# Einführung und Ziele

* **zugrunde liegende Geschäftsziele:**
* Kostengünstige Ride-Sharing App
* Benutzer mit nahegelegenen Fahrern verbinden, und dadurch
* Anzahl der Autos auf der Straße verringern
* Alternative Transportationsmöglichkeit für Benutzer die keinen Zugang zu einem Auto haben
* **wesentliche Aufgabenstellungen:**
* Entwicklung einer iOS / Android App
* Zahlungssystem
* Echtzeit-Updates (Verfügbarkeit von Fahrten, Ankunftszeiten…)
* **essenzielle fachliche Anforderungen an das System:**
* Benutzerregistrierung & Profilerstellung
* Fahrtensuche / Buchung
* Echtzeit-Tracking (von Fahrten & jeglichen Updates)
* Verifizierungsprozess (Fahrer/Mitfahrer)
* Bewertungs-/Rezensionssystem
* **Qualitätsziele für die Architektur:**
* Sicherheit & Datenschutz (Zahlungsdetails, persönliche Informationen)
* Skalierbarkeit (wachsende Nutzeranzahl)
* Performance (keine Verzögerungen)
* Kompatibilität (iOS/Android)
* Zuverlässigkeit (bei Ausfall 🡪 Wiederherstellungsfähigkeit)
* **relevante Stakeholder und deren Erwartungshaltung:**
* Maryam Patel (Investor): Entwicklungsplan, gute Rendite
* Raj Gupta (Geschäftsentwicklung): App soll lokalen Transportbedürfnissen entsprechen
* Megan Chen (UX-Designerin): Benutzerfreundliches Design
* Amirah Rahman (Fahrervertreter): App soll Bedürfnissen & Anforderungen der Fahrer gerecht werden
* Javier Gomez (Umweltaktivist): Reduzierung von Emissionen/Verkehr

## Aufgabenstellung

**Inhalt:**

Die RideShare-App soll eine mobile Applikation sein, die Benutzer mit nahegelegenen Fahrern für Mitfahrgelegenheiten verbindet. Sie soll nicht nur kostengünstiger sein, sondern gleichzeitig auch Emissionen und Verkehr reduzieren.

**Motivation:**

Durch immer mehr Verkehr und den damit verbundenen Umweltauswirkungen besteht immer mehr Bedarf an Alternativen für den Personenverkehr. RideShare soll nicht nur die Verkehrsbelastung reduzieren, sondern auch Personen die keinen Zugang zu einem Auto haben die Möglichkeit geben, für persönliche Zwecke eines benutzen zu können (ohne eines kaufen zu müssen).

**Use-Cases:**

## Use Case 1: Benutzerregistrierung

Ein neuer Benutzer kann sich registrieren und ein Profil erstellen

## Use Case 2: Fahrtensuche und Buchung

Ein Benutzer kann nach verfügbaren Fahrten suchen und eine Fahrt buchen

## Use Case 3: Echtzeit-Tracking

Benutzer können den Standort des Fahrzeugs in Echtzeit verfolgen

## Use Case 4: Zahlungsabwicklung

Nach Abschluss der Fahrt kann der Benutzer über die App bezahlen

## Qualitätsziele

|  |  |
| --- | --- |
| Priorität 1: Sicherheit | Szenario: Ein Benutzer will auf private Daten eines anderen Nutzers zugreifen aber wird durch das System daran gehindert. |
| Priorität 2: Performance | Szenario: Ein Benutzer fordert eine Fahrt an – die Antwortzeit unserer App soll unter 3 Sekunden betragen. |
| Priorität 3: Benutzerfreundlichkeit | Ein neuer Benutzer sollte es schaffen, innerhalb von 10 Minuten nach dem runterladen problemlos durch die App navigieren zu können. |
| Priorität 4: Skalierbarkeit | Szenario: Nach einem guten Marketing, sollte das System problemlos tausende neue Benutzer innerhalb einer Stunde aushalten können. |
| Priorität 5: Zuverlässigkeit | Szenario: Bei einem Netzwerkausfall sollte das System innerhalb von 2 Minuten wieder problemlos funktionieren. |

## Stakeholder

|  |  |  |
| --- | --- | --- |
| Rolle | Kontakt | Erwartungshaltung |
| *Investor* | *maryam.patel@venturecapital.xyz* | *Skalierbare Architektur* |
| *Business-Development*  *UX-Designer*  *Fahrervertreter*  Umweltaktivist | *raj.gupta@lta.com*  [*megan.chen@creativeagency.abc*](mailto:megan.chen@creativeagency.abc)  [*amirah.rahman@rideshare.drivers*](mailto:amirah.rahman@rideshare.drivers)  *javier.gomez@sustainable.xyz* | *Benutzerfreundliche Architektur*  *Flexible Architektur*  *Zuverlässige Architektur*  *Umweltfreundliche Architektur* |

# Randbedingungen

|  |  |
| --- | --- |
| Technische Randbedingungen | Kompatibilität mit iOS und Android |
| Organisatorische Randbedingungen | Einhaltung von Datenschutzrechten |
| Politische Randbedingungen | Einhaltung der Verkehrsregeln |

# Kontextabgrenzung

Passagier:

* Eingabe: Anfrage für eine Fahrt (Ort, Ziel), Buchungsanfrage, Zahlungsart
* Ausgabe: Verfügbare Fahrer/Autos, Bestätigung (der Fahrt & Zahlung), Fahrtdetails

Fahrer:

* Eingabe: Fahrtenangebot, Annahme/Ablehnung von Fahrt
* Ausgabe: Auszahlung, Buchungsanfrage, Buchungsbestätigung

*Schnittstellen zu Nachbarsystemen:*

Zahlungssystem:

* Eingabe: Zahlungsanfrage, Zahlungsart (PayPal, Bankomatkarte, Guthaben)
* Ausgabe: Zahlungsstatus, Fehlermeldungen, Guthaben

Technische Schnittstellen:

* Internetverbindung: Datenübertragung
* GPS: Standortbestimmung
* Protokolle: HTTPS/Sockets für Kommunikation & Datenübertragung
* Smartphones: Für Benutzung der App erforderlich
* Server: Datenbank & Backend-Logik

## Fachlicher Kontext

|  |  |  |
| --- | --- | --- |
| ***Kommunikationsbeziehung*** | ***Eingabe*** | ***Ausgabe*** |
| Mitfahrer / Passagier | Suchanfrage für Fahrten, Buchungsanfrage, Zahlungsdetails | Liste von verfügbaren Fahrern in der Nähe, Buchungsbestätigung, Zahlungsbestätigung, Fahrtdetails |
| Fahrer | Fahranfragen, Annahme/Ablehnung | Buchungsanfragen, Auszahlung, Buchungsbestätigung |
| Zahlungssystem | Zahlungsanfrage, Details, Zahlungsart | Zahlungsstatus, Guthaben, Fehlermeldungen |

## Technischer Kontext

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Technische Schnittstelle | Eingabe | Ausgabe | Übertragungsmedium | Protokoll |
| Internet (4G/5G/Wi-Fi) | Suchanfrage für Fahrten, Buchungsanfrage, Annahme/Ablehnung, Zahlung | Verfügbare Fahrten, Bestätigung der Buchung und Zahlung, Zahlungsstatus | Drahtlos | HTTPS |
| GPS | Standort des Passagiers/Fahrers | Standort des Fahrers für den Passagier, Standort des Passagiers für den Fahrer | Satellit | NMEA 0183 |
| Benachrichtigungs-system | Buchungsanfrage, Zahlungsbestätigung | Buchungsbestätigung, Annahme/Ablehnung | Drahtlos | FCM / APNs |

**Mapping für die technischen Schnittstellen:**

* Internet:
* Suchanfrage für eine Fahrt: App sendet Anfrage an Server und findet eine verfügbare Fahrt
* Buchungsanfrage: Auswahl einer Fahrt 🡪 Buchungsanfrage an Server
* Zahlungsanfrage: Nach einer Fahrt 🡪 Zahlungsanfrage an Zahlungsprozessor
* GPS:
* Standort Passagier: App erfasst Standort 🡪 Server (verfügbare Fahrten finden)
* Standort Fahrer: App erfasst Standort 🡪 informiert Passagier über Ankunftszeit
* Benachrichtigungssystem:
* Buchungsanfrage: Fahrer erhält Benachrichtigung
* Buchungsbestätigung: Fahrer akzeptiert eine Anfrage 🡪 Passagier erhält Bestätigung

# Lösungsstrategie

**Technologieentscheidungen**:

* Frontend: React (plattformübergreifende Mobilität)
* Backend: Node.js (Realtime Verarbeitung der Daten)
* Zahlungssystem: Stripe (Sicherheit, Benutzerfreundlichkeit)
* Datenbank: MongoDB (Skalierbarkeit)

**Top-Level Zerlegung des Systems:**

* Model-View-Controller (MVC) für Trennung von UI, Logik und Daten
* Client-Server Architektur

**Qualitätsanforderungen:**

* Sicherheit: 2-Schritt-Authentifizierung (z.B. OAuth) und JWT für Datenübertragung
* Skalierbarkeit: Cloud-basierte Datenbank
* Benutzerfreundlichkeit: Intuitives UX-Design

**Organisatorisches:**

* Agile Entwicklungsmethode: iteratives Feedback nach jedem Schritt
* Aufgabeneinteilung: Spezialisierte Teams arbeiten an verschiedenen Tasks

# Bausteinsicht

1. **Module:**

* Frontend-Modul: Beinhaltet Benutzeroberfläche, Authentifizierung, Zahlungsabwicklung, Fahrtsuche, Buchung, und Echtzeit-Tracking
* Backend-Modul: API-Server, Authentifizierungsservice, Fahrerverifizierung, Zahlungsabwicklung, Datenbank-Management

1. **Komponenten:**

* Mobile App (Frontend): Benutzer-Interface, Authentifizierungsmodul, Zahlungsmodul
* Server (Backend): RESTful API, Authentifizierungsserver, Datenbank-Server

1. **Subsysteme:**

* Authentifizierungssubsystem: OAuth 2.0 und JWT-basierte Authentifizierung
* Zahlungssubsystem: Integration mit Stripe oder anderen Zahlungsdienstleistern

1. **Klassen:**

* Spezifisch definiert für Funktionen wie Benutzerverwaltung, Fahrtsuche, Buchungsmanagement, Echtzeit-Tracking

1. **Pakete / Bibliotheken:**

* Frontend: React Native-Framework und zugehörige Bibliotheken
* Backend: Node.js, Express.js, und weitere NPM-Pakete

1. **Frameworks:**

* Frontend: React Native
* Backend: Node.js

1. **Schichten:**

* Präsentationsschicht: Mobile App Benutzeroberfläche
* Logikschicht: Serverseitige Business-Logik
* Datenschicht: MongoDB-Datenbank

1. **Partitionen / Tiers:**

* Client-Side (Frontend): Mobile Anwendung auf Nutzergeräten
* Server-Side (Backend): Server-Infrastruktur und Dienste

1. **Funktionen / Operationen:**

* Frontend: Registrierung, Login, Fahrtsuche, Buchung, Echtzeit-Tracking
* Backend: Authentifizierung, Datenverarbeitung, Zahlungsabwicklung

1. **Datenstrukturen:**

* Datenbank-Strukturen: Benutzerprofile, Fahrtdaten, Bewertungen, Transaktionen

Die Bausteinsicht in unserer Architekturdokumentation für das RideShare-System zielt darauf ab, die statische Struktur des Systems klar und verständlich zu machen. Dieser Ansatz ist wesentlich, da er:

1. Komplexität reduziert: Indem das System in Module, Komponenten und Subsysteme unterteilt wird, wird das Verständnis der verschiedenen Teile und deren Zusammenwirken erleichtert
2. Kommunikation fördert: Die abstrakte Darstellung erlaubt Teammitgliedern aus verschiedenen Bereichen, effektiv über die Architektur zu kommunizieren und ein gemeinsames Verständnis zu entwickeln
3. Entscheidungsfindung unterstützt: Die Bausteinsicht dient als Basis für architektonische Entscheidungen, hilft bei der Identifikation von Problemen und fördert informierte Entscheidungen
4. Dokumentation und Wartbarkeit verbessert: Nützliche Dokumentation für das Team und unterstützt die langfristige Wartbarkeit

## Whitebox Gesamtsystem

Begründung

Die Zerlegung des Gesamtsystems in die Hauptkomponenten Frontend, Backend, Datenbank und externe Dienste folgt dem Prinzip der Trennung von Anliegen. Jeder Baustein hat eine klar definierte Verantwortlichkeit, was die Wartbarkeit und Skalierbarkeit des Systems verbesserten. Diese Struktur unterstützt auch eine agile Entwicklungsmethode, da Teams unabhängig an verschiedenen Bausteinen arbeiten können.

Enthaltene Bausteine

* Frontend-App (Blackbox)
  + Hauptverantwortlichkeiten: Benutzeroberfläche, Client-seitige Logik, Authentifizierung, Fahrtsuche, Buchungs- und Zahlungsabwicklung.
  + Schnittstellen: RESTful API-Aufrufe zum Backend.
* Backend-Server (Blackbox)
  + Hauptverantwortlichkeiten: Verarbeitung von Geschäftslogik, Authentifizierung, Datenpersistenz, Integration von Zahlungsdiensten.
* Datenbank (Blackbox)
  + Hauptverantwortlichkeiten:Persistente Speicherung von Benutzerdaten, Fahrtendaten, Transaktionshistorie und Bewertungen.
  + Schnittstellen:Anfragen vom Backend-Server über ein Datenbankverwaltungssystem (DBMS).
* Externe Dienste
  + Zahlungsdienstleister:Abwicklung finanzieller Transaktionen.
  + Standortdienste:Bereitstellung von Echtzeit-Standortdaten für Fahrzeuge und Nutzer.

Wichtige Schnittstellen

* Frontend zu Backend RESTful API
  + Syntax/Semantik: HTTP-Anfragen mit JSON-Body für Authentifizierung, Benutzeraktionen wie Fahrtsuche und Buchung.
  + Protokolle: HTTPS für sichere Datenübertragung.
  + Fehlerverhalten: Angemessene HTTP-Statuscodes für Fehlerzustände (z.B. 400er-Serie für Client-Fehler, 500er-Serie für Server-Fehler).
* Backend zu Datenbank
  + Syntax/Semantik: CRUD-Operationen über eine ORM-Schicht oder direkte Datenbankabfragen
  + Protokolle: Datenbankspezifische Protokolle, häufig TCP/IP-basiert
  + Fehlerverhalten: Transaktionsmanagement mit Rollback bei Fehlern, Wiederversuche bei temporären Ausfällen
* Backend zu Externen Diensten
  + Syntax/Semantik: API-Schlüssel und Token-basierte Authentifizierung, spezifische Anforderungsformate je nach Dienst
* Protokolle: Oftmals RESTful APIs oder SOAP für Web Services, Verwendung von Dienst-spezifischen SDKs
* Fehlerverhalten: Graceful Degradation bei Ausfall externer Dienste, Einsatz von Circuit Breakern zur Vermeidung von Kaskadenfehlern

|  |  |
| --- | --- |
| Name | Verantwortung |
| *Frontend-App* | *Die Frontend-App bietet die Benutzeroberfläche für das RideShare-System. Sie verantwortet die Benutzerinteraktion, darunter Authentifizierung, Fahrtsuche, Buchung und Zahlungsabwicklung. Sie kommuniziert mit dem Backend-Server, um Dienste zu nutzen und Daten auszutauschen.* |
| *Backend-Server* | *Der Backend-Server ist das zentrale Nervensystem des RideShare-Systems. Er verarbeitet Geschäftslogik, Authentifizierungsanfragen, Datenpersistenz und Integration externer Dienste wie Zahlungsdienstleister. Er stellt APIs bereit, die vom Frontend für sämtliche operationelle Aktionen aufgerufen werden.* |

**Frontend-App**

Zweck/Verantwortung

Die Frontend-App dient als primärer Kontaktpunkt für den Benutzer. Sie ermöglicht es dem Benutzer, sich zu registrieren, anzumelden, Fahrten zu suchen und zu buchen sowie Zahlungen vorzunehmen. Sie stellt eine intuitive und responsive Benutzeroberfläche bereit, die auf verschiedenen Geräten und Plattformen funktioniert.

Schnittstelle(n)

* Restful API für die Kommunikation mit dem Backend-Server.
* OAuth für Authentifizierung und Autorisierung.
* Stripe API für Zahlungsabwicklungen.

(Optional) Qualitäts-/Leistungsmerkmale

* Verfügbarkeit**:** Entwickelt für hohe Verfügbarkeit und geringe Ausfallzeiten.
* Performance**:** Schnelle Antwortzeiten, mit dem Ziel, Benutzeranfragen innerhalb von 3 Sekunden zu beantworten.
* Benutzerfreundlichkeit**:** Einfache und intuitive Navigation und Bedienbarkeit.

(Optional) Ablageort/Datei(en)

* Der Quellcode befindet sich in einem Versionierungssystem wie Git, gehostet auf Plattformen wie GitHub oder GitLab.
* Assets und Ressourcen werden in entsprechenden Cloud-Speicherdiensten abgelegt.

(Optional) Erfüllte Anforderungen

* Entspricht den funktionalen und nicht-funktionalen Anforderungen wie in den Projektunterlagen und dem Lastenheft definiert.
* Erfüllt die Sicherheitsanforderungen für die Authentifizierung und Datenübertragung.
* Erfüllt die Zugänglichkeitsstandards für mobile Anwendungen.

(Optional) Offene Punkte/Probleme/Risiken

* Anpassungen an die sich schnell ändernden Größen und Auflösungen von Handys / mobilen Geräten
* Abhängigkeit von Drittanbieter-APIs kann zu unerwarteten Ausfallzeiten führen
* Risiken im Zusammenhang mit der Skalierung der Infrastruktur bei steigender Benutzerzahl

**Backend-Server**

Zweck/Verantwortung

Der Backend-Server verwaltet die zentrale Logik und Datenverarbeitung des RideShare-Systems. Er bearbeitet Anfragen von der Frontend-App, führt Authentifizierungsprozesse durch, interagiert mit der Datenbank zur Datenspeicherung und handhabt die Integration von externen Zahlungssystemen.

Schnittstelle(n)

* Bietet RESTful APIs für die Frontend-App zur Durchführung von Operationen wie Benutzerauthentifizierung, Fahrtensuche, Buchung und Zahlungsabwicklung
* Datenbankschnittstelle für CRUD-Operationen
* Integrationsschnittstellen für externe Dienste wie Zahlungsabwickler (Stripe, PayPal etc.)

(Optional) Qualitäts-/Leistungsmerkmale

* Sicherheit**:** Implementiert Sicherheitsmaßnahmen wie SSL/TLS-Verschlüsselung, mehrstufige Authentifizierung und regelmäßige Sicherheitsaudits
* Skalierbarkeit**:** Fähigkeit, Lastspitzen durch Skalierung zu bewältigen und tausende gleichzeitige Benutzeranfragen zu unterstützen
* Zuverlässigkeit**:** Gewährleistung einer hohen Verfügbarkeit und schnellen Wiederherstellung bei Ausfällen durch Backup- und Recovery-Strategien

(Optional) Ablageort/Datei(en)

* Serverseitiger Code befindet sich in einem gesicherten Repository, zugänglich nur für berechtigte Developer-Teams
* Konfigurationsdateien und Skripte für die Deployment-Pipeline sind ebenfalls in Versionskontrollsystemen gespeichert

(Optional) Erfüllte Anforderungen

* Kompatibilität mit gängigen Webstandards und Sicherheitsprotokollen
* Erfüllung der Performance-Kriterien gemäß Service-Level-Agreements (SLAs)
* Implementierung von APIs gemäß den OpenAPI-Spezifikationen für eine klare und konsistente Schnittstellenbeschreibung

(Optional) Offene Punkte/Probleme/Risiken

* Abhängigkeit von der Verfügbarkeit und Stabilität externer APIs kann zu einer Beeinträchtigung der Dienstleistungen führen
* Potenzielle Leistungseinbußen bei nicht optimiertem Datenbankdesign und schlecht verwalteten Ressourcen
* Risiko von Datenverlust oder -korruption bei unzureichender Transaktionssteuerung und fehlerhafter Datenhandhabung

## Ebene 2

Beschreiben Sie den inneren Aufbau (einiger) Bausteine aus Ebene 1 als Whitebox.

### Whitebox *Frontend-App*

### Ein Bild, das Text, Diagramm, Screenshot, Reihe enthält. Automatisch generierte Beschreibung

### Whitebox *Backend Server*

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

## Ebene 3

Beschreiben Sie den inneren Aufbau (einiger) Bausteine aus Ebene 2 als Whitebox.

**Whitebox Baustein 1: Benutzerauthentifizierung**

|  |
| --- |
| * Benutzermeldung * Registrierung * OAuth Token Austausch |

**Whitebox Baustein 2: Fahrtensuche und Buchung**

|  |
| --- |
| * Suchanfragen bearbeiten * Fahrt buchen * Zahlungsabwicklung |

**Whitebox Baustein 3: Sicherheitsmechanismen**

|  |
| --- |
| * SSL/TLS Verschlüsserung * Mehrstufige Authentifizierung |

# Laufzeitsicht

## *Authentifizierung und Nutzermeldung*

* **Ablaufbeschreibung:** 
  + Der Nutzer öffnet die Anwendungsseite und gibt Benutzername und Passwort ein
  + Das Anwendungs-Frontend übermittelt die Anmeldedaten an den Authentifizierungsdienst
  + Der Authentifizierungsdienst überprüft die Anmeldedaten gegenüber der Benutzerdatenbank
  + Bei erfolgreicher Authentifizierung erstellt der Authentifizierungsdienst ein Token und sendet es an das Anwendungs-Frontend
  + Das Anwendungs-Frontend speichert den Token und ermöglicht dem Nutzer Zugriff auf geschützte Ressourcen
* **Besonderheiten:**

## Der Authentifizierungsdienst kommuniziert direkt mit der Benutzerdatenbank, um die Anmeldedaten zu überprüfen

## Der Token basiert auf einem sicheren Authentifizierungsprotokoll (z.B. OAuth 2.0)

## Das Anwendungs-Frontend muss sicherstellen, dass das Token sicher gespeichert und bei jeder Anfrage an geschützte Ressourcen übermittelt wird

## *Fahrtenbuchung in der Ride-Sharing-App*

* **Ablaufbeschreibung:**
  + Der Nutzer öffnet die Ride-Sharing-App und gibt den gewünschten Abhol- und Zielort ein
  + Das App-Frontend sendet die Anfrage an den Fahrtenvermittlungsdienst
  + Der Fahrtenvermittlungsdienst analysiert die Anfrage und sucht verfügbare Fahrer in der Nähe des Abholorts
  + Der Fahrtenvermittlungsdienst wählt einen geeigneten Fahrer aus und sendet die Anfrage zur Fahrtannahme
  + Der ausgewählte Fahrer erhält die Anfrage und kann sie akzeptieren oder ablehnen
  + Bei Annahme informiert das App-Frontend den Nutzer über den akzeptierten Fahrer und die voraussichtliche Ankunftszeit
  + Der Fahrer folgt der Navigation zum Abholort und holt den Nutzer ab
  + Die App verfolgt die Fahrt in Echtzeit und zeigt dem Nutzer den Fortschritt auf der Karte an
  + Nach Abschluss der Fahrt erfolgt die Zahlung über die App und beide, Fahrer und Nutzer, haben die Möglichkeit, sich zu bewerten
* **Besonderheiten:**
  + Der Fahrtenvermittlungsdienst muss effizient Fahrer in der Nähe finden und berücksichtigt dabei verschiedene Faktoren wie Verfügbarkeit, Entfernung und aktuelle Verkehrslage
  + Die Echtzeitverfolgung der Fahrt erfordert eine zuverlässige Kommunikation zwischen App-Frontend, dem Fahrer und dem Fahrtenvermittlungsdienst
  + Die Zahlungsabwicklung sollte sicher und transparent sein, unter Verwendung von sicheren Zahlungsmethoden

# Verteilungssicht

*Ein Bild, das Text, Screenshot, Diagramm, Rechteck enthält.

Automatisch generierte Beschreibung*

Begründung

Die Verteilung des Gesamtsystems auf verschiedene Umgebungen und Rechner erfolgt, um eine hohe Verfügbarkeit, Skalierbarkeit und Ausfallsicherheit sicherzustellen. Zudem ermöglicht die gezielte Verteilung die optimale Nutzung von Ressourcen und eine verbesserte Lastenverteilung.

Qualitäts- und/oder Leistungsmerkmale

* Skalierbarkeit: Die Infrastruktur muss dazu in der Lage sein, mit steigender Nutzerzahl zu skalieren.
* Verfügbarkeit: Die Architektur strebt eine hohe Verfügbarkeit an, um Unterbrechungen und Ausfälle zu minimieren
* Sicherheit: Die Verteilung berücksichtigt Sicherheitsaspekte, um sensible Daten angemessen zu schützen
* Leistung: Durch die Verteilung werden eine optimale Leistung und Antwortzeit angestrebt

Zuordnung von Bausteinen zu Infrastruktur

* Frontend-Server: Die Komponenten des Frontend-Servers werden auf den Webservern verteilt, um die Benutzerschnittstelle effizient bereitzustellen
* Backend-Services: Die Backend-Services laufen auf separaten Anwendungsservern, um die Geschäftslogik zu verarbeiten
* Datenbank: Die Datenbank wird auf einem dedizierten Datenbankserver betrieben, um Datenpersistenz zu gewährleisten
* Load Balancer: Zur Optimierung der Lastenverteilung werden Load Balancer in der Verteilungsstruktur eingesetzt
* Authentifizierungsdienst: Der Authentifizierungsdienst wird auf verschiedenen Servern verteilt, um zu jeder Zeit Verfügbarkeit sicherzustellen

## Infrastruktur Ebene 2

An dieser Stelle können Sie den inneren Aufbau (einiger) Infrastrukturelemente aus Ebene 1 beschreiben.

Für jedes Infrastrukturelement kopieren Sie die Struktur aus Ebene 1.

**Webserver**

### Der Webserver bildet eine zentrale Komponente auf der Ebene 2 der Infrastruktur. Er ist für die Bereitstellung des Frontend-Clients verantwortlich. Mehrere Instanzen des Webservers sind auf verschiedenen Rechnern in der Umgebung verteilt, um eine gleichmäßige Lastenverteilung zu gewährleisten. Ein Load Balancer leitet die Anfragen an die verfügbaren Webserver weiter.

**Anwendungsserver**

Die Anwendungsserver stellen die Plattform für die Ausführung der Backend-Services bereit. Sie sind auf dedizierten Rechnern verteilt, um die Verarbeitung der Geschäftslogik zu optimieren. Die Anwendungsserver kommunizieren sowohl mit den Frontend-Webservern als auch mit der Datenbank, um die notwendigen Informationen zu erhalten bzw. zu persistieren.

# 

# Querschnittliche Konzepte

**Konzepte und Lösungsansätze:**

* 1. ***Fachliche Modelle*:** Verwendung eines Domain-Driven Designs für eine bessere Strukturierung und Modellierung der Geschäftslogik
  2. ***Sicherheitskonzepte*:** Implementierung von 2-Step Authentifizierung (z.B. OAuth 2.0) für eine sichere Authentifizierung und Autorisierung des Benutzers.
  3. ***Betriebskonzepte***: Einsatz von DevOps Praktiken 🡪 Kontinuierliche Integration und Bereitstellung von neuen Features oder Bugfixes
  4. ***Entwicklungskonzepte***: Agile Entwicklungsmethoden 🡪 Flexible und Iterative Produktentwicklung
  5. ***Architekturmuster***: Einsatz von Microservices für die Skalierbarkeit und unabhängige Entwicklung verschiedener Bestandteile der App

# Entwurfsentscheidungen

**Liste nach Wichtigkeit sortiert:**

* ***React Native für Frontend Entwicklung*:** Plattformübergreifende Entwicklung möglich mit einer Codebasis
* ***Node.js im Backend*:** Echtzeit-Datenverarbeitung und Skalierbarkeit
* ***MongoDB als Datenbank*:** Flexibilität und Skalierbarkeit
* ***Cloud-basierter Ansatz*:** Zuverlässigkeit und Skalierbarkeit

# Qualitätsanforderungen

Neben den bereits in Abschnitt 1.2. genannten Qualitätszielen wären folgende Anforderungen ebenfalls von großer Bedeutung:

* Wartbarkeit des Codes
* Barrierefreiheit der App für alle möglichen Benutzer
* Multilinguale App 🡪 Unterstützung mehrerer Sprachen

## Qualitätsbaum

Qualität ist bei uns aufgeteilt in:

## ├ Performance

## │ ├ Schnelle Antwortzeit bei der Suche einer Fahrt

## │ └── Effiziente Datenverarbeitung

## ├── Sicherheit

## │ ├ Datenschutz

## │ ├ Mehrstufige Authentifizierung durch OAuth

## │ └── Sichere Zahlungsabwicklung

## ├ Benutzerfreundlichkeit

## │ ├ Intuitive Benutzeroberfläche

## │ ├ Einfacher Registrierungsprozess

## │ └── Leichte Auffindbarkeit von Funktionen

## ├ Zuverlässigkeit

## │ ├ hohe Verfügbarkeit

## │ └── Schnelle Wiederherstellung bei Ausfällen

## └── Skalierbarkeit

## └── Keine Ausfälle bei hoher Auslastung durch neue Nutzer

## Qualitätsszenarien

Deep-Dive in die Konkretisierung der in 1.2. genannten Punkte:

* + 1. ***Performance***: Innerhalb von 3 Sekunden sollte das System dazu in der Lage sein eine Anfrage nicht nur zu bearbeiten, sondern diese auch dem jeweiligen Benutzer, ob Fahrer oder Mitfahrer, anzuzeigen
    2. ***Zuverlässigkeit***: Bei einem Ausfall des Systems sollte innerhalb von 2 Minuten ein Backup-System die Kontrolle übernehmen
    3. ***Sicherheit***: Durch die mehrstufigen Authentifizierungsmaßnahmen sollte unberechtigter Zugriff auf Nutzerdaten verhindert werden können
    4. ***Benutzerfreundlichkeit***: Innerhalb von 10 Minuten nach dem Download sollte ein neuer Benutzer bereits dazu in der Lage sein, wesentlichste Funktionen der App zu nutzen
    5. ***Skalierbarkeit***: Unser System sollte in der Lage sein, innerhalb einer Stunde eine plötzliche Zunahme von 1000% zu bewältigen (z.B. nach einem Marketing), ohne dass es zu einem Systemausfall kommt

# Risiken und technische Schulden

Nach Prioritäten geordnete Liste:

* ***Konkurrenz***: Es besteht das Risiko, dass der Markt bereits gesättigt ist und man nur deshalb nur schwer an Kunden kommt, da diese bereits eine der „Big-Player“ Apps benutzen könnten
* ***Externe API-Probleme***: Durch die Benutzung externer Dienste könnte unsere App Probleme haben, wenn die externen Dienste nicht Verfügbar sind
* ***Datenschutz***: Risiko eines Datenlecks, besonders personenbezogene Daten oder Zahlungsinformationen
* ***Skalierungsprobleme***: Technische Schulden durch Entwurfsentscheidungen am Anfang, die eine Skalierung erschweren könnten
* ***Rechtliche Anforderungen***: Einhaltung der lokalen Verkehrs- und Datenschutzgesetze

# Glossar

Begriff Definition

|  |  |
| --- | --- |
| Microservices | Eine Architektur die die Anwendung in mehrere kleineren, unabhängigen Dienste aufteilt |
| OAuth 2.0 | Ein Standardprotokoll welches Benutzerauthentifizierung ermöglicht und dem Client Aktionen ermöglicht, ohne die Anmeldedaten des Nutzers zu teilen |
| JWT (JSON Web Token) | Ein Token-Format für geschützte Übertragung von Informationen |
| DevOps | Praktiken, die die Softwareentwicklung und Betriebsprozesse in der IT vereinen |
| React Native | Framework welches zur plattformübergreifenden Entwicklung von mobilen Apps benutzt wird |
| Node.js | Serverseitige Plattform welche zur Ausführung von JavaScript benutzt wird |
| MongoDB | NoSQL-Datenbank welche ideal für die Verarbeitung großer Datenmengen gedacht ist |
| Cloud Computing | Die Bereitstellung von IT-Infrastruktur und Diensten über das Internet |