



CARA Dokumentation

1. Projektübersicht

Projektname: CARA – Caretaker Assignment & Resource Allocation

Zweck der Anwendung:

Die Anwendung dient der **Optimierung und Effizienzsteigerung bei der Zuweisung von Betreuungspersonen (Assistenzkräften) zu Kindern**. Jeder zu betreuende Fall verfügt über einen von staatlicher Seite festgelegten Qualifikationsbedarf der Assistenzkraft. Es existieren drei Qualifikationsstufen:

- **ReKo** (Regelkompetenz)
- **QHK** (Qualifizierte Hilfeleistungskraft)
- **FK** (Fachkraft)

Hauptfunktionen:

- **Import von Daten** (Kinder und Betreuungspersonen) über **CSV-Dateien** oder manuell über eine **benutzerfreundliche Web-Oberfläche**.
- **Verwaltung der Daten** (Hinzufügen, Bearbeiten und Löschen von Kindern und Betreuungspersonen direkt in der Anwendung).
- **Ermittlung von Reisezeiten** zwischen Betreuungspersonen und Kindern mithilfe der **Google Distance Matrix API**, inklusive **Adressvalidierung bis auf Gebäudeebene**.
- **Erstellung optimaler Zuordnungen** (Matching) zwischen Betreuungspersonen und Kindern basierend auf gewichteten Kriterien:
 - **Reisezeit**
 - **Qualifikation**
 - **Über- und Unterstunden** der Betreuungspersonen
- **Manuelle und automatische Paarbildung:**

- **Manuelle Paarbildung:** Möglichkeit, einzelne Betreuungsverhältnisse zu erstellen und diese auf Machbarkeit prüfen zu lassen.
- **Automatische Paarbildung:** Einsatz von **Linearer Optimierung (AMPL)** über eine spezielle Logik, die verschiedene Parameter berücksichtigt und eine optimale Gesamtzuordnung erstellt.
- **Flexible Auswahlfunktion:** Nutzer können gezielt auswählen, welche Kinder und Betreuungspersonen in die automatische Zuordnung einbezogen werden sollen (z. B. über Checkboxes in einer Tabelle).

Technologie-Stack:

- **Infrastruktur:** Docker-Netzwerk mit separaten Containern
 - **Frontend:** React (TypeScript), DaisyUI, Tailwind CSS
 - **Backend:** Python (FastAPI)
 - **Optimierung:** AMPL (über Python-Integration, erreichbar via FastAPI-Container)
- **Verwendete Sprachen:** TypeScript, Python, AMPL
- **Frameworks & Tools:** FastAPI, React, Google Distance Matrix API

2. Systemarchitektur

Überblick:

Die Anwendung basiert auf einer **containerisierten Web-Architektur**, die in einem **Docker-Netzwerk** ausgeführt wird. Sie folgt dem **MVC-Muster (Model-View-Controller)**, wobei:

- **Frontend (View)** die Benutzeroberfläche bereitstellt,
- **Backend (Controller/Model)** die Geschäftslogik und Datenverwaltung übernimmt,
- **Optimierungsmodul (AMPL)** als spezialisiertes Berechnungsmodul dient.

Hauptkomponenten:

1. **Frontend-Container (React, TypeScript, DaisyUI, Tailwind CSS)**
 - Präsentiert die Benutzeroberfläche (UI)

- Kommuniziert ausschließlich mit dem Backend über interne API-Endpoints
- Läuft hinter einem **NGINX-Reverse-Proxy**, der Anfragen an das Backend weiterleitet

2. Backend-Container (Python FastAPI)

- Enthält die Geschäftslogik und Controller
- Verantwortlich für Datenvalidierung, Verarbeitung der Nutzeranfragen und Koordination mit dem Optimierungsmodul
- Ist **nur innerhalb des Docker-Netzwerks erreichbar** (nicht direkt von außen)

3. Optimierungs-Container (AMPL + FastAPI-Schnittstelle)

- Führt **lineare Optimierungen** durch, um optimale Zuweisungen zu berechnen
- Kann nur vom **Backend-Container** aus aufgerufen werden

4. Datenbank (z. B. PostgreSQL)

- Speichert alle relevanten Daten zu Kindern, Betreuungspersonen und Zuordnungen
- Zugriff erfolgt ausschließlich über den Backend-Container

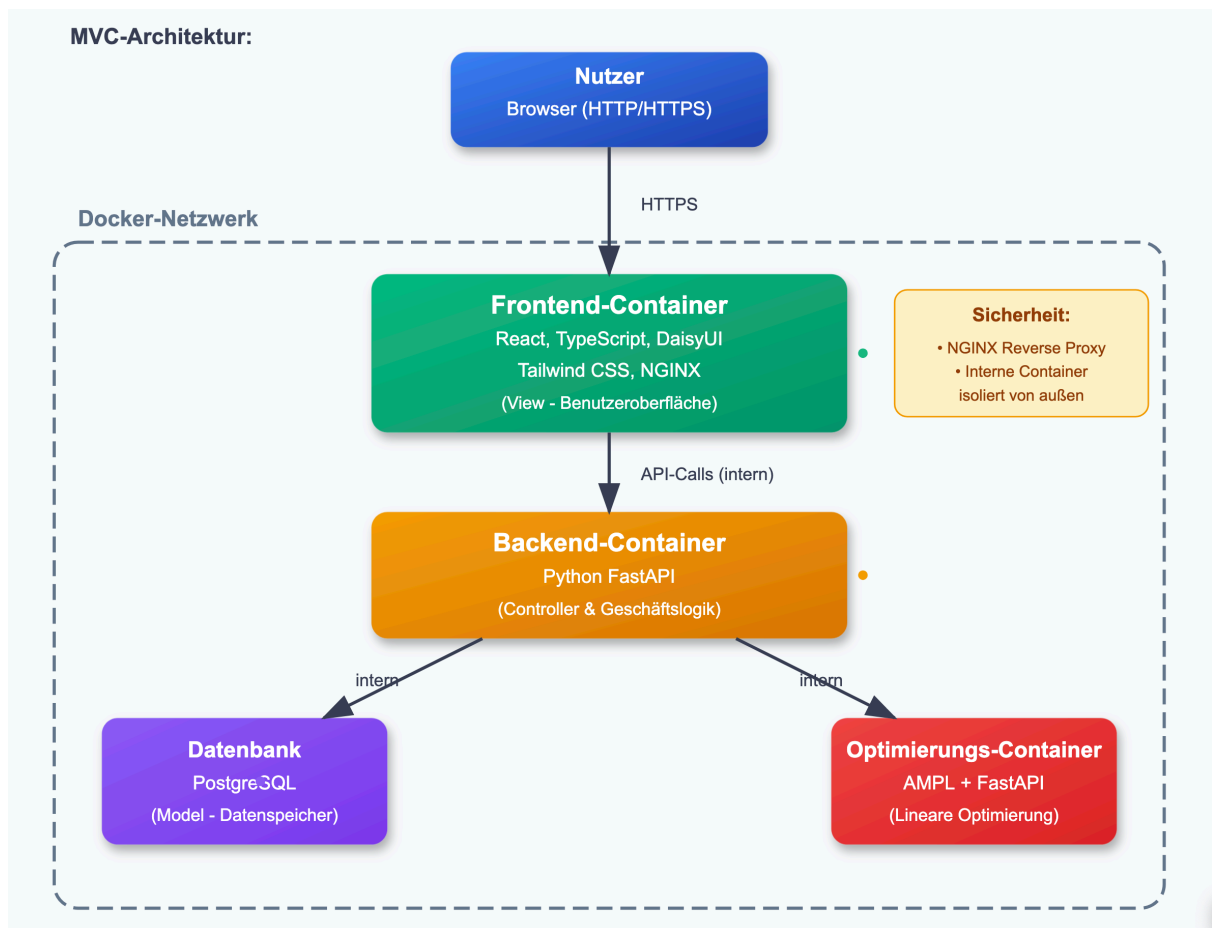
5. Hilfs-Container (z. B. Adminer)

- Dient der Verwaltung und dem Debugging der Datenbank
- Ebenfalls nur innerhalb des Docker-Netzwerks zugänglich

6. Sicherheitskonzept:

- **Reverse Proxy (NGINX):** Externe Clients können ausschließlich über das Frontend auf die Anwendung zugreifen
- **Interne Container-Kommunikation:** Backend und Optimierungsmodul sind nach außen hin isoliert

Diagramm:



Datenbank Diagramm



2.1 Google API Anforderungen

🔑 Erforderlich: Google API-Setup

Bevor Sie neue Kinder oder Betreuer importieren können, **benötigen Sie zwingend** einen gültigen Google API-Schlüssel mit diesen aktivierten Diensten:

Erforderliche Google APIs:

1. **Distance Matrix API** - Zur Berechnung von Reisezeiten zwischen Adressen
2. **Address Validation API** - Zur Überprüfung, dass Adressen real und korrekt sind

Kostenlose Kontingente:

- **Address Validation API:** 5.000 Aufrufe pro Monat (kostenlose Stufe)
- **Distance Matrix API:** 5.000 Aufrufe pro Monat (kostenlose Stufe)
- **Batch-Verarbeitung:** Jeder Distance Matrix-Aufruf kann bis zu 100 Ziele verarbeiten, wodurch Massenimporte kosteneffizient werden

⚠ **Wichtig:** Ohne einen gültigen API-Schlüssel oder wenn diese Dienste nicht aktiviert sind, **können Sie keine** neuen Kinder oder Betreuer in das System importieren.

2.2 Funktionsweise des Systems

Adressverwaltung (Intelligente Ressourcenoptimierung)

CARA behandelt Adressen als unabhängige Objekte, die mehreren Personen zugeordnet werden können. Dieses Design spart Google API-Aufrufe und verbessert die Performance:

1. **Adressvalidierung:** Beim Hinzufügen einer neuen Person wird deren Adresse zuerst mit Googles Address Validation API validiert
2. **Adresswiederverwertung:** Falls dieselbe Adresse bereits in der Datenbank existiert, wird sie ohne weiteren API-Aufruf wiederverwendet
3. **Entfernungsberechnung:** Nur bei wirklich neuen Adressen berechnet das System Entfernungen zu allen existierenden Adressen

Der Importprozess

Beim Import eines **neuen Kindes oder Betreuers:**




Neue Person → Adressvalidierung → Datenbankpeicherung → Entfernungsberechnung

1. **Adressvalidierung:** Googles Address Validation API prüft, ob die Adresse real und korrekt ist
2. **Datenbankpeicherung:** Bei Gültigkeit wird die Adresse in der Datenbank gespeichert
3. **Distance Matrix-Update:** Das System berechnet Reisezeiten von dieser neuen Adresse zu allen existierenden Adressen





4. **Routenspeicherung:** Reisezeiten und Entfernungen werden lokal für zukünftige Nutzung gespeichert

Anwendungsworkflow

Was Google API benötigt:

-  Import neuer Kinder
-  Import neuer Betreuer
-  Hinzufügen/Bearbeiten einzelner Datensätze mit neuen Adressen

Was ohne Google API funktioniert:

-  **Pairs Manager** - Immer funktionsfähig, unabhängig von Google API
-  **Lineare Programmierungsoptimierung** - Nutzt vorberechnete Entfernungen aus der Datenbank
-  **Anzeige bestehender Daten**
-  **Erstellen von Zuordnungen** aus existierenden Kindern und Betreuern

Typische Nutzungsszenarien

Szenario 1: Ersteinrichtung

1. Google API-Schlüssel in den Anwendungseinstellungen konfigurieren
2. Ihre anfänglichen Kinder und Betreuer importieren (Adressen werden validiert)
3. System baut automatisch eine interne Entfernungsmatrix auf
4. Pairs Manager zur Erstellung optimaler Zuordnungen verwenden

Szenario 2: Neue Personen hinzufügen

1. Neues Kind oder Betreuer über die Web-Oberfläche hinzufügen
2. Adresse wird automatisch validiert
3. System berechnet Entfernungen zu allen existierenden Adressen
4. Neue Person ist sofort für Zuordnungserstellung verfügbar

Szenario 3: Arbeiten mit bestehenden Daten

1. Keine Google API-Aufrufe erforderlich
2. Pairs Manager zur Erstellung von Zuordnungen verwenden
3. Optimierungsalgorithmen mit gespeicherten Entfernungsdaten ausführen
4. Zuordnungen nach Bedarf anpassen

Kostenspartipps

Massenimporte

- Mehrere Kinder/Betreuer auf einmal importieren statt einzeln
- Das System verarbeitet Adressen in 100er-Batches für Distance Matrix API-Effizienz

Adresswiederverwertung

- Das System erkennt automatisch doppelte Adressen (gleiche Koordinaten)
- Keine zusätzlichen API-Aufrufe für Personen an derselben Adresse (Familien, geteilte Büros)

Einschränkungen & Überlegungen

Genauigkeit der Entfernungsdaten

- Entfernungs- und Reisezeitdaten werden einmalig beim Hinzufügen von Adressen berechnet
- Routen können sich über die Zeit ändern, aber eine Neuberechnung ist in dieser Version nicht enthalten
- Dieser Kompromiss priorisiert Kosteneinsparungen und Performance über Echtzeit-Routenaktualisierungen

Erste Schritte

1. **Setup:** Sicherstellen, dass die Docker-Umgebung läuft
2. **API-Konfiguration:** Google API-Schlüssel in den Anwendungseinstellungen hinzufügen
3. **Dienste überprüfen:** Prüfen, dass Distance Matrix und Address Validation APIs aktiviert sind

4. **Daten importieren:** Mit CSV-Massenimporten für beste API-Effizienz beginnen
5. **Zuordnungen erstellen:** Pairs Manager für optimales Kind-Betreuer-Matching verwenden

Das System ist darauf ausgelegt, effizient mit der Google API-Nutzung umzugehen und gleichzeitig mächtige Optimierungsfähigkeiten für Betreuerzuordnungen zu bieten.

3. Installation & Setup

Voraussetzungen:

- Installierte **Docker** und **Docker Compose** Umgebung auf dem Zielsystem:
 - Docker installieren: <https://docs.docker.com/get-started/get-docker/>

Installationsschritte:

1. Projekt-Repository herunterladen oder klonen:

```
git clone https://github.com/ozhadykov/cara.git
cd cara
```

2. Docker-Container erstellen und starten:

Mit folgendem Befehl werden alle benötigten Services (Frontend, Backend, Datenbank, Optimierungsmodul, Adminer) automatisch gebaut und im Hintergrund gestartet:

```
docker compose up --build -d
```

3. Überprüfung des Status:

Um zu überprüfen, ob alle Container laufen:

```
docker ps
```

Erwartete Container:

- **frontend** (React + NGINX)

- **backend** (FastAPI)
- **ampl** (Optimierungsmodul)
- **database** (MySQL/PostgreSQL)
- **db_adminer** (Datenbank-UI)

Wird ungefähr so aussehen:

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
1ba8d5207a33	phenix_dockerized-backend	"uvicorn app.main:ap..."	10 minutes ago
Up 10 minutes	0.0.0.0:8080→8080/tcp	backend	
47fe1635ccbc	phenix_dockerized-db	"docker-entrypoint.s..."	10 minutes ago
Up 10 minutes	3306/tcp, 33060/tcp	database	
de325b8cd504	phenix_dockerized-frontend	"/docker-entrypoint...."	10 minutes ago
Up 10 minutes	0.0.0.0:80→80/tcp	frontend	
73aed94b1ef1	phenix_dockerized-ampl	"uvicorn app.main:ap..."	10 minutes ago
Up 10 minutes	8000/tcp	ampl	
de59d59a2ad4	adminer	"entrypoint.sh docke..."	10 minutes ago
Up 10 minutes	0.0.0.0:8081→8080/tcp	db_adminer	

4. Zugriff auf die Anwendung:

- **Frontend/Web-App:** <http://localhost> oder http://vm's_ip_address (man kann das mit Kommand finden)

```
hostname -I
```

- **Datenbankverwaltung (Adminer):** <http://localhost:8081>

Umgebungsvariablen:

Es sind **keine zusätzlichen** `.env` -Dateien notwendig, da alle Variablen im `docker-compose.yml` definiert sind.

Stoppen und Neustarten:

- Stoppen der Anwendung:

```
docker compose down -v
```

- Neu starten:

```
docker compose up --build -d
```

4. Benutzerhandbuch / Nutzung der Anwendung

Nach erfolgreichem Start der Anwendung über Docker Compose ist die Webanwendung über den Browser erreichbar:

<http://localhost>

4.1 Anmeldung und Startseite

- Beim Aufruf der Anwendung wird die Startseite angezeigt.
- Hier befinden sich Hauptnavigationen wie **Kinderverwaltung**, **Betreuerverwaltung** und **Zuordnung**.

4.2 Kinder- und Betreuerverwaltung

- Import per CSV:
 - Über die jeweilige Importfunktion können **Kinder-** und **Betreuerdaten** als **CSV-Dateien** hochgeladen werden.
- Manuelle Verwaltung:
 - Neue Datensätze können über Formulare hinzugefügt werden.
 - Bestehende Einträge lassen sich **bearbeiten** oder **löschen** direkt in der UI.

4.3 Adressvalidierung und Fahrzeiten

- Bei der Eingabe oder Bearbeitung von Adressen wird die Adresse bis auf **Hausnummer-Ebene validiert**.
 - **Fahrzeiten zwischen Betreuern und Kindern** werden automatisch über die **Google Distance Matrix API** berechnet und angezeigt.
-

4.4 Erstellung von Betreuungsverhältnissen (Matching)

Manuelle Zuordnung:

- Nutzer können einzelne **Kinder-Betreuer-Paare** auswählen und erstellen.
- Die Anwendung prüft, ob das Paar gemäß Qualifikation und Verfügbarkeit zulässig ist.

Automatische Zuordnung:

- Es können gezielt **Kinder und Betreuer** markiert werden, die in die Berechnung einbezogen werden sollen (Auswahl über Checkboxen in einer Tabelle).
 - Die Berechnung erfolgt mithilfe von **Linearer Optimierung (AMPL)** und berücksichtigt:
 - **Reisezeit**
 - **Qualifikationen**
 - **Über- und Unterstunden** der Betreuer
 - Nach Abschluss wird ein **optimales Matching** angezeigt, das direkt übernommen werden kann.
-

4.5 Datenbank-Management (optional)

- Für Administratoren steht **Adminer** zur Verfügung:
 - URL: <http://localhost:8081>
-

4.6 Beispiel CSV

Der CSV-Import für Kinder muss in dieser Form erfolgen:

```
first_name,family_name,required_qualification,street,street_number,city,zip_
_code,requested_hours
Lena,Meyer,2,Lessingstraße,10,"Halle (Saale)",06114,20
Paul,Schmidt,1,Goethestraße,5,"Halle (Saale)",06110,25
Sophie,Weber,3,Hallorenring,1,"Halle (Saale)",06108,15
Max,Fischer,1,Merseburger Straße,50,"Halle (Saale)",06112,30,
```

Der CSV-Import für Betreuer muss in dieser Form erfolgen:

```
first_name,family_name,qualification,has_car,street,street_number,city,zip_
code,min_capacity,max_capacity
Sophie,Schneider,2,0,Lessingstraße,15,"Halle (Saale)",06114,25,35
Michael,Weber,1,1,Goethestraße,8,"Halle (Saale)",06110,15,25
Laura,Fischer,3,0,Hallorenring,3,"Halle (Saale)",06108,35,40
Daniel,Meyer,1,1,Merseburger Straße,60,"Halle (Saale)",06112,25,25
```

!!

Um Zahlen wie 25,56 einzugeben, muss man sie in Anführungszeichen setzen. Beispiel: „25.56“ wird als 25.56 ausgelesen. Auch die Zahlen müssen „.“ statt „,” enthalten, damit sie als Float ausgelesen werden.

4.7 Optimierungsmodell - Übersicht

Grundprinzip

Das CARA-System verwendet ein **mathematisches Optimierungsmodell** basierend auf **Linearer Programmierung (AMPL)**, um die bestmöglichen Zuordnungen zwischen Kindern und Betreuern zu finden. Das Modell

berücksichtigt dabei mehrere gewichtete Kriterien gleichzeitig und findet die global optimale Lösung.

Zielfunktion

Das Modell maximiert einen Gesamt-Score, der sich aus dem Nutzen aller Zuordnungen zusammensetzt, abzüglich von Strafpunkten für ineffiziente Ressourcennutzung:

$$\max Z = \sum(\text{Nutzen}_{ij} \times x_{ij}) - \sum(\text{Straf_Überstunden} + \text{Straf_Unterstunden})$$

Wo:

- x_{ij} = Binärvariable (1 wenn Betreuer j Kind i zugewiesen wird, sonst 0)
- Nutzen_{ij} = Kombiniertes Score aus Reisezeit und Qualifikationspassung

Berücksichtigte Faktoren

1. Qualifikationsanforderungen

- **Perfekter Match** (gleiche Qualifikation): +100 Punkte
- **Eine Stufe überqualifiziert**: +66.6 Punkte
- **Zwei Stufen überqualifiziert**: +33.3 Punkte
- **Unterqualifiziert**: -99999 Punkte (praktisch ausgeschlossen)

2. Reisezeiten

- Normalisierter Score zwischen 0 und 1
- Kurze Reisezeiten (≤ 15 min) = Score 1.0
- Lange Reisezeiten (≥ 90 min) = Score 0.0
- Lineare Interpolation dazwischen

3. Arbeitszeit-Optimierung

- **Überstunden** werden bestraft (konfigurierbares Gewicht)
- **Ungenutzte Stunden** werden bestraft (konfigurierbares Gewicht)
- Ziel: Gleichmäßige Auslastung der Betreuer

Wichtige Nebenbedingungen

Versorgungspflicht

$\sum x_{ij} \geq 1$ (Jedes Kind muss mindestens einen Betreuer erhalten)

Flexible Betreuung

$\sum x_{ij} \leq \text{MaxBetreuer}_i$ (Kinder mit hohem Bedarf können 2 Betreuer erhalten)

Ressourcenbilanz

$\sum (\text{Zeitbedarf}_i \times x_{ij}) + \text{Unterzeit}_j - \text{Überzeit}_j = \text{Zeitkapazität}_j$

Intelligente Funktionen

Automatische Aufteilung bei hohem Betreuungsbedarf

- Kinder mit >20 Stunden Bedarf (ist auch anpassbar) können automatisch auf 2 Betreuer aufgeteilt werden
- Das Modell entscheidet selbst, ob eine Aufteilung vorteilhaft ist

Berücksichtigung bestehender Zuordnungen

- Bereits fest zugewiesene Paare werden aus der Optimierung herausgenommen
- Kapazitäten werden entsprechend angepasst




Skalierbare Gewichtung


- Reisezeit-Wichtigkeit ist konfigurierbar
- Strafgewichte für Über-/Unterstunden sind anpassbar

Ergebnis

Das Modell liefert eine **mathematisch optimale Gesamtzuordnung**, die:

-  Alle Qualifikationsanforderungen erfüllt

-  Reisezeiten minimiert
-  Arbeitszeiten gleichmäßig verteilt
-  Gesamtnutzen maximiert

 **Detaillierte Dokumentation:** Eine vollständige mathematische Beschreibung des Modells inklusive aller Formeln und Algorithmen finden Sie im separaten Dokument "*Detaillierte Beschreibung der Datenvorbereitung und des mathematischen Zuweisungsmodells*", das dieser Dokumentation beiliegt.

5. Wartung & Weiterentwicklung

Dieser Abschnitt richtet sich an Entwickler oder Administratoren, die die Anwendung warten oder erweitern möchten.

```
/cara
/frontend/app    - React-Frontend (UI, Styling mit Tailwind CSS, DaisyUI)
/backend/app     - FastAPI-Backend (Controller, Logik, API-Routen)
/backend/ampl_app - Optimierungsmodul (AMPL + FastAPI-Schnittstelle)
/database        - Datenbank-Konfiguration (Dockerfile, Initialisierungen)
/docker-compose.yml - Definition aller Container und Netzwerke
```

5.2 Updates & Änderungen

Frontend ändern:

- Änderungen an der Benutzeroberfläche erfolgen im Verzeichnis `/frontend`.
- Nach Änderungen:

```
docker compose up --build -d
```

Backend ändern:

- Neue Endpunkte oder Logik in `/backend/app` implementieren.
- Nach Änderungen:

```
docker compose up --build -d
```


Optimierungsmodul erweitern:

- Anpassungen an den Matching-Algorithmen oder AMPL-Modellen in `/backend/ampl_app`

5.3 Häufige Probleme

- **Container starten nicht:**
 - Prüfen, ob Docker läuft und Ports nicht blockiert sind. (`docker ps` und checken Ports 80, 3306, 8081)
- **Keine Verbindung zur Datenbank:**
 - Sicherstellen, dass der Container `database` aktiv ist (`docker ps`).
- **Optimierung schlägt fehl:**
 - Logs des `ampl` Containers prüfen (`docker logs ampl`).

5.4 Backup & Datenverwaltung

- **Backup der Datenbank:**

```
docker exec -t database mysqldump -u root -p phenix_mysql > backup.sql
```

- **Wiederherstellen:**

```
docker exec -i database mysql -u root -p phenix_mysql < backup.sql
```

5.5 Backend API Endpoints - Overview

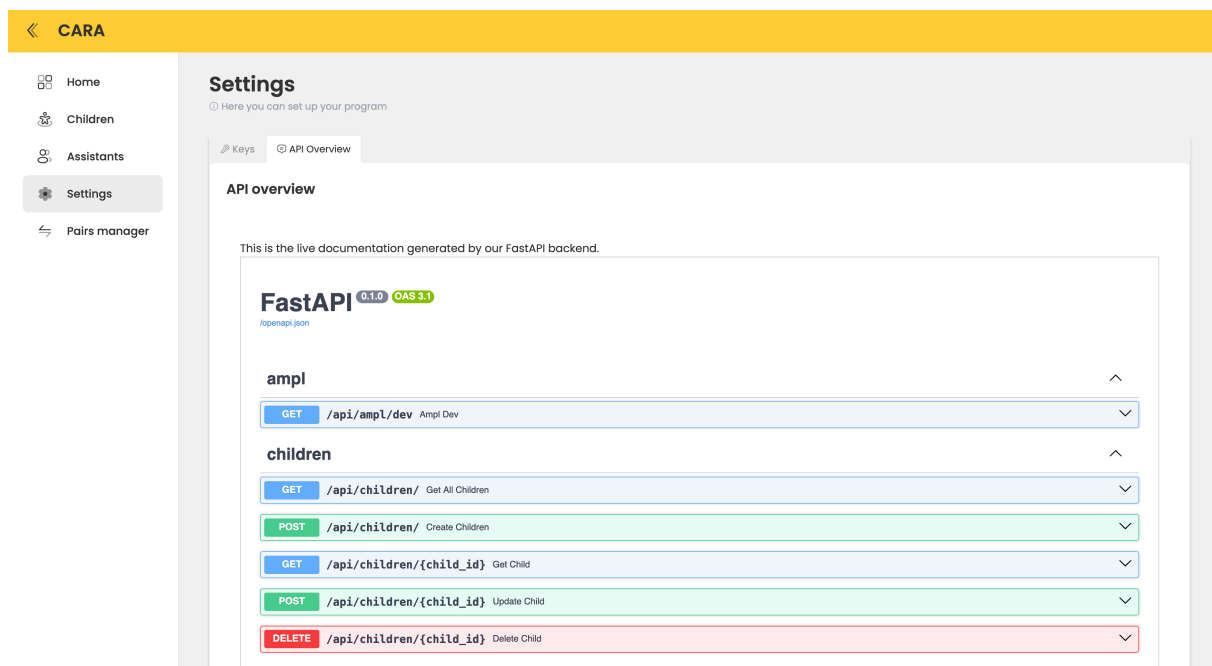
Das Backend stellt verschiedene API-Endpunkte bereit, die in logische Routen unterteilt sind. Diese ermöglichen den Zugriff auf und die Verwaltung aller relevanten Daten in der Anwendung:

- **Children Routes** – Verwaltung von Kinderdaten (Erstellen, Bearbeiten, Löschen, Abrufen).

- **Assistants Routes** – Verwaltung von Betreuerdaten mit denselben CRUD-Funktionen.
- **Pairs Routes** – Verwaltung von Betreuungsverhältnissen, inklusive manueller und automatischer Zuweisung.
- **Settings Routes** – Nur für Google API Key.

Alle verfügbaren Endpunkte, ihre HTTP-Methoden und Parameter finden Sie im **Settings-Tab** der Anwendung unter **API Overview**. Dort werden die einzelnen Routen, erforderlichen Parameter und Rückgabewerte übersichtlich dokumentiert.

Hier einmal wie das aussieht:



Endpoint Info:

Nach dem Öffnen des Übersichts-Tabs erhalten Sie Informationen zum jeweiligen Endpoint, einschließlich der erwarteten Antwort und der erforderlichen Daten.

This is the live documentation generated by our FastAPI backend.

FastAPI 0.1.0 OAS 3.1
/openapi.json

children

GET

/api/children/

Get All Children

^

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	<div>Successful Response</div> <div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <div>"string"</div>	No links
404	nothing found in children service	No links

POST

/api/children/

Create Children

v