



Dokumentation

Stammdatenverwaltung

Inhaltsverzeichnis

Einleitung	1
Projektziel	1
Vorteile	1
OptiBus	2
Was ist OptiBus?	2
Definitionsphase	2
Was soll am Ende des Projektes erreicht sein?	2
Welche Anforderungen müssen erfüllt sein?	2
Welche Einschränkungen müssen berücksichtigt werden?	2
Projektkostenkalkulation	2
Kosten	2
Amortisation	2
Rechnung	3
Make or Buy?	3
Konzeption	4
Benutzeroberfläche	4
API-Kommunikation	4
Datenbank	4
Backend	5
Entwicklungsumgebung	5
Aufbau der Daten	Fehler! Textmarke nicht definiert.
Funktionalitäten	6
API	6
UI/UX	7
UI	7

Corporate Design	7
Die Farbe des Wassers	7
Durchführung	8
Testphase	9
Einleitung.....	9
Durchführung.....	9
Unit-Test	9
Leichter Integrationstest	9
Ergebnisse	9
Installation und Einrichtung	10
Anlagen	11
Benutzerhandbuch	11
Glossar	11
Zeitplan.....	12
http-Statuscodes	13
UML-Klassendiagramm	13
Aktivitätsdiagramm.....	14
Use-Case Diagramm.....	14
Datenschutzrichtlinien	15
Einführung	15
Verantwortlichkeit für die Datenverarbeitung.....	15
Zweck der Datenverarbeitung	15
Arten der verarbeiteten Daten.....	15
Rechtsgrundlage für die Datenverarbeitung	15
Sicherheitsmaßnahmen	16
Datenweitergabe an Dritte.....	16
Aufbewahrung von Daten.....	16
Rechte der betroffenen Personen	16

Änderungen dieser Datenschutzrichtlinie	16
Entwicklerdokumentation:	17
Reisebusunternehmen Stammdaten Verwaltungsprogramm	17
Übersicht.....	17
Ordnerstruktur	17
Setup-Anweisungen	17
Verwendung von Supabase	17
Testen	18
Deployment.....	18
Bekannte Probleme und Einschränkungen	Fehler! Textmarke nicht definiert.
Benutzerhandbuch	19

Bitte beachten Sie, dass die **farbig** markierten Wörter im Glossar mit entsprechenden Erklärungen zu finden sind.

Einleitung

Diese Projektdokumentation bietet einen umfassenden Einblick in die Entwicklung einer intuitiven Benutzeroberfläche zur Verwaltung von Stammdaten. Das Projekt wurde initiiert, um die Effizienz der Datenverwaltung zu verbessern und das Unternehmen bei der besseren Nutzung seiner Ressourcen zu unterstützen. Die derzeitige Situation erfordert eine Neuentwicklung der Verwaltungssoftware von Stammdaten für den **Stakeholder** der OptiBus GmbH. Die bisherigen Schwierigkeiten bei der Pflege der Stammdaten haben die Notwendigkeit eines umfassenden Tools zur Verwaltung und Aktualisierung dieser Daten aufgezeigt. Das geplante Projekt wird von Grund auf als Teil der Neuentwicklung betrachtet und ist klar von den bestehenden Altsystemen abgegrenzt. Die Stammdaten werden informell als betriebliche Daten definiert, die statisch im System vorhanden sind. Beispiele hierfür sind Daten, die in einer Datenbank gespeichert werden und die für den reibungslosen Ablauf der Geschäftsprozesse von entscheidender Bedeutung sind. Dies ist ein Teilprojekt der Neuentwicklung.

Projektziel

Das Hauptziel dieses Projekts ist es, eine zentrale und benutzerfreundliche Plattform zu schaffen, die es dem Unternehmen ermöglicht, die Verwaltung seiner Daten zu optimieren. Die Lösung soll dazu beitragen, den Geschäftsbetrieb effizienter zu gestalten und die Zufriedenheit zu steigern. Der zu entwickelnde Web-Client soll es den Benutzern auf eine einfache und benutzerfreundliche Weise ermöglichen auf die Stammdaten zuzugreifen, sie zu bearbeiten und zu aktualisieren. Die intuitive Benutzeroberfläche reduziert die Einarbeitungszeit und erhöht die Effizienz bei der Datenverwaltung erheblich.

Vorteile

Durch die Einführung einer Verwaltungssoftware wird der Verwaltungsaufwand geringer und die Personalkosten sinken. Zudem können Daten genormt erfasst werden, um **Anomalien** zu vermeiden. Anomalien treten immer dann auf, wenn beim Einfügen, Ändern oder Löschen des Datensatzes Daten fehlen, fehlerhaft oder unvollständig sind.

OptiBus

Was ist OptiBus?

Ein mittelständisches Unternehmen mit etwas mehr als 50 Mitarbeitern. Gegründet in Mainz und ist eine fiktive Firma für dieses Schulprojekt. Schwerpunkte der Firma sind dabei u.a. die Beförderung der Kunden zu ihrem Zielort und dessen Verwaltung.

Definitionsphase

Was soll am Ende des Projektes erreicht sein?

Ein intuitiver Web-Client zur Verwaltung der [Stammdaten](#), dessen Kommunikation an eine Backend Schnittstelle (Datenbank) mittels API läuft.

Welche Anforderungen müssen erfüllt sein?

- Daten sollen per [CRUD](#) (Create-Read-Update-Delete) verwaltet werden können.
- Daten mittels [API](#) an ein Backend kommunizieren.

Welche Einschränkungen müssen berücksichtigt werden?

Zeitliche Aspekte:

Da das Abschlussprojekt für eine Neuentwicklung eine kurze Bearbeitungszeit hat, ist der Umfang entsprechend begrenzt. Durch den modularen Aufbau des Projektes ist eine einfache Weiterentwicklung gewährleistet.

Projektkostenkalkulation

Die Kosten werden auf Basis der geleisteten Arbeitsstunden während des Projektes berechnet. Dabei rechnen wir mit fiktiven Werten, da dieses Projekt keine echten Kosten verursacht.

Kosten

Als Richtwert für den Stundenlohn nehmen wir den aktuellen Mindestlohn von (Stand 2023) 12 €. Es ist zu berücksichtigen, dass das Projektteam aus zwei Personen besteht, von denen eine nur die Hälfte der Zeit anwesend sein wird.

Amortisation

Angenommen, wir betrachten einen imaginären Lizenzwert von 500€ pro Monat.

Rechnung

<u>Kosten</u>	<u>Einnahmen</u>
2*(20h* 12€)	50€ Lizenzkosten
+ (40h* 12€)	
= 960€	
960€ / 50€	
<u>= 19,2 Monate</u>	

Bei der Kalkulation wurden reine fiktive Nettozahlen verwendet, um die Komplexität gering zu halten.

Beginnend zum 20. Monat wären die Kosten des Projekts amortisiert und ab diesem Zeitpunkt werden Gewinne erzielt.

Make or Buy?

Nun stellt sich die Frage, welche Argumente dafürsprechen, dass sich eine Eigenentwicklung lohnt, oder ob die Entscheidung letztlich doch auf den Kauf fällt

KRITERIEN	MAKE	BUY
KOSTEN	Entwicklungskosten sind hoch	Anschaffungskosten sind hoch, aber keine Entwicklungskosten
Zeit	Lange Entwicklungszeit	Sofortige Verfügbarkeit
Anpassbarkeit	Vollständige Anpassbarkeit an spezifische Anforderungen	Begrenzte Anpassbarkeit, je nach gekaufter Lösung
Kontrolle	Volle Kontrolle über die Entwicklung und den Code	Geringere Kontrolle über Funktionsweise und Updates der gekauften Software
Risiko	Höheres Risiko bei der Entwicklung	Geringeres Risiko, da bereits etablierte Lösung
Support	Eigenes Support-Team notwendig	Externer Support verfügbar
Skalierbarkeit	Kann an zukünftige Bedürfnisse angepasst werden	Abhängig von der Flexibilität der gekauften Lösung

Konzeption

Unsere Technologiestack besteht aus modernen und bewährten Technologien, die speziell für die Anforderungen dieses Projekts ausgewählt wurden.

Benutzeroberfläche

Im Bereich Frontend setzen wir auf die mächtige Kombination aus Next.js, einem React-Framework, das speziell für Anforderungen wie im Bereich Routing entwickelt wurde und React welches das Herzstück unserer Oberflächenentwicklung ist. Es ermöglicht uns, wiederverwendbare Objekte zu erstellen, die ein konsistentes und effizientes Design unserer Website gewährleisten. Die visuelle Gestaltung unserer Benutzeroberfläche wird durch React in Kombination mit TailwindCSS erreicht. TailwindCSS übernimmt dabei das Inline-Styling, d.h. die Anpassung des Erscheinungsbildes erfolgt direkt im jeweiligen Objekt selbst. Diese Vorgehensweise ermöglicht nicht nur eine hohe Flexibilität, sondern erleichtert auch die Wartung und Weiterentwicklung unseres Frontends. Ein weiterer wichtiger Aspekt unserer Frontend-Entwicklung ist die Verwendung von TypeScript. TypeScript erweitert unsere Entwicklung um eine zusätzliche Ebene - der Typisierung. Durch die explizite Angabe von Typen für Variablen und Objekte erhöht TypeScript die Codequalität und macht den Entwicklungsprozess effizienter. Dies ist insbesondere bei größeren Projekten von Vorteil, da potenzielle Fehler frühzeitig erkannt und vermieden werden können.

Zusammengefasst lässt sich sagen, dass unsere Frontend-Entwicklung mit Next.js, React, TailwindCSS und TypeScript eine solide Basis für eine benutzerfreundliche, ansprechende und gut strukturierte Oberfläche bietet. Diese Technologien gewährleisten eine optimale User Experience (UX) und vereinfachen gleichzeitig die Entwicklungsprozesse.

API-Kommunikation

Unsere API-Kommunikation wird durch die Funktionen mittels NextJS und Supabase bereitgestellt. Dabei bieten beide Seiten jeweils eigene API-Endpunkte an, die wir sowohl als Server oder auch als Client ansprechen können.

Datenbank

[Supabase](#) ist eine Open-Source-Plattform für die Entwicklung von Anwendungen, die auf [PostgreSQL](#)-Datenbanken basieren. Es bietet eine Vielzahl von Tools und Diensten, die Entwicklern helfen, Datenbanken schnell einzurichten, zu verwalten und mit

Anwendungen zu verbinden. Es kombiniert die Leistungsfähigkeit von relationalen Datenbanken mit der Flexibilität und Einfachheit von Backend-Services, und das alles in einer einzigen, benutzerfreundlichen Plattform. Durch die Verwendung von PostgreSQL als zugrunde liegende Datenbank bietet Supabase die volle Leistungsfähigkeit einer ausgereiften relationalen Datenbank. Dies ermöglicht komplexe Abfragen, Transaktionen und Datenintegrität.

Backend

Backend als Datenübersetzer: Unser Backend nimmt eine entscheidende Rolle als Schnittstelle zwischen Datenbank und Frontend ein. Er fungiert nicht nur als Übersetzer, sondern übernimmt auch die Aufgabe, möglicherweise falsch formatierte Daten abzufangen und zu korrigieren. Das Backend basiert auf NodeJS und ist nah am NextJS angeknüpft.

Entwicklungsumgebung

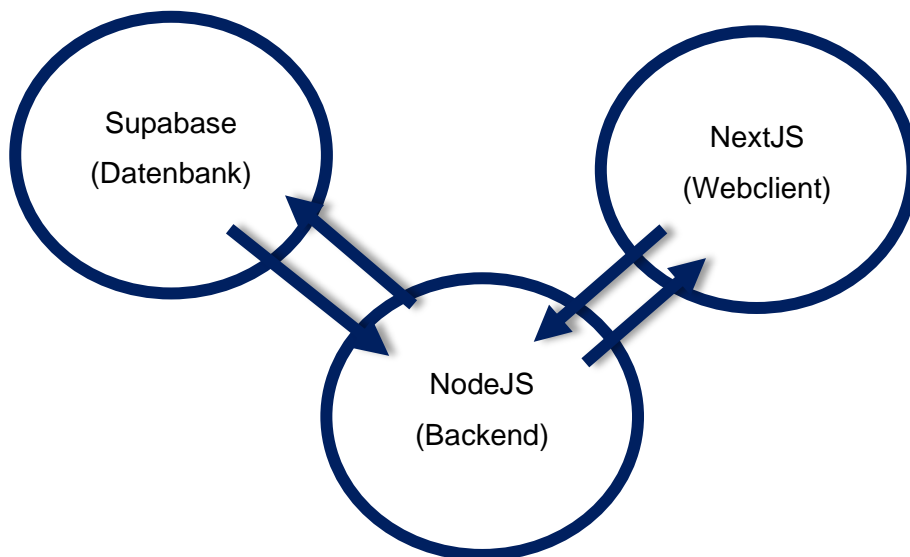
Entwicklung: Für die Entwicklung im Frontend setzen wir auf Visual Studio Code, eine Entwicklungsumgebung (IDE). Visual Studio Code bietet eine Vielzahl von Funktionen, die die Frontend-Entwicklung erleichtern, darunter eine benutzerfreundliche Oberfläche, integrierte [Git](#)-Unterstützung, [Syntax](#)-Hervorhebung für verschiedene Programmiersprachen und eine breite Palette an Erweiterungen.

Versionsverwaltung: Zur Versionsverwaltung nutzen wir Git über [GitHub](#). Git ist ein bewährtes Tool für die Zusammenarbeit in der Softwareentwicklung und bietet eine Versionsverwaltung. Funktionen wie Git-Repository-Verwaltung, [CI/CD](#), Fehler-Tracking und vieles mehr nutzen wir zusammen durch die Kombination mittels GitHub. Die Verwendung von Git ermöglicht es unserem Entwicklungsteam, effizient zusammenzuarbeiten, Änderungen nachzuverfolgen und eine konsistente Versionskontrolle über den gesamten Entwicklungszyklus hinweg aufrechtzuerhalten.

Architektur

Bus, Personal, Kunde, Bezahlung, Reise, Ticket, Fahrer. Jede dieser Datenarten erfüllt spezifische Aufgaben und trägt zur Funktionalität unseres Systems bei. Um Daten dynamisch zu verwalten benutzen wir hierbei das MVC pattern. Die Daten werden dabei zusätzlich klar von Server und Client getrennt verarbeitet. Unsere Architektur besteht

aus einer Datenbank (Supabase), einem Server (NodeJS) und einem Frontend-Webclient (NextJS):



Funktionalitäten

Mittels Software sollen **Stammdaten** per **CRUD** verwaltet werden können. Dabei müssen die Typisierungen eingehalten werden um Daten konsistent zu halten. So dürften z.B. keine Boolean Werte in einem Datum abgespeichert werden.

API

Unsere **API** bietet eine Schnittstelle für die Kommunikation zwischen dem Frontend und der Datenbank. Hier sind einige der Endpunkte und Funktionen (als Beispiel mittels Bus), die über die API verfügbar sind:

getAll	Dieser Endpunkt gibt eine Liste aller Bus-Objekte im System zurück.
get(id)	Gibt das genaue Bus-Objekt mit der entsprechenden ID zurück.
Insert(data)	Fügt ein neues Bus-Objekt hinzu.
update(data)	Inhaltliches bearbeiten vom Bus.
remove(data)	Löscht einen Bus aus der Datenbank.

Ähnliche Endpunkte sind für die Verwaltung von den restlichen Tabellen verfügbar. Unsere API folgt bewährten REST-Prinzipien (**CRUD**) und verwendet HTTP-Statuscodes wie 200, 404, ..., um den Status der Anfragen anzuzeigen.

UI/UX

UI

Für die UI entschieden wir uns für ein **Flat-UI** Design. Dabei werden 2-dimensionale Icons und Komponente benutzt. Im Gegensatz zu früheren Designstilen wie **Skeuomorphismus**, bei denen Elemente realistisch und dreidimensional dargestellt wurden, betont das flache Design die Schlichtheit und klare Linien.

Corporate Design

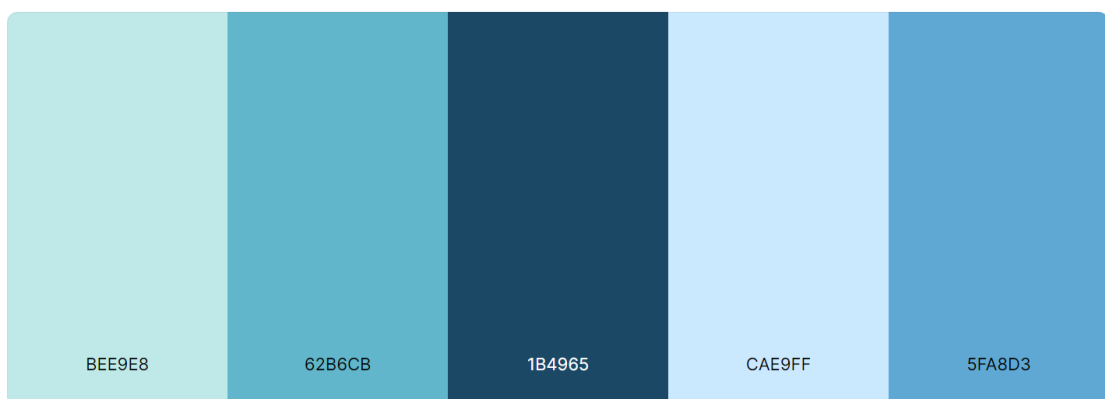
Entdecken Sie Unser Blaues Corporate Design:

Eine Ode an Harmonie, Urlaub und Meer

Willkommen in einer Welt, in der Farben Geschichten erzählen und Emotionen wecken. Bei unserer Entscheidung für ein blaues Corporate Design drehte sich alles um die Sehnsucht nach Ruhe, Entspannung und der endlosen Weite des Ozeans. Lassen Sie uns gemeinsam eintauchen in die Gründe, warum Blau für uns nicht nur eine Farbe, sondern ein Gefühl von Harmonie und Freiheit verkörpert.

Die Farbe des Wassers

Blaue Töne umgeben uns mit der sanften Beruhigung des Meeres. Sie symbolisieren Reinheit, Klarheit und Tiefe. Gleichzeitig erinnern sie uns an den unendlichen Horizont, der uns einlädt, die Grenzen unserer Vorstellungskraft zu überschreiten. Unser blaues Corporate Design soll genau dieses Gefühl von Weite und Unendlichkeit vermitteln.



Durchführung

In Bezug auf das Projekt zur Erstellung einer [Stammdatenverwaltung](#) benutzen wir das [Wasserfallmodell](#).

Anforderungsanalyse: In der ersten Phase, der Anforderungsanalyse, ist es entscheidend, die Anforderungen der Datenbank, des Frontends, der API und anderer Komponenten sorgfältig zu erfassen. Dies umfasst die Identifizierung der erforderlichen Datenfelder, Funktionen und Elemente in der Benutzeroberfläche. Da das Wasserfallmodell eine detaillierte Anforderungsspezifikation betont, muss sichergestellt werden, dass alle Anforderungen klar definiert sind.

Systemdesign: In der Systemdesign-Phase konnten wir die Software-Architektur und das Datenbankdesign detailliert planen. So erstellen wir spezifische Pläne für das Frontend, sowie die Struktur der Datenbanktabellen. Die Phase sorgt für eine solide Grundlage, um sicherzustellen, dass die Systemkomponenten gut aufeinander abgestimmt sind.

Implementierung: Während der Implementierungsphase erfolgt die eigentliche Entwicklung der Softwarekomponenten. Dabei werden die Pläne aus den vorherigen Phasen in funktionierenden Code umgesetzt.

Testen: Nach der Implementierung folgt die Testphase, in der die entwickelte Software umfassend getestet wird, um sicherzustellen, dass sie den Anforderungen entspricht und fehlerfrei funktioniert. Hierbei handelt es sich um eine entscheidende Phase, um die Qualität des Systems sicherzustellen. Um Software-Code zu testen, werden Unit-Tests und Integrationstests durchgeführt. Die Coverage gibt dabei an, wie viel des Codes dabei erfolgreich abgedeckt wurde.

Abnahme und Freigabe: Sobald die Tests erfolgreich abgeschlossen sind und das System den Anforderungen entspricht, wird es für den produktiven Einsatz freigegeben. Dies erfolgt bei uns nach einer endgültigen Abnahme und Überprüfung durch den [Stakeholder](#).

Testphase

Einleitung

Die Integrationstestphase im Rahmen unseres Projekts war entscheidend, um sicherzustellen, dass das Frontend und Backend nahtlos miteinander interagieren und alle Funktionen ordnungsgemäß ausgeführt werden. In dieser Dokumentation werden die Schritte und Überlegungen während dieser Testphase detailliert beschrieben, insbesondere im Hinblick auf die Abstimmung von **CRUD**-Operationen.

Durchführung

Unit-Test

Ein Unit-Test ist eine Softwaretestmethode, bei der einzelne Komponenten, in der Regel Funktionen oder Methoden, auf ihre korrekte Funktionalität überprüft werden.

Leichter Integrationstest

Leicht daher, da wir nur im Rahmen des Projektes auf CRUD und den Eingaben der Formulare getestet haben.

- Create: Das Frontend sendet Anfragen zum Erstellen neuer Einträge an das Backend. Wir überprüften, ob die erstellten Einträge ordnungsgemäß in der Datenbank gespeichert werden.
- Read: Das Frontend fordert Daten vom Backend an und präsentiert sie dem Benutzer. Es wurde sichergestellt, dass die angeforderten Daten korrekt und vollständig vom Backend bereitgestellt werden.
- Update: Das Frontend sendet Aktualisierungsanfragen an das Backend, und wurden überprüft, ob die entsprechenden Datensätze erfolgreich aktualisiert werden.
- Delete: Löschanfragen vom Frontend wurden daraufhin überprüft, ob die zugehörigen Daten im Backend gelöscht werden.

Die einheitliche Handhabung von Datumsformaten war entscheidend, um Probleme bei der Datenaustauschkompatibilität zu vermeiden.

Ergebnisse

Die Integration war mit wenig Komplikationen erfolgreich, Fehler konnten schnell gefunden und behoben werden. Alle **CRUD**-Operationen wurden erfolgreich aufeinander

abgestimmt. Durch diesen Test konnten zukünftige Probleme frühzeitig erkannt werden und sorgen für eine problemlose Fortsetzung der Entwicklung im späteren Verlauf der Entwicklung.

Installation und Einrichtung

Um das System einzurichten, müssen Entwickler [Node.js](#) und [npm](#) auf ihren Rechnern installiert haben. Nach dem Klonen (kopieren) des Projekts aus dem [Repository](#) (Projektumgebung), können die erforderlichen Abhängigkeiten mit `<npm install>` installiert werden.

Anlagen

Benutzerhandbuch

Unser Benutzerhandbuch bietet detaillierte Anweisungen für die Verwendung des Systems. Es enthält Schritt-für-Schritt-Anleitungen zur Verwaltung von [Stammdaten](#). Screenshots und Beispiele erleichtern die Verwendung des Systems für Endbenutzer.

Glossar

Anomalien	Fehlverhalten oder fehlende Informationen beim Einfügen, Ändern oder Löschen von Daten wenn Daten fehlen, fehlerhaft oder unvollständig sind.
API	Schnittstelle, die den Datenaustausch zwischen Programmen ermöglicht.
BSON	Binary JSON, eine JSON in binär umgewandelt.
CI/CD	Continuous Integration / Continuous Delivery (Fortlaufende Integration / Fortlaufende Verteilung)
CRUD	Create-Read-Update-Delete
Supabase	Eine webbasierte PostgreSQL Datenbank.
Flat-UI Design	Simple gehalten, vereinfacht und in 2D.
Git	Eine Versionsverwaltungssoftware.
GitHub	Versionsverwaltung, kollaborator Entwicklungsplattform, mit Möglichkeiten zur CI/CD.
JSON	Dokument mit key-value Paaren, um Datenstrukturen vorzugeben und Datenaustausch zwischen Anwendungen zu standardisieren.
Node.js	Runtime Environment für serverseitiges JavaScript programmieren.
NPM	Node Package Manager zum Verwalten von Projekt Abhängigkeiten.
Repository	Der Container zum Speichern der Projekt Daten.
Skeuomorphismus	Design im gewohnten 3D, bei denen realitätsechte Objekte verwendet werden. Z.B. Tastentelefon für Anruficon.
Stakeholder	Ein Interessensvertreter des Projektes. Diese Person kann direkt von einer Entscheidung betroffen sein.

Stammdaten	Stammdaten werden informell als betriebliche Daten definiert, die statisch im System vorhanden sind.
Syntax	Kombination von Zeichen, die dem Sprachformat entsprechen.
Wasserfallmodell	Klassische Methode mit klar definierten Projektphasen: Anforderungen, Entwurf, Implementation, Überprüfung und Wartung.

Zeitplan

- Definitionsphase (20h)
 - Anforderungsanalyse
 - Projektkosten
 - Architekturentwurf
- Konzeptionsphase (20h)
 - Benutzeroberflächendesign
 - Grobe Datenflussfestlegung
 - Corporate Design
- Implementierungsphase (20h)
 - Typescript
 - Logik Implementierung
 - Erstellung der einzelnen UI-Komponenten
 - Verheiratung der UI und der Logik
- Testen und Qualitätssicherung (12h)
 - Unittests
 - Fehlerbehebung
 - Einfache Performance-Optimierung
 - Bereitstellung der Projektdaten
- Dokumentation (8h)
 - Projektdokumentation
 - Entwicklerdokumentation
 - Benutzerhandbuch

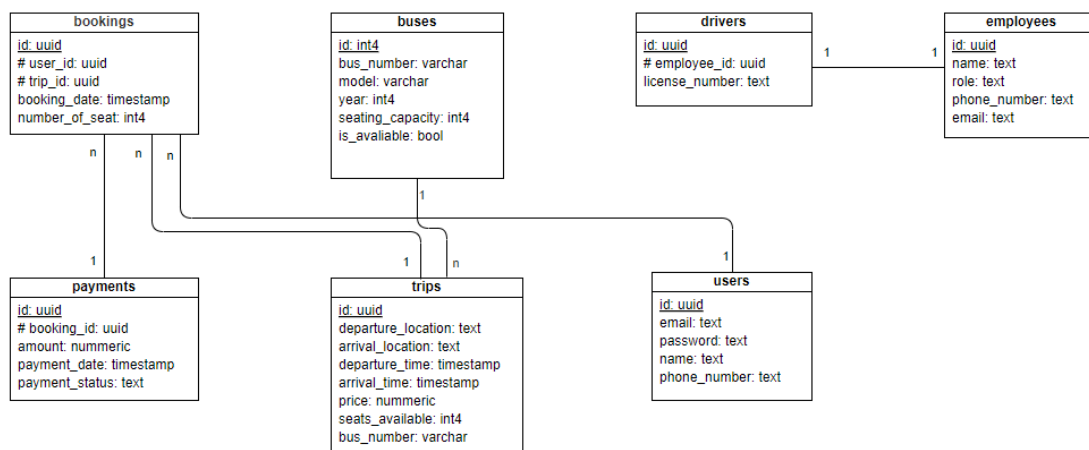
http-Statuscodes

Dies ist die Auflistung aller grundlegend relevanten Statuscodes für die Entwicklung bei Statuscode Abfragen für API-Requests:

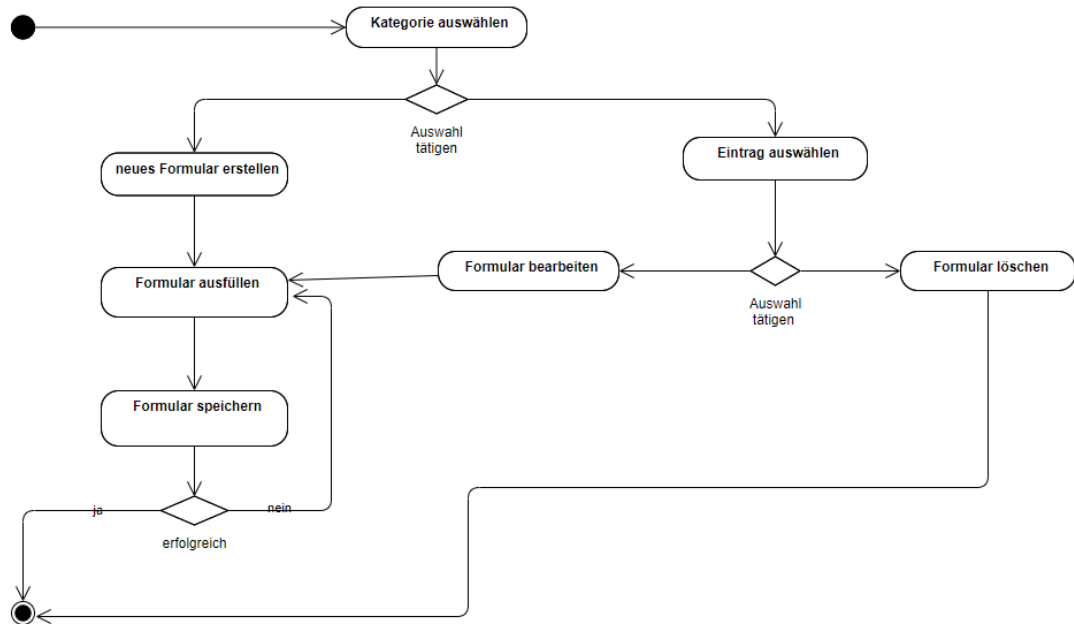
200	OK	Die Anfrage war erfolgreich
201	Created	Die Anfrage hat <i>etwas</i> erfolgreich erstellt.
204	No Content	Die Anfrage war erfolgreich, gibt aber keine Daten zurück.
400	Bad Request	Die Anfrage war fehlerhaft.
401	Unauthorized	Der Zugriff erfordert eine Authentifizierung.
403	Forbidden	Der Zugriff ist auch mit derzeitiger Authentifizierung nicht erlaubt.
404	Not Found	Die angeforderte Ressource wurde nicht gefunden.
405	Method Not Allowed	Die Methode ist nicht für diese Ressource erlaubt.
500	Internal Server Error	Ein allgemeiner Serverfehler ist aufgetreten.
502	Bad Gateway	Der Server hat eine ungültige Antwort erhalten.
503	Service Unavailable	Der Server kann nicht erreicht werden / ist überlastet.

Eine Auflistung aller Statuscodes kann unter <https://datatracker.ietf.org/doc/html/rfc7231> gefunden werden.

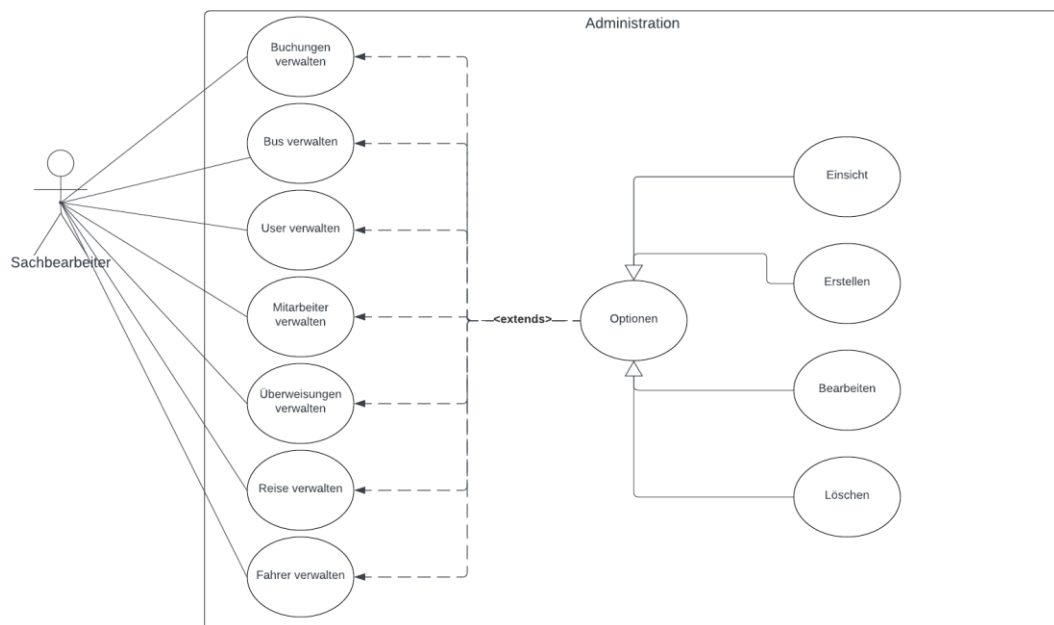
UML-Klassendiagramm



Aktivitätsdiagramm



Use-Case Diagramm



Datenschutzrichtlinien

Einführung

OptiBus Reisebusunternehmen (nachfolgend "das Unternehmen") verpflichtet sich zum Schutz der Privatsphäre und personenbezogenen Daten seiner Kunden und Geschäftspartner. Diese Datenschutzrichtlinie beschreibt, wie das Unternehmen personenbezogene Daten in Zusammenhang mit seiner Stammdatenverwaltungssoftware gemäß den Anforderungen der Datenschutz-Grundverordnung (DSGVO) der Europäischen Union verarbeitet und schützt.

Verantwortlichkeit für die Datenverarbeitung

Das Unternehmen ist für die Verarbeitung personenbezogener Daten im Rahmen seiner Stammdatenverwaltungssoftware verantwortlich und stellt sicher, dass alle Mitarbeiter, die Zugriff auf diese Daten haben, die Datenschutzrichtlinien und -verfahren des Unternehmens einhalten.

Zweck der Datenverarbeitung

Die Stammdatenverwaltungssoftware wird genutzt, um Kundeninformationen, Buchungsdaten, Reiserouten und andere relevante Informationen im Zusammenhang mit den Dienstleistungen des Unternehmens zu verwalten. Die Verarbeitung personenbezogener Daten erfolgt ausschließlich zum Zweck der Vertragserfüllung, Kundenbetreuung, Buchungsabwicklung und internen Verwaltung.

Arten der verarbeiteten Daten

Die Stammdatenverwaltungssoftware kann folgende Arten von personenbezogenen Daten enthalten:

- Name, Adresse, Kontaktinformationen der Kunden und Geschäftspartner
- Reisedaten, Buchungsinformationen, Präferenzen
- Gegebenenfalls besondere Kategorien personenbezogener Daten gemäß Artikel 9 DSGVO, wie gesundheitliche Einschränkungen oder besondere Bedürfnisse im Zusammenhang mit der Reise

Rechtsgrundlage für die Datenverarbeitung

Die Verarbeitung personenbezogener Daten erfolgt auf Grundlage einer oder mehrerer der folgenden Rechtsgrundlagen gemäß Artikel 6 DSGVO:

- Die Verarbeitung ist zur Erfüllung eines Vertrags erforderlich, dessen Vertragspartei die betroffene Person ist.

- Die Verarbeitung ist zur Erfüllung rechtlicher Verpflichtungen erforderlich, denen das Unternehmen unterliegt.
- Die betroffene Person hat ihre Einwilligung zur Verarbeitung ihrer personenbezogenen Daten für einen oder mehrere bestimmte Zwecke gegeben.

Sicherheitsmaßnahmen

Das Unternehmen ergreift angemessene technische und organisatorische Maßnahmen, um die Sicherheit und Vertraulichkeit der personenbezogenen Daten zu gewährleisten. Dazu gehören unter anderem Zugriffsbeschränkungen, Verschlüsselungstechnologien, Schulungen der Mitarbeiter und regelmäßige Überprüfungen der Sicherheitsmaßnahmen.

Datenweitergabe an Dritte

Das Unternehmen gibt personenbezogene Daten nur an Dritte weiter, wenn dies zur Erfüllung der Vertragsleistung oder gesetzlicher Verpflichtungen erforderlich ist. Dritte, die Zugriff auf personenbezogene Daten haben, sind vertraglich verpflichtet, die Datenschutzstandards des Unternehmens einzuhalten.

Aufbewahrung von Daten

Personenbezogene Daten werden nur so lange gespeichert, wie es für die Erfüllung der festgelegten Zwecke erforderlich ist oder gesetzliche Aufbewahrungsfristen dies vorschreiben. Nach Ablauf der Aufbewahrungsfristen werden die Daten sicher gelöscht oder anonymisiert.

Rechte der betroffenen Personen

Gemäß der DSGVO haben betroffene Personen bestimmte Rechte in Bezug auf ihre personenbezogenen Daten, einschließlich des Rechts auf Auskunft, Berichtigung, Löschung und Widerspruch gegen die Verarbeitung. Betroffene Personen können diese Rechte ausüben, indem sie sich an das Unternehmen wenden.

Änderungen dieser Datenschutzrichtlinie

Das Unternehmen behält sich das Recht vor, diese Datenschutzrichtlinie jederzeit zu ändern oder zu aktualisieren, um rechtlichen, technologischen oder geschäftlichen Anforderungen gerecht zu werden. Jegliche Änderungen werden auf der Unternehmenswebsite veröffentlicht.

Entwicklerdokumentation:

Reisebusunternehmen Stammdaten Verwaltungsprogramm

Übersicht

Das Reisebusunternehmen - Stammdaten Verwaltungsprogramm, ist eine Webanwendung, die entwickelt wurde, um die Stammdaten eines Reisebusunternehmens zu verwalten. Die Anwendung wurde mit Next.js, React, TypeScript und Tailwind CSS erstellt und verwendet die Supabase-API für die Datenhaltung.

Ordnerstruktur

Die Ordnerstruktur des Projekts ist wie folgt organisiert:

- **src/**
 - **app/**: Enthält die Seiten (Pages) und deren Layout.
 - **components/**: Enthält wiederverwendbare UI-Komponenten, die von Seiten und Layouts verwendet werden.
 - **lib/**: Enthält Supabase-Endpunkte mit Methoden für das Abrufen, Einfügen, Aktualisieren und Löschen von Daten.
 - **styles/**: Enthält die **globals.css**-Datei für globale Stileinstellungen.
 - **types/**: Enthält TypeScript-Typdefinitionen für Supabase.Komponenten und Styling
- **Seitenstruktur**: Jede Seite muss die **<Card>**-Komponente nutzen, die eine **<CardHeader>** für Titel und eine **<CardContent>** für Inhalte enthält.
- **Styling**: Die Farben, die in der **globals.css**-Datei definiert sind, werden in Tailwind CSS mit der Syntax **className="text-primary"** aufgerufen.

Setup-Anweisungen

1. Kclone das Repository von GitHub auf deinen lokalen Computer.
2. Navigiere in das Projektverzeichnis.
3. Führe **npm install** aus, um alle Abhängigkeiten zu installieren.
4. Kopiere ggf. die erforderlichen Umgebungsvariablen in die **.env**-Datei.

Verwendung von Supabase

Die Supabase-API wird verwendet, um Daten zu lesen, einzufügen, zu aktualisieren und zu löschen. Die entsprechenden Methoden sind im **lib/**-Ordner implementiert.

Testen

Es liegen keine spezifischen Informationen zu automatisierten Tests vor. Entwickler werden ermutigt, Tests gemäß den Anforderungen des Projekts hinzuzufügen.

Deployment

Das Deployment erfolgt auf dem gewünschten Hosting-Dienst. Eine Konfiguration für die Bereitstellung wird nicht spezifiziert.

Diese Dokumentation soll als Orientierungshilfe für Entwickler dienen, die am Projekt arbeiten oder es übernehmen möchten. Bei Bedarf können weitere Details hinzugefügt oder angepasst werden.

Benutzerhandbuch

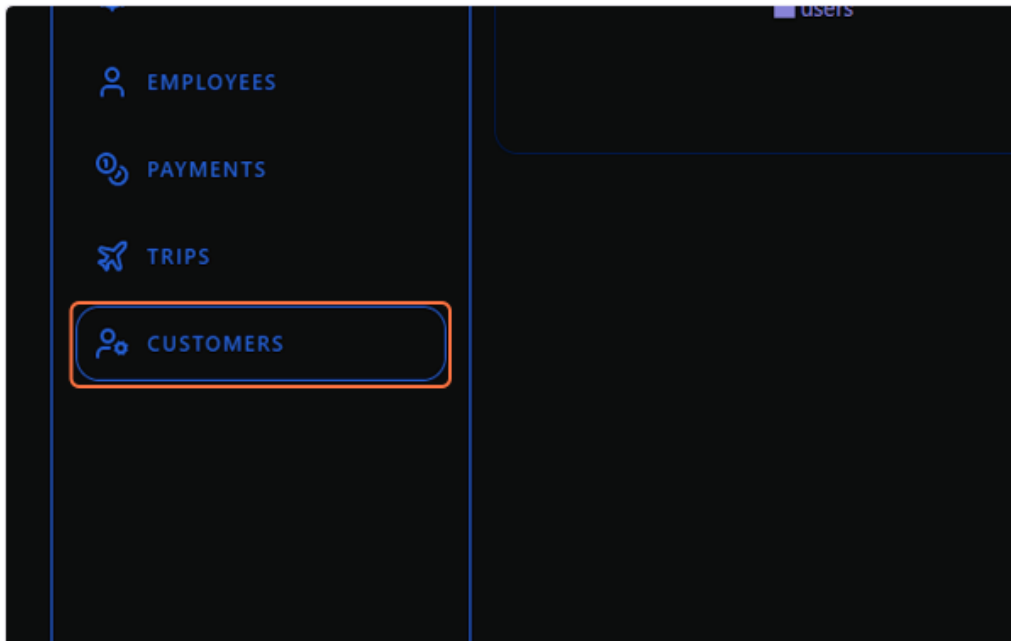
Using OptiBus: A Step-by-Step Tutorial

Creation Date: February 29, 2024

Created By: Nathalie S

OptiBus

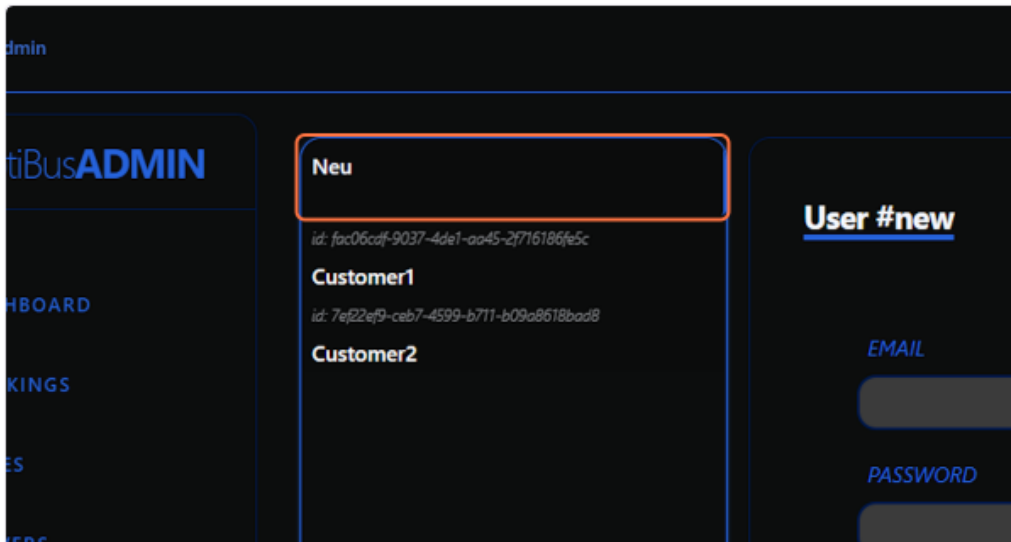
1. Click on CUSTOMERS



Tango

Created with Tango.us

2. Click on Neu



3. Type "max.mustermann@mail.com"

The screenshot shows a dark-themed user registration form titled "User #new". The form contains four input fields: "EMAIL", "PASSWORD", "NAME", and "PHONE_NUMBER". The "EMAIL" field is highlighted with an orange border and contains the text "max.mustermann@mail.com". A "copy" button is visible to the right of the email field. The "PASSWORD", "NAME", and "PHONE_NUMBER" fields are empty. The form is created with Tango.us.

User #new

EMAIL

max.mustermann@mail.com copy

PASSWORD

NAME

PHONE_NUMBER

Tango Created with Tango.us

4. Type "mypassword"

The screenshot shows the same dark-themed user registration form titled "User #new". The "EMAIL" field is now filled with "max.mustermann@mail.com" and has a "copy" button. The "PASSWORD" field is highlighted with an orange border and contains the text "mypassword". A "copy" button is also visible to the right of the password field. The "NAME" and "PHONE_NUMBER" fields remain empty. The form is created with Tango.us.

User #new

EMAIL

max.mustermann@mail.com copy

PASSWORD

mypassword copy

NAME

PHONE_NUMBER

Tango Created with Tango.us

5. Type "max mustermann"



EMAIL

max.mustermann@mail.com copy

PASSWORD

mypassword copy

NAME

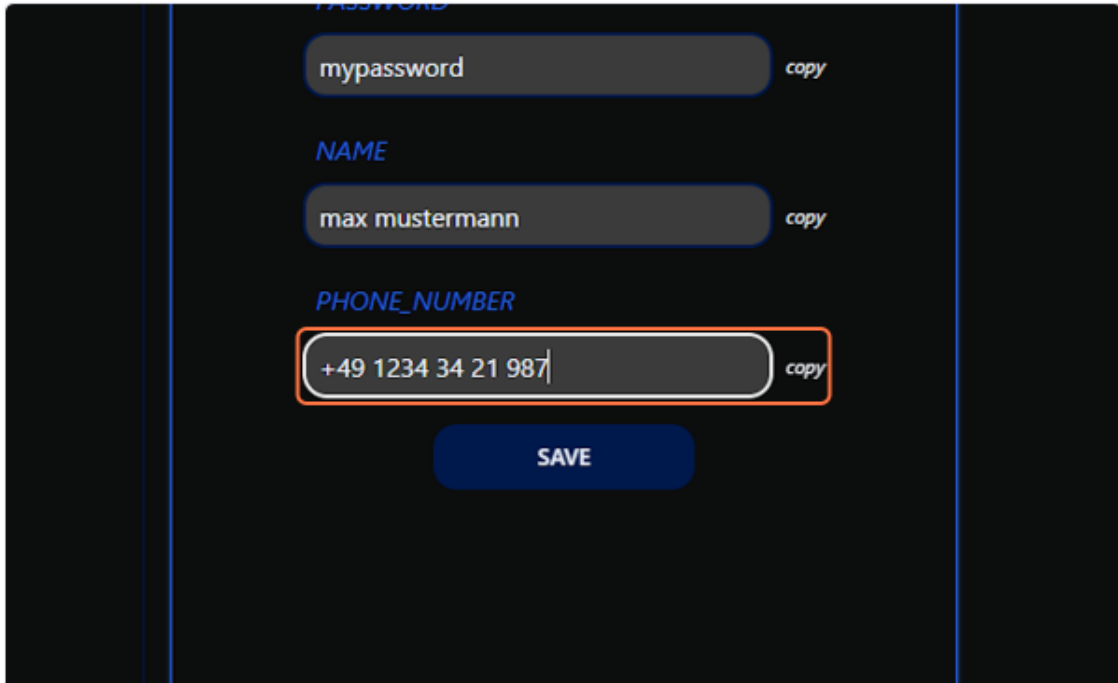
max mustermann copy

PHONE_NUMBER

SAVE

Tango Created with Tango.us

6. Type "+49 1234 34 21 987"



PASSWORD

mypassword copy

NAME

max mustermann copy

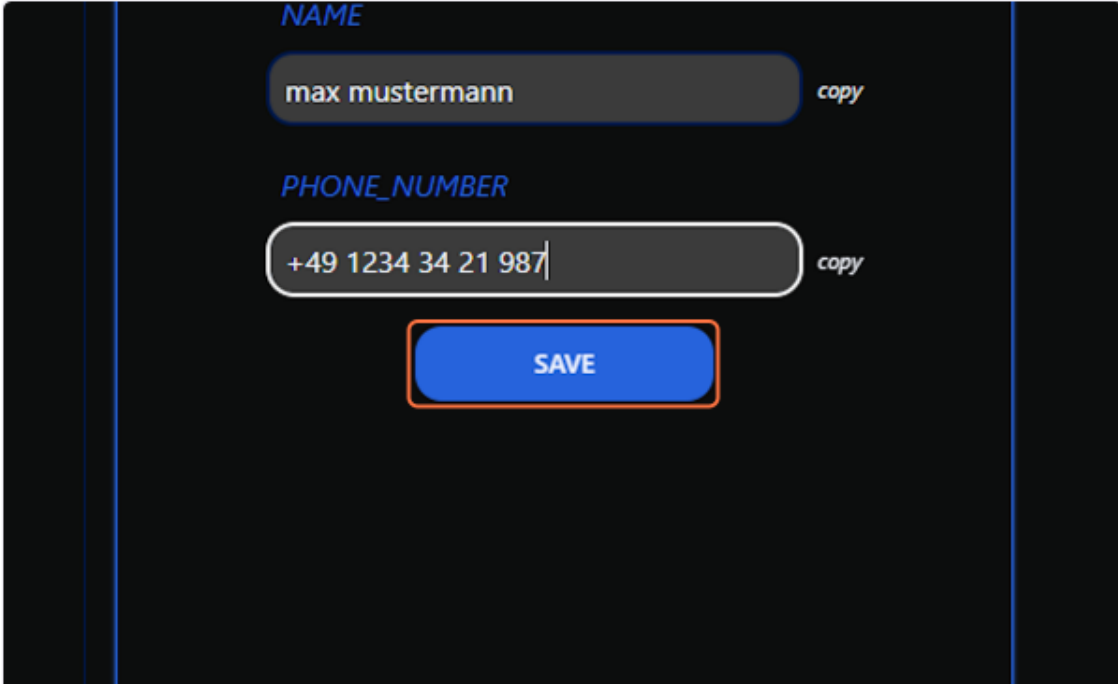
PHONE_NUMBER

+49 1234 34 21 987 copy

SAVE

Tango Created with Tango.us

7. Click on SAVE



A screenshot of a form with a dark background. The form has two input fields: one for 'NAME' containing 'max mustermann' and one for 'PHONE_NUMBER' containing '+49 1234 34 21 987'. Both fields have a 'copy' button to their right. Below the fields is a blue 'SAVE' button with an orange border. The form is framed by a light blue border.

NAME

max mustermann copy

PHONE_NUMBER

+49 1234 34 21 987 copy

SAVE

Tango Created with Tango.us

8. Click on id: 188f0991-2554-4fa6-a33f-0c0cc2e06c00...



A screenshot of a user management interface. On the left is a sidebar with a 'tiBusADMIN' header and a list of menu items: 'HBOARD', 'KINGS', 'ES', 'VERS', 'LOYEES', and 'MENTS'. The main area is divided into two columns. The left column, titled 'Neu', contains a list of users: 'Customer1' (id: fac06cdf-9037-4de1-aa45-2f716186fe5c), 'Customer2' (id: 7ef22ef9-ceb7-4599-b711-b09a8618bad8), and 'max mustermann' (id: 188f0991-2554-4fa6-a33f-0c0cc2e06c00). The 'max mustermann' entry is highlighted with an orange border. The right column, titled 'User #new', contains a form with fields for 'EMAIL', 'PASSWORD', and 'NAME', each with a corresponding input field.

tiBusADMIN

HBOARD

KINGS

ES

VERS

LOYEES

MENTS

Neu

id: fac06cdf-9037-4de1-aa45-2f716186fe5c

Customer1

id: 7ef22ef9-ceb7-4599-b711-b09a8618bad8

Customer2

id: 188f0991-2554-4fa6-a33f-0c0cc2e06c00

max mustermann

User #new

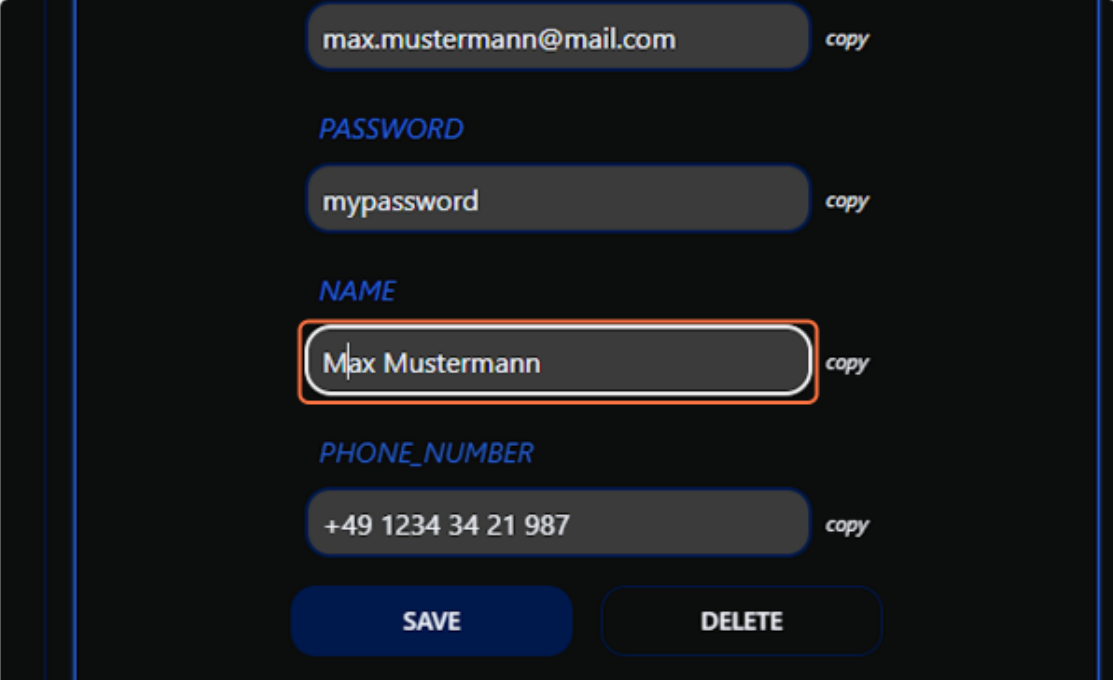
EMAIL

PASSWORD

NAME

Tango Created with Tango.us

9. Type "Max Mustermann"



max.mustermann@mail.com *copy*

PASSWORD

mypassword *copy*

NAME

Max Mustermann *copy*

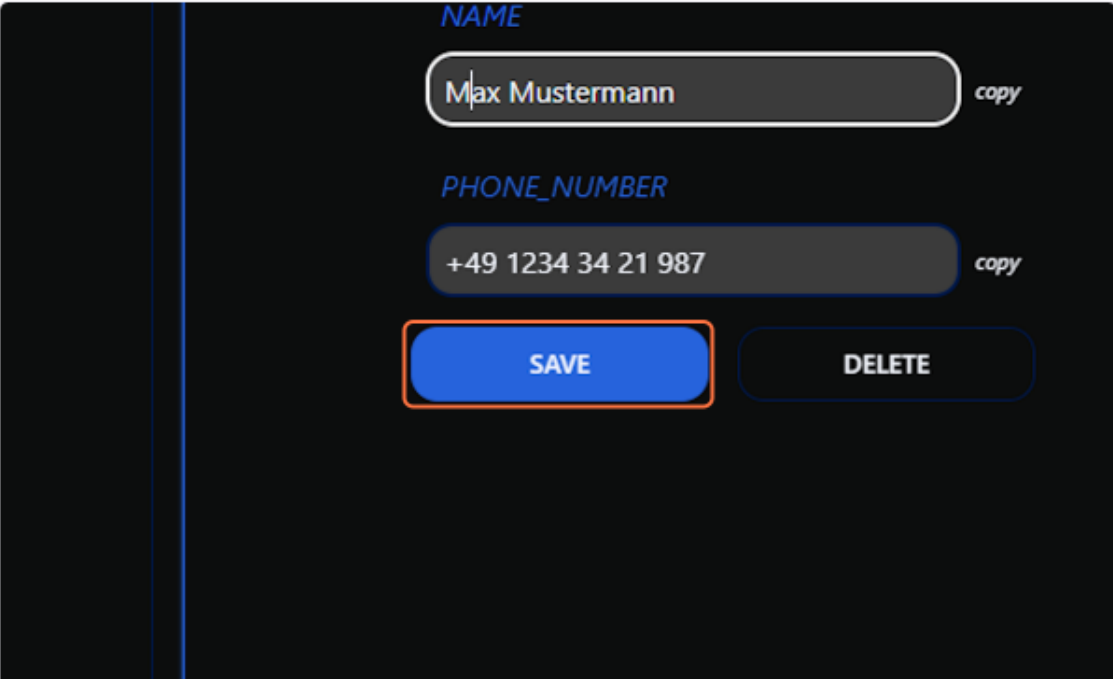
PHONE_NUMBER

+49 1234 34 21 987 *copy*

SAVE **DELETE**

Tango Created with Tango.us

10. Click on SAVE



NAME

Max Mustermann *copy*

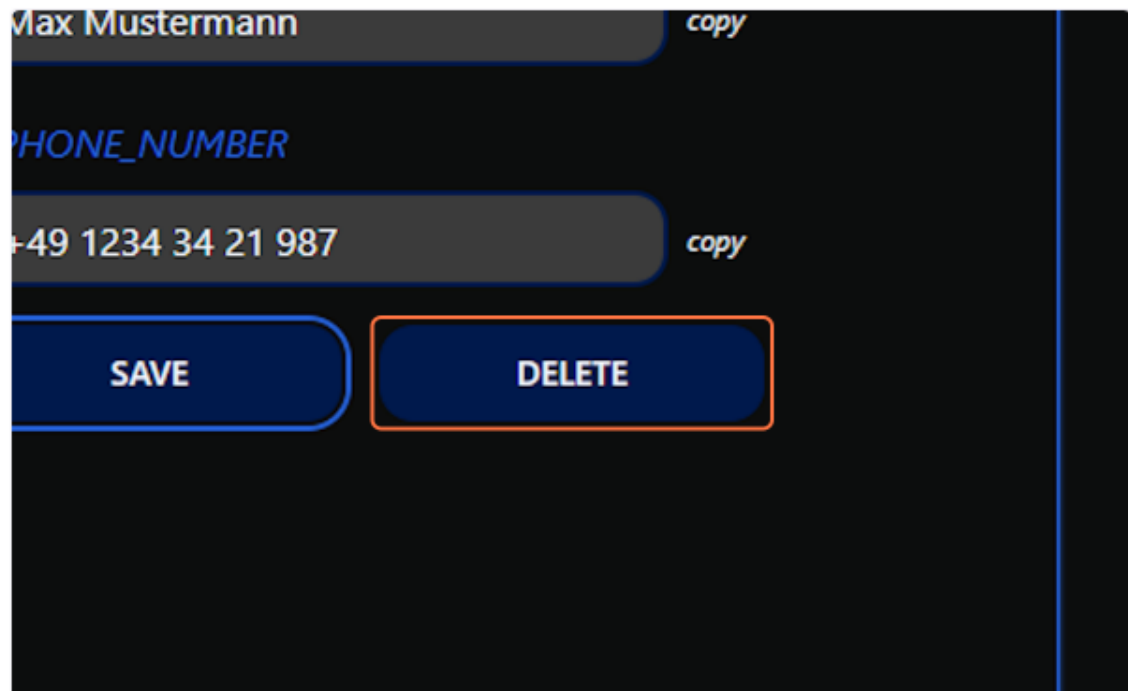
PHONE_NUMBER

+49 1234 34 21 987 *copy*

SAVE **DELETE**

Tango Created with Tango.us

11. Click on DELETE

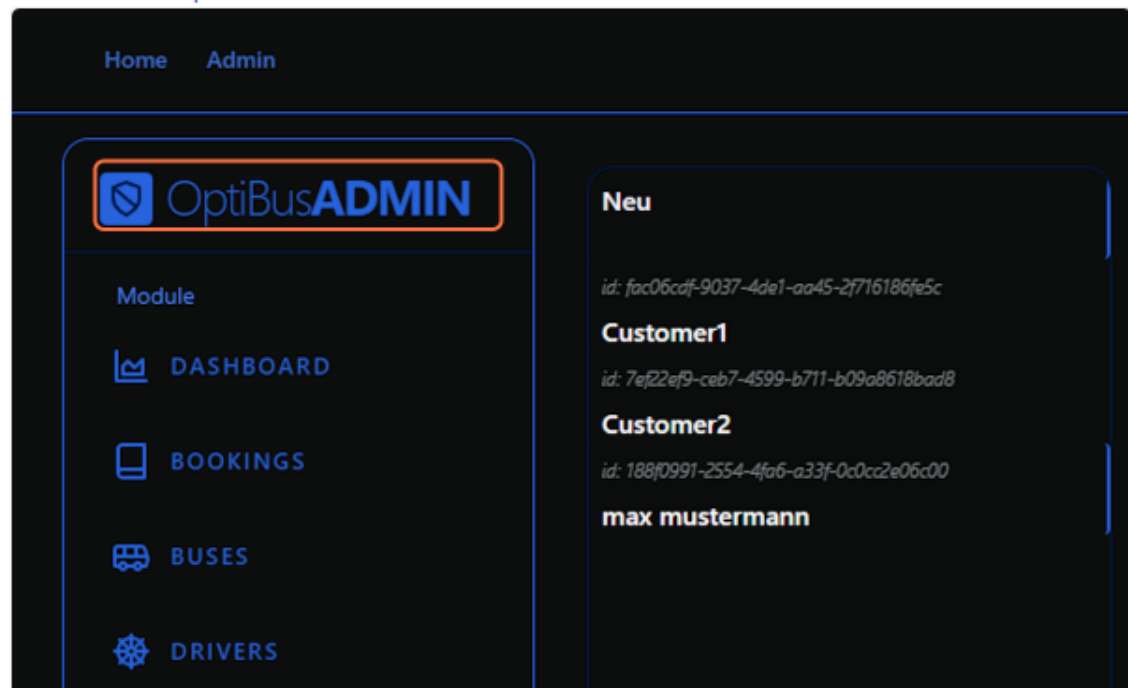


A screenshot of a user profile form. At the top, the name "Max Mustermann" is displayed in a dark grey box with a "copy" link to its right. Below this, the label "PHONE_NUMBER" is shown in blue. The phone number "+49 1234 34 21 987" is in a dark grey box with a "copy" link to its right. At the bottom, there are two buttons: "SAVE" and "DELETE". The "DELETE" button is highlighted with a red rectangular border.

Tango

Created with Tango.us

12. Click on OptiBus...

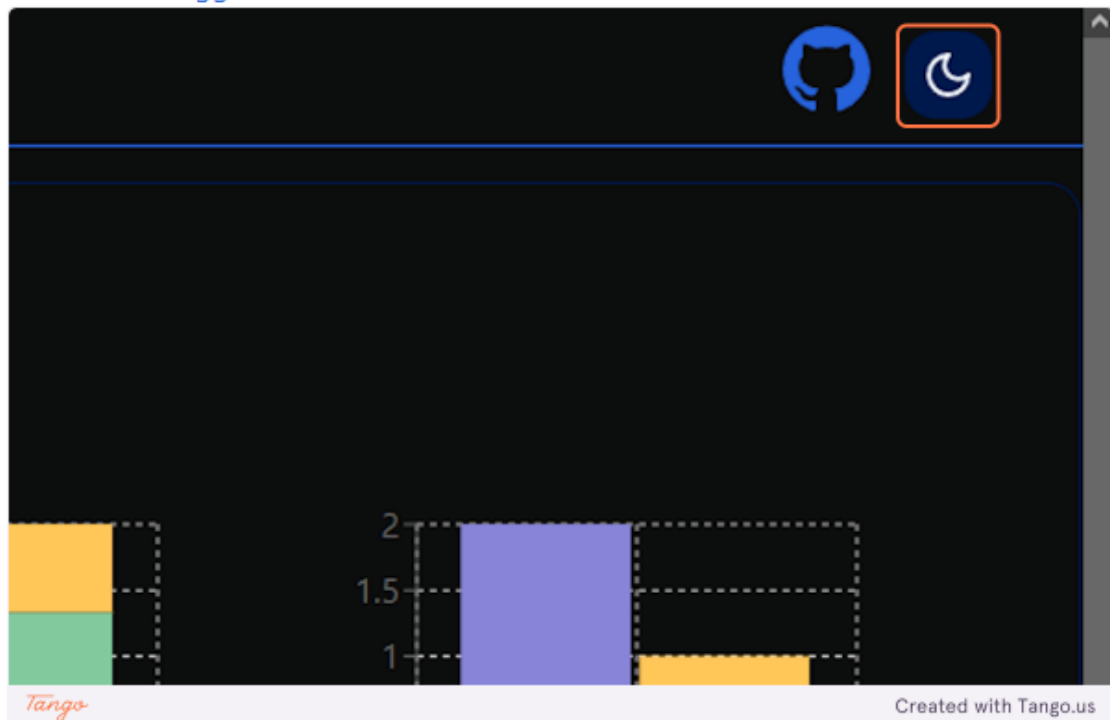


A screenshot of the OptiBusADMIN dashboard. The top navigation bar includes "Home" and "Admin". The main content area is divided into two columns. The left column, titled "Module", contains a list of menu items: "DASHBOARD", "BOOKINGS", "BUSES", and "DRIVERS", each with a corresponding icon. The right column, titled "Neu", displays a list of new entries. The first entry is "Customer1" with ID "fac06cdf-9037-4de1-aa45-2f716186fe5c". The second entry is "Customer2" with ID "7ef22ef9-ceb7-4599-b711-b09a8618bad8". The third entry is "max mustermann" with ID "188f0991-2554-4fa6-a33f-0c0cc2e06c00".

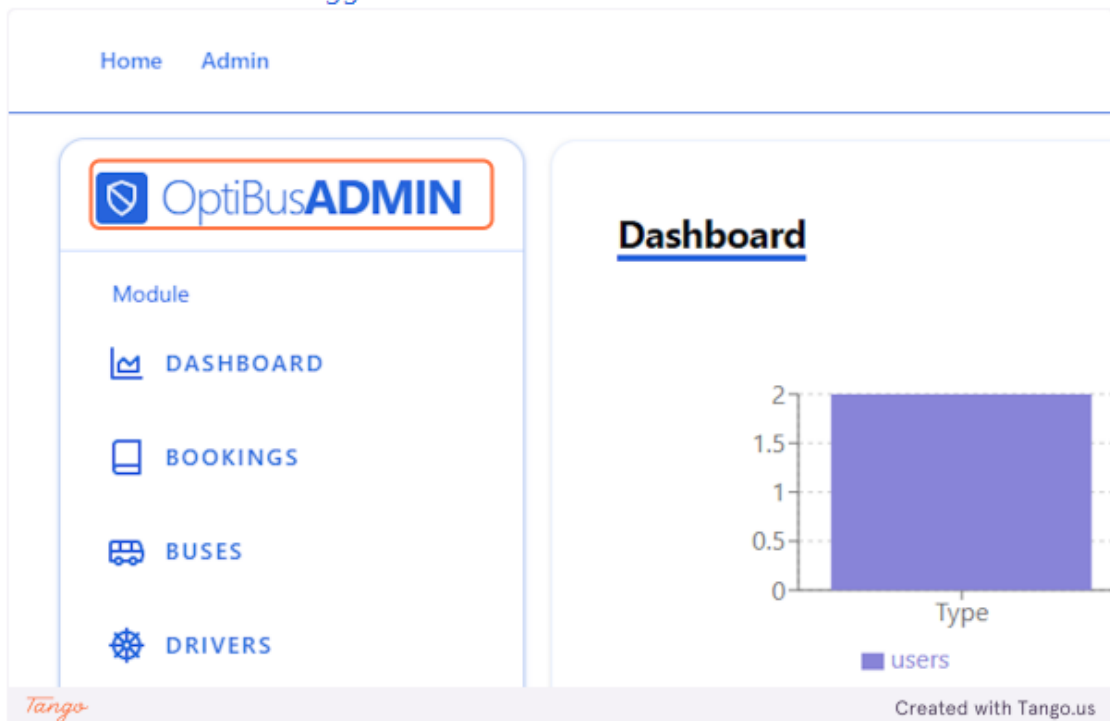
Tango

Created with Tango.us

13. Click on Toggle theme



14. Now the Theme is toggled



Codesnippets

Die folgenden Ausschnitte der Busse sind für alle anderen Entitäten gleichwertig und können in Zukunft einfach um weitere Entitäten erweitert werden.

Formular Beispiel eines Busses:

```
'use client';
import { FormInput } from '@components/core/form-input';
import { SubmitButton } from '@components/ui/SubmitButton';
import {
  deleteBus,
  getDefaultBus,
  insertBus,
  updateBus
} from '@lib/supabase/buses';
import React, { FC, useState } from 'react';
import { Form } from '../Form';
import { DeleteButton } from '@components/ui/DeleteButton';
import { toast } from 'react-toastify';

export interface BusProps {
  bus?: any;
}

const BusForm: FC<BusProps> = ({ bus }) => {
  const [formData, setFormData] = useState<any>(bus || getDefaultBus());

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    const promise = formData.id
      ? await updateBus(formData)
      : await insertBus(formData);
    if (promise) toast.success('Erfolgreich gespeichert');
    else toast.error('Fehler beim speichern');
  };

  const handleDelete = async (e: React.FormEvent) => {
    e.preventDefault();
    const promise = await deleteBus(formData.id);
    if (promise) toast.success('Erfolgreich gelöscht');
    else toast.error('Fehler beim löschen');
  };

  const handleInputChange = (key: string, value: any) => {
    setFormData({ ...formData, [key]: value });
  };
};
```

```

return (
  <Form
    onSubmit={handleSubmit}
    title={`Bus #${formData.id ? formData.id : 'new'}`}
  >
    {Object.keys(formData).map((key) => (
      <FormInput
        key={key}
        name={key}
        handleChange={handleInputChange}
        value={formData[key]}
        required
      />
    ))}

    <SubmitButton />

    {formData.id && <DeleteButton onClick={handleDelete} />}
  </Form>
);
};

export { BusForm };

```

Supabase API call:

```

import { UUID } from 'crypto';
import { supabase } from './supabase';
import { Database } from '@types/supabase';

export type Bus = Database['public']['Tables']['buses']['Row'];

function getDefaultBus() {
  return {
    bus_number: 0,
    model: 'Mercedes',
    year: 2000,
    seating_capacity: 50,
    is_available: false
  };
}

async function getBuses() {
  try {
    let { data, error } = await supabase.from('buses').select('*');

```

```
    if (error) {
      throw error;
    }

    return (data as Bus[]) || [];
  } catch (error: any) {
    console.error('Error fetching data:', error);
    return [];
  }
}

async function getBus(id: UUID) {
  try {
    const { data, error } = await supabase
      .from('buses')
      .select('*')
      .eq('id', id)
      .single();

    if (error) {
      throw error;
    }

    return data;
  } catch (error: any) {
    console.error('Error fetching data:', error);
    return false;
  }
}

async function updateBus(buses: any) {
  try {
    const { error } = await supabase
      .from('buses')
      .update(buses)
      .eq('id', buses.id);

    if (error) {
      throw error;
    }

    return true;
  } catch (error: any) {
    console.error('Error update data:', error);
    return false;
  }
}
```



```
}

async function insertBus(buses: any) {
  try {
    const { error } = await supabase.from('buses').insert(buses);

    if (error) {
      throw error;
    }

    return true;
  } catch (error: any) {
    console.error('Error insert data:', error);
    return false;
  }
}

async function deleteBus(id: UUID) {
  try {
    const { error } = await supabase.from('buses').delete().eq('id', id);

    if (error) {
      throw error;
    }

    return true;
  } catch (error: any) {
    console.error('Error delete data:', error);
    return false;
  }
}

export { getDefaultBus, getBuses, getBus, insertBus, updateBus, deleteBus };
};
```

Jest test:

```
import { v4 as uuidv4 } from 'uuid';
import {
  getDefaultBus,
  getBuses,
  getBus,
  insertBus,
  updateBus,
  deleteBus
```

```
} from './buses';
import { supabase } from './supabase';

jest.mock('./supabase', () => ({
  supabase: {
    from: jest.fn(() => ({
      select: jest.fn(() => ({
        eq: jest.fn(() => ({
          single: jest.fn()
        })
      })
    })),
    insert: jest.fn(),
    update: jest.fn(),
    delete: jest.fn()
  })
}));

describe('Bus functions', () => {
  const mockBus = {
    id: uuidv4(),
    bus_number: 123,
    model: 'Mercedes',
    year: 2000,
    seating_capacity: 50,
    is_available: false
  };

  it('should fetch buses', async () => {
    const mockData = [mockBus];
    const { data } = await getBuses();
    expect(data).toEqual(mockData);
  });

  it('should fetch a bus by id', async () => {
    const { data } = await getBus(mockBus.id);
    expect(data).toEqual(mockBus);
  });

  it('should insert a bus', async () => {
    const success = await insertBus(mockBus);
    expect(success).toBe(true);
  });

  it('should update a bus', async () => {
    const success = await updateBus(mockBus);
    expect(success).toBe(true);
  });
});
```

```
});  
  
it('should delete a bus', async () => {  
  const success = await deleteBus(mockBus.id);  
  expect(success).toBe(true);  
});  
});
```