Höhere Technische Bundes-Lehr- und Versuchsanstalt Villach

Abteilung für Informatik

Diplomarbeit

Truck-Routing

Arduvi

Eingereicht am 20. 04. 2021 von

Florian Eder 5BHIF

Geb. am 15.06.2001

Betreut und beurteilt von

Prof. Mag. Gerald Ortner

Eidesstattliche Erklärung

Ich versichere, dass ich diese Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen haben wir alle gekennzeichnet und im Literaturverzeichnis angeführt. Diese Arbeit wurde noch an keiner anderen Stelle zur Beurteilung eingereicht.

_____

Florian Eder, geb. am 15.06.2001

Villach, am 21. 04. 2021

Danksagung

An dieser Stelle möchte ich mich bei jenen Personen bedanken, welche mich beider vorliegende Arbeit unterstütz haben. Ohne diese, wäre dieses Projekt nicht in diesem Ausmaß und dieser Qualität zustande gekommen.

Danke sagen möchte ich an

- Herrn Martin Pichler, Geschäftsführer von Arduvi GmbH, für die Bereitstellung des Themas und die Unterstützung bei der Ausarbeitung, Planung und Durchführung des Projektes
- Prof. Mag. Gerald Ortner für die Betreuung der Diplomarbeit.
- meine Familie für die schulische Unterstützung in den letzten Jahren.
- alle Professoren der HTL-Villach für die lehrreichen Unterrichtsstunden.
- meine Mitschüler für den guten Zusammenhalt und die gute Zusammenarbeit während unserer gemeinsamen schulischen Laufbahn.

Abstract

Arduvi is an online B2B platform for timber sales. Since Arduvi's delivery system from the sawmills to the carpentries had some problems and usually resulted in high delivery costs for the wood processors, a system was built that would utilize the trucks in the best possible way, thus saving time and costs, as well as reducing the environmental impact.

According to the requirements, the carpentries, as well as the sawmills, should be offered a simple and user-friendly way to manage their orders and then create a route for trucks.

Difficulties were posed by the order of loading the wood and creating the route for a truck, as it cannot or is not allowed to drive on all available roads.

A truck route (Abbildung 4) is created for each order, which can then be merged with other truck routes under certain conditions. Furthermore, it is possible for the sawmill to publish a truck route so that other carpentries can order on this truck route.

Internally, a graph is created with duration and distance from one supplier to all carpentries and from all carpentries to all other carpentries within a radius of 800 km. (Abbildung 5) This helps to calculate the route and whether it meets the requirements, such as the maximum allowed driving time.

Kurzfassung

Arduvi ist eine Online B2B-Platform für Holzverkauf. Da das Liefersystem bei Arduvi von den Sägewerken zu den Zimmereien einige Probleme aufwies und es meist zu hohen Lieferkosten für die Holzverarbeiter kam, wurde ein System gebaut, welches die LKWs bestmöglich auslastet und somit Zeit und Kosten spart, sowie die Umweltbelastungen verringert.

Laut Anforderungen sollte den Zimmereien, wie auch den Sägewerken eine einfache und benutzerfreundliche Möglichkeit geboten werden, ihre Bestellungen zu verwalten und daraufhin eine Route für LKWs zu erstellen.

Schwierigkeiten stellte die Reihenfolge der Beladung und die Route eines LKWs dar, da er nicht auf allen verfügbaren Straßen fahren kann oder darf.

Zu jeder Bestellung wird eine Truck-Route angelegt, welche dann mit weiteren Truck-Routen unter bestimmten Bedingungen zusammengefügt werden kann. Weiteres, ist es dem Sägewerk möglich, eine Truck-Route zu veröffentlichen, damit weitere Zimmereien auf diese Truck-Route bestellen können.

Intern wird ein Graph mit Dauer und Distanz von einem Supplier zu allen Zimmereien und von allen Zimmereien zu alle anderen Zimmereien im Umkreis von 800 km aufgestellt. Dieser Hilft beim Berechnen der Route und ob diese den Bestimmungen, wie zum Beispiel der maximal erlaubten Fahrzeit, entspricht.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

1 Einleitung und Überblick

Da ich vor diesem Projekt bereits drei Ferialpraktika bei Arduvi GmbH absolviert habe, ist es mir sehr leichtgefallen, Zugang zu finden. Die Codebasis und deren Struktur war mir bereits bekannt und ich musste mich nur in einige Änderungen einarbeiten.

Nun möchte ich die Problemstellung genauer erläutern und auch eine kurze Beschreibung über Arduvi GmbH geben.

1.1 Problemstellung

Die Lieferung zu Bestellungen musste vor dieser Arbeit separat ausgemacht werden. Dies führte meist zu hohen Lieferkosten, da ein LKW meist mit nur einer Bestellung beladen wurde. Ein weiterer Aspekt ist die Umweltbelastung. Bei Lieferungen, speziell über längere Strecken, ist es wichtig diese auch bestmöglich auszulasten.

Als Versender einer Ware möchte ich die Lieferkosten so gering wie möglich halten, da ich das Produkt anschließend günstiger verkaufen kann. Ebenfalls

Für den Empfänger bedeutet dies eine möglicherweise schnellere Lieferung und weniger Lieferkosten, da sich mehrere Bestellungen in einem LKW befinden. Des Weiteren ist es wichtig ökologisch zu denken und zu handeln. Die Umwelt wird dadurch weniger belastet und der Schadstoffausstoß minimiert.

1.2 Arduvi und das Unternehmen

Arduvi GmbH wurde Ende 2017 gegründet und hat ihren Sitzt in der Steiermark. ([1]) Arduvi bietet einen Online B2B Beschaffungsplattform für Holzbaustoffe an, die sich einerseits an Lieferanten (z.B.: Holzindustrie, Sägewerke) und andererseits an entsprechende Verarbeiter (z.B.: Holzbaubetriebe, Fertighausindustrie, Bauunternehmen, Dachdecker, Metallindustrie) wendet.

Dieses Unternehmen stellt die passende Lösung zur Optimierung der Einkaufsprozesse für Holzbaustoffe zur Verfügung. Die Plattform verbindet Produzenten mit einer Vielzahl von Holzbaubetrieben und bündelt Bestell- und Abrechnungsprozesse an einem zentralen Ort.

Verarbeiter haben die Möglichkeit, ohne Zwischenhandel, direkt bei ihren bevorzugten Lieferanten zu bestellen und somit die Waren in ihrer gewünschten Qualität zu empfangen. Des Weiteren bietet Arduvi Möglichkeiten zur Kommunikation mit dem Vertragspartner und Hilfestellungen für die logistische Abwicklung von Bestellungen an.

2 Digitalisierung im Handel

Vor 30 Jahren war es unvorstellbar, Sachgüter oder Dienstleistungen über unser Smartphone oder unseren Computer zu erwerben. Mittlerweile ist es gang und gäbe, Käufe über unser Mobiltelefon abzuwickeln.

In den nächsten Unterkapiteln möchte nun über die Aspekte des digitalen Handels und die immer steigende Digitalisierung schreiben.

2.1 Der Online-Handel

„1989 gingen die ersten Webseiten ans Netz und nur kurze Zeit später wurden auch die ersten Online-Shops eröffnet. Zu Beginn waren das jedoch nur einfache Listen, die am ehesten mit den heutigen Zeitungsinseraten vergleichbar sind. Wer Interesse an einem Produkt hatte, musste sich also selbstständig mit dem Verkäufer in Verbindung setzen und weitere Informationen erfragen.“

(Quelle: [2], Die ersten Online-Shopps)

1994 wurde der wohl bekannteste Online-Shop von dem reichsten Mann der Welt, Jeff Bezos, in seiner Garage in Seattle gegründet. Alles startete als Bücherverleih, doch bereits vier Jahre später wurde der millionste Kunde gezählt. Heutzutage ist Amazon weltbekannt und zählt zu den größten Unternehmen der Welt.

([3], Unsere Geschichte: Was aus einer Garagen-Idee werden kann?)

Bald darauf kam auch schon die Internet-Auktionsplattform eBay ins Netz. Dort werden bis heute Waren von Firmen und Privatpersonen verkauft oder versteigert.

([4], eBay)

Ohne die Wohnung zu verlassen, ist es möglich, sich von der Zahnpaste bis zum Auto alles direkt nach Hause liefern zu lassen. Besonders in der Corona-Pandemie haben viele Leute diesen Service genutzt und die Größe des Online-Handels ist weiter angewachsen. Wie sollte man sich Sachgüter kaufen können, wenn zum einen die Geschäfte Großteiles geschlossen haben und man womöglich in Quarantäne ist? Andererseits hat diese Zeit auch vielen Menschen gezeigt, wie wichtig der Einkauf und die Beratung im stationären Handel ist.

2.1.1 Vorteile des Online-Handels

Der Online-Handel ermöglicht uns eine Vielzahl von Vorteilen und Erleichterungen für unser tägliches Leben. Ich möchte nun einige aufzählen und diese näher erläutern.

Bequemes, zeitlich und örtlich unabhängiges Einkaufen

Gemütlich von der Couch einen neuen Fernseher bestellen oder ohne viel Aufwand verschiedenste Artikel zu vergleichen ist dank des Online-Handels 24/7 möglich geworden. Es bietet viel Komfort und Erleichterung im Alltag, da es Zeit und teilweise auch Kosten wie Treibstoff oder Parkgebühren spart.

Möglichkeit für kranke oder beeinträchtigte Menschen

Beeinträchtigten oder kranken Menschen, welche zum Beispiel das Haus nicht verlassen können, wird eine Möglichkeit geboten, trotzdem ihre Ware selbst auszusuchen und diese dann in weiterer Folge auch zu bestellen. Viele Lebensmittelgeschäfte bieten einen Bestell- und Lieferdienst an, um den Einkauf direkt vor die Tür zu befördern.

Preisvergleich und große Auswahl verschiedenster Anbieter

Dank Online-Preisvergleichsportalen ist es sehr einfach und bequem die Preise eines Artikels, welcher bei verschiedenen Online-Anbieter zu erwerben ist, zu vergleichen und dadurch Kosten zu sparen. Ebenfalls ist es möglich, unterschiedliche Modelle miteinander zu vergleichen und darauf basierend eine Entscheidung zu fällen.

Im stationären Handel muss der Kunde ein Geschäft aufsuchen, sich beraten lassen und anhand der Fakten, welche der Verkäufer nennt, eine Kaufentscheidung fällen.

Ein Großteil der Menschen informiert sich über das jeweilige Produkt im Vorhinein und vergleicht auch dessen Preise bei verschiedensten Anbietern.

Verkaufsmöglichkeit für kleine Unternehmen

Startups oder kleine Unternehmen haben meist nicht die Ressourcen, um sich eine eigene Verkaufsfläche zu kaufen oder es würde sich schlicht weg nicht rentieren, da sich das Unternehmen auf den internationalen Markt fokussiert. Immer öfter werden Artikel nur noch ausschließlich über das Internet vertrieben, da es für die Unternehmer Geld, wie auch Zeit spart.

Des Weiteren können sich Firmen einen größeren Kundenstamm, weit über die Region, aufbauen und somit wachsen.

2.1.2 Nachteile des Online-Handels

Natürlich bringt der Online-Handel nicht nur Vorteile mit sich. Viele teilweise kleine selbstständige Unternehmer kämpfen unter dem Druck des Online-Handels.

Virtuelle Kontakt mit realen Waren

Eines der größten Nachteile ist der fehlende physische Kontakt zu den bestellten Sachgütern. Die Qualität oder Verarbeitung dieser, kann im Vorhinein nur sehr schwer bestimmt werden. Erst wenn das Produkt zu Hause ankommt, hat der Käufer die Möglichkeit, dieses zu Untersuchen. Mängel können erst zu diesem Zeitpunkt erkannt werden. Dies kann unnötige Kosten und Zeit des Kunden beanspruchen.

Nachteile für lokale und heimische Geschäfte

Aufgrund der großen Auswahl und Preisunterschiede wird der Online-Handel immer beliebter. Dadurch verlieren die lokalen Geschäfte Kunden, wie auch Umsatz. Oft läuft es darauf hinaus, dass diese schließen müssen und daraus folgend verlieren Leute ihre Arbeitsplätze.

2.2 Digitalisierung im stationären Handel

Damit der stationäre Handel weiter überleben kann, müssen diese ebenfalls in den Online-Handel einsteigen. Zum Beispiel, wie bereits genannt, die Lebensmittelgeschäfte, welche Lieferungen nach Hause anbieten.

Ein weiteres Beispiel wäre die Vorteilskarten auf einem Smartphone. So gut wie jeder größere Handel bietet mittlerweile eine Kundenkarten-App an, womit der Kunde seine Daten verwalten kann und gewisse Vorteile erhält.

Weiteres werden teilweise Newsletter oder Werbungen per E-Mail, anstatt per Post, verschickt. Dies Hilft der Umwelt und spart dem Unternehmen Kosten für den Druck und den Versand der Zeitschriften.

Eines ist gewiss: Der stationäre Handel steht in einer digitalen Transformation und muss mit dem Online-Handel mitziehen, um weiterhin wettbewerbsfähig zu bleiben.

2.3 Verbindung zu Arduvi

Arduvi ist ein moderner Online-Handel. Diese B2B Beschaffungsplattform für Holzbaustoffe verknüpft den Kunden direkt mit den Herstellern. Kunden können somit direkt mit den Sägewerken kommunizieren und dadurch auch genau absprechen, was sie benötigen.

Früher mussten Zimmereien bei einem Holzhändler die Waren erwerben. Der Holzhändler musste die Waren im Vorhinein bestellen oder die Bestellung einer Zimmerei den Sägewerken übergeben. Dies kann zu Fehler oder Missverständnissen führen. Des Weiteren ist die Handelsspanne meist hoch, da der stationäre Holzhändler Lagerkosten oder auch Verkaufsflächen zu bezahlen hat.

Bei Arduvi werden die Lieferungen beim Sägewerk abgeholt und auf direktem Wege den Zimmereien geliefert. Somit werden Lagerkosten gespart.

Ein weiterer Vorteil dieser Plattform ist die Nähe von Sägewerken und Kunden. Diese erhalten Feedback oder Änderungsvorschläge direkt von ihren Abnehmern.

3 LKW-Fahrverbote und sonstige Einschränkungen

Für die Erstellung von LKW-Routen müssen natürlich auch die vorhandenen Bedingungen, wie zum Beispiel das maximal zugelassene Gesamtgewicht oder die Pausenregelungen berücksichtigt werden.

3.1 LKW-Arten

Es gibt eine Vielzahl an unterschiedlichen LKW-Arten. Generell können diese Typen in fünf Klassen unterschieden werden.

Fahrzeugklasse	zulässiges Gesamtgewicht	Beispiele für die Nutzung
N1 - Leichte Nutzfahrzeuge	bis 3,49 t	Lieferfahrzeuge
N2 - Leichte LKW	3,5 t bis 11,49 t	Lieferungen im Nah- und Regionalverkehr
N3 - Schwere LKW	ab 12 t	Baustellenverkehr, Güterfernverkehr

([5], Welche LKW-Arten gibt es?)

3.2 Ruhezeiten und Ruhepause

Der Fahrer eines LKWs muss per Gesetz bestimmte Pausenzeiten und die maximale Fahrzeit einhalten. Natürlich ist dies auch enorm wichtig für dieses Projekt, da ansonsten die Routen teilweise gesetzwidrig wären.

3.2.1 Tägliche und wöchentliche Ruhezeit

Die tägliche und wöchentliche Ruhezeit musste nicht beachtet werden, da eine Lieferung eines LKWs immer nur für einen Tag bestimmt ist.

3.2.2 Ruhepause

Da in unserem Projekt eine Tagesarbeitszeit bis maximal 10 Stunden nicht ausgeschlossen wird, muss nach mindestens 4,5 Stunden eine Ruhepause von mindestens 45 Minuten eingehalten werden. Dies muss nicht am Stück passieren, jedoch muss der erste Teil mindestens 15 Minuten und der zweite Teil mindestens 30 Minuten betragen.

([6], Lenkpausen)

3.3 Fahrverbote

Natürlich gibt es unterschiedliche Fahrverbote für unterschiedliche Fahrzeugtypen. Bei Lastkraftfahrzeugen ist dies strenger geregelt als bei Personenkraftwagen.

3.3.1 Nachtfahrverbot

Grundsätzlich gilt in ganz Österreich ein Nachtfahrverbot auf allen Straßen von 22 bis 5 Uhr für LKWs, welche mit mehr als 7,5 Tonnen höchstzulässiges Gesamtgewicht eingetragen sind.

Ausgenommen von dieser Regelung sind Fahrzeuge des Straßendienstes, Bundesheeres oder auch der Feuerwehr.

([7], Nachtfahrverbot in ganz Österreich)

3.3.2 Wochenendfahrverbot

Das Wochenendfahrverbot auf allen Straßen Österreichs beginnt samstags ab 15 Uhr und endet sonntags um 22 Uhr. An gesetzlichen Feiertagen beginnt diese bereits um 0 Uhr und dauert bis 22 Uhr an.

Dies gilt für LKW mit Anhänger, wenn das Höchstzulässige Gesamtgewicht 3,5 t übersteigt und für LKW, Sattelkraftfahrzeuge und selbstfahrende Arbeitsmaschinen mit einem höchstzulässigen Gesamtgewicht von mehr als 7,5 t. Einige Ausnahmen erlauben trotz der Regelung Wochenendfahrten.

([7], Wochenendfahrverbot in ganz Österreich für alle Straßen)

3.3.3 Sonstige Fahrverbote

Weiteres sind Fahrverbote auf Straßen zu berücksichtigen, welche nicht für einen Lastkraftwagen gebaut sind und Fahrverbote laut des Fahrverbotskalender.

([7], Fahrverbote zur Verhinderung des Maut-Ausweichverkehrs, Fahrverbotskalender 2020)

3.4 Gesamtgewicht

Das höchstzulässige Gesamtgewicht ist an die Fahrzeugklasse gebunden. Das Gewicht der Ladung muss deshalb im Vorhinein berechnet und dann überprüft werden, ob es für diese LKW-Art zulässig ist. Bei einer maßgeblichen Übertretung des höchstzulässigen Gesamtgewichts kann die Weiterfahrt untersagt werden. Ebenfalls wirkt es sich auf das Fahrverhalten des Fahrzeuges aus und kann zum Beispiel den Bremsweg maßgeblich verlängern.

([8], Überladung von Lkw, Pkw und Anhänger)

4 Verwendete Technologien

In diesem Kapitel stelle ich die verwendeten Technologien für mein Projekt vor.

Die Anzahl an verschiedenen Möglichkeiten ein Projekt umzusetzen, ist in keiner Branche so hoch, wie in der Informatik. Es ist wichtig, von Beginn an Informationen über potenzielle Lösungen zu suchen und diese zu evaluieren.

4.1 Backend

Serverseitig ist die Logik implementiert. Die richtige Auswahl des Backends und deren Aufbau ist enorm wichtig für einen reibungslosen Ablauf. Falsche Entscheidungen können enorme Folgen auslösen.

Die Struktur des Backends war bereits vorhanden und auf dieser wurde aufgebaut.

4.1.1 ASP.NET Core

Active Server Pages .NET Core, kurz ASP.NET Core, ist der Nachfolger von ASP.NET und wird von Microsoft entwickelt. Dieses kostenlose Open-Source-Webframework bietet die Möglichkeit plattformunabhängige Applikationen zu entwickeln. Es bietet eine modulare Software-Entwicklung, da eine Vielzahl von fertigen Paketen verschiedenster Art zur Verfügung stehen.

([9], ASP.NET Core)

Da dieses Framework von Arduvi bereits verwendet wurde, ist mir die Entscheidung, welches Webframework für meine Diplomarbeit in Frage kommen würde, abgenommen worden.

4.1.2 Datenspeicherung

Die Datenverwaltung, wie auch die Datenspeicherung ist ein zentraler Bestandteil einer Applikation. Sie dient dazu, Inhalte permanent abzuspeichern, um diese jederzeit wieder abrufen zu können. Es gibt verschiedene Methoden, wie Daten abgespeichert werden können. In meinem Fall habe ich den Cloud-Dienst Azure verwendet.

Azure Cosmos DB

Azure Cosmos DB ist ein global verteiltes, schema-freies und horizontal skalierbares Datenbankservice aus dem Hause Microsoft. Das Datenbankservice ist Teil der Cloud-Computing-Plattform namens Azure. Mithilfe verschiedener APIs können Daten bestmöglich verwaltet werden.

([10], Cosmos DB)

Da der Cloud-Dienst Azure komplett in die Applikation integriert ist, war es selbstverständlich, diesen auch zu nutzen.

Azure Cosmos DB SQL-API

Mithilfe der SQL-API können Daten dokumentenbasiert im Format JSON in die Azure Cosmos DB eingetragen werden. Dies erfolgt in sogenannten Dokumenten. Weiteres können diese Dokumente in Partitionen unterteilt werden. Die einzelnen Einträge innerhalb eines Dokumentes werden Items genannt. Dokumente werden in einem Container abgespeichert und Container wiederum in Datenbanken.

([11], Hierarchical data)

Die Schnittstelle zwischen Applikation und Datenbank war bereits vorhanden und es mussten lediglich die Methoden für die Verwaltung der Routendaten implementiert werden.

Azure Cosmos DB Gremlin-API

Die Gremlin-API, welche auf die Azure Cosmos DB zugreift, wird verwendet, um Graphen zu persistieren. Meist wird es benutzt, wenn die Beziehung zwischen Entitäten eine große Rolle spielt. Die sogenannten Property-Graphen bestehen aus folgenden Elementen:

- Knoten (Vertex)
- Kanten (Edge)
- Labels
- Eigenschaften (Properties)

Grundsätzlich besteht der Graph aus Vertices und Edges. Mithilfe des Labels wird ihnen ein Name zugeteilt. Dieser bestimmt den Typ eines Vertex bzw. einer Edge. Properties bestehen aus Key-Value-Pairs, welche einem Vertex oder einer Edge angehören.

([12], Ein paar Grundlagen)

Da die Schnittstelle zwischen Applikation und Datenbank noch nicht vorhanden war, musste diese komplett neu implementiert werden.

4.2 Client

Die Darstellung und Strukturierung der Daten wird clientseitig vollzogen. Dazu wurde der sogenannte "living standard" verwendet. Dieser beschreibt die Kombination aus HTML, JavaScript und CSS und wird vom World Wide Web Consortium (W3C) ständig weiterentwickelt.

4.2.1 HTML

Die Hypertext Markup Language, kurz HTML, ist eine Auszeichnungssprache zum Darstellen und Strukturieren von Elementen wie zum Beispiel Paragrafen, Listen, Tabellen oder auch Buttons.

Sie ist die Sprache des World Wide Webs und wird von allen gängigen Browsern unterstützt. Allerdings dient HTML rein zur Strukturierung, denn die visuelle Darstellung wird mit CSS umgesetzt. ([13], Hypertext Markup Language)

4.2.2 CSS

Cascading Style Sheets, kurz CSS, wird zur visuellen Gestaltung von HTML- oder XML-Code verwendet. Damit sollte das Aussehen von der Strukturierung der Inhalte getrennt werden. Mittels CSS können zum Beispiel Farben angepasst, Animationen eingefügt oder auch Abstände zwischen Elementen werden.

([14], Cascading Style Sheets)

4.2.3 JavaScript

Die Skriptsprache JavaScript, kurz JS, welche dynamisch typisiert, objektorientiert und klassenlos ist. Ursprünglich wurde sie im Jahr 1995 von Netscape entwickelt, um dynamische Webseiten zu bauen. Mittlerweile allerdings kommt sie auch serverseitig oder für Microcontroller Einsatz. Objektorientiertes, prozedurales oder auch funktionales programmieren ist mit JavaScript möglich.

([15], JavaScript)

4.2.4 Razor Pages

Razor ist eine Markup-Syntax, welche die Möglichkeit bietet, serverseitigen Code in Webseiten zu integrieren. Es ist somit keine Programmiersprache, sondern eine serverseitige Auszeichnungssprache. Üblicherweise wird ´.cshtml´ als Dateiendung verwendet. Bevor die Website an den Browser gesendet wird, wird der serverseitige Code (meist C#) ausgeführt.

([16], ASP.NET Razor
[17], Was sind ASP.NET Web Pages bzw. die Razor Syntax?)

4.2.5 Template Inspinia Admin Theme

Das Inspinia Admin Theme ist ein responsives Admin-Dashboard-Theme, das auf dem Bootstrap 4.x Framework aufbaut. Eine Vielzahl an Webframeworks und HTML-Templates wird somit bereits mitgeliefert.

([18], Introduction)

4.2.6 Weitere Frameworks und Libraries

Natürlich reichen diese fünf oben genannten Technologien aus, um eine voll fortschrittliche, voll funktionsfähiges Frontend zu bauen. Es gibt allerdings eine Vielzahl an JavaScript Libraries, welche einem viel Arbeit abnehmen und somit Zeit eingespart wird. Nun möchte ich einige der wichtigsten Bibliotheken für mein Projekt aufzählen.

Bootstrap

Bootstrap ist eines der beliebtesten open-source Frameworks für die Entwicklung von modernen Webseiten mittels HTML, CSS und JS. Eines der wichtigsten Features sind Media-Queries, die für die responsive Darstellung eingesetzt werden können.

([19], Für alle, überall.)

jQuery

jQuery ist eine umfangreiche JavaScript Library für einfacheres und schnelleres Zugreifen auf den HTML Dombaum. Viele JavaScript Frameworks bauen auf jQuery und somit ist es in einer Vielzahl von Projekten vertreten.

([20], What is jQuery?)

5 Projektaufbau und Implementierung

Der Projektaufbau gliedert sich grob in vier verschiede Layer. Jeder Bereich hat seine eigenen klar strukturierten Aufgaben und kommuniziert mit dem darüber- und dem darunterliegenden Layer.

In diesem Kapitel möchte ich den Projektaufbau, sowie einige Teile meines Quellcodes genauer erläutern und beschreiben.

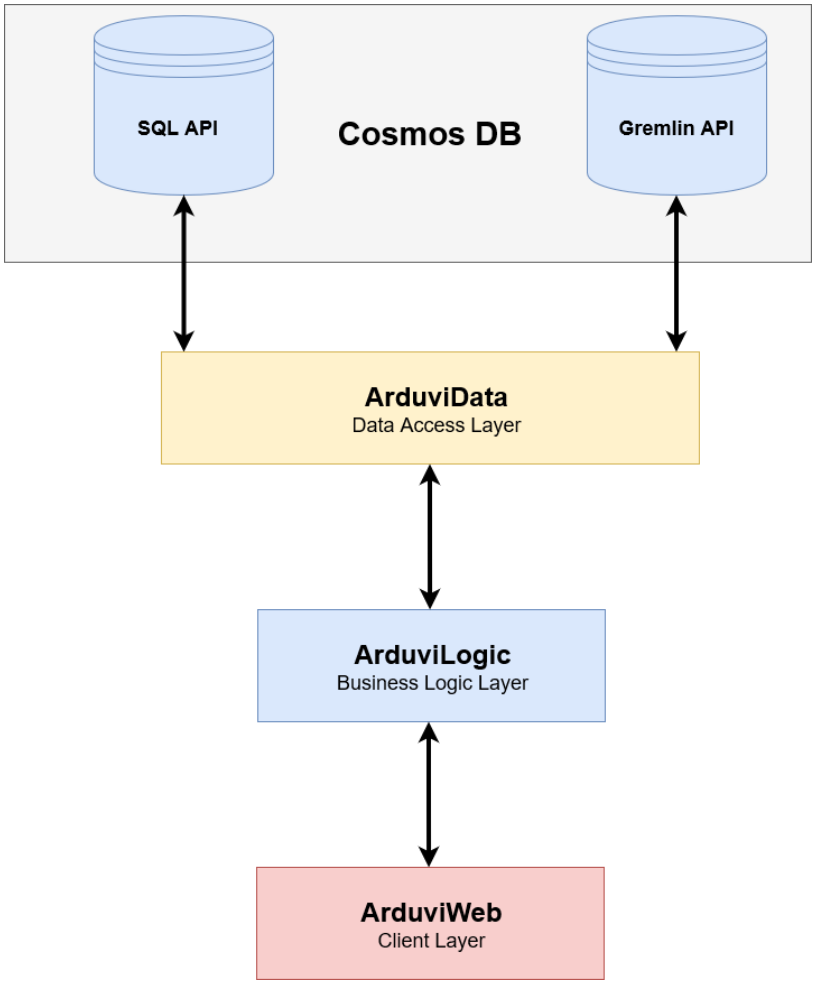


Abbildung 1: Einfache dargestellte Layer Architecture des gesamten Projektes

5.1 Persistence Layer (Cosmos DB)

Im Persistence Layer befinden sich die zwei verwendeten Datenbanken, welche für das Projekt verwendet wurden. Wie bereits im Abschnitt 4.1.2 Datenspeicherung erwähnt, biete Cosmos DB mehrere unterschiedliche Datenbank APIs an. Anstatt diese zu erklären möchte ich erläutern, warum ich mich für diese entschieden habe und was ihr Verwendungszweck innerhalb dieses Projektes ist.

5.1.1 SQL-API

Sobald eine neue Bestellung getätigt wird, wird automatisch eine Truck-Route (Abbildung 4) erstellt und in der Datenbank persistiert. Die Anbindung der Datenbank an den Data-Access-Layer wird unter Punkt 5.2 Data Access Layer (ArduviData) genauer erklärt.

5.1.2 Gremlin-API

Mithilfe der Gremlin-API können Graphen in der Cosmos DB abgespeichert werden.

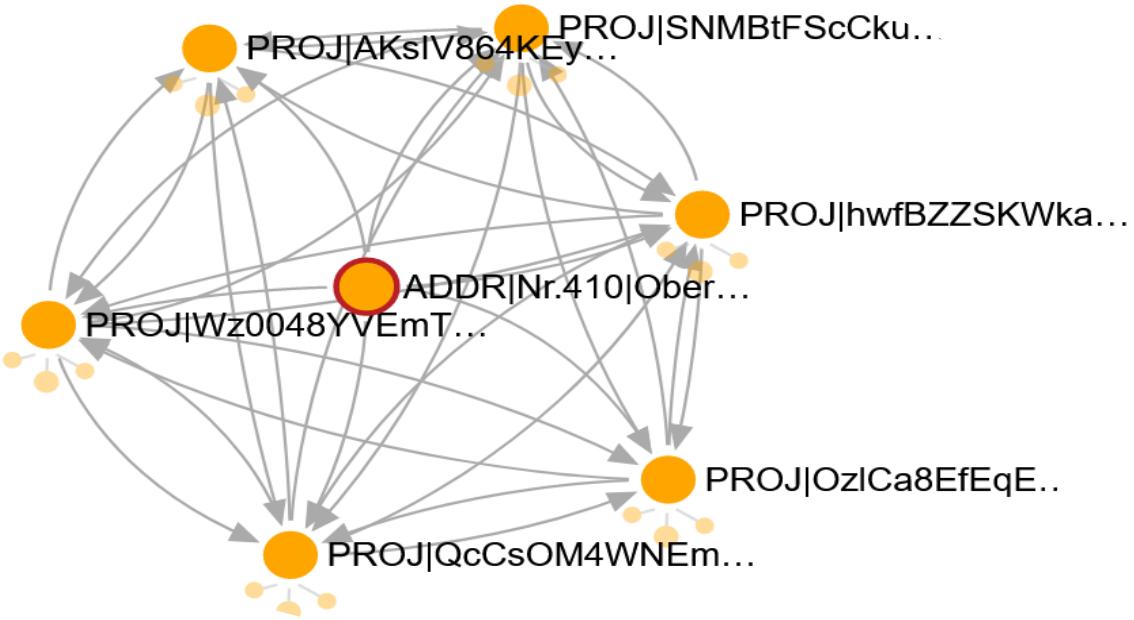
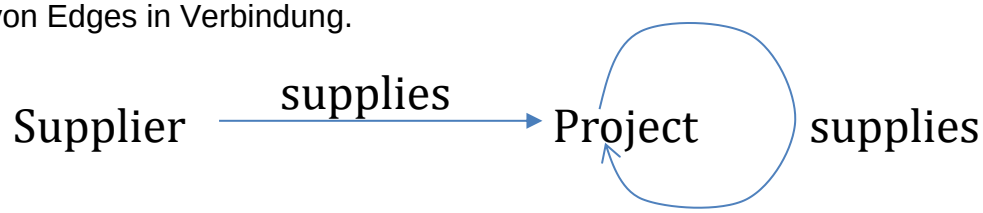


Abbildung 2: Visualisierter Graph

Der Daten-Explorer von Azure bietet die Möglichkeit den Graphen zu visualisieren. Weiteres können damit Abfragen und Manipulationen durchgeführt werden. Die Vertices sind mit gelben Punkten dargestellt. Jeder Vertex steht mit jedem anderen Vertex mithilfe von Edges in Verbindung.



Ein Lieferant beliefert Projekte und diese Projekte wiederum haben Verbindungen zu allen weiteren Projekten, damit die Strecke und Dauer der Route berechnet werden kann. Somit kann mithilfe eines Algorithmus bestimmt werden, ob eine Route den LKW-Bedingungen entspricht und weiteres, welche umliegenden Projekte noch eine Bestellung zur Route hinzufügen könnten.

5.2 Data Access Layer (ArduviData)

Der Data Access Layer kümmert sich um die Datenabfrage. Er bietet der Businesslogik Methoden an, um Daten in die Datenbank einzupflegen, zu ändern, zu löschen oder abzufragen. Des Weiteren werden alle Datenbank Models, welche für die Logik wichtig sind im DAL abgespeichert. Dieser sollte so konzipiert sein, dass der Persistence Layer ohne viel Aufwand beliebig erweitert werden kann.

5.2.1 Models

Es muss zwischen zwei verschiedenen Models unterschieden werden. Bei Abbildung 4 wird das Model für die SQL-API erläutert und bei Abbildung 5 das Model für die Gremlin-API.

Truck-Route

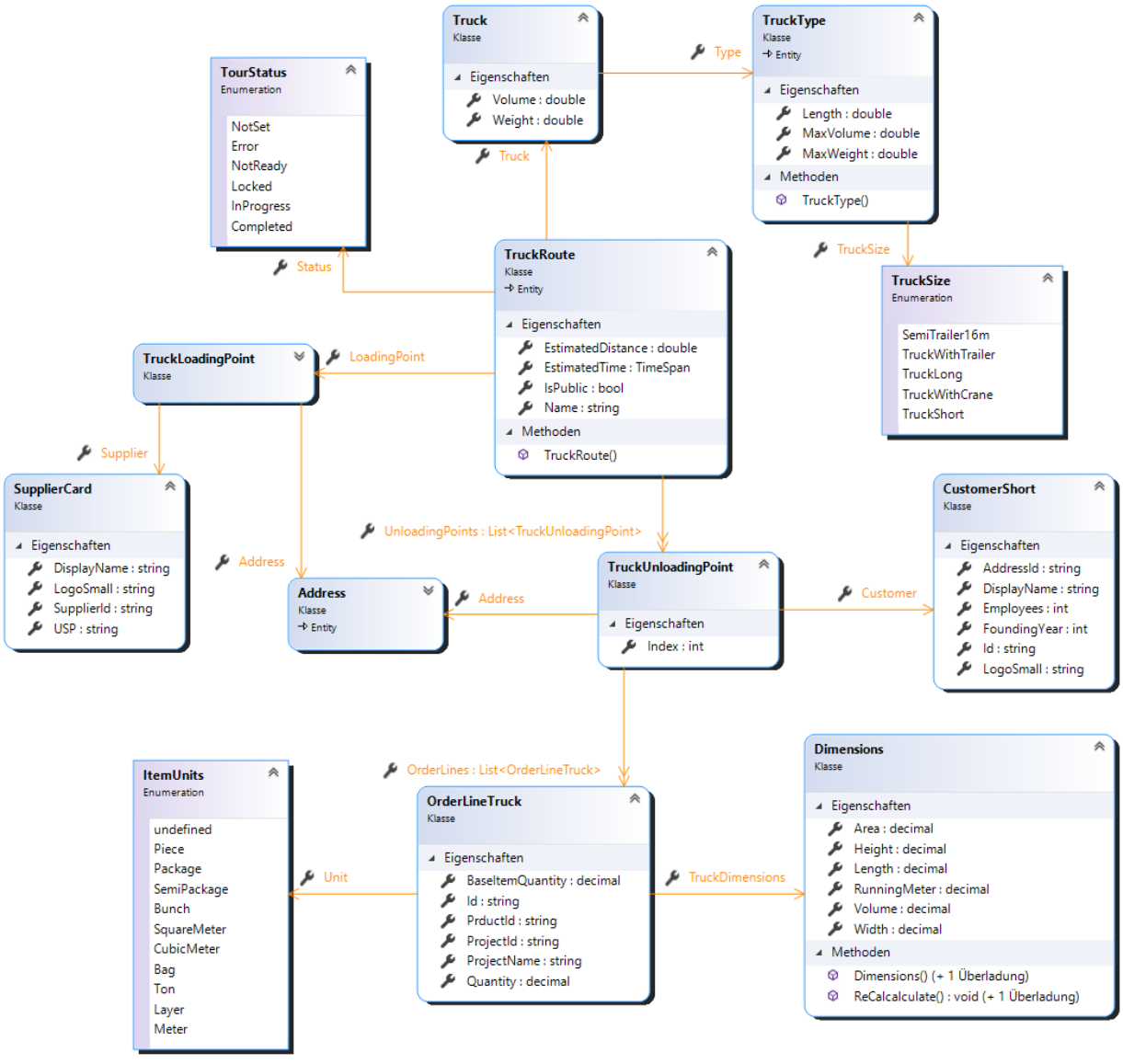


Abbildung 4: Klassendiagramm einer Truck-Route

Diese Abbildung zeigt alle Models in Verbindung mit einer Route. Der zentrale Teil ist die *TruckRoute.* Eine *TruckRoute* speichert bereits die geschätzte Distanz, sowie die geschätzte Fahrzeit. Weiteres beinhaltet sie einen Status, wie auch den LKW und weitere Details zum LKW. Außerdem ist ein Aufladepunkt und maximal vier Abladepunkte enthalten. Diese besitzen jeweils eine Adresse und Informationen zu den dazugehörigen Unternehmen. Weiteres speichert jeder Abladepunkt eine Menge von Bestellungen und Details, wie zum Beispiel Volumen und Gewicht.

Models für den Graphen

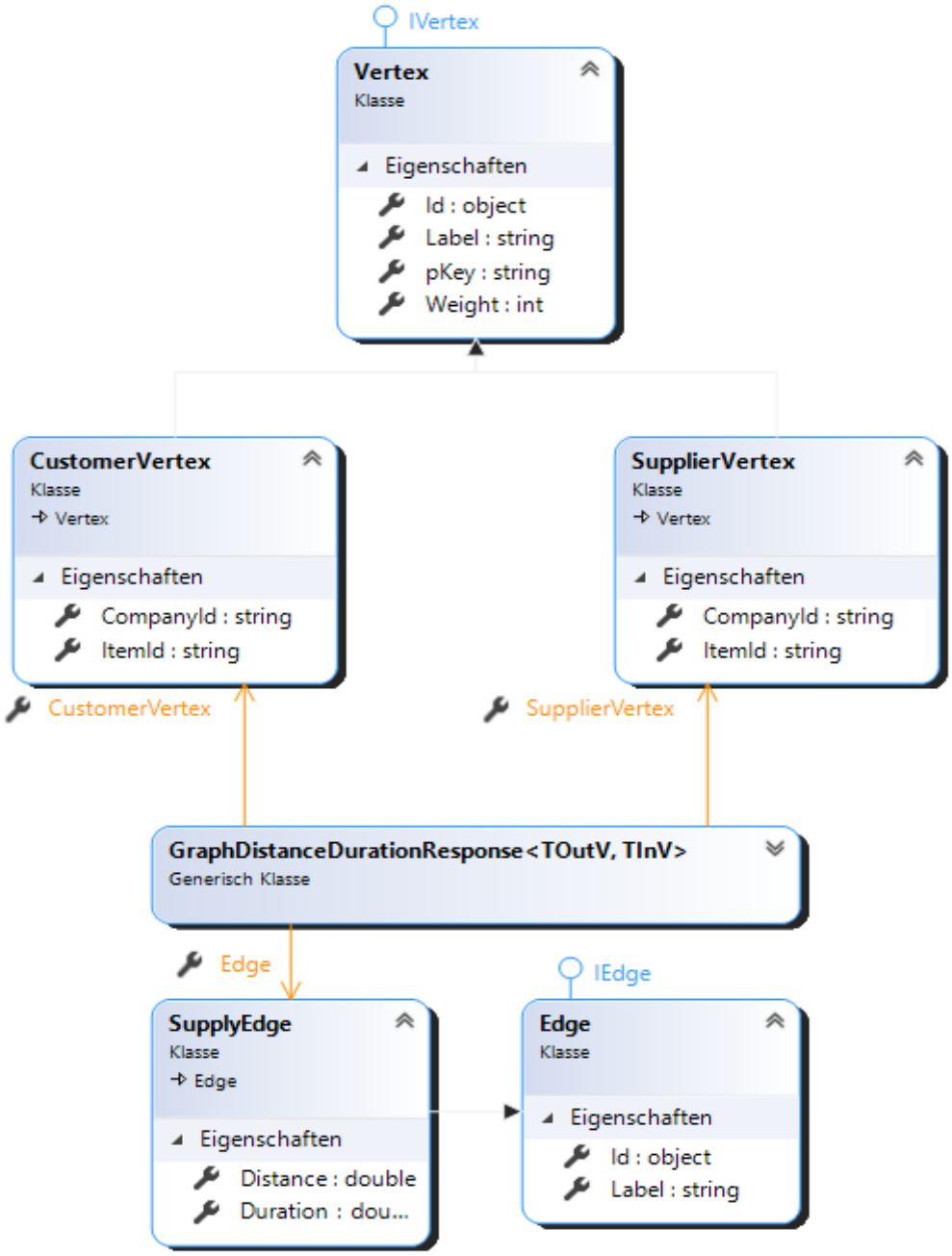


Abbildung 5: Klassendiagramm des Graphen

Für die Anbindung an die Datenbank wurden die Pakete ExRam.Gremlinq.Core und ExRam.Gremlinq.Providers.CosmosDb verwendet. Dieses stellt auch die Interfaces *IVertex* und *IEdge* bereit. Generell wird zwischen *SupplierVertex* und *CustomerVertex* unterschieden. Diese Erben von , welcher wiederrum das Interface *IVertex*

implementiert. Der *CustomerVertex* wird für die Projekte der Holzverarbeiter verwendet. Standorte der Sägewerke sind im *SupplierVertex* vermerkt. Die *SupplyEdge* speichert die Wegstrecke und Fahrzeit zwischen zwei Vertices.

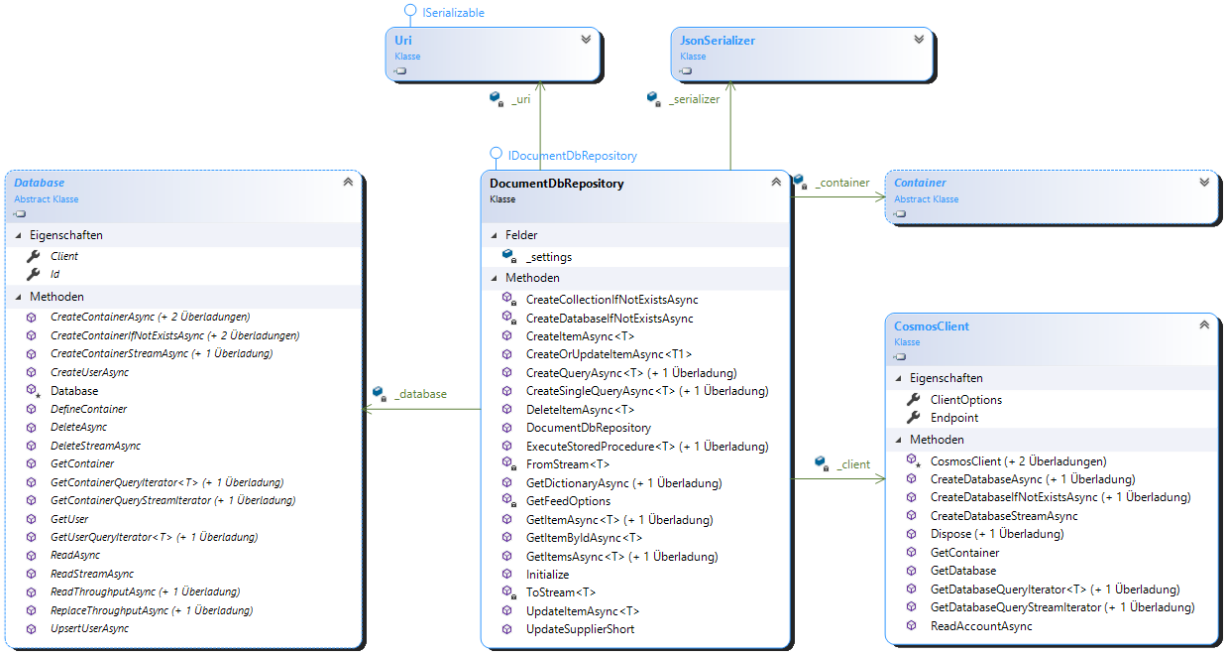
Die Vertices und die *SupplyEdge* werden im *GraphDistanceDurationResponse* Objekt zusammengeführt, welches im weiteren Verlauf an die Logik weitergegeben wird.

5.2.2 Services

Die Services im Data-Access-Layer kümmern sich um die Anbindung an die Datenbanken, das Versenden von E-Mails oder auch das Cachen von Daten in Redis.

SQL-API Anbindung

Das Service *DocumentDbRepository* stellt die Schnittstelle zur Cosmos DB SQL-API her. Es beinhaltet Methoden zur Abfrage und Manipulation von Daten. Dies war bereits fertig implementiert und wird für das Abspeichern und Verwalten der *TruckRoute* verwendet.



Gremlin-API Anbindung

Der *GraphDbClient* wurde komplett neu implementiert, da davor noch keine Graphdatenbank verwendet wurde. Er managt das Abfragen, Hinzufügen und Löschen von Graphen, Vertices und Edges. *GeremlinQuerySource* vom Paket ExRam.Gremlinq.Core liefert die notwendigen Methoden, für das Verwalten von Graphen.

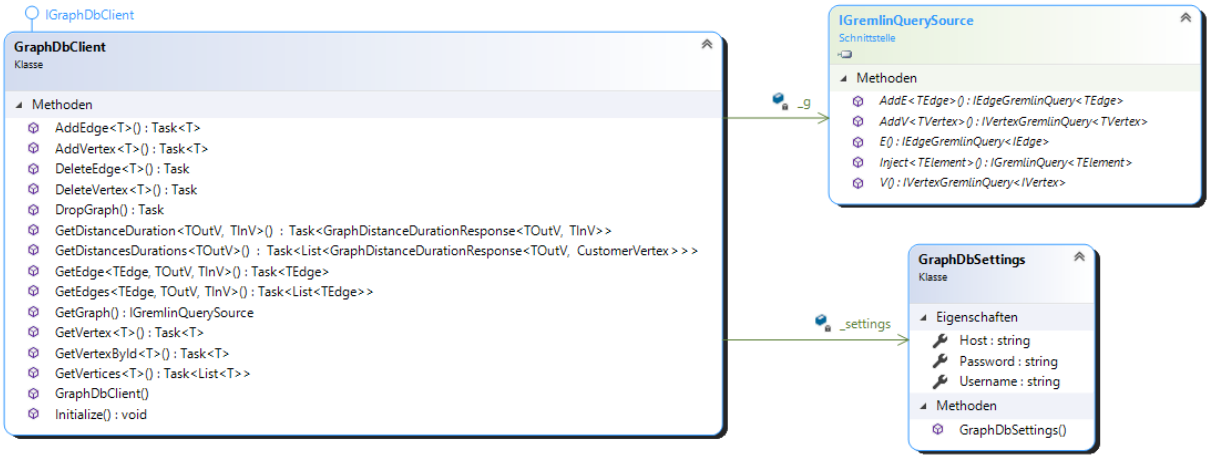


Abbildung 7: Klassendiagramm vom GraphDbClient

5.2.3 Data-Access-Klassen

Die Data-Access-Klassen stellen die Methoden für das Empfangen und Bearbeiten der Daten von Models zur Verfügung. Jedes Model hat eine eigene Data-Access-Klasse. Die Services, wie *DocumentDbRepository* und *GraphDbClient* werden verwendet, um auf die Datenbank zuzugreifen. Als Beispiel möchte ich *SupplierVertexAccess* und *OfferAccess* aufzeigen.

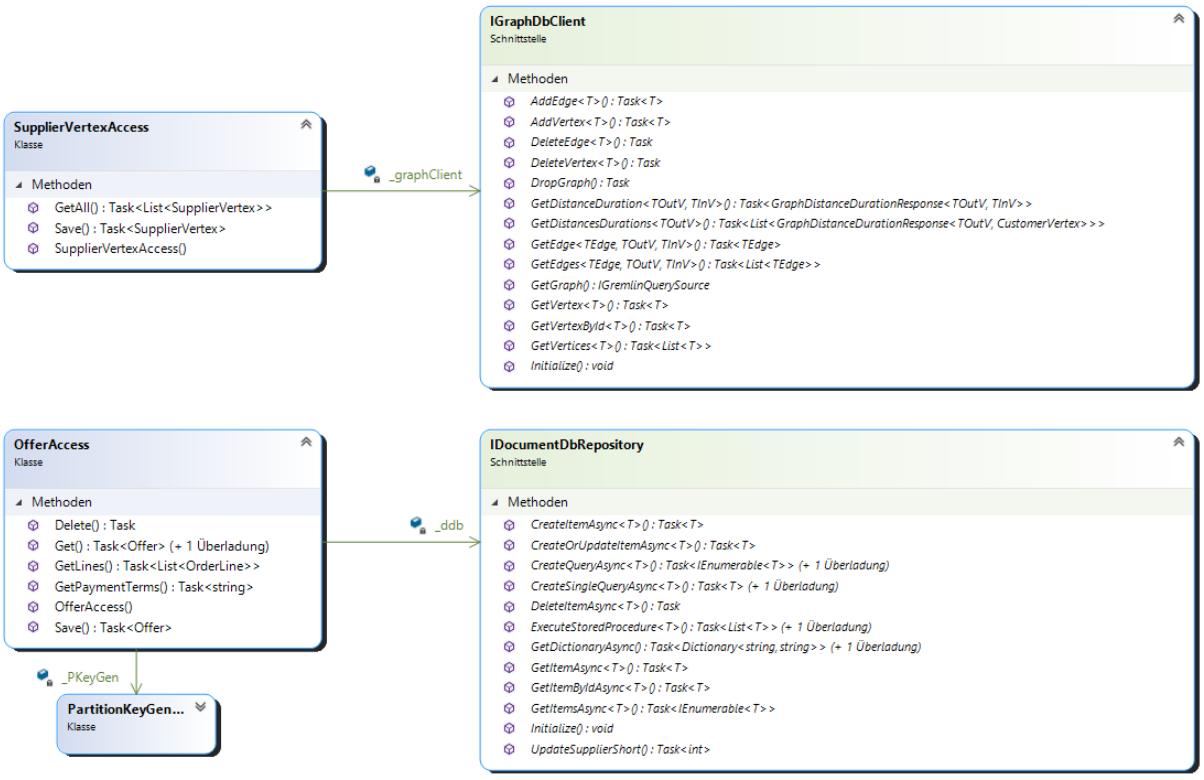


Abbildung 8: Beispiele von Klassendiagrammen der Data-Access-Klassen

5.3 Business Logic Layer (ArduviLogic)

Die meiste Logik des Projektes steckt im Business Logic Layer. Dieser sollte eigenständig und ohne einer festgelegten Darstellungsebene arbeiten können, damit das Gerät oder die Technologien, welche für die Darstellung der Daten (Mobile App, Webseite, Desktopanwendung) zuständig sind, jederzeit ausgetauscht werden können. Mithilfe der Models und der Data-Access-Klassen können Daten aus dem Persistence Layer abgefragt und manipuliert werden. Die Logik sollte sich nicht darum kümmern, wie oder wo die Daten abgespeichert werden, denn dafür ist der Data Access Layer zuständig.

5.3.1 Logik für die Manipulation des Graphen

Diese Klasse stellt die notwendigen Methoden für den Zugriff auf die Data-Access-Klassen des Graphen zur Verfügung.

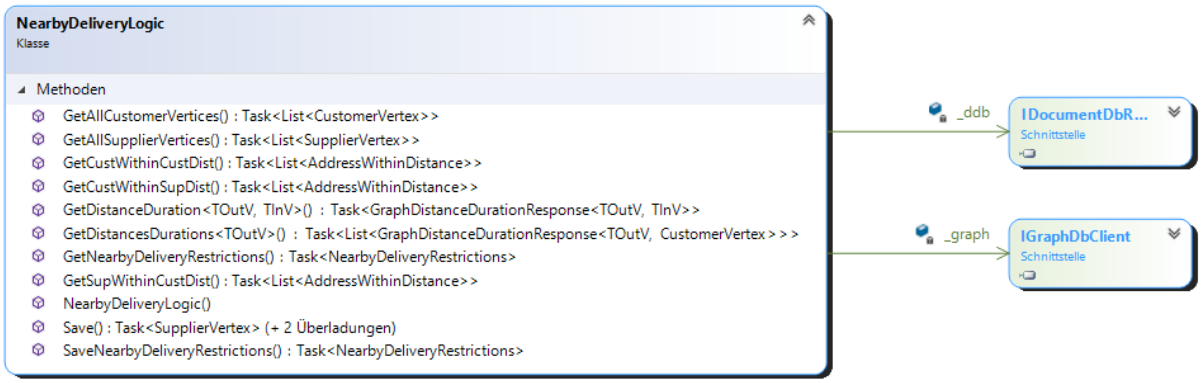
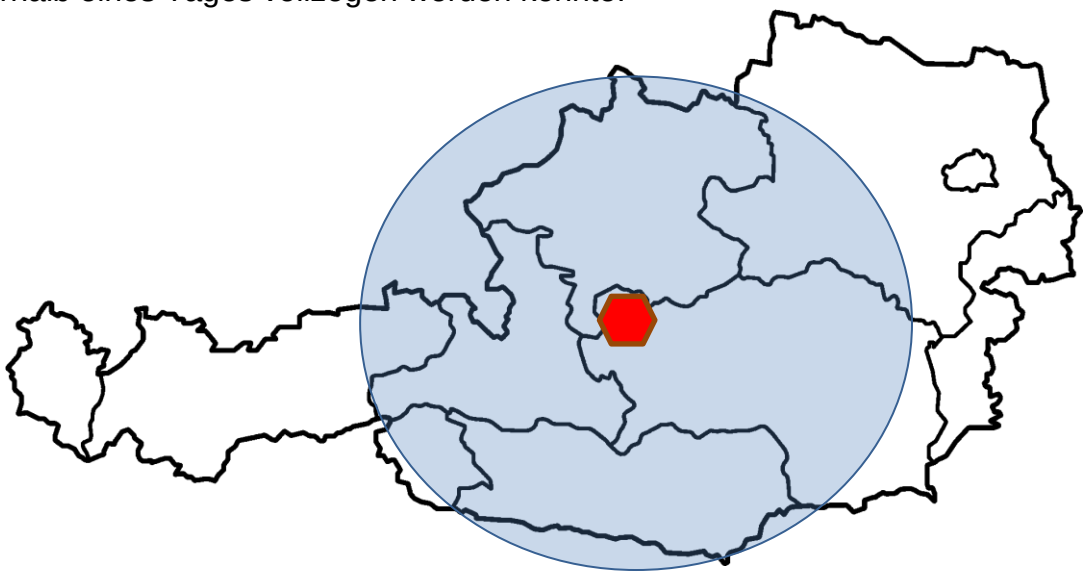


Abbildung 9: Klassendiagramm zur Logik des Graphen

Unter anderem beinhaltet diese auch die Methoden *GetCustWithinCustDist* und *GetSupWithinCustDist*. Diese liefern die Projekte und Sägewerke innerhalb eines bestimmten Radius zurück, da es ab einem gewissen Luftweg keinen Sinn macht die Entfernung mithilfe einer *Distance Matrix API* zu berechnen, weil die Lieferung nicht innerhalb eines Tages vollzogen werden könnte.



```
public async Task<List<AddressWithinDistance>> GetSupWithinCustDist(Location location)
{
        return (List<AddressWithinDistance>)await
        _ddb.CreateQueryAsync<AddressWithinDistance>
                ($"SELECT c.ownerId as companyId, c.gps as location, c.id as itemId" +
                $"ST_DISTANCE({{'type': 'Point', 'coordinates':c.gps.coordinates}}, " +
                $"{{'type': 'Point',
                'coordinates':{JsonConvert.SerializeObject(location.Coordinates)}}}) " +
                $"as distance FROM c " +
                $"WHERE c.type=\"Address\" AND c.ownerType=\"Supplier\" AND
                c.addressType=\"Invoice\" AND " +
                $"ST_DISTANCE({{ 'type': 'Point', 'coordinates':c.gps.coordinates}}, " +
                $"{{ 'type': 'Point',
                'coordinates':{JsonConvert.SerializeObject(location.Coordinates)}}}) <
                700000", null);
}
```

Quellcode 1: Select-Befehl für die Abfrage von Sägewerken in einem Umkreis von 700km Luftlinie

5.3.2 Logik für das Erstellen und Verwalten von Routen

Der zentrale Bestandteil dieses Projektes ist die *TruckRoutingLogic* Klasse, welche sich um die Logik für das *TruckRoute* Objekt kümmert. Nun möchte ich den Aufbau der Logik und den groben Ablauf, wie der Graph erstellt wird erklären.

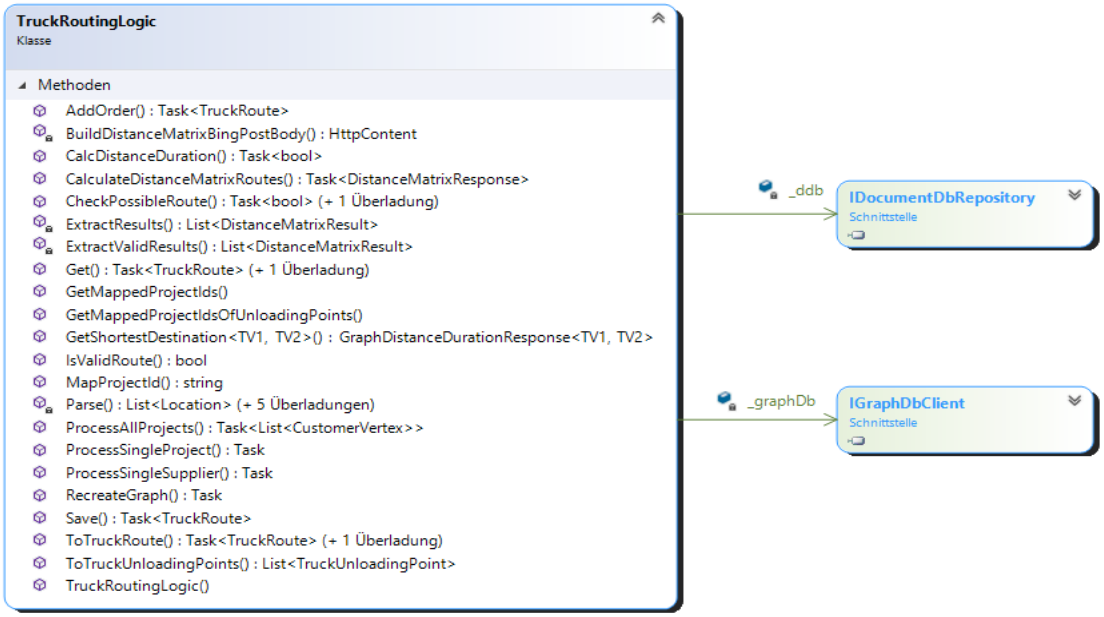


Abbildung 11: Klassendiagramm zur Logik für das Erstellen und Verwalten einer Route

Erstellung und Berechnung des Graphen

Der Kern dieser Klasse ist die Methode *CalculateDistanceMatrixRoutes.* Sie übermittelt die Koordinaten an die *Distance Matrix API* und liefert die Fahrstrecke und die Fahrzeit dieser Wegpunkte zurück. Anschließend wird das Ergebnis zu den jeweiligen *Vertices* in die Graphdatenbank gespeichert.

```
{
   "origins": [{
      "latitude": lat0,
      "longitude": lon0
   },
   {
      "latitude": latM,
      "longitude": lonM
   }],
   "destinations": [{
      "latitude": lat0,
      "longitude": lon0
   },
   {
      "latitude": latN,
      "longitude": lonN
   }],
   "travelMode": travelMode,
   "startTime": startTime
}
```

Quellcode 2: POST Body Template für eine Abfrage auf die Distance Matrix API

Dies ist ein Template für einen POST Request auf die *Distance Matrix API* von Bing Maps.

Origins sind die Startpunkte und *Destinations sind die* Ziele einer jeweiligen Route. Ein Wegpunkt besteht aus Längengrad und Breitengrad.

Ebenfalls ist es möglich einen *TravelMode* mitzugeben. Dieser kann folgendes beinhalten:

- driving
- walking
- transit

StartTime ist ein optionaler Parameter. Wenn dieser angegeben wird, werden prädiktive Verkehrsdaten mit in die Berechnung einbezogen.

([21], URL Template, Template Parameters)

Als *Response* erhält man ein *Array* bestehend aus in *Quellcode 3* vermerkten Feldern.

Der *OriginIndex* und der *DestinationIndex* beinhalten die Reihenfolge der angegebenen Startpunkte und Ziele im POST Body des Requests.

TravelDistance und *TravelDuration* gibt die Fahrstrecke in Kilometern und die Fahrzeit in Minuten an.

```
{
   "originIndex": 0,
   "destinationIndex": 1,
   "travelDistance": 54,
   "travelDuration": 42,
   "totalWalkDuration": 0,
   "hasError": false
}
```

```
public async Task<DistanceMatrixResponse> CalculateDistanceMatrixRoutes(
    List<AddressWithinDistance> origins,
    List<AddressWithinDistance> destinations)
{
    string url =
$"https://dev.virtualearth.net/REST/v1/Routes/DistanceMatrix?key={API_Key}";


    origins = new List<AddressWithinDistance>() { origins.First() };
    DistanceMatrixResponse result = new DistanceMatrixResponse();

    /* Parse addresses within distance into DistanceMatrixItems for request body of
     * DistanceMatrixAPI */
    List<DistanceMatrixItem> originsWithIdx = null;
    List<DistanceMatrixItem> destinationsWithIdx = null;

    List<Location> originLocations = origins
        .Select(x => x.Location).ToList();
    List<Location> destinationLocations = destinations
        .Select(x => x.Location).ToList();

    using (HttpClient httpClient = new HttpClient())
    {
        // Building JSON body and sending HTTP POST call to DistanceMatrixAPI
        HttpResponseMessage response = await httpClient.PostAsync(url,
            BuildDistanceMatrixBingPostBody(originLocations, destinationLocations));
        DistanceMatrixBingResponse responseObject = null;
        if (response.StatusCode == HttpStatusCode.OK)
        {
            // Convert JSON response into C# object
            responseObject = JsonConvert.DeserializeObject<DistanceMatrixBingResponse>
                (await response.Content.ReadAsStringAsync());
            if (responseObject != null)
            {
                /* Parse addresses within distance into DistanceMatrixItems with
                 * indizes */
                originsWithIdx = Parse(origins);
                destinationsWithIdx = Parse(destinations);

                // Extract results array of DistanceMatrixResponse
                result.Results = ExtractResults(
                    responseObject,
                    originsWithIdx,
                    destinationsWithIdx);
            }
        }
    }
    return result.Results != null ? result : null;
}
```

Quellcode 4: Methode für den Call an die Distance Matrix API und das anschließende Parsen des JSON-Responses in ein C# Objekt

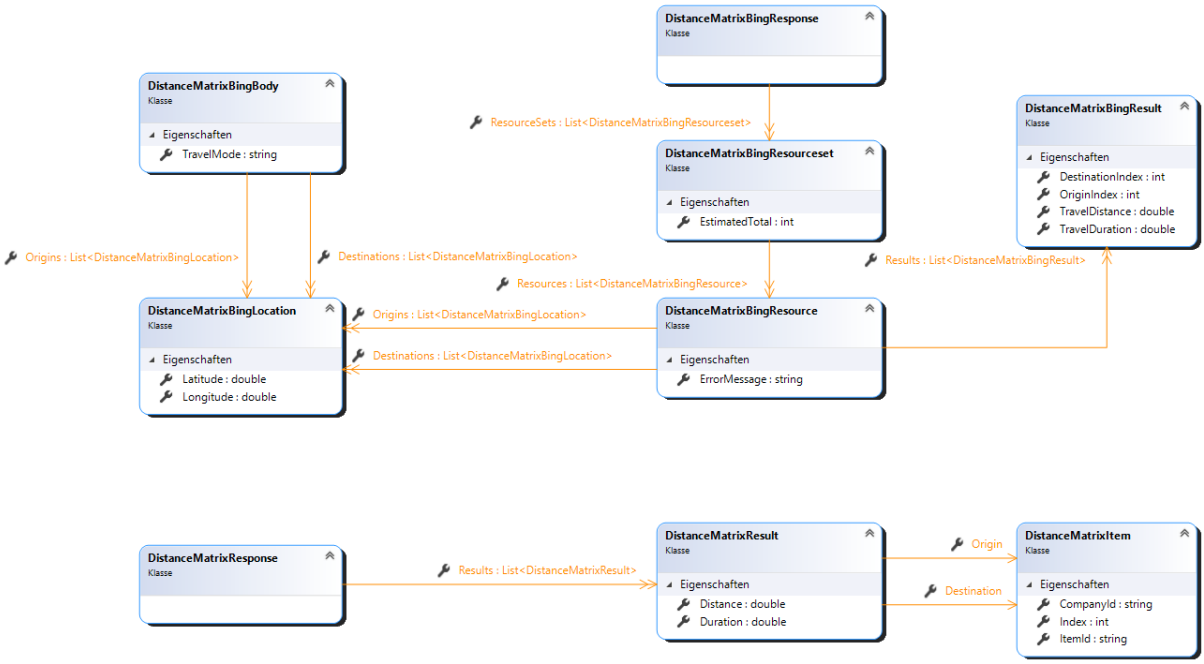
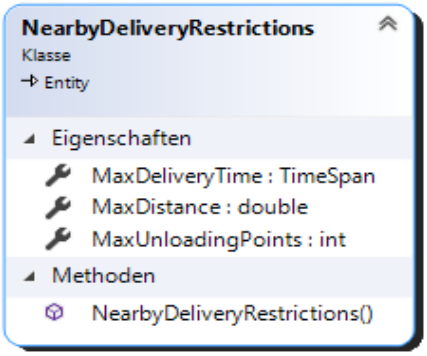


Abbildung 12: Klassendiagramm des Responses der Distance Matrix API und der fertig umgewandelten Ergebnisse aus dem Response

Das obere Klassendiagramm (*DistanceMatrixBingResponse*) wird rein für das Umwandeln des JSON-Responses der Distance Matrix API verwendet. Die Klasse *DistanceMatrixResponse* bildet die extrahierten Ergebnisse des *DistanceMatrixBingResponse* ab. Nun hat man eine Liste von Objekten, welche einen Ursprung, ein Ziel, die Distanz und die Dauer speichern.



Als Nächstes müssen die Resultate auf ihre Gültigkeit überprüft werden. Dafür musste ein neues Objekt für das Speichern der Restriktionen implementiert werden.

Mittels der *NearbyDeliveryRestrictions* wird festgelegt, welche Konditionen eine Route erfüllen muss, um gültig zu sein. Es besteht aus der maximalen Lieferzeit, der maximalen Fahrstrecke und der maximalen Abladepunkte.

In dem nachfolgenden Codebeispiel (Quellcode 5) wird jedes Ergebnis aus der Distance Matrix API als einzelne Route behandelt. Diese Methode sortiert Routen aus, welche bereits ohne weiteren Abladepunkten nicht den Restriktionen entspricht, da eine zu lange Route, kombiniert mit einer zweiten Route nicht kürzer werden kann.

```
private List<DistanceMatrixResult> ExtractValidResults(
    List<DistanceMatrixResult> results,
    NearbyDeliveryRestrictions restrictions)
{
    List<DistanceMatrixResult> validResults = new List<DistanceMatrixResult>();
    foreach (DistanceMatrixResult item in results)
    {
        // Check if route would be valid according to restrictions
        if (item.Distance <= restrictions.MaxDistance &&
            item.Duration <= restrictions.MaxDeliveryTime.TotalMinutes)
        {
            validResults.Add(item);
        }
    }
    return validResults;
}
```

Quellcode 5: Überprüfen der Distance-Matrix-API-Ergebnisse auf Gültigkeit laut Restriktionen

```
await nearbyDeliveryLogic.Save(
    new SupplyEdge
    {
        Distance = r.Distance,
        Duration = r.Duration
    },
    supVertex,
    destinationV);
```

Darauffolgend kann über die *NearbyDeliveryLogic* Klasse die Edge in den Graphen gespeichert werden. Falls diese Verbindung zwei *CustomerVertices*, also Projekte, miteinander verbindet, muss die Edge bidirektional hinzugefügt werden, da sich die Reihenfolge der Abladepunkte innerhalb einer Route verändern kann. Somit wird der Algorithmus, welcher den optimalen Lieferweg berechnet, schneller und effizienter.

Vor diesem Aufruf, muss sichergestellt werden, dass die zwei Knoten, welche verbunden werden sollen, in der Graphdatenbank existieren. Sollte dies nicht der Fall sein, werden die Vertices im Vorhinein noch über die *NearbyDeliveryLogic* Klasse erstellt.

Dies war der ein grober Überblick wie die Fahrstrecke und Fahrzeit zwischen zwei Vertices abgespeichert wird. Für jedes Sägewerk wird ein eigener Graph aufgebaut, damit die darauffolgende Berechnung für eventuelle Routen möglichst einfach funktioniert

Validierung und Berechnung der schnellsten Route

Sobald der Graph einmal aufgebaut ist und neu erstellte Projekte oder auch Sägewerke automatisch hinzugefügt werden, kann die Evaluierung verschiedenster Routen für unterschiedliche Bestellungen angefangen werden.

Als Beispiel möchte ich die Methode *AddOrder* (Quellcode 7) aufzeigen. Sie wird verwendet, um weitere Bestellungen an eine *TruckRoute* anzuhängen. Speziell ist darauf zu achten, dass eine andere Bestellung nicht zwangsweise eine andere Abladestelle hat, da mehrere Holzverarbeiter an einem Projekt arbeiten könnten oder ein bereits in der Route enthaltener Kunde erneut auf dieselbe Adresse bestellt.

Ein weiterer Punkt ist das Sägewerk. Pro Route kann es nur einen Aufladepunkt geben. Somit gibt es pro *TruckRoute* immer nur einen Holzproduzenten.

```
public async Task<TruckRoute> AddOrder(TruckRoute route, ArduviData.Models.Order toAd-
dOrder)
{
    NearbyDeliveryLogic nearbyDeliveryLogic = new NearbyDeliveryLogic(_ddb, _graphDb);
    NearbyDeliveryRestrictions restrictions =
       await nearbyDeliveryLogic.GetNearbyDeliveryRestrictions();

    // Check if to add order is from the supplier of the loadingpoint
    if (toAddOrder.SupplierId != route.LoadingPoint.Supplier.SupplierId)
        return null;

    // Converting to add order to TruckUnloadingPoints
    List<TruckUnloadingPoint> toAddUnloadingPoints = ToTruckUnloadingPoints(
        toAddOrder.CustomerShort,
        toAddOrder.Projects,
        OrderLineLogic.FromOrderLine(toAddOrder.Lines, toAddOrder.Projects));

    // Creating a new list and truckroute because the original data will get modified
    List<TruckUnloadingPoint> tempUnloadingPoints =
        new List<TruckUnloadingPoint>(route.UnloadingPoints);
    TruckRoute tempRoute = new TruckRoute()
    {
        LoadingPoint = route.LoadingPoint,
        UnloadingPoints = tempUnloadingPoints,
        EstimatedDistance = route.EstimatedDistance,
        EstimatedTime = route.EstimatedTime
    };

    // Adding new unloading point(s) to original route
    toAddUnloadingPoints.ForEach(x =>
    {
        TruckUnloadingPoint point = route.UnloadingPoints
            .Find(y => y.Address.GoogleFormatted == y.Address.GoogleFormatted);
        if (point == null)
            route.UnloadingPoints.Add(x);
        else if (x.Customer.Id == point.Customer.Id)
            point.OrderLines.AddRange(x.OrderLines);
    });

        /* Check if there are less unloading points at all than in
        * the restrictions defined AND
        * Calculate distance and duration of new route */
    if (route.UnloadingPoints.Count + toAddUnloadingPoints.Count
            >= restrictions.MaxUnloadingPoints &&
        !await CalcDistanceDuration(route))
    {
        // If route is not possible, rollback all changes and return null
        route.LoadingPoint = tempRoute.LoadingPoint;
        route.UnloadingPoints = tempRoute.UnloadingPoints;
        route.EstimatedDistance = tempRoute.EstimatedDistance;
        route.EstimatedTime = tempRoute.EstimatedTime;
        return null;
    }

    return route;
}
```

Quellcode 7: Hinzufügen einer Bestellung zu einer Route

Nun möchte ich den groben Ablauf dieses Codes beschrieben. Als erstes wird überprüft, ob der Lieferant von der Route mit dem Lieferanten aus der Bestellung übereinstimmt. Darauffolgend werden die Abladepunkte aus der zu hinzufügenden Bestellung herausgefiltert. Im nächsten Schritt wird eine Kopie der originalen Liste der

Abladepunkte erzeugt, da dieser danach die neuen Abladepunkte hinzugefügt werden. Bevor die Berechnung der schnellsten Route starten kann, muss die Anzahl an Abladepunkten mithilfe der *NearbyDeliveryRestrictions* überprüft werden. Zu guter Letzt kann die Evaluierung starten. Falls keine valide Route möglich ist, wird die originale Route wieder zurückgesetzt. Anderenfalls wird die neu ermittelte Wegstrecke zurückgegeben.

Schlussendlich möchte ich noch einige Details zur Routenberechnung loswerden. Um zu bewerten, wie schnell ein Weg ist, wird die ermittelte Fahrstrecke mit der Fahrzeit multipliziert. Dieser Wert wird anschließend mit weiteren Werten verglichen. Der kleinste Wer gewinnt das Rennen und der erste Abladepunkt steht fest. Dieses Verfahren wird für alle weiteren Abladepunkte durchgeführt, bis keiner mehr übrig ist oder die Route den Restriktionen nicht gerecht wird.

5.4 Client Layer (ArduviWeb)

Abschließend sind wir beim *ASP.NET CORE* Projekt angelangt. Der *Client Layer* verwaltet die Darstellung der Daten und übermittelt Benutzerinteraktionen an die *Business Logic*. Er besteht aus *Model* (*ViewModel*), *View* und *Controller* (kurz *MVC*).

Das *ViewModel* ist für die Bereitstellung der Daten zuständig. Meist ergeben mehrere Models aus *ArduviData* ein *ViewModel* in *ArduviWeb*. Mithilfe von *ArduviLogic* werden die Daten in die *ViewModels* geladen.

Wie die Webseite schlussendlich aussieht, bestimmt die *Ansicht* (*View*). Dies sind ‚.cshtml'-Dateien, welche ich bereits unter 4.2.4 Razor Pages genauer erläutert habe. Jedem *View* wird ein konkretes *ViewModel* zugewiesen.

Die *Controller* sind der logische Teil des *Client Layers.* Sie verwalten die Erstellung des *ViewModels,* stellen die HTTP-Methoden für die *Views* bereit, kommunizieren mit *ArduviLogic* und kümmern sich um clientspezifische Logik.

5.4.1 Aufbau des Client Layers

Für die Darstellung und Verwaltung der *TruckRoutes* sind zwei *Views* vorhanden. Ein *View* zeigt eine Übersicht über alle erstellten *Routen*. Das andere ist eine Detailansicht einer bestimmten *TruckRoute*. Diese beiden Views werden über einen gemeinsamen Controller verwaltet.

5.4.2 ViewModels

Wie bereits unter 5.4 Client Layer (ArduviWeb) erwähnt, sind die ViewModels für die Strukturierung der Daten zuständig. Beim Erstellen der HTML-Seite werden diese mitübergeben und anschließend kann per Razor auf die Felder zugegriffen werden.

Auf der anschließenden Abbildung 14 sind die beiden *ViewModels* in der Mitte des Diagramms zu erkennen. Diese sind grundsätzlich ähnlich aufgebaut. Das *TruckRoutingIndexViewModel* enthält eine Liste von *TruckRoutes* und *TruckRouteCoordinates* und das *TruckRoutingDetailsViewModel* nur einfache Objekte. Das ist deshalb so, da das *TruckRoutingIndexViewModel* für die Überblickseite und das *TruckRoutingDetailsViewModel* für die Detailansicht verwendet wird.

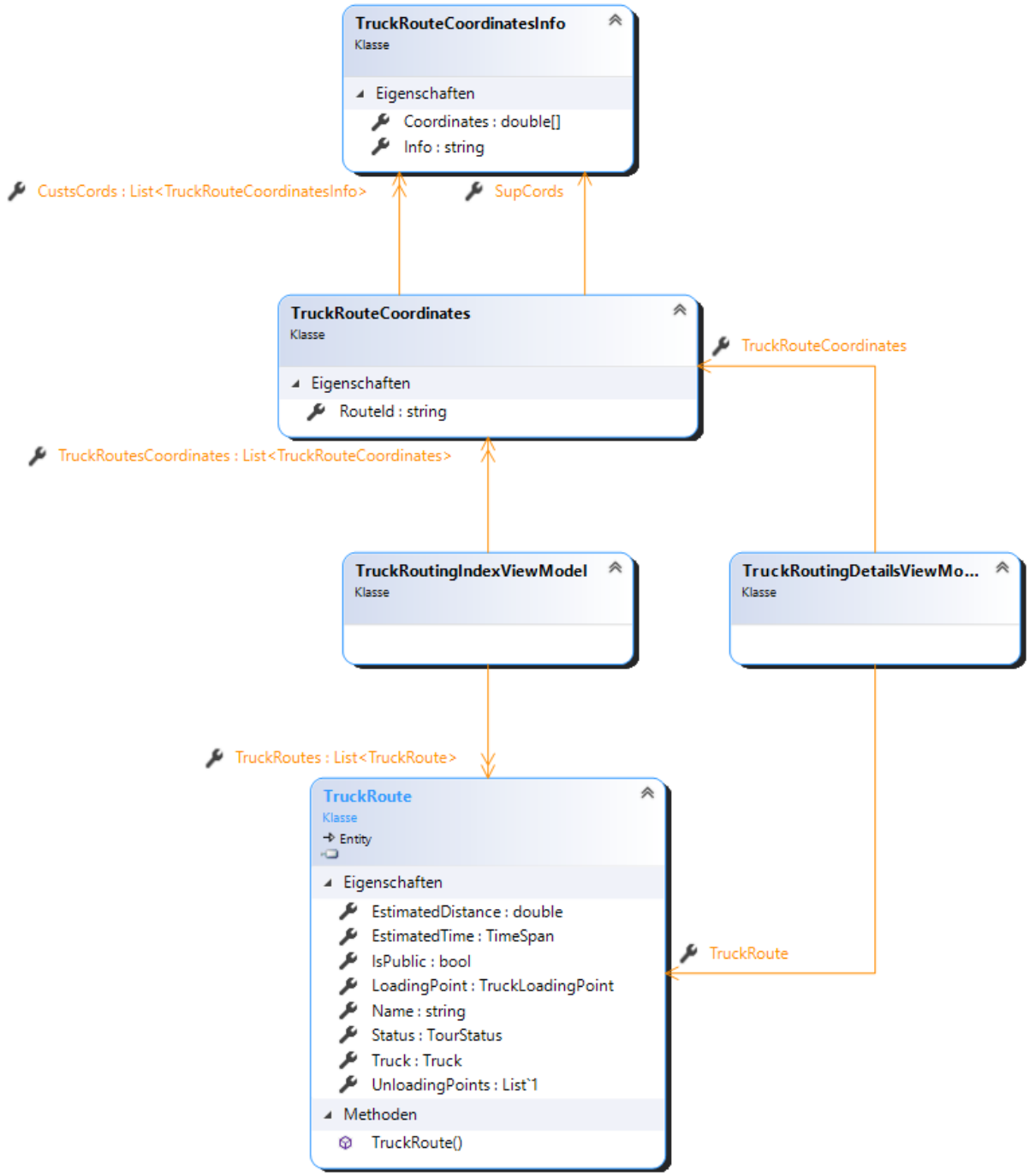


Abbildung 14: Klassendiagramm zu ViewModels

5.4.3 Controller

Der *TruckRoutingController* holt sich die vorhandenen *TruckRoutes* des jeweiligen Sägewerks und erstellt daraufhin das nötige *ViewModel* für die *Anzeige*. Er enthält alle notwendigen *Felder*, wie zum Beispiel die *Datenbankschnittstelle* oder den *Usermanager*, welche der Logik in weiterer Folge übergeben werden müssen.

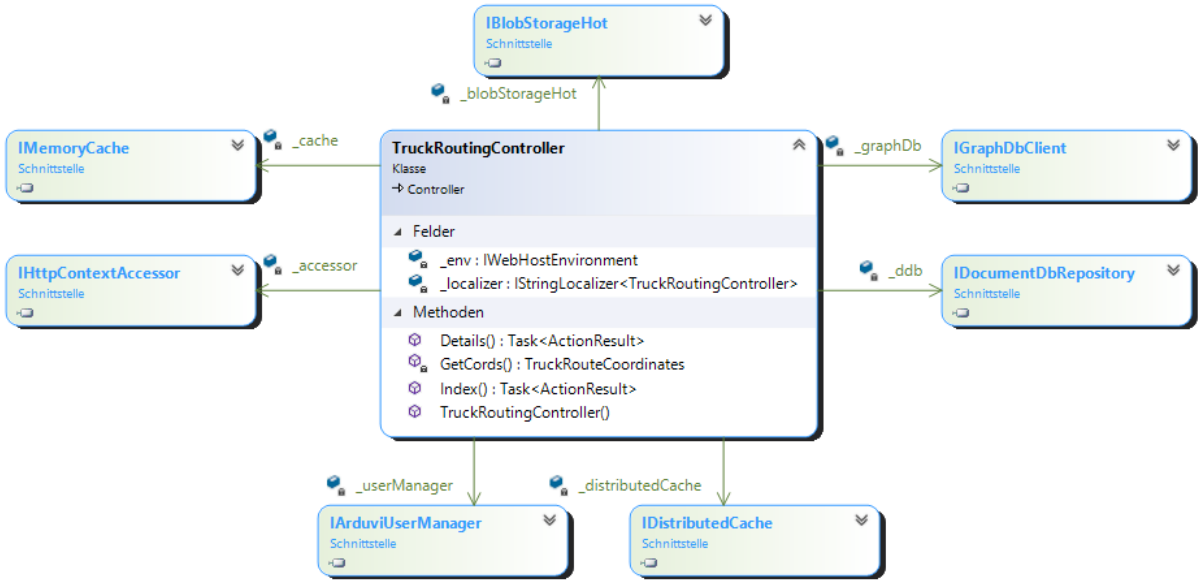


Abbildung 15: Klassendiagramm vom TruckRoutingController

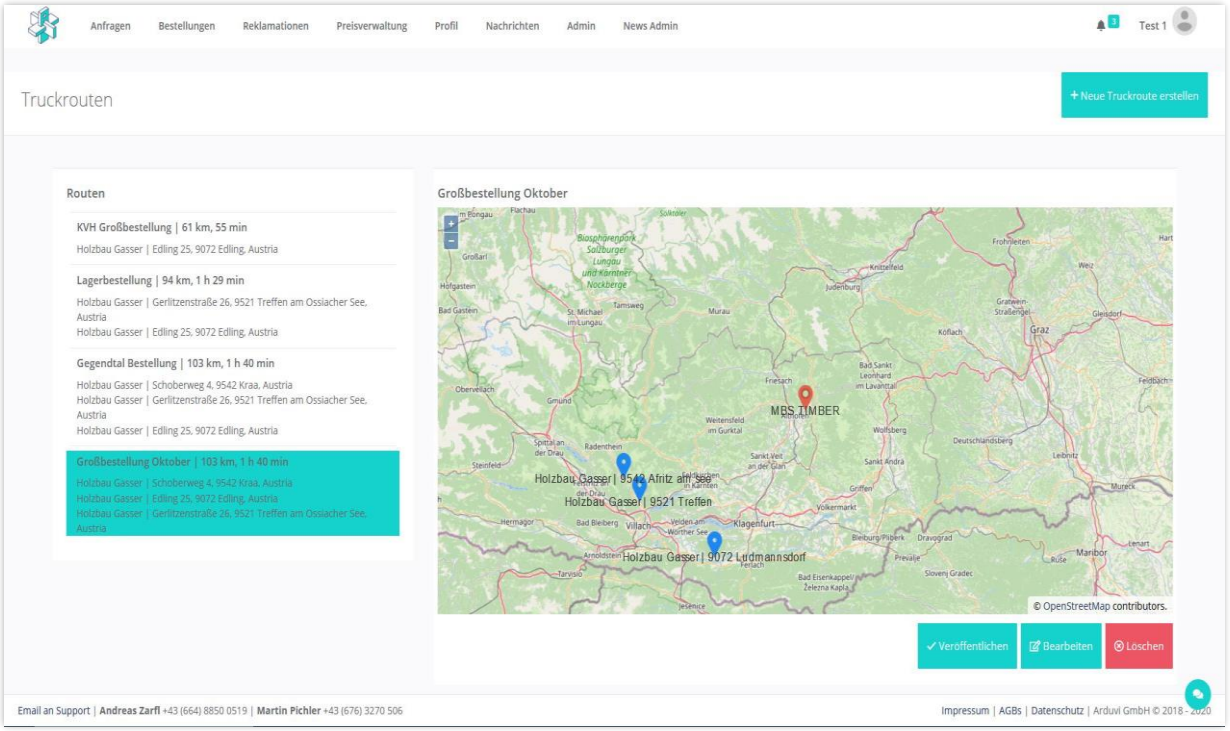
Der Name der *Methode* muss denselben Namen haben, wie die dazugehörige *Ansicht.* Bei Quellcode 8 ist gut erkennen, dass der Controller lediglich auf die Business Logic und nicht auf den Data Access Layer zugreift.

```
public async Task<ActionResult> Index()
{
    TruckRoutingIndexViewModel viewModel = new TruckRoutingIndexViewModel();
    TruckRoutingLogic truckRoutingLogic = new TruckRoutingLogic(_ddb, _graphDb);
    NearbyDeliveryLogic nearbyDeliveryLogic = new NearbyDeliveryLogic(_ddb, _graphDb);

    // Check if user is supplier
    ArduviUser user = await _userManager.GetAsync(User);
    if (user.CompanyType != CompanyType.Supplier)
        return NotFound();

    // get all routs of current logged in user
    List<TruckRoute> routes = await truckRoutingLogic.Get(user.CompanyId);
    if (routes == null)
        return View();

    // add routes to viewModel
    viewModel.TruckRoutes = routes;
    viewModel.TruckRoutesCoordinates = (routes.Select(route =>
       GetCords(route))).ToList();
    return View(viewModel);
}
```

Quellcode 8: TruckRoutingController – Übersichtseite

5.4.4 Views

Zu guter Letzt, kommen wir zu der Spitze des Eisberges, denn nur das bekommt der Benutzer am Ende zu Gesicht. Als Admin Theme Template wurde *Inspinia* verwendet. Dies ist eine kostenpflichtige Designvorlage, welche auf *Bootstrap* aufbaut. Es stellt fertige HTML-Elemente, wie Tabellen oder Dashboards bereit.




```
@* creating html list view for truckRoutes *@
@foreach (TruckRoute route in Model.TruckRoutes)
{
    ARDUVIWeb.Models.TruckRoutingViewModels.TruckRouteCoordinates routeCords =
        Model.TruckRoutesCoordinates.FirstOrDefault(x => x.RouteId.Equals(route.Id));
    <hr class="m-xs" />
    <div id="@(route.Id)" name="truckRoute" class="col-12 p-w-sm"
            onclick="truckRouteClicked(
            '@route.Name',
            this.id,
            @(Json.Serialize(routeCords.SupCords).ToString()),
            @(Json.Serialize(routeCords.CustsCords).ToString()))">
        @* prints the name of the truckroute *@
        <h4 class="inline">@route.Name</h4>
        <p>
            @* print unloadingPoints below name of the truckRoute *@
            @foreach (TruckUnloadingPoint unloadingPoint in route.UnloadingPoints)
            {
                @Html.Raw(unloadingPoint.Customer.DisplayName + " | " +
                    unloadingPoint.Address.GoogleFormatted); <br />
            }
        </p>
    </div>
}
```

Quellcode 9: Beispiel von Razor-Syntax

C#-Befehle können mit einem @ eingeleitet werden. Bevor der HTML-Code an den *Client* (*Browser) geschickt wird, wird serverseitig die Razor-Syntax* ausgeführt und in HTML umgewandelt.

In dem Quellcode 9 Beispiel wird die Liste, welche auf Abbildung 16 zu sehen ist, erstellt.

6 Resümee

In diesem Projekt habe ich erneut gemerkt, wie wichtig es ist, die Grundlagen der Informatik zu verstehen und sie auch einzusetzen zu wissen. Ich musste mich über neue Technologien informieren und diese dann auch erlernen.

Ebenfalls konnte ich in Erfahrung bringen, wie wichtig die vorläufige Planung für die anschließende Umsetzung ist. Auf der anderen Seite ist es auch von großer Bedeutung flexibel und anpassungsfähig bei Änderungen zu sein.

Eine Datenbank, welche auf Graphen basiert, hatte ich im Vorhinein noch nicht verwendet. Diese erwies sich allerdings als sehr sinnvoll, zeitsparend und effizient für diesen Anwendungsfall.

7 Anhang

7.1 Glossar

7.2 Literaturverzeichnis

[1] „Arduvi GmbH in Lannach," Firmen ABC, 04 16 2021. [Online]. Available: https://www.firmenabc.at/arduvi-gmbh_OSCI.

[2] „Die ersten Online Shops sperren auf," Geschichte Österreich, 10 04 2021. [Online]. Available: https://www.geschichte-oesterreich.com/internet/shopping.html.

[3] „About Amazon," Amazon, 10 04 2021. [Online]. Available: https://www.aboutamazon.de/%C3%BCber-amazon/unsere-geschichte-was-aus-einer-garagen-idee-werden-kann.

[4] „eBay," Wikipedia, 10 04 2021. [Online]. Available: https://de.wikipedia.org/wiki/EBay.

[5] „LKW-Bußgeldkatalog," Mobilitätsmagazin, 23 03 2021. [Online]. Available: https://www.bussgeldkatalog.org/lkw-arten/#welche_lkw-arten_gibt_es.

[6] „Arbeitszeittabelle: Lenkzeiten," WKO, 07 04 2021. [Online]. Available: https://www.wko.at/branchen/transport-verkehr/gueterbefoerderungsgewerbe/Arbeitszeittabelle_Lenkzeiten.html.

[7] „Lkw-Fahrverbote in Österreich: Überblick," WKO, 07 04 2021. [Online]. Available: https://www.wko.at/service/verkehr-betriebsstandort/LKW_Fahrverbote_in_Oesterreich_Ueberblick.html.

[8] „Überladung von Lkw, Pkw und Anhänger," lasiprofi, 07 04 2021. [Online]. Available: https://www.ladungssicherung.eu/ratgeber/ladung/ueberladen/.

[9] „ASP.NET Core," Wikipedia, 20 01 2021. [Online]. Available: https://en.wikipedia.org/wiki/ASP.NET_Core.

[10] „Cosmos DB," Wikipedia, 20 01 2021. [Online]. Available: https://en.wikipedia.org/wiki/Cosmos_DB.

[11] „Understanding the differences between NoSQL and relational databases," Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/azure/cosmos-db/relational-nosql.

[12] „Gremlins im Cosmos: Graphverarbeitung mit CosmosDB," Informatik Aktuell, 21 01 2021. [Online]. Available: https://www.informatik-aktuell.de/betrieb/datenbanken/graphverarbeitung-mit-cosmosdb.html.

[13] „Hypertext Markup Language," Wikipedia, 21 01 2021. [Online]. Available: https://de.wikipedia.org/wiki/Hypertext_Markup_Language.

[14] „Cascading Style Sheets," Wikipedia, 22 01 2021. [Online]. Available: https://de.wikipedia.org/wiki/Cascading_Style_Sheets.

[15] „JavaScript," Wikipedia, 22 01 2021. [Online]. Available: https://de.wikipedia.org/wiki/JavaScript.

[16] „ASP.NET Razor," Wikipedia, 22 01 2021. [Online]. Available: https://en.wikipedia.org/wiki/ASP.NET_Razor.

[17] „Einführung in ASP.NET Web Pages und Razor Syntax," centron, 22 01 2021. [Online]. Available: https://www.centron.de/2013/06/18/einfuehrung-in-asp-net-web-pages-und-razor-syntax/.

[18] „Inspinia - Responsive Admin Template," WrapBootstrap, 23 01 2021. [Online]. Available: https://wrapbootstrap.com/theme/inspinia-responsive-admin-template-WB0R5L90S.

[19] „Landing page," Bootstrap, 23 01 2021. [Online]. Available: http://holdirbootstrap.de/.

[20] „What is jQuery?," jQuery, 23 01 2021. [Online]. Available: https://jquery.com/.

[21] „Calculate a Distance Matrix," Microsoft, 18 04 2021. [Online]. Available: https://docs.microsoft.com/en-us/bingmaps/rest-services/routes/calculate-a-distance-matrix.