Ressourcen

Funktionalität	HTTP Verb	URI	Format
ein Benutzerprofil erstellen	post	`/user`	JSON
ein spezifisches Benutzerprofil abrufen	get	`/user/:user_id`	JSON
ein spezifisches Benutzerprofil löschen	delete	`/user/:user_id`	JSON
ein spezifisches Benutzerprofil updaten	put	`/user/:user_id`	JSON
eine Bewertung für einen Benutzer erstellen	post	`/user/:user_id/rating`	JSON
alle Bewertungen eines Benutzers abrufen	get	`/user/:user_id/rating`	JSON
eine spezifische Bewertung löschen	delete	`/user/:user_id/rating/:rating_id`	JSON
eine spezifische Bewertung aktualisieren	put	`/user/:user_id/rating/:rating_id`	JSON
eine neue Fahrt erstellen	post	`/fahrten`	JSON
alle Fahrten eines Benutzers abrufen	get	`/fahrten?user_id=int`	JSON
eine spezifische Fahrt updaten	put	`/fahrten/:fahrt_id`	JSON
eine spezifische Fahrt löschen	delete	`/fahrten/:fahrt_id`	JSON
einen neuen Chatraum erzeugen	post	/chats/	JSON
alle Chats an denen ein Benutzer teilnimmt abrufen	get	/chats?user_id=int	JSON
eine neue Nachricht in einem Chatraum erzeugen	post	/chats/:chat_id	JSON
alle Nachrichten eines	get	/chats/:chat_id	JSON

spezifischen Chats abrufen			
einen spezifischen Chat löschen	delete	/chats/:chat_id	JSON
eine spezifische Chatnachricht ändern	put	/chats/chat_id/:message_id	JSON
eine spezifische Chatnachricht löschen	delete	/chats/chat_id/:message_id	JSON
Erhalte die nächsten 10 Fahrten die von Start zum Zielbahnhof statfinden	get	`/trips?departure=string&target=string&date=string&time=string&ticket=boolean&uid=int`	JSON
Erhalte alle den Parametern entsprechenden Matches	get	`/matches?hasSeasonTicket=boolean&utid=int&dsid=int&tsid=int&uid=int`	JSON

Anwendungslogik

Client

Umkreissuche

#aktuelle Position ermitteln

#area	-der Radius in dem Haltestellen ermittelt werden sollen
#maxNumberStations	-die Maximale Anzahl an Haltestellen die ermittelt werden soll
#stations[Name, lat, lon]	-ein Array das alle Bahnhöfe inklusive Koordinaten enthält

```
#Ermittelt die nächsten X Station in einem spezifischem Radius
findStationsInArea(area, maxNumberStations, stations)
        #eigene Position auslesen
        lon = position.lon
        lat = position.lat

        #Position in x,y,z, Koordinaten ändern
        position = changeKoordinates(lon,lat)

        stationsInArea[Name][distance]
        foreach station in stations
                stationPosition = changeKoordinates(station.lon,station.lat)
                distance= sqroot((position.x-stationPosition.x)^2+(position.y-
                        stationPosition.y)^2+(position.z-stationPosition.z)^2)
                if(distance > radius)
                        stationsInArea.add[station.Name, distance]

        stationsInArea.sort(distance)

        while(stationsInArea.length > maxNumberStations
                stationsInArea.remove(last)
```

```
#Variablen
#lon                    -Der Längengrad
#lat                    -Der Breitengrad

#Ermittelt die zu Längengrad und Breitengrad passenden Koordinaten in einem dreidimensionalem Koordinatensystem
changeKoordinates(lon, lat)
        lon = position.lon
        lat = position.lat



        #Gradmaß ins Bogenmaß umrechnen
        lambda = lon*pi/180
        phi = lat*pi/180

        #Umwandlung in ein dreidimensionales Koordinatensystem
        xPosition = 6371*cos(phi)*cos(lambda)
        yPosition = 6371*cos(phi)*sin(lambda)
        zPosition = 6371*sin(phi)

        return koordinates [xPosition][yPosition][zPosition]
```

Autocomplete

Variablen:

#stops[] -Ein Array das alle Bahnhofsnamen enthält

#stopsFiltered[] -Ein Array welches die gefilterten Bahnhofsnamen enthält

#versetzt das gefilterte Array in einen ungefilterten Zustand, wird beim Anklicken des zu filtenden Textfeldes und beim Löschen von Buchstaben aufgerufen

```
reset(stops, stopsFiltered)
    stopsFiltered = stops
```

#wird bei jeder Inhaltsänderung des Textfeldes aufgerufen

```
filter(stops, stopsFiltered)
    #der Inhalt des Textfeldes
    content = textField.content
    #nicht passende Bahnhofsnamen werden entfernt
    foreach stop in stopsFiltered
        if(!(stop enthält Buchstabenfolge aus content))
            stopsFiltered.remove(stop)
```

Server

Trips

#Variablen:

#departureStationName	-ID des Startbahnhofs
#targetStationName	-ID des Zielbahnhofs
#departureTime	-Abfahrtszeit
#departureDate	-Abfahrtsdatum
#hasSeasonticket	-Boolean der Auskunft über den Ticketstatus des
#userID	-Die eindeutige ID des Benutzers

#Ermittelt die nächsten Zehn Fahrten von Bahnhof a nach Bahnhof b ab einer spezifischen Zeit und einem spezifischem Datum

findTrips(departureStationName, targetStationName, departureTime, departureDate, hasSeasonticket, userID)

departureStationID = stops-Datenbankabfrage(departureStationName)

targetStationID = stops-Datenbankabfrage(targetStationName)

#alle Fahrten die an einem Bahnhof halten

departureTrips[tripID][departureTime][sequenceID] = stop_times-DBabfrage(departureStationID)

targetTrips[tripID][arrivalTime][sequenceID] = stop_times-DBabfrage(targetStationID)

#alle Fahrten die an beiden Bahnhöfen halten

trips[tripID][departureTime][arrivalTime][sequenceIDDepartureStation][sequenceIDTargetStation][serviceID]

foreach departuteTrip in departureTrips

foreach targetTrip in targetTrips

if(departureTrip.ID = targetTrip.ID)

trips.add(departureTrip.tripID, departureTrip.departureTime, targetTrip.arrivalTime, departureTrip.sequenceID, targetTrip.sequenceID, trips-DBabfrage(departureTrip.tripID)

```
#hier werden die nächsten 10 Fahrten von Startbahnhof zum Zielbahnhof gespeichert
uniqueTrips[uniqueTripID][tripID][departureTime][arrivalTime][sequenceIDDeparture
Station][sequenceIDTargetStation][departureDate][numberMatches]

#es gibt keine Linie die an beiden Bahnhöfen hält
if(trips.length<=0)
        Error(Es gibt keine Linie, welche die beiden Bahnhöfe miteinander verbindet)
else
        trips.sort(arrivalTime)
        #Fahrten die am selben Tag stattfinden
        foreach trip in trips
                if(trip.departureTime >= departureTime &
                calendar-Datenbankabfrage(trip.serviceID) am Wochentag von
                departureDate = 1) &
                calendar_dates-Datenbankabfrage(trip.serviceID) an
                departureDate = 1 | null)
                        if(trip.departureTime >= 24:00:00 )
                                #erstellen einer Einzigartigen ID, Zug ist am Vortag losgefahren
                                uniqueTripID= trip.tripID + (departureDate-1)
                        else
                                #erstellen einer Einzigartigen ID, Zug ist am selben Tag losgefahren
                                uniqueTripID= trip.tripID + departureDate

                        uniqueTrips.add(uniqueTripID, trip.tripID,
                        trip.departureTime, trip.arrivalTime,
                        trip.sequenceIDDepartureStation,
                        trip.sequenceIDTargetStation, trip.departureDate, null)

        #suche weitere Fahrten bis insgesamt 10 gefunden wurden
        currentDepartureDate = departureDate
        while(uniqueTrips.length <= 10)
                departureDate++;
```

```
        #keine Fahrten die weiter als eine Woche vom Startdatum
        entfernt sind
        if(currenDepartureDate <= departureDate+7)
                break
        else
            foreach trip in trips
                if(calendar-Datenbankabfrage(trip.serviceID) am
                Wochentag von departureDate = 1) &
                calendar_dates-Datenbankabfrage(trip.serviceID) an
                departureDate = 1 | null))
                        if(trip.departureTime >= 24:00:00 )
                                #erstellen einer Einzigartigen ID, Zug ist am
                                Vortag losgefahren
                                uniqueTripID= trip.tripID +
                                (currentDepartureDate-1)
                        else
                                #erstellen einer Einzigartigen ID, Zug ist am
                                selben Tag losgefahren
                                uniqueTripID= trip.tripID +
                                currentDepartureDate

                uniqueTrips.add(uniqueTripID, trip.tripID,
                trip.departureTime, trip.arrivalTime,
                trip.sequenceIDDepartureStation,
                trip.sequenceIDTargetStation, trip.currentDepartureDate,
                null)

if(hasSeasonticket)
        foreach uniqueTrip in uniqueTrips
                uniqueTrip.numberMatches =
                searches(trip.sequenceIDDepartureStation,
                trip.sequenceIDTargetStation, trip.tripID, trip.uniqueTripID,
                userID)
else
        foreach uniqueTrip in uniqueTrips
                uniqueTrip.numberMatches =
                offers(trip.sequenceIDDepartureStation,
```

```
                    trip.sequenceIDTargetStation, trip.tripID, trip.uniqueTripID,
                    userID)
return uniqueTrips
```

Erweiterte Matching

#Variablen:

#sequenceIDDepartureStation	-Sequence-ID des Startbahnhofs
#sequenceIDTargetStation	-Sequence-ID des Zielbahnhofes
#uniqueTripID	-Abfahrtsdatum
#userID	-Eindeutige ID des Benutzers
#hasSeasonticket	-Boolean der Angibt ob der Benutzer ein Dauerticket besitzt

#Ermittelt alle Matches zu einer bestimmten Strecke

findMatch(sequenceIDDepartureStation, sequenceIDTargetStation, uniqueTripID, userID)

matches[] = dt_trips-DBabfrage(uniqueTripID)

#überprüfung ob der Dauerticketbesitzer die gesamte Strecke seines Mitfahrers im Zug mitfährt, nicht selber die Fahrt initiiert hat oder die Fahrt noch nicht gematcht wurde

#Dauerticket vorhanden

if(hasSeasonTicket)

foreach match in matches

if (sequenceIDDepartureStation > match.sequenceIDDepartureStation & sequenceIDTargetStation < match.sequenceIDTargetStation & match.userID != userID & match.partner.userID = null)

matches.remove(match)

#kein Dauerticket vorhanden

if(!hasSeasonTicket)

foreach match in matches

if (sequenceIDDepartureStation > match.sequenceIDDepartureStation & sequenceIDTargetStation < match.sequenceIDTargetStation & match.userID != userID & match.partner.userID = null)

matches.remove(match)

return matches

Bewerten

#Variablen:

#ratingID -Die eindeutige ID des Ratings

#stars -Die Anzahl an Sternen die für die Bewertung vergeben wurden

#comment -Das Kommentar der Bewertung

#userID -Die eindeutige ID des Benutzers der bewertet wurde

#wird aufgerufen um eine Bewertung in die Datenabank einzutragen und die durchschnittliche Bewertung des bewerteten Benutzers anzupassen

```
rateUser(ratingID, stars, comment, userID)
    rating-DB.add(ratingID, userID, stars, comment)

    ratings[] = rating-DBabfrage(userID)

    sumStars
    foreach rating in ratings
        sumStars += rating.stars

    averageRating = sumStars/ratings.length

    rating-DB.update(userID, averageRating)
```

Suchagenten

#Variablen:

#sequenceIDepartureStation	-Sequence-ID des Startbahnhofs
#sequenceIDTargetStation	-Sequence-ID des Zielbahnhofes
#uniqueTripID	-Abfahrtsdatum
#userID	-Eindeutige ID des Benutzers der sich eingetragen hat
#hasSeasonticket	-Boolean der Angibt ob der Benutzer ein Dauerticket besitzt

#wird Aufgerufen wenn sich ein Benutzer für einen Trip in die Datenbank einträgt

noticeMatchesOffers

noticeMatches((sequenceIDepartureStation, sequenceIDTargetStation, uniqueTripID, userID)

```
matches[] = findMatch(sequenceIDepartureStation, sequenceIDTargetStation, uniqueTripID, userID)

foreach match in matches
    ping(match.userID,"Es wurde ein neuer Match gefunden")
```