundert er sich nach Projektende, warum die Anwendung nicht auch Funktion y wie das Konkurrenzprodukt unterstützt. Um diesem Problem Abhilfe zu verschaffen bietet es sich an, die Kano-Modell Befragung in regelmäßigen Intervallen, z.B. nach jedem Sprint, erneut durchzuführen, um mit dem sich schnell veränderten Markt und den sich veränderten Erwartungen des Kunden mithalten zu können.

Letzter Vorschlag führt jedoch gleich zur nächsten Problematik, die Durchführung von Kano-Befragungen ist, aufgrund der Vielzahl an zu befragenden Personen, ressorcen- und zeitaufwändig, was insbesondere im Hinblick auf ein agiles Entwicklungsumfeld verlangsamende Auswirkungen auf den Gesamtprozess haben kann. Idealerweise sollte die Befragung daher automatisiert erzeugt, durchgeführt und ausgewertet werden.

Neben den schnell alternden Analyseergebnissen, stellt auch der ausschließliche Fokus auf Features ein weiteres Problem für das Kano-Modell dar. Die Gefahr: das ganzheitliche Nutzerlebnis gerät aus den Augen der Entwickler, stattdessen wird zu viel Zeit in die Optimierung einzelner Features gesteckt. Um dem entgegenzuwirken sollten nutzerzentrierte Methoden, wie z.B. User Story Mapping oder Customer Journey Analysen, mit dem Kona-Modell kombiniert werden und im Kontext von GreenRide, Feature-Bewertungen immer in den Kontext der gesamten Mobilitätserfahrung gesetzt werden.

Aufgabe 2:

a)

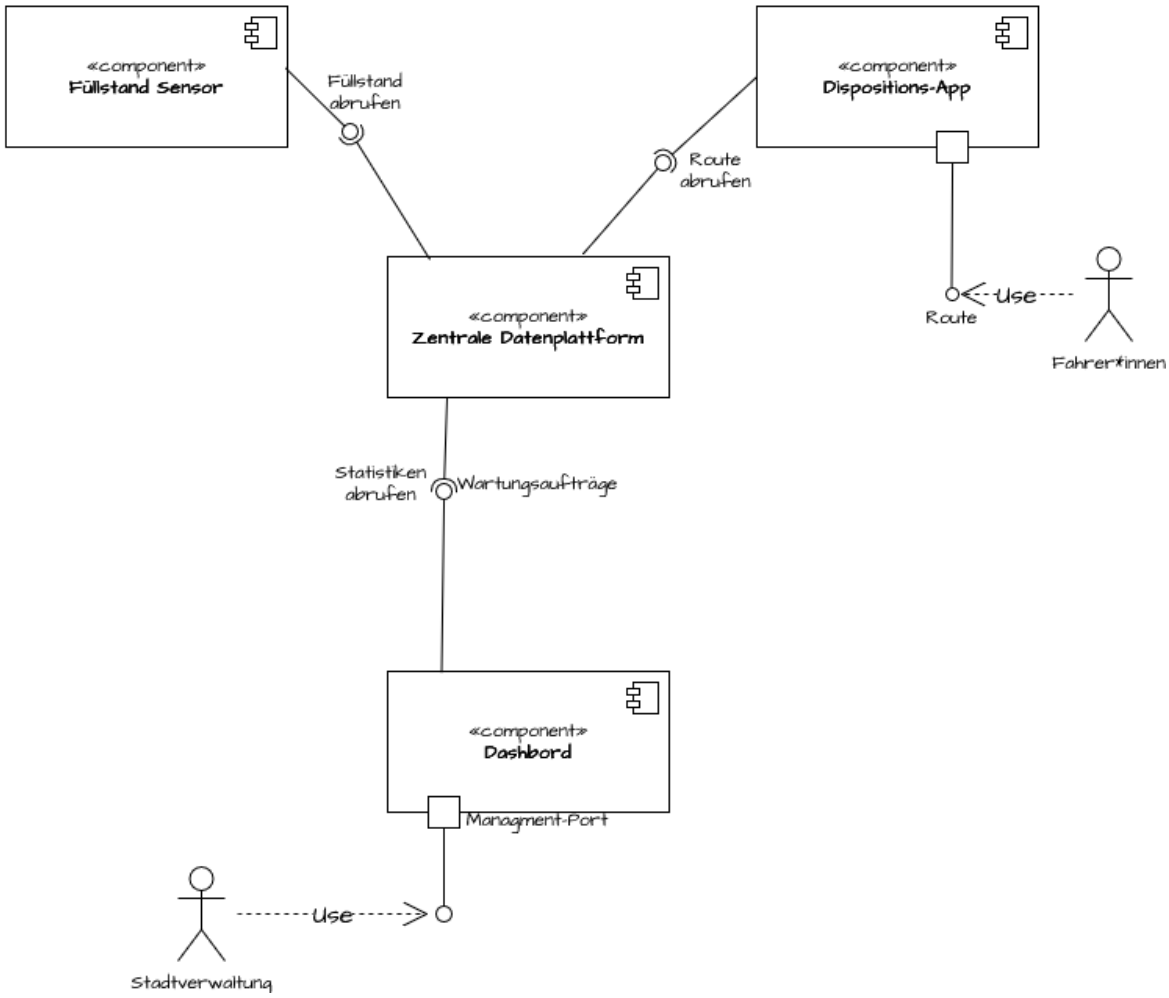

Abbildung 1: Komponentendiagramm

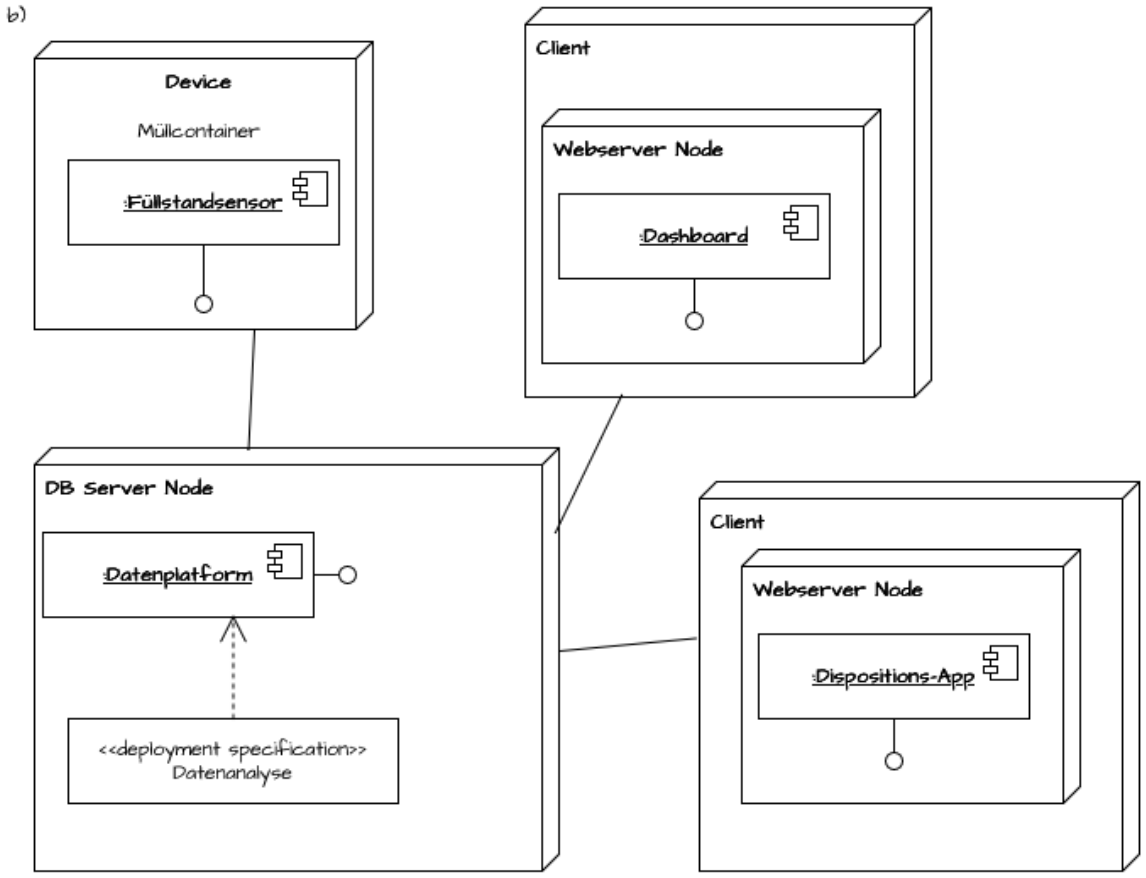

Abbildung 2: Deploymentdiagramm

c) Ein kritischer Engpass ist die Datenplattform, da sie sowohl die Sensordaten als auch die Anfragen entgegennehmen muss. Zusätzlich müssen die Sensordaten verarbeitet werden. Kommen nun noch sehr viele Statistik- und Routenanfragen gleichzeitig von Dashboard und App hinzu, könnte dies zu einer Leistungsminderung führen.

d) Ein geeignetes Entwurfsmuster besteht darin, die einzelne Datenplattform in mehrere kleinere Plattformen zu unterteilen, z.B. eine pro Stadtteil. Dadurch wird die zu tragende Last aufgeteilt, und die horizontale Architektur ermöglicht eine einfache Skalierung. Das System kann so flexibel an wachsende Städte angepasst werden. Die Ausfallsicherheit wird dadurch verbessert, dass der Ausfall einer Datenplattform nicht zum Ausfall des gesamten Systems führt, sondern nur zu einem Teilausfall des betroffenen Bereichs.

Aufgabe 3:

a) Man sieht schnell dass es sehr wirr ist, Entwickler haben eigene REST-Endpunkte implementiert. Zudem wissen die Entwickler selbst nicht genau wie andere Module funktionieren und arbeiten nur an ihrer eigenen Komponenten. Dies führt zu vielen Fehlern und Problemen, sowie zu einem hohen Wartungsaufwand. Außerdem wird es sehr sicher zu Mergekonflikten kommen.

b) **Wartbarkeit:**
Diese ist wie schon vorher erwähnt, nicht wirklich gegeben, da die Entwickler selbst nicht einen guten Überblick über das Gesamtprojekt haben, das erschwert natürlich die Wartung und es erhöht auch die Komplexität des Systems. Zudem denke ich auch, dass es zu Mergekonflikten kommen wird, da anscheinend nicht viel Kommunikation zwischen den Entwicklern stattfindet.

Skalierbarkeit:
Durch das manuelle Deployment auf nur einen zentralen Server, wird es sehr sicher zu Problemen kommen, denn die Infrastruktur und der Server limitiert ist. Die Datenbanken von den Microservices könnten einzelne Services ein wenig verbessern.

Sicherheit:
Laut dem Sicherheitsbeauftragten, sind die Kommunikationswege zwischen Maschinen und der Cloud unverschlüsselt. Das ist generell ein großes Sicherheitsrisiko, da so ein Angreifer leicht an die Daten kommen kann. Hier sollte man decryption und encryption Verfahren benutzen, die wir in GIS beigebracht bekommen.

c) Wie schon gerade erwähnt, kann ein Angreifer leicht an die Daten kommen, da die Kommunikation unverschlüsselt ist. Dadurch können senible Daten von wie Produktionsdaten abgefriffen werden, welche ein großes Sicherheitsrisiko darstellen. Durch die Verwaltung bzw. das Deployment auf nur einem Server, kann es zu einem Single Point of Failure kommen, also wenn der Server ausfällt, fällt das ganze System aus. Zudem wenn nur ein einziger Service ausfällt muss man den Server auch komplett neustarten, d.h. auch die anderen Services sind dann nicht mehr verfügbar, bis der Server wieder online ist. Das letzte Risiko, ist das durch die schlechte Kommunikation zwischen den Enwicklern, Fehler im Symstem entstehen können, die ggf. auch zu Sicherheitslükcne führen können. Dadurch dass die einzelnen Entwickler nicht mal einen Überblick über mehrere Komponenten haben, können sich Sicherheitslücken an den Schnittstellen einschleichen.

d) In der Vorlesung wurden bisher keine konkreten Architekturen vorgestellt, deshalb habe ich nach einiger Recherche folgende Vorschläge: Eine Architektur mit Microservices mit Orchestrierung und Zero Trust. Durch Microservices-Architektur reduzieren wir die Gefahr von einem kompletten Serverneustarts. Jeder Service läuft also unabhängig voneinander. Wenn ein Service ausfällt, laufen die anderne einfach weiter. Durch ein API Gateway kann man die Sicherheit erhöhen. Es terminiert die Kommunikation von außen und kann TLS/SSL Verschlüsselung erzwingen. Durch Zero-Trust kann man auch die ende-zu-ende Verschlüsselung und Authentifizierung erzwingen. Also muss man sich vorher authentifizieren bevor man Daten abrufen kann. Man könnte z.B. OAuth 2.0 verwenden.

e) **Architekturrevies:**
Ich würde Architekturrevies vorschlagen, da so die Entwickler gezwungen sind sich mit dem Gesamtsystem auseinanderzusetzen und nicht nur mit ihrer eigenen Komponente. Dadurch können Fehler und Probleme frühzeitig erkannt werden. Zudem ist es so möglich, dass keine Abweichungen zum Endprodukt entstehen, da die Entwickler sich regelmäßig mit dem Gesamtsystem beschäftigen müssen.

Verschlüsselung:
Es sollte eine schnellstmögliche Verschlüsselung (beispielsweise TLS oder SSL) verwendet werden, damit die komplette Kommunikation zwischen den Maschinen und der Cloud verschlüsselt ist. Das könnte man über ein API Gateway machen.

Dokumentation:
Durch eine gute Dokumentation können die Entwickler besser verstehen, was die anderen Komponenten machen und wie sie richtig ohne Bugs funktionieren. Dadruch könnte man seine Komponente besser an die anderen Komponenten anpassen und so erstenes Fehler und Debugging Zeit sparen und zweitens das Risiko von Sicherheitslücken an den Schnittstellen verringern.

Kommunikation:
Wie es aus den Interviews hervorgeht, gibt es anscheinend wenig Kommunikation innerhalb des Teams.Der Projektleiter möchte, dass die Softwareplattform innerhalb von drei Monaten beim ersten Kunden lauffähig ist. Aus den Interviews geht hervor, dass hier noch viel Arbeit zu tun ist und generell der Überlick anscheinend über das Projekt fehlt, also der aktuelle Stand des Projekts. Hierzu wären regelmäßige Meetings mit sinnvollen Strukturen wie Scrummaster etc. sinnvoll.