



Abschlussprüfung Sommer 2023

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Parkhaus Simulationssoftware

Implementierung eines neuen Parkhauses mithilfe einer Simulationssoftware

Abgabedatum: den 22.05.2023

Prüfungsbewerber:

Daniel-Alfonsin Massa
Hammer Baum 23
20537 Hamburg

IHK-Nummer: 131
Prüflingsnummer: 54190
Prüfungsausschuss: FachinAnw20 018ort

Ausbildungsbetrieb:

Grassau GmbH
Gerlindweg 29
22559 Hamburg



Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis.....	I
Abbildungsverzeichnis.....	IV
Tabellenverzeichnis.....	V
Verzeichnis der Listings	VI
Abkürzungsverzeichnis	VII
1 Einleitung	8
1.1 Projektumfeld.....	8
1.2 Projektbegründung	8
1.3 Projektschnittstellen.....	8
1.4 Projektabgrenzung	9
2 Projektplanung	9
2.1 Abweichungen vom Projektantrag.....	9
2.2 Ressourcenplanung.....	9
2.3 Make-or-Break-Eventpoints.....	9
2.4 Entwicklungsprozess	10
2.5 Ist-Analyse	10
2.6 Wirtschaftlichkeitsanalyse	10
2.7 Make or Buy-Entscheidung	10
2.8 Nutzwertanalyse	11
2.9 Anwendungsfälle	11
2.10 Qualitätsanforderungen	11
2.11 Lastenheft/Fachkonzept	12
3 Entwurfsphase	12
3.1 Zielplattform	12
3.2 Architekturdesign.....	12
3.3 Entwurf der Benutzeroberfläche	12
3.4 Geschäftslogik	12
3.5 Maßnahmen zur Qualitätssicherung	12
3.6 Pflichtenheft/Datenverarbeitungskonzept.....	13
4 Implementierungsphase	13
4.1 Implementierung der Datenstrukturen.....	13
4.2 Implementierung der Benutzeroberfläche	13
4.3 Implementierung der Geschäftslogik	13
5 Abnahmephase	13
6 Einführungsphase	13
7 Dokumentation	14
8 Fazit	14

Inhaltsverzeichnis

8.1	Soll-/Ist-Vergleich	14
8.2	Lessons Learned	15
8.3	Ausblick	15
8.4	Literaturverzeichnis	16
8.5	Eidesstattliche Erklärung	17
Anhang.....		18
A1	Auftrag	18
A2	Detaillierte Zeitplanung.....	18
A3	Kosten- und Ressourcenplan	20
A4	Projektablaufplan als Gantt-Diagramm	21
A5	Projektstrukturplan.....	22
A6	Zustands-Diagramm	23
A7	Sequenzdiagramm Entwicklungsvorgang der Software.....	24
A8	Package-Diagramm.....	24
A9	Lastenheft (Auszug)	26
A10	Use-Case-Diagramm	26
A11	Tabellen für die Parkhaus Simulationssoftware in MySQL	27
A12	Pflichtenheft (Auszug)	30
A13	Datenbankmodell	31
A14	Entity-Relationship-Model.....	32
A15	Sequenzdiagramm für die Parkhaus Simulationssoftware.....	33
A16	Ereignisgesteuerte Prozesskette.....	36
A17	Aktivitätsdiagramm	38
A18	Screenshots der Anwendung	40
A19	Das Domain-Modell der Parkhaus-Simulationssoftware	41
A20	Entwicklerdokumentation (Auszug)	41
A21	Klassenmodell von der Parkhaus Simulationssoftware strukturiert im MVVC-Model.	45
A22	Entity-Relationship-Model.....	46
A23	Meilensteinplan.....	47
A24	Klasse: ParkhausSimulationssoftwareTest	47
A25	Klasse "ParkingSpace"	48
A26	Klasse "GarageGUI"	51
A27	Klasse Garage	54
A28	Klasse "Reservation"	55
A29	Erweiterung der Klasse "ParkingGarage" mit Reservierungsfunktionalität	56
A30	Das Strategy Pattern	57
A31	Model View Control Pattern angewandt am Parkhaus Projekt	60
A32	Das Model-View-ViewModel (MVVM) Design Pattern	60
A33	Strategy Pattern Design angewendet am Parkhaus Projekt:	62
A34	Zusammenfassung	62

Inhaltsverzeichnis

A35	Klassendiagramm	63
A36	Benutzerdokumentation (Auszug)	65
A37	Parkplatz Design	67
A38	Use-Case-Diagramm mit Methoden/Prozeduren	68
A39	Codeschnipsel, der die Einfahrt eines Parkhauses simuliert	69
A40	Das Parkplatzreservierungssystem	69
A41	Flowchart Diagramme	69
A42	Kompositionsstruktur-Diagramm	71
A43	Objektdiagramm	72
A44	Interaktionsdiagramm	73
A45	Verteilungsdiagramm.....	73
A46	Komponentendiagramm	74
A47	Kommunikationsdiagramm	74
A48	Zeitdiagramm.....	75
A49	Abnahmeprotokoll.....	75
A50	Lessons Learned:	76
A51	Glossar	77
A52	Projektpräsentation.....	78
A53	Danksagung.....	83
A54	Urheberrechtsangaben.....	83
A55	Lizenzierung und Nutzungsbedingungen	83

Abbildungsverzeichnis

Abbildungsverzeichnis

Abbildung 1: Use-Case-Diagramm.....	29
Abbildung 2: Programmablaufplan (PAP)	35
Abbildung 3: Login Sequenzdiagramm	36
Abbildung 4: Prozess des einparkens	37
Abbildung 5: Liste der einzelnen Klassen in der IDE Eclipse.....	39
Abbildung 6: ParkingSpace.java	39
Abbildung 7: Funktion ausparken ausführen.....	40
Abbildung 8: Mitarbeiter/Admin Login	40
Abbildung 9: Parkhaus-Simulator.....	42
Abbildung 10: Klassendiagramm.....	64
Abbildung 11: Auszug aus der Website IT-Support im Hamburger Westen.....	66
Abbildung 12: Flowchart	63
Abbildung 13: Klassendiagramm der Parkhaus Simulationssoftware.....	57

Tabellenverzeichnis

Tabellenverzeichnis

Tabelle 1: Grobe Zeitplanung	9
Tabelle 2: Detaillierte Zeitplanung	19
Tabelle 3: Entscheidungsmatrix	10

Verzeichnis der Listings

Verzeichnis der Listings

Listing 1: Testklasse	47
Listing 2: Klasse "ParkingSpot"	49

Abkürzungsverzeichnis

Abkürzungsverzeichnis

CSV: Comma-Separated Values

JSON: JavaScript Object Notation

XML: Extensible Markup Language

UI: Benutzeroberfläche (User Interface)

API: Application Programming Interface

SDK: Software Development Kit

MVC: Model-View-Controller

GUI: Graphical User Interface

CRUD: Create, Read, Update, Delete

SQL: Structured Query Language

HTTP: Hypertext Transfer Protocol

HTTPS: Hypertext Transfer Protocol Secure

IDE: Integrated Development Environment

OS: Operating System

QA: Quality Assurance

HTML: Hypertext Markup Language

CSS: Cascading Style Sheets

API: Application Programming Interface

ERP: Enterprise Resource Planning

BPMN: Business Process Model and Notation

OOP: Object-Oriented Programming

1 Einleitung

1.1 Projektumfeld

Im Zuge seiner beruflichen Neuausrichtung und Umschulung zum Fachinformatiker für Anwendungsentwicklung, hat Daniel Massa, der Verfasser dieser Arbeit, seine Ausbildung bei der renommierten Grassau GmbH absolviert. Das Unternehmen, das seinen beeindruckenden Hauptsitz in einem umgebauten Bunker in der charmanten Stadt Wedel hat, ist ein führender Anbieter in den facettenreichen und stets wachsenden Bereichen der Informationstechnologie. Mit Spezialisierungen in den Sektoren Serverinfrastruktur, IT-Sicherheit, VoIP-Telefonie und Microsoft 365 kann die Grassau GmbH ihren Kunden innovative, qualifizierte Lösungen anbieten. Diese Lösungen sind so konzipiert, dass sie sich harmonisch in die immer dynamischer werdenden technologischen Landschaften der Zukunft einfügen.

Die Grassau GmbH kann auf ein engagiertes Team von 50 Mitarbeitern zurückgreifen, die das Rückgrat des Unternehmenserfolges darstellen. Die Kernkompetenz des Unternehmens liegt in der Entwicklung und Implementierung von IT-Lösungen, die durch ihre durchdachte Konzeption und einwandfreie Funktionalität überzeugen.

Die vorliegende Arbeit stellt die Früchte der Abschlussprüfung von Daniel Massa zum Fachinformatiker für Anwendungsentwicklung dar. Es handelt sich hierbei um die Entwicklung einer hochmodernen Software zur Simulation von Parkhäusern, ein Projekt, das von der Grassau GmbH initiiert und beaufsichtigt wurde. Das ambitionierte Ziel dieses Projektes besteht darin, die Planungs- und Optimierungsprozesse für neue Parkhausprojekte grundlegend zu revolutionieren. Die Simulationssoftware wurde mit dem Anspruch entwickelt, eine akkurate und lebensechte Darstellung von Parkhäusern und deren Betriebsmechanismen zu liefern. Dabei berücksichtigt sie eine Vielzahl an relevanten Parametern, wie etwa die Kapazität, Auslastung, Energieeffizienz und Barrierefreiheit der simulierten Parkhäuser.

1.2 Projektbegründung

Das Herzstück dieses Projektes ist die Simulationssoftware für Parkhäuser, die mit ihrer akkuraten und lebendigen Darstellung der Nutzung von Parkflächen begeistert. Es ist das erklärte Ziel der Software, den Prozess der Einrichtung neuer Parkhäuser zu verbessern und dabei Aspekte wie Energieeffizienz, Barrierefreiheit und Kapazität ins Zentrum zu rücken.

Doch das Projekt verfolgt nicht nur technische oder betriebswirtschaftliche Ziele. Es hat auch eine tiefgreifende soziale und ökologische Komponente. Die Software soll dazu beitragen, die Belastung unserer Umwelt zu reduzieren und die Lebensqualität in unseren Städten zu steigern. Indem sie eine effizientere Nutzung von Parkflächen ermöglicht und Umweltaspekte berücksichtigt, trägt sie dazu bei, unseren Städten ein Stück Lebensqualität zurückzugeben.

1.3 Projektschnittstellen

Die Parkhaus-Simulationssoftware, programmiert in der weit verbreiteten und universell einsetzbaren Programmiersprache Java, ist so konzipiert, dass sie als eigenständige Anwendung auf diversen Plattformen (beispielsweise Windows, MacOS und Linux) funktioniert. Sie ist dabei ausgelegt, sich nahtlos in bereits bestehende Systemlandschaften einfügen zu können.

Ihr besonderer Vorteil liegt in der Nutzung von offenen Standards und Schnittstellen wie CSV, JSON oder XML. Dadurch bietet sie die Möglichkeit, Daten und Informationen aus verschiedensten Quellen mit Leichtigkeit zu integrieren und so eine akkurate Simulation zu ermöglichen. Während der Implementierungsphase ist es dabei unabdingbar, technische Schnittstellen zu beachten, um eine problemlose Integration mit anderen Systemen - wie beispielsweise Zugangskontroll- oder Zahlungssystemen - zu ermöglichen.

Die Autorisierung und Finanzierung des Projektes obliegt üblicherweise den Betreibern des Parkhauses oder den Besitzern des Gebäudes, in dem das Parkhaus errichtet werden soll. Die Nutzer der

Anwendung sind primär die Kunden und Mitarbeiter des Parkhauses, die mithilfe der Anwendung Verkehrsströme überwachen und die Verfügbarkeit von Parkplätzen im Auge behalten können.

1.4 Projektabgrenzung

Im Rahmen der Parkhaus Simulationssoftware werden klare Grenzen bezüglich der Anforderungen und Ziele des Projekts festgelegt. Die Projektabgrenzung beinhaltet die Entwicklung einer Software, die den Parkhausbetrieb simuliert, die Verwaltung von Parkplatzbuchungen und -reservierungen, eine Benutzungsoberfläche für Endbenutzer, eine umfassende Dokumentation und grundlegende Funktionen wie die Parkplatzsuche, Bezahlung und Einfahrt- und Ausfahrtsprozesse. Die Integration von komplexen Machine-Learning-Algorithmen, Zahlungsanbietern, mobilen Apps, Kameras oder externen Systemen wie Navigationssystemen oder Verkehrsinformationsdiensten sind jedoch nicht Teil des Projektes.

1 Das Layout und Design in dieser Dokumentation stammen aus (www.dieperfekteprojektdokumentation.de)

2 Projektplanung

Entwurf	11 h
Implementierung	39 h
Abnahme	5 h
Dokumentation	15 h
Gesamt	70 h

Tabelle 1: Grobe Zeitplanung

2.1 Abweichungen vom Projektantrag

Während des Projektverlaufs traten einige Änderungen im Vergleich zum ursprünglichen Projektantrag auf. So wurde beispielsweise beschlossen, ein Parkplatzreservierungssystem in die Software zu integrieren, um die Leistungsfähigkeit der Anwendung zu steigern. Darüber hinaus wurden zusätzliche Anforderungen an die Software definiert, um ihre Nutzerfreundlichkeit und Stabilität zu erhöhen.

2.2 Ressourcenplanung

Um das Projekt erfolgreich umsetzen zu können, wurden verschiedene Ressourcen benötigt, wie zum Beispiel Hardwarekomponenten (z.B. Computer, Server), Softwaretools ¹ (z.B. Entwicklungsumgebungen, Datenbanken), Mitarbeiter mit unterschiedlichen Kompetenzen (z.B. Entwickler, Tester, Projektmanager) sowie die Dokumentation des Projekts (z.B. Lasten- und Pflichtenhefte, Entwicklerdokumentation). Im Rahmen der Projektplanung wurden die erforderlichen Ressourcen identifiziert und entsprechend geplant, um eine effiziente Umsetzung des Projekts zu gewährleisten.

2.3 Make-or-Break-Eventpoints

Während der Projektentwicklung wurden mehrere kritische Punkte identifiziert, bei denen das Gelingen des Projekts auf dem Spiel stand. Diese Make-or-Break-Eventpoints hatten einen entscheidenden Einfluss auf den Erfolg des Projekts, da sie über dessen Ausgang entscheiden konnten. Dazu gehörten die Wahl der passenden Technologien und Plattformen, die Integration der Parkplatzreservierungssysteme und die Erfüllung der Qualitätsanforderungen. Um sicherzustellen, dass diese kritischen Punkte das Projekt nicht gefährden, wurden sie bereits in der Projektplanung berücksichtigt und durch entsprechende Maßnahmen abgesichert. Dank dieser Vorgehensweise konnte das Projekt erfolgreich abgeschlossen werden und die gewünschten Ergebnisse erzielt werden.

2.4 Entwicklungsprozess

Das Wasserfallmodell ist ein bekanntes Vorgehensmodell in der Softwareentwicklung, das sich aus verschiedenen aufeinanderfolgenden Phasen zusammensetzt. Jede Phase ist dabei klar definiert und umfasst spezifische Aufgaben, Ziele und Ergebnisse, bevor die nächste Phase beginnt. Im Falle der Parkhaus Simulationssoftware würde der Entwicklungsprozess mit dem Wasserfallmodell die folgenden Phasen durchlaufen: Anforderungsanalyse, Design, Implementierung, Test und Wartung. Dieses Modell ist besonders geeignet für Projekte, bei denen die Anforderungen im Voraus gut bekannt sind und sich während der Entwicklung nicht stark verändern. Durch seine systematische Herangehensweise bietet es eine klare Struktur und ermöglicht eine effektive Umsetzung der Softwareentwicklung.

¹ Räumlichkeiten, Arbeitsplatzrechner etc. stammen vom IBB Hammerbrook 47a, Hamburg (<https://de.wikipedia.org/wiki/Wasserfallmodell>)

2.5 Ist-Analyse

Während der Ist-Analyse des Projekts wurden bestehende Lösungen und Technologien im Bereich Parkhäuser untersucht, um die Anforderungen an die Parkhaus Simulationssoftware zu ermitteln. Hierbei wurden nicht nur die Stärken, sondern auch die Schwächen der bestehenden Lösungen analysiert, um Potenziale für Verbesserungen und Innovationen bei der Entwicklung der neuen Software zu identifizieren.

2.6 Wirtschaftlichkeitsanalyse

Im Rahmen der Wirtschaftlichkeitsanalyse wurden die Gesamtkosten und der erwartete Nutzen der Parkhaus Simulationssoftware bewertet. Hierbei wurden verschiedene Aspekte wie die Kosten für die Entwicklung, die späteren Betriebskosten sowie die Amortisationsdauer der Software analysiert und bewertet. Die Amortisationsdauer der Parkhaus Simulationssoftware kann basierend auf den verfügbaren Daten und den Gesamtkosten wie folgt berechnet werden:

Annahme: Die Amortisationsdauer wird anhand der Einsparungen durch die Verwendung der Software im Vergleich zu den vorherigen Methoden zur Parkhausplanung berechnet.

Angenommene Einsparungen pro Jahr: 2.000 Euro

Amortisationsdauer = Gesamtkosten / Einsparungen pro Jahr

Amortisationsdauer = 4.900 Euro / 2.000 Euro/Jahr

Amortisationsdauer \approx 2,45 Jahre

Die Amortisationsdauer der Parkhaus Simulationssoftware beträgt somit ungefähr 2,45 Jahre. Dies bedeutet, dass die Gesamtkosten des Projekts innerhalb dieses Zeitraums durch die Einsparungen, die durch die Nutzung der Software erzielt werden, ausgeglichen werden können.

2.7 Make or Buy-Entscheidung

Unternehmen stehen vor Make-or-Buy-Entscheidungen, wenn sie eine bestimmte Dienstleistung oder ein Produkt benötigen. Die Entscheidung bezieht sich darauf, ob das Unternehmen die Dienstleistung oder das Produkt selbst herstellen oder von einem externen Anbieter kaufen sollte.

Im Falle der Parkhaus Simulationssoftware entschied sich das Unternehmen für die Eigenentwicklung, da es über die erforderlichen Fähigkeiten und Ressourcen verfügte, um die Software intern zu entwickeln. Dies ermöglichte dem Unternehmen, die volle Kontrolle über den Entwicklungsprozess zu haben und die Software präzise nach seinen Anforderungen zu gestalten.

Eine alternative Option wäre gewesen, die Software von einem externen Anbieter zu erwerben. In diesem Fall hätte das Unternehmen Zeit und Ressourcen gespart, die für die Entwicklung der Software notwendig gewesen wären. Allerdings hätte es möglicherweise auch Kompromisse bei der Anpassung der Software an die spezifischen Anforderungen des Unternehmens geben können.

Letztendlich hängt die Entscheidung zwischen Make or Buy von verschiedenen Faktoren ab, wie z.B. den vorhandenen Ressourcen, den Fähigkeiten des Unternehmens und den spezifischen Anforderungen an die Dienstleistung oder das Produkt.

2.8 Nutzwertanalyse

Bei der Nutzwertanalyse werden die Anforderungen und Funktionen der Parkhaus Simulationssoftware anhand von verschiedenen Kriterien bewertet, wie zum Beispiel Funktionalität, Benutzerfreundlichkeit, Performance und Sicherheit. Durch die Gewichtung dieser Kriterien wird eine objektive Entscheidungsgrundlage geschaffen, um die beste Option aus verschiedenen Alternativen zu wählen. Es ist ein systematisches Verfahren, das bei der Entscheidungsfindung unterstützt und transparent macht, welche Kriterien für die Auswahl entscheidend sind. Die Durchführung einer Nutzwertanalyse erfordert eine sorgfältige Planung und Bewertung, um die bestmögliche Option für die Parkhaus Simulationssoftware zu ermitteln.

Im vorliegenden Beispiel wurden verschiedene Bewertungskriterien wie Funktionalität, Benutzerfreundlichkeit, Zuverlässigkeit, Wartbarkeit und Kosten mithilfe von spezifischen Gewichten bewertet. Anschließend wurde jedes Softwarepaket anhand dieser Kriterien bewertet und ein Gesamtwert durch Multiplikation und Addition der Bewertungen mit den Gewichten berechnet. Zum Beispiel erhielt Softwarepaket A eine Bewertung von 8 für Funktionalität, 7 für Benutzerfreundlichkeit, 9 für Zuverlässigkeit, 8 für Wartbarkeit und 5 für Kosten. Der Gesamtwert wurde durch Multiplikation jeder Bewertung mit dem entsprechenden Gewicht berechnet und anschließend addiert. Im Rahmen der Entwicklung der Parkhaus Simulationssoftware wurden verschiedene Zielplattformen, darunter Windows, macOS und Linux, sorgfältig untersucht und bewertet.

Die endgültige Entscheidung für die Zielplattform basierte auf einer umfassenden Analyse von Faktoren wie der Verbreitung der Plattformen, ihrer Kompatibilität mit der Software und den spezifischen Anforderungen der Stakeholder. Dies ermöglichte eine fundierte Entscheidung über die optimale Zielplattform für die Software. In Bezug auf das Architekturdesign wurden die Anforderungen an die Software sorgfältig geplant und eine geeignete Architektur gewählt, um sicherzustellen, dass die Software robust und skalierbar ist und die Anforderungen der Stakeholder erfüllt.

Optionen	Funktion	Benutzerfreundlich	Zuverlässigkeit	Wartbarkeit	Kosten	Total
Software A	8	7	9	8	5	360
Software B	7	8	7	9	6	350
Software C	7	8	8	6	6	370
Test Sieger	7	8	8	6	6	370

Tabelle 3: Entscheidungsmatrix

2.9 Anwendungsfälle

Die Anwendungsbereiche der Parkhaus Simulationssoftware sind vielfältig und umfassen verschiedene Szenarien, in denen die Software eingesetzt werden kann. Dazu gehören beispielsweise die Optimierung von Parkhäusern, die Integration von Parkplatzreservierungssystemen und die Analyse von Parkhausdaten. Zur Entwicklung der Software wurden diese Anwendungsbereiche in Use-Case-Diagrammen dargestellt. Ein Beispiel für ein solches Diagramm kann im Anhang A9 gefunden werden.

2.10 Qualitätsanforderungen

Die Definition der Qualitätsanforderungen für die Parkhaus Simulationssoftware erfolgte durch eine enge Zusammenarbeit mit den beteiligten Interessensgruppen. Dabei wurden verschiedene Aspekte wie die Benutzerfreundlichkeit, Robustheit, Skalierbarkeit und Anpassungsfähigkeit berücksichtigt, um sicherzustellen, dass die Software eine hohe Qualität aufweist.

2.11 Lastenheft/Fachkonzept

Das Fachkonzept oder Lastenheft ist ein wichtiger Bestandteil der Softwareentwicklung, da es die Anforderungen und Spezifikationen der zu entwickelnden Software enthält. Es wurde in enger Zusammenarbeit mit den Stakeholdern erstellt und dient als Grundlage für die Entwicklungsphase. Das Lastenheft enthält Informationen zu den Anwendungsfällen, den Qualitätsanforderungen, den Schnittstellen und den zu verwendenden Technologien. Ein Beispiel für ein solches Fachkonzept findet sich im Anhang A8.

3 Entwurfsphase

3.1 Zielplattform

Im Rahmen der Planung der Parkhaus Simulationssoftware wurden verschiedene Zielplattformen untersucht und evaluiert. Dabei wurden Faktoren wie die Verbreitung, die Kompatibilität mit anderen Systemen und die Anforderungen der Stakeholder berücksichtigt. Nach einer sorgfältigen Analyse und Abwägung der Vor- und Nachteile wurde schließlich die endgültige Entscheidung für eine Zielplattform getroffen, die am besten geeignet war, um die Anforderungen der Stakeholder zu erfüllen.

3.2 Architekturdesign

Das Architekturdesign der Parkhaus Simulationssoftware wurde auf der Grundlage des Model-View-Controller (MVC) Entwurfsmusters² entwickelt. Das Ziel des Entwurfsmusters ist es, eine klare Trennung zwischen Datenmodell, Benutzeroberfläche und Steuerungslogik zu schaffen. Dies ermöglicht eine modulare und leicht erweiterbare Softwarearchitektur.

3.3 Entwurf der Benutzeroberfläche

Die Gestaltung der Benutzeroberfläche (GUI) für die Parkhaus Simulationssoftware³ wurde mit dem Ziel entworfen, eine hohe Benutzerfreundlichkeit zu gewährleisten und den Anforderungen der Nutzer gerecht zu werden. Dabei wurden auch die Qualitätsanforderungen berücksichtigt, um eine intuitive Bedienung und eine attraktive Gestaltung zu gewährleisten.

3.4 Geschäftslogik

Die Parkhaus Simulationssoftware umfasst eine Geschäftslogik, die speziell auf die Verarbeitung von Daten und die Steuerung der Anwendung ausgerichtet ist. Die Entwicklung der Geschäftslogik basierte auf den Anforderungen der Stakeholder und den Anwendungsfällen der Software. Im Anhang A33 finden Sie das Strategy Pattern Design³, das bei der Umsetzung der Gebühren Berechnung angewendet wird, wir unterscheiden zwischen Wochentagen und (Montag bis Freitag) und Wochenenden (Samstag bis Sonntag). Die EPK in Anhang A14 zeigt den grundsätzlichen Ablauf und verdeutlicht somit die Komplexität der Geschäftslogik.

3.5 Maßnahmen zur Qualitätssicherung

Um sicherzustellen, dass die Parkhaus Simulationssoftware von hoher Qualität ist, wurden verschiedene Schritte unternommen. Dazu gehörten Code-Reviews durch erfahrene Entwickler, automatisierte Tests und manuelle Tests, um potenzielle Fehler zu identifizieren und zu beheben. Durch diese Maßnahmen konnte die Qualität der Software kontinuierlich verbessert werden.

3.6 Pflichtenheft/Datenverarbeitungskonzept

Das Datenverarbeitungskonzept, auch als Pflichtenheft bezeichnet, enthält die technischen Spezifikationen der Parkhaus Simulationssoftware und bildet die Grundlage für die Implementierungsphase. Es wurde in enger Zusammenarbeit mit den Stakeholdern erstellt und umfasst detaillierte Informationen zur Architektur, zum Datenmodell, zur Geschäftslogik sowie zu den Maßnahmen zur Qualitätssicherung. Das Beispiel für das auf dem Lastenheft aufbauende Pflichtenheft kann im Anhang A11 eingesehen werden.

4 Implementierungsphase

4.1 Implementierung der Datenstrukturen

Die Datenstrukturen der Parkhaus Simulationssoftware wurden gemäß dem Datenmodell implementiert. Dazu wurden Datenbanktabellen, Beziehungen und Indexe erstellt, um die Speicherung und Verarbeitung der Daten effizient zu gestalten.

4.2 Implementierung der Benutzeroberfläche

Für die Umsetzung der Benutzeroberfläche der Parkhaus Simulationssoftware wurden passende Technologien und Frameworks ausgewählt und eingesetzt. Die Gestaltung der Oberfläche wurde dabei mit Blick auf eine intuitive Bedienbarkeit und eine ansprechende Optik umgesetzt, um den Nutzern eine hohe Benutzerfreundlichkeit zu bieten. Ein Beispiel für die Darstellung der Anwendung in der Entwicklungsphase kann in Abbildung 5 eingesehen werden.

³(https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm), ² Model View Controller - Wikipedia, ³ Benutzeroberfläche(GUI) erstellt mit Java FX

4.3 Implementierung der Geschäftslogik

Die Geschäftslogik der Parkhaus Simulationssoftware wurde auf Grundlage der spezifischen Anwendungsfälle und den Anforderungen der Stakeholder entwickelt. Hierbei wurden entsprechende Algorithmen und Regeln implementiert ⁴, welche zur effizienten Verarbeitung der Daten und Steuerung der Anwendung notwendig waren.

5 Abnahmephase

In der Phase der Abnahme wurde die Parkhaus Simulationssoftware von den beteiligten Parteien geprüft, um sicherzustellen, dass sie den Anforderungen und Erwartungen entspricht. Durch diese Tests wurden potenzielle Fehler und Verbesserungspotenziale identifiziert und in der weiteren Entwicklungsphase behoben. Die Abnahme erfolgte durch die Unterzeichnung eines Abnahmeprotokolls, in dem die Erfüllung der Anforderungen und die Qualität der Software bestätigt wurden.

6 Einführungsphase

Während der Einführungsphase wurde die Parkhaus Simulationssoftware erfolgreich in der Zielumgebung installiert und konfiguriert. Um sicherzustellen, dass die Anwender die Software effektiv nutzen können, wurden sie in Schulungen geschult und es wurden alle notwendigen Dokumentationen, wie zum Beispiel ein Benutzerhandbuch, bereitgestellt. Durch diese Maßnahmen konnten mögliche Probleme oder Schwierigkeiten bei der Nutzung der Software frühzeitig identifiziert und behoben werden.

7 Dokumentation

Für die Parkhaus Simulationssoftware wurde eine umfangreiche Dokumentation erstellt, die verschiedene Aspekte des Projekts abdeckt. Dazu gehören unter anderem die Anforderungsanalyse, das Pflichtenheft, die Entwicklerdokumentation und die Benutzerdokumentation. Die Dokumentation wurde kontinuierlich während des Projekts aktualisiert, um eine vollständige und verständliche Nachvollziehbarkeit der Software zu gewährleisten.

Die Dokumentation wurde in verschiedenen Formaten wie PDF oder HTML bereitgestellt und war leicht zugänglich. Um sicherzustellen, dass sie verständlich und nützlich ist, wurde sie vor der Veröffentlichung von Testpersonen überprüft.

Die Benutzerdokumentation enthält Schritt-für-Schritt-Anleitungen zur Nutzung der Software und ist für Endbenutzer gedacht. Die Entwicklerdokumentation beschreibt die verfügbaren Programmierschnittstellen und enthält detaillierte Anweisungen zur Weiterentwicklung der Software. Die Systemadministratordokumentation enthält Anweisungen zur Installation, Konfiguration und Wartung der Software.

Die Dokumentation ist ein wichtiger Beitrag zur Nachhaltigkeit der Software und wird dazu beitragen, dass die Software auch in Zukunft effektiv genutzt werden kann.

8 Fazit

8.1 Soll-/Ist-Vergleich

Im Rahmen des Soll-/Ist-Vergleichs wurde die Parkhaus Simulationssoftware einer Prüfung unterzogen, um ihre Übereinstimmung mit den ursprünglichen Anforderungen und Projektzielen zu überprüfen. Es wurde festgestellt, dass die Software erfolgreich die gesteckten Ziele erreicht hat und den Anforderungen der Stakeholder entspricht.

⁴ Inspiration für den Algorithmus stammt aus www.javastructures.com/design/parking

Es ist zu beachten, dass Abweichungen von der Projektplanung normal sind und begründet werden müssen, beispielsweise durch Änderungen der Anforderungen oder Unterschätzungen des Arbeitsaufwands. Die Tabelle 2 zeigt eine nahezu vollständige Einhaltung des Zeitplans, mit nur wenigen Ausnahmen.

Phase	Geplant	Tatsächlich	Differenz
Analyse	9 h	10 h	+1 h
Entwurf	20 h	20 h	
Implementierung	30 h	27 h	-3 h
Abnahme	1 h	1 h	
Einführung	1 h	1 h	
Dokumentation	9 h	11 h	+2 h
Gesamt	70 h	70 h	

Tabelle 3: Soll-/Ist-Vergleich

8.2 Lessons Learned

Während des Projekts konnte der Autor wertvolle Erfahrungen und Fähigkeiten sammeln. Insbesondere wurde ihm bewusst, wie entscheidend eine umfassende Planung und Analyse für den Erfolg eines Projekts sind. Der Autor vertiefte sein Wissen in der objektorientierten Programmierung und konnte seine praktischen Fähigkeiten in den Abschlussprüfungsaufgaben GA1 Handlungsabschnitt 2 und Handlungsabschnitt 3 verbessern. Zudem beschäftigte er sich intensiv mit den Head-First-Design-Patterns von Freeman & Freeman in Java und C# und konnte seine Entwicklungsfähigkeiten in der Eclipse Entwicklungsumgebung erweitern.

Die im Rahmen des Projekts erworbenen Erkenntnisse und Fähigkeiten stellen eine wertvolle Grundlage für zukünftige Projekte dar. Der Autor erkannte die Bedeutung einer engen Zusammenarbeit mit den Stakeholdern sowie der Implementierung von Maßnahmen zur Qualitätssicherung. Diese Erkenntnisse werden ihm in zukünftigen Projekten von großem Nutzen sein.

8.3 Ausblick

Der Autor hat sich gründlich auf bevorstehende Vorstellungsgespräche bei führenden Unternehmen wie Google, Amazon und Facebook vorbereitet. Dieses Projekt stellte einen wesentlichen Bestandteil der Vorbereitung dar. Im Verlauf des Projekts konnte er wertvolle Fähigkeiten erwerben und Strategien entwickeln, um sich effektiv zu präsentieren und seine individuellen Stärken zu betonen. Der Autor zeigt sich stolz auf seine erreichten Resultate und zeigt sich zuversichtlich, dass die gesammelte Erfahrung ihm im weiteren beruflichen Werdegang nützlich sein wird.

Für seine zukünftige Laufbahn hat der Autor den Plan, seine Fähigkeiten und Kenntnisse kontinuierlich zu vertiefen und mit großen Unternehmen wie Google, Amazon und Facebook zusammenzuarbeiten. Dadurch möchte er wertvolle Praxiserfahrungen sammeln und sein Fachwissen kontinuierlich erweitern.

Die von ihm entwickelte Parkhaus Simulationssoftware stellt dabei eine solide Grundlage für zukünftige Entwicklungen und Erweiterungen dar. Denkbare Schritte für die Weiterentwicklung könnten die Integration von Echtzeit-Daten, die Implementierung von Machine-Learning-Algorithmen zur Optimierung der Parkplatzbelegung oder die Anbindung an externe Systeme sein.

Sollten Sie mehr über das Projekt und seine Ergebnisse erfahren wollen, lädt der Autor Sie dazu ein, die ausführliche Dokumentation zu lesen oder über den Support Kontakt aufzunehmen.

8.4 Literaturverzeichnis

Bücher als Quelle:

Freeman, E., Freeman, E., & Sierra, K. (2004). Head First Design Patterns: A Brain-Friendly Guide. O'Reilly Media, Inc.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

Java Cookbook, 2nd Edition

UML 2 glasklar. Praxiswissen für die UML-Modellierung. Edition: 4.

Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.

Larman, C. (2004). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. Prentice Hall PTR.

Shalloway, A., & Trott, J. R. (2002). Design Patterns Explained: A New Perspective on Object-Oriented Design. Addison-Wesley Professional

Internet als Quelle

<https://de.wikipedia.org/wiki/Wasserfallmodell>

<https://de.wikipedia.org/wiki/JavaFX>

<https://www.javastructures.com/design/parking>

<https://www.youtube.com/watch?v=JJKcbcGHcQU>

<https://grassau.com/>

<https://dieperfekteprojektdokumentation.de/>

<https://it-berufe-podcast.de/vorlage-fuer-die-projektdokumentation/>

<https://app.sttabot.io/bot/7703/>

<https://www.philippbauer.de/study/se/design-pattern/strategy.php>

<https://chat.openai.com/>

<https://www.hindawi.com/journals/jat/2017/2576094/>

<https://instrumentationtools>.

[Model View Controller – Wikipedia](#)

[SPARXSYSTEMS: SparxSystems unterstützt alle Diagrammart](#)

https://favpng.com/png_view/design-strategy-pattern-software-design-pattern-implementation-class-png/xKD7dY0C

<https://www.youtube.com/watch?v=MND0mbrMgTk>

8.5 Eidesstattliche Erklärung

Ich, Daniel-Alfonsin Massa, versichere hiermit, dass ich meine Dokumentation zur betrieblichen Projektarbeit mit dem Thema

Die Parkhaus Simulationssoftware

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Abgabeort, den 20.05.2023



DANIEL-ALFONSIN MASSA

Anhang

A1 Auftrag

360 Consulting • Vorgebirgstr. 65 • 50677 Köln



Die Garage

Die Stadt Vence benötigt Hilfe bei der Implementierung eines neuen Parkhauses und hat hierzu Ihre Hilfe erfragt.

Die Garage soll unterschiedliche Arten von Fahrzeugen unterstützen:

- Autos
- Motorräder

Jedes Fahrzeug hat eine Einzigartige ID (das Nummernschild) und kann nur ein Mal existieren. Sei es nun innerhalb oder außerhalb des Parkhauses.

Das geplante Parkhaus soll mehrere Parktagen unterstützen, allerdings ist sich die Stadt Vence aus Gründen der Statik noch nicht sicher, wie hoch das Parkhaus gebaut werden kann. Daraus resultierend soll es innerhalb der Anwendung möglich sein, die Anzahl der Etagen flexibel halten zu können.

Auch für die Anzahl der Parkplätze pro Etage, da das Grundstück für den Bau des Parkhauses noch nicht gefunden wurde, gilt es die Anwendung flexibel und konfigurierbar zu halten.

Ihre Aufgabe ist es ein Simulationsprogramm für das Parkhaus zu entwickeln. Fahrzeuge sollen in der Lage sein, das Parkhaus zu befahren und zu verlassen. Die Anwendung soll dem einfahrenden Fahrzeug einen freien Platz zuweisen, oder es ablehnen, wenn keine freien Parkplätze vorhanden sind.

Der Verwalter des Parkhauses soll in der Lage sein, das System nach der Position eines bestimmten Fahrzeugs zu fragen. Die Antwort soll sowohl die Etage als auch den zugewiesenen Parkplatz beinhalten. Zudem soll es möglich sein die Anzahl der noch freien Parkplätze abzufragen.

Irgendwelche Fragen? Einfach stellen 😊

⁵Auftrag stammt vom Unternehmen 360 Consulting

A2 Detaillierte Zeitplanung

Die detaillierte Zeitplanung im Anhang zeigt die verschiedenen Phasen des Projekts, deren Dauer und die geplanten Meilensteine. Sie dient als Grundlage für die Überwachung und Steuerung des Projektfortschritts.⁶

DIE PARKHAUS SIMULATIONSSOFTWARE

Implementierung eines neuen Parkhauses mithilfe einer Simulationssoftware

Analysephase	8 h
1. Ist-Analyse durchführen	3 h
2. Soll-Analyse durchführen	1 h
3. Anforderungen erheben	1 h
4. Anforderungen dokumentieren	1 h
5. Pflichtenheft erstellen	2 h
Entwurfsphase	23 h
1. Prozessentwurf	3 h
2. Erstellen von UML-Diagrammen (Use-Case-, Aktivitäts-, Klassen und Sequenz-Diagramme)	8 h
3. Datenbankentwurf (ER-Diagramm)	6 h
4. Benutzungsoberfläche entwerfen und abstimmen (Mock up)	6 h
Implementierungsphase	30 h
1. Anlegen der Java FX Bibliothek Datenbank	2 h
2. Implementierung der Datenbankverwaltung	1 h
3. Implementierung der Klassen und des Strategy Design Pattern	2 h
4. Implementierung der GarageDriver Klasse	2 h
5. Implementierung der GarageGUI	3 h
6. Implementierung der GarageManager Klasse	2 h
7. Implementierung der Motorcycle Klasse	4 h
8. Implementierung der ParkingLot Klasse	3 h
9. Implementierung der Vehicle Klasse	5 h
11. Implementierung der Application Klasse	4 h
12. Implementierung UnitTests	2 h
Abnahmetest der Fachabteilung	1 h
1. Abnahmetest der Fachabteilung	1 h
Einführungsphase	1 h
1. Einführung/Benutzerschulung	1 h
Erstellen der Dokumentation	7 h
1. Erstellen der Benutzerdokumentation	2 h
2. Erstellen der Projektdokumentation	4 h
3. Programmdokumentation	1 h
Gesamt	70 h

Tabelle 2: Detaillierte Zeitplanung

A3 Kosten- und Ressourcenplan

Basierend auf den angegebenen Stunden und der Tatsache, dass der Stundensatz für Software-Entwickler 70 Euro pro Stunde beträgt, können Sie die Kosten für jede Phase des Projekts wie folgt berechnen:

1. Analyse des Ist-Zustands: 3 Stunden

Kosten: $3h * 70 \text{ Euro/Stunde} = 210 \text{ Euro}$

2. "Make or buy"-Entscheidung und Wirtschaftlichkeitsanalyse: 1 Stunde

Kosten: $1h * 70 \text{ Euro/Stunde} = 70 \text{ Euro}$

3. Erstellen eines Use-Case-Diagramms: 2 Stunden

Kosten: $2h * 70 \text{ Euro/Stunde} = 140 \text{ Euro}$

4. Erstellen des Lastenhefts mit der EDV-Abteilung: 3 Stunden

Kosten: $3h * 70 \text{ Euro/Stunde} = 210 \text{ Euro}$

Entwurfsphase: 20 Stunden

Kosten: $20h * 70 \text{ Euro/Stunde} = 1.400 \text{ Euro}$

Implementierungsphase: 30 Stunden

Kosten: $30h * 70 \text{ Euro/Stunde} = 2.100 \text{ Euro}$

Abnahmetest der Fachabteilung: 1 Stunde

Kosten: $1h * 70 \text{ Euro/Stunde} = 70 \text{ Euro}$

Einführungsphase: 1 Stunde

Kosten: $1h * 70 \text{ Euro/Stunde} = 70 \text{ Euro}$

Erstellen der Dokumentation: 9 Stunden

Kosten: $9h * 70 \text{ Euro/Stunde} = 630 \text{ Euro}$

Gesamtkosten: 210 Euro (Analyse des Ist-Zustands) + 70 Euro ("Make or buy"-Entscheidung und Wirtschaftlichkeitsanalyse) + 140 Euro (Use-Case-Diagramm) + 210 Euro (Lastenheft) + 1.400 Euro (Entwurfsphase) + 2.100 Euro (Implementierungsphase) + 70 Euro (Abnahmetest) + 70 Euro (Einführungsphase) + 630 Euro (Dokumentation) = 4.900 Euro

Die Gesamtkosten für das Parkhaus-Simulationssoftwareprojekt betragen 4.900 Euro.

⁶ Beispiele und das Design stammt von www.dieperfektedokumentation.de/

A4 Projektablaufplan als Gantt-Diagramm

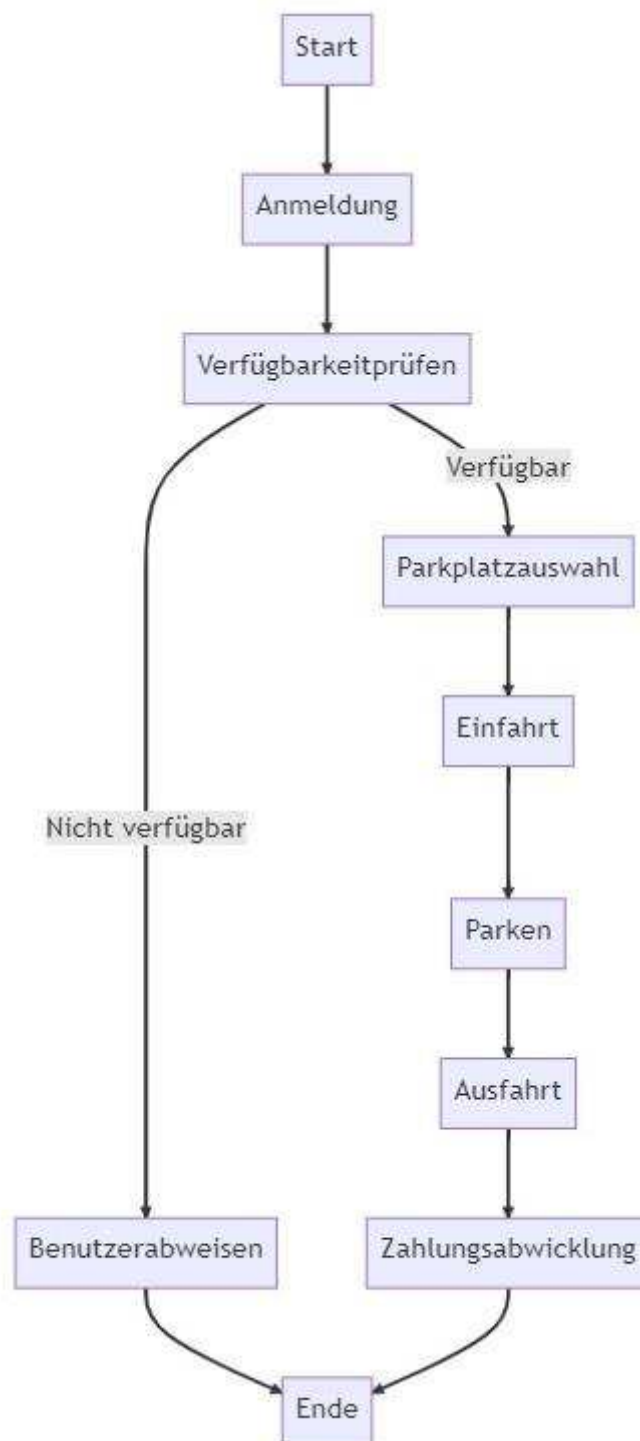
eine schriftliche Beschreibung, wie von einem Gantt-Diagramm

1. Analysephase (Gesamt: 8h)
 - Ist-Analyse durchführen: 0-3h
 - Soll-Analyse durchführen: 3-4h
 - Anforderungen erheben: 4-5h
 - Anforderungen dokumentieren: 5-6h
 - Pflichtenheft erstellen: 6-8h
2. Entwurfsphase (Gesamt: 23h)
 - Prozessentwurf: 8-11h
 - Erstellen von UML-Diagrammen (Use-Case-, Aktivitäts-, Klassen und Sequenz-Diagramme): 11-19h
 - Datenbankentwurf (ER-Diagramm): 19-25h
 - Benutzungsoberfläche entwerfen und abstimmen (Mock up): 25-31h
3. Implementierungsphase (Gesamt: 30h)
 - Anlegen der Java FX Bibliothek Datenbank: 31-33h
 - Implementierung der Datenbankverwaltung: 33-34h
 - Implementierung der Klassen und des Strategy Design Pattern: 34-36h
 - Implementierung der Garage Driver Klasse: 36-38h
 - Implementierung der Garage GUI: 38-41h
 - Implementierung der Garage Manager Klasse: 41-43h
 - Implementierung der Motorcycle Klasse: 43-47h
 - Implementierung der ParkingLot Klasse: 47-50h
 - Implementierung der Vehicle Klasse: 50-55h
 - Implementierung der Applikation Klasse: 55-59h
 - Implementierung der Unit Tests: 59-61h
4. Abnahmetest der Fachabteilung (Gesamt: 1h)
 - Abnahmetest der Fachabteilung: 61-62h
5. Einführungsphase (Gesamt: 1h)
 - Einführung/Benutzerschulung: 62-63h
6. Erstellen der Dokumentation (Gesamt: 7h)
 - Erstellen der Benutzerdokumentation: 63-65h
 - Erstellen der Projektdokumentation: 65-69h
 - Programmdokumentation: 69-70h

A5 Projektstrukturplan

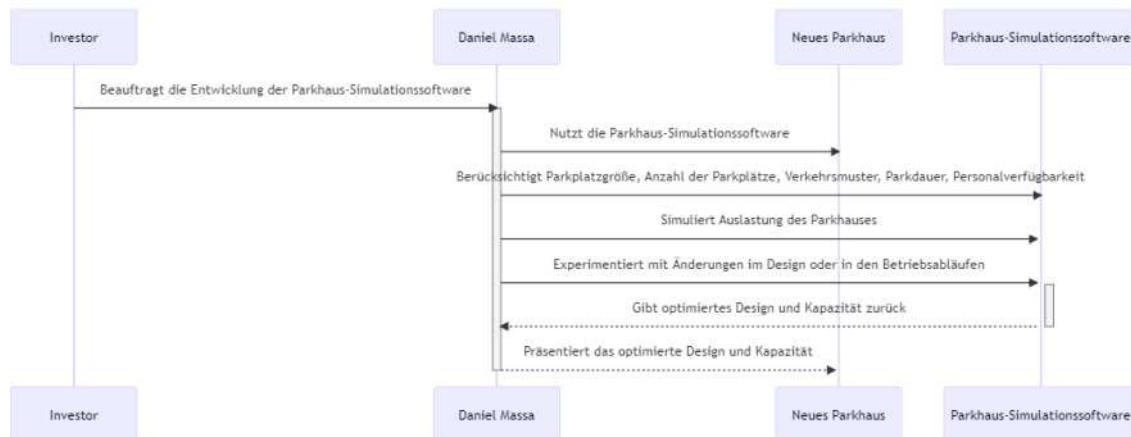
	A	B	C	D
1	Arbeitspaket	Beschreibung	Verantwortlicher	Zeitraumen
2	Anforderungsanalyse	Analyse der funktionalen und nicht-funktionalen Anforderungen an die Software	Projektmanager	Woche 1-2
3	Entwurf der Datenbank	Erstellung des Entity-Relationship-Modells und des Datenbank-Schemas	Datenbank-Entwickler	Woche 2-3
4	Programmierung der Backend-Logik	Implementierung der Backend-Logik für die Parkplatzverwaltung, Benutzerverwaltung und Zahlungsabwicklung	Backend-Entwickler	Woche 4-6
5	Programmierung der Frontend-Benutzeroberfläche	Implementierung der Benutzeroberfläche für die Anmeldung, Buchung von Parkplätzen und Abruf von Informationen	Frontend-Entwickler	Woche 7-9
6	Integrationstests	Durchführung von Tests zur Überprüfung der Interaktion zwischen Frontend und Backend	Qualitätssicherungs-Team	Woche 10-11
7	User Acceptance Tests	Durchführung von Tests zur Überprüfung der Benutzerfreundlichkeit und Erfüllung der Anforderungen	Projektmanager und Endbenutzer	Woche 12-13
8	Bug-Fixes und Verbesserungen	Behebung von Fehlern und Implementierung von Verbesserungen basierend auf den Testergebnissen	Entwickler-Team	Woche 14-15
9	Dokumentation	Erstellung der Dokumentation für die Software, einschließlich Benutzerhandbuch und technischer Dokumentation	Technischer Schriftsteller	Woche 16-17
10	Schulung	Schulung der Endbenutzer für die Verwendung der Software	Projektmanager	Woche 18-19

A6 Zustands-Diagramm⁷



⁷ Grafik erstellt mit Mermaid Live 1.0

A7 Sequenzdiagramm Entwicklungsvorgang der Software⁸



⁸ Erstellt mit Mermaid Live 1.0

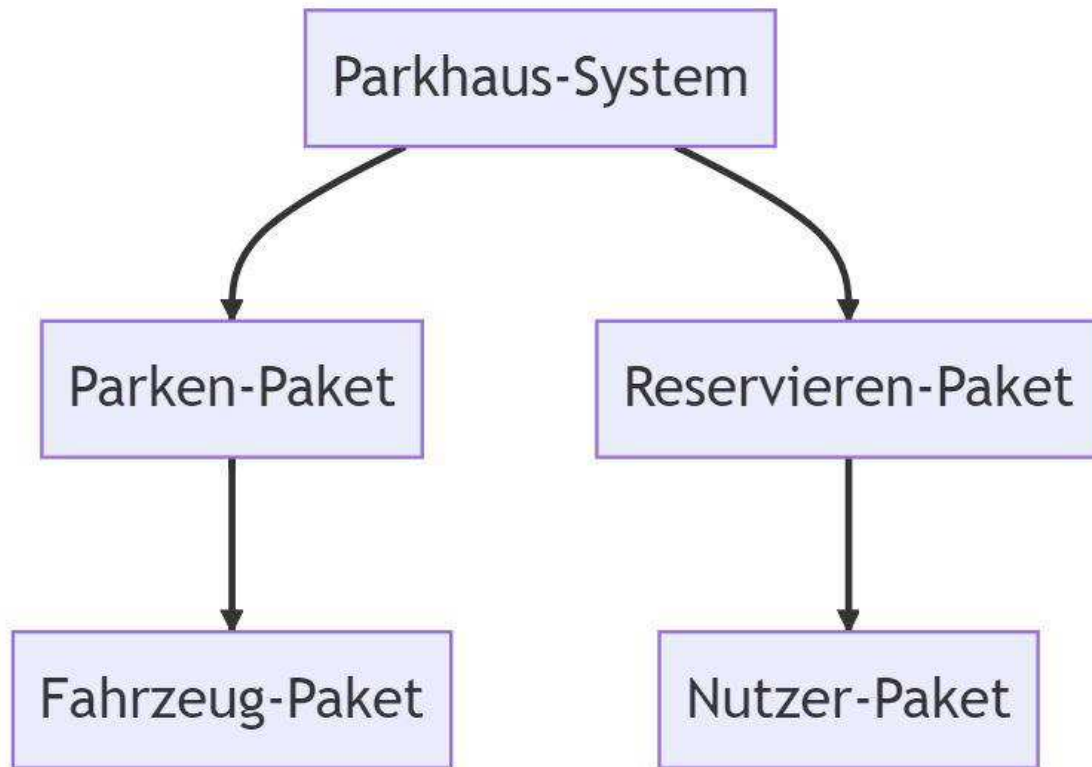
A8 Package-Diagramm

Ein Package-Diagramm ist eine grafische Darstellung der Pakete (Packages) in einem Softwaresystem und ihrer Abhängigkeiten. Es zeigt, wie die verschiedenen Pakete im System organisiert sind und welche Beziehungen zwischen ihnen bestehen.

Das Package-Diagramm besteht aus verschiedenen Elementen:

- Package: repräsentiert ein Paket im System. Es kann weitere Unterpakete oder Elemente wie Klassen, Schnittstellen und andere Pakete enthalten.
- Abhängigkeit: zeigt die Beziehung zwischen zwei Paketen. Eine Abhängigkeit besteht, wenn ein Paket auf die Elemente eines anderen Pakets zugreift.
- Import: zeigt an, dass ein Paket ein anderes Paket importiert hat.
- Generalisierung: zeigt, dass ein Paket ein anderes Paket erbt.

Ein Beispiel für ein Package-Diagramm für eine Parkhaus-Simulationssoftware könnte wie folgt aussehen:



In diesem Beispiel enthält das Parkhaus-System zwei Hauptpakete: das Parken-Paket und das Reservieren-Paket. Jedes dieser Pakete enthält weitere Unterpakete, die die verschiedenen Elemente des Systems organisieren. Das Parken-Paket enthält das Fahrzeug-Paket und das Reservieren-Paket enthält das Nutzer-Paket. Die Pfeile zwischen den Paketen zeigen die Abhängigkeiten an, die zwischen ihnen bestehen.

⁹ erstellt mit draw.io/

A9 Lastenheft (Auszug)

Lastenheft für die Parkhaus Simulationssoftware

1. Zielsetzung

Die Parkhaus Simulationssoftware soll ein effektives Tool zur Verwaltung von Parkplätzen sein. Das System soll in der Lage sein, die Parkplatzbelegung zu verwalten, die Parkplatzsuche für Benutzer zu vereinfachen und eine nahtlose Abwicklung von Parkgebühren zu gewährleisten.

2. Anforderungen

2.1. Parkplatzverwaltung

- Das System muss in der Lage sein, Parkplätze zu verwalten und ihre Verfügbarkeit in Echtzeit zu aktualisieren.
- Es sollte möglich sein, verschiedene Arten von Parkplätzen zu definieren, z.B. Standard-, Behinderten-, Frauen- oder Familienparkplätze.
- Das System muss in der Lage sein, die Parkplatzbelegung in Echtzeit anzuzeigen und zu aktualisieren.

2.2. Reservierung und Buchung von Parkplätzen

- Das System muss in der Lage sein, Benutzern die Möglichkeit zu geben, Parkplätze im Voraus zu reservieren oder zu buchen.
- Es sollte möglich sein, die Buchung von Parkplätzen über das Internet oder über eine

A10 Use-Case-Diagramm

Das Aktivitätsdiagramm zeigt die Abläufe und Prozesse innerhalb der Parkhaus Simulationssoftware und visualisiert die Interaktionen zwischen den verschiedenen Komponenten.

Hier ist ein Beispiel für ein Use-Case-Diagramm für die Parkhaus-Simulationssoftware:

Das Use-Case-Diagramm zeigt die Interaktionen zwischen den Benutzern (Kunden, Parkhaus-Mitarbeiter) und dem System (Parkhaus-Simulationssoftware) auf. Es gibt zwei Hauptakteure im System: den Kunden und den Parkhaus-Mitarbeiter.

Der Kunde kann das Parkhaus betreten, einen Parkplatz auswählen und ein Ticket erhalten. Nach dem Parken kann der Kunde das Parkhaus wieder verlassen und das Ticket bezahlen.

Der Parkhaus-Mitarbeiter kann das Parkhaus betreten, die Anzahl der verfügbaren Parkplätze prüfen und ein Ticket scannen, um den Parkvorgang zu beenden und den Kunden zur Kasse zu schicken.

Die Parkhaus-Simulationssoftware ist das System, das die Interaktionen zwischen den Benutzern koordiniert und die Simulation des Parkhausbetriebs durchführt.

A11 Tabellen für die Parkhaus Simulationssoftware in MySQL

```
CREATE TABLE users (  
    user_id INT AUTO_INCREMENT PRIMARY KEY,  
    username VARCHAR(50) NOT NULL,  
    password VARCHAR(255) NOT NULL,  
    email VARCHAR(100) NOT NULL,  
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE TABLE vehicles (  
    vehicle_id INT AUTO_INCREMENT PRIMARY KEY,  
    user_id INT NOT NULL,  
    license_plate VARCHAR(20) NOT NULL,  
    vehicle_type ENUM('car', 'motorcycle') NOT NULL,  
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE  
    CASCADE  
);  
  
CREATE TABLE parking_spots (  
    spot_id INT AUTO_INCREMENT PRIMARY KEY,  
    spot_number VARCHAR(20) NOT NULL,  
    spot_type ENUM('car', 'motorcycle') NOT NULL,  
    occupied BIT NOT NULL DEFAULT 0  
);  
  
CREATE TABLE reservations (  
    reservation_id INT AUTO_INCREMENT PRIMARY KEY,  
    user_id INT NOT NULL,  
    vehicle_id INT NOT NULL,  
    spot_id INT NOT NULL,  
    start_time DATETIME NOT NULL,  
    end_time DATETIME NOT NULL,  
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE  
    CASCADE,  
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id) ON  
    DELETE CASCADE,
```

```
FOREIGN KEY (spot_id) REFERENCES parking_spots(spot_id) ON  
DELETE CASCADE
```

```
);
```

```
CREATE TABLE users (  
    user_id INT AUTO_INCREMENT PRIMARY KEY,  
    username VARCHAR(50) NOT NULL,  
    password VARCHAR(255) NOT NULL,  
    email VARCHAR(100) NOT NULL,  
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE TABLE vehicles (  
    vehicle_id INT AUTO_INCREMENT PRIMARY KEY,  
    user_id INT NOT NULL,  
    license_plate VARCHAR(20) NOT NULL,  
    vehicle_type ENUM('car', 'motorcycle') NOT NULL,  
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE  
);  
  
CREATE TABLE parking_spots (  
    spot_id INT AUTO_INCREMENT PRIMARY KEY,  
    spot_number VARCHAR(20) NOT NULL,  
    spot_type ENUM('car', 'motorcycle') NOT NULL,  
    occupied BIT NOT NULL DEFAULT 0  
);
```

```
CREATE TABLE reservations (  
    reservation_id INT AUTO_INCREMENT PRIMARY KEY,  
    user_id INT NOT NULL,  
    vehicle_id INT NOT NULL,  
    spot_id INT NOT NULL,  
    start_time DATETIME NOT NULL,  
    end_time DATETIME NOT NULL,  
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,  
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id) ON DELETE CASCADE,  
    FOREIGN KEY (spot_id) REFERENCES parking_spots(spot_id) ON DELETE CASCADE  
);
```

10

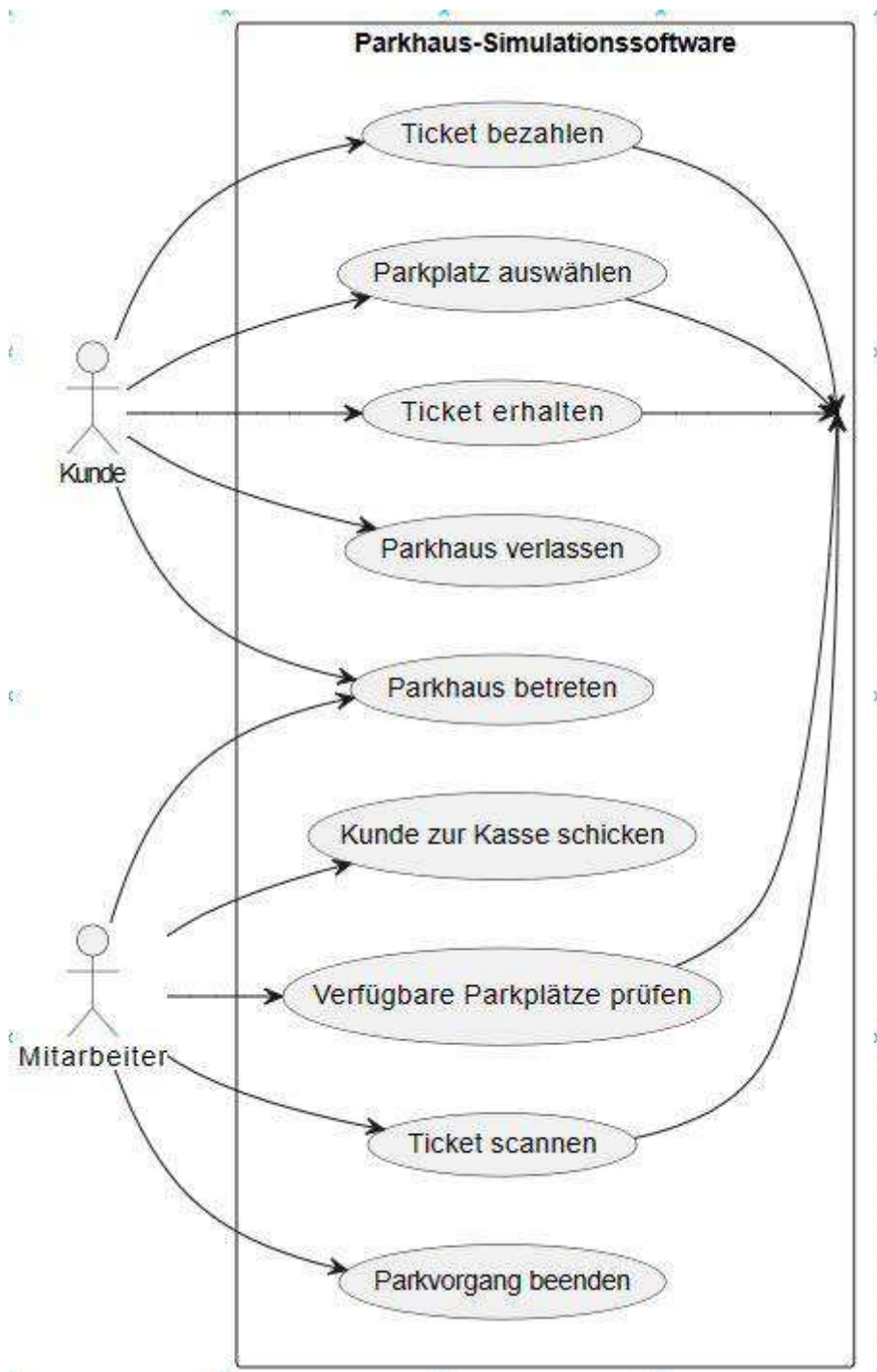


Abbildung 1: Use-Case-Diagramm

A12 Pflichtenheft (Auszug)

Pflichtenheft für die Parkhaus-Simulationssoftware

1. Einleitung

Die Parkhaus-Simulationssoftware soll dazu dienen, die Verwaltung von Parkplätzen und Parkhauskapazitäten zu erleichtern und zu optimieren. Das System soll eine benutzerfreundliche Oberfläche für die Anmeldung, Buchung von Parkplätzen und den Abruf von Informationen bereitstellen.

2. Zielgruppe

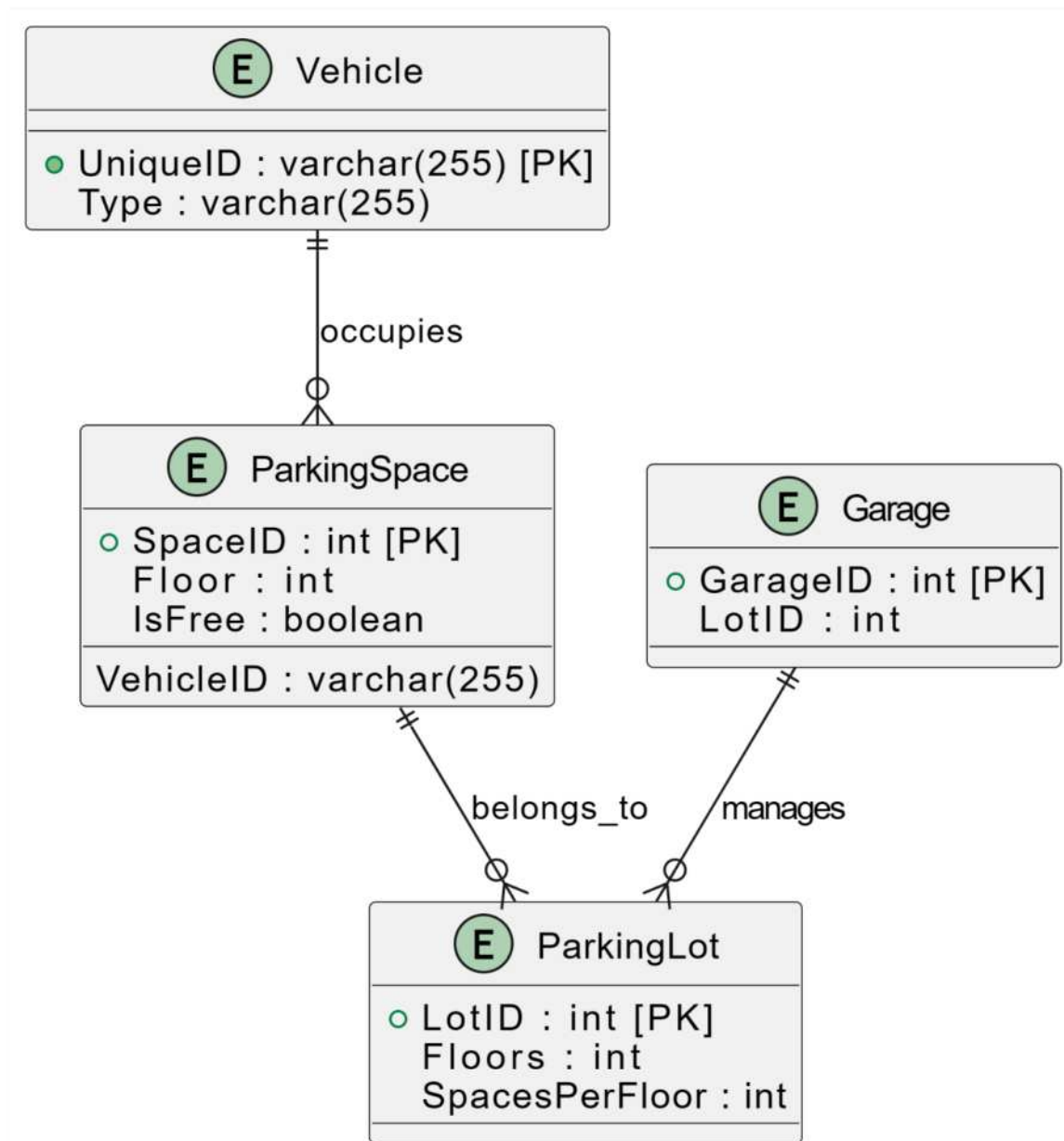
Die Zielgruppe für die Parkhaus-Simulationssoftware sind Unternehmen und Organisationen, die ein Parkhaus betreiben oder verwalten.

3. Anforderungen

Die folgenden Anforderungen wurden an die Parkhaus-Simulationssoftware gestellt:

3.1. Funktionale Anforderungen

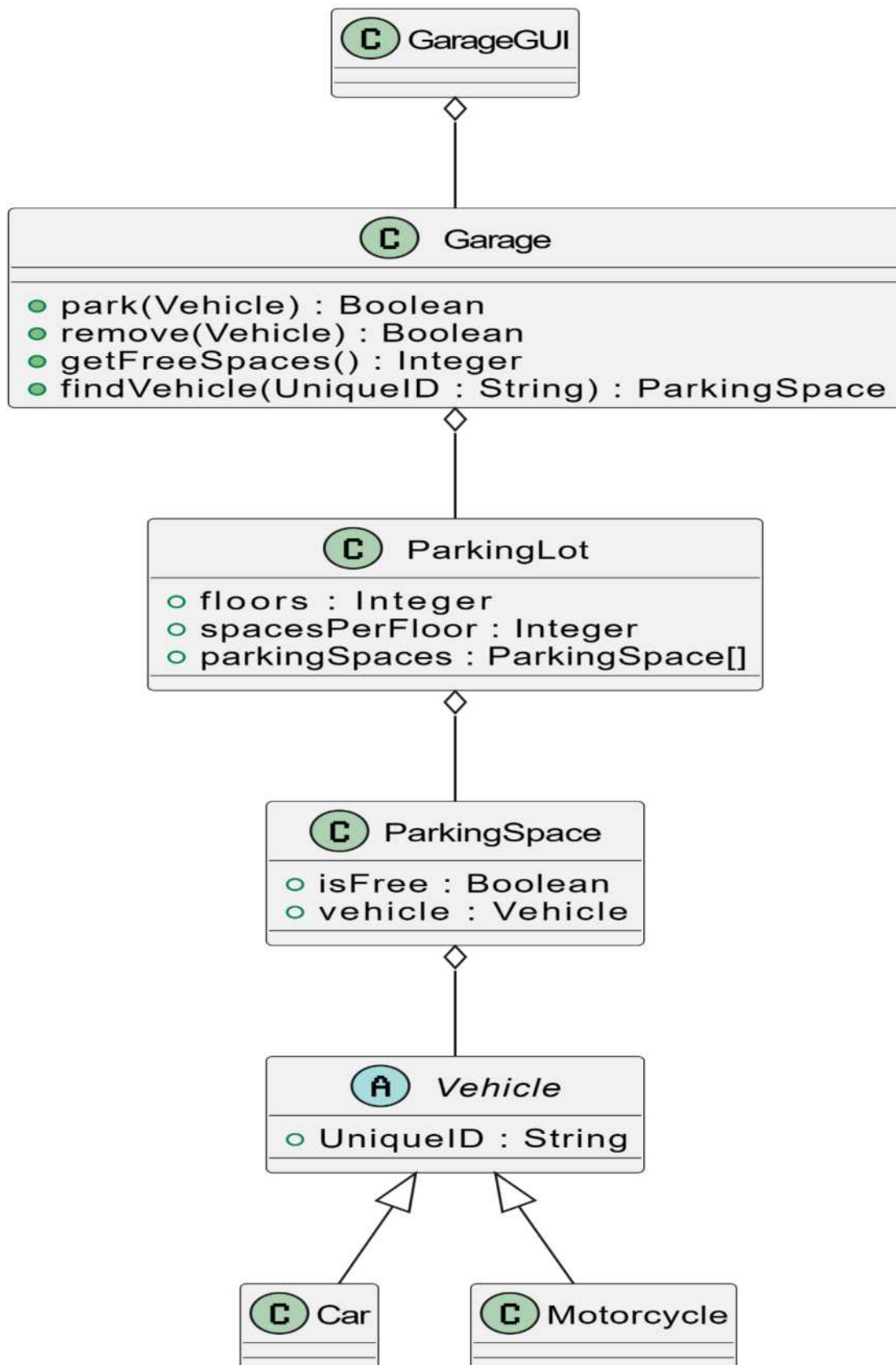
1. Das System soll eine Anmeldung für Benutzer bereitstellen.
2. Das System soll eine Buchung von Parkplätzen ermöglichen.
3. Das System soll den Abruf von Informationen über verfügbare Parkplätze bereitstellen.
4. Das System soll die Verwaltung von Parkhauskapazitäten erleichtern.
5. Das System soll die Abwicklung von Zahlungen für Parkplätze ermöglichen.
6. Das System soll eine Verwaltung von Benutzerkonten und Passwörtern bereitstellen.

A13 Datenbankmodel¹¹

Die Datenbank besteht aus vier Tabellen: "Vehicle", "ParkingSpace", "ParkingLot" und "Garage"

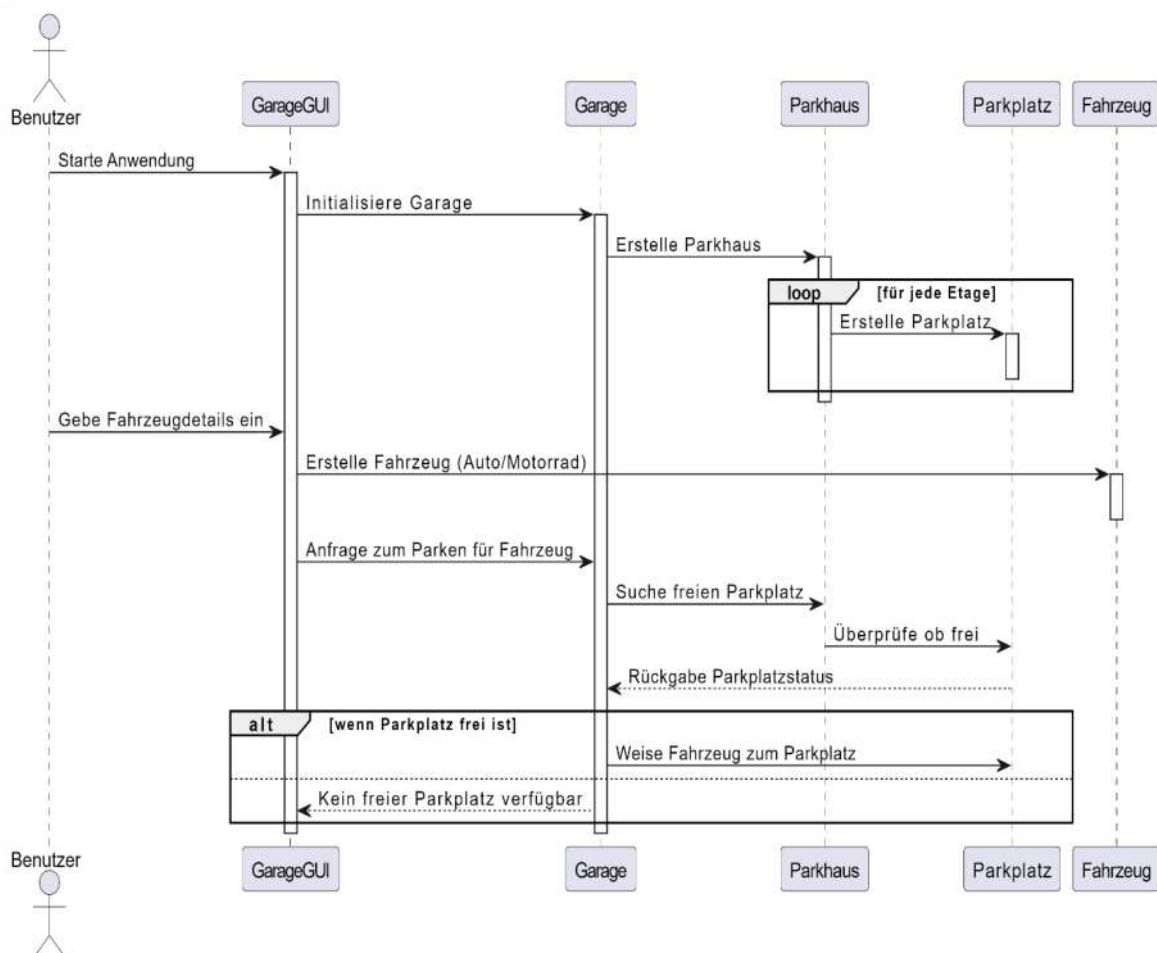
¹¹ Datenbankmodel erstellt mit draw.io

A14 Entity-Relationship-Model



A15 Sequenzdiagramm für die Parkhaus Simulationssoftware

12



In diesem Diagramm stellt die Pfeilrichtung den Nachrichtenfluss dar, und die Boxen, die sich von den Teilnehmerlinien erstrecken, repräsentieren die Zeit, in der ein Teilnehmer aktiv ist.

Bitte beachten Sie, dass dieses Diagramm ein vereinfachter Ablauf ist und möglicherweise nicht alle Funktionen oder Details abdeckt.

Die "User"-Tabelle enthält Informationen über die registrierten Benutzer, wie ihren Benutzernamen, ihre E-Mail-Adresse und ihr Passwort.

Die "Vehicle"-Tabelle enthält Informationen über die Fahrzeuge, die im Parkhaus geparkt sind, wie die Fahrzeugkennung und die Größe des Fahrzeugs.

Die "ParkingSpot"-Tabelle enthält Informationen über die Parkplätze im Parkhaus, wie ihre eindeutige Nummer und ihre Größe.

¹² Datenbankmodel erstellt mit draw.io

Die "Reservation"-Tabelle enthält Informationen über die Parkplatzreservierungen, die von den Benutzern vorgenommen wurden, wie das Datum und die Uhrzeit der Reservierung und die ID des Benutzers und des Parkplatzes.

¹² Datenbankmodel erstellt mit draw.io

Die Tabellen sind über Fremdschlüssel miteinander verknüpft, um eine Beziehung zwischen den Daten herzustellen. So ist beispielsweise in der "Vehicle"-Tabelle ein Fremdschlüssel auf die "ParkingSpot"-Tabelle verweist, um den Parkplatz zuzuweisen, auf dem das Fahrzeug geparkt ist.

Die Kardinalitäten zwischen den Tabellen in dem Datenbankmodell der Parkhaus Simulationssoftware könnten wie folgt aussehen:

Eine Instanz der Tabelle "User" kann mehrere Instanzen der Tabelle "Vehicle" haben, aber jedes Fahrzeug ist nur einem Benutzer zugeordnet (1:n-Beziehung).

Eine Instanz der Tabelle "Vehicle" kann nur einem Parkplatz in der Tabelle "ParkingSpot"

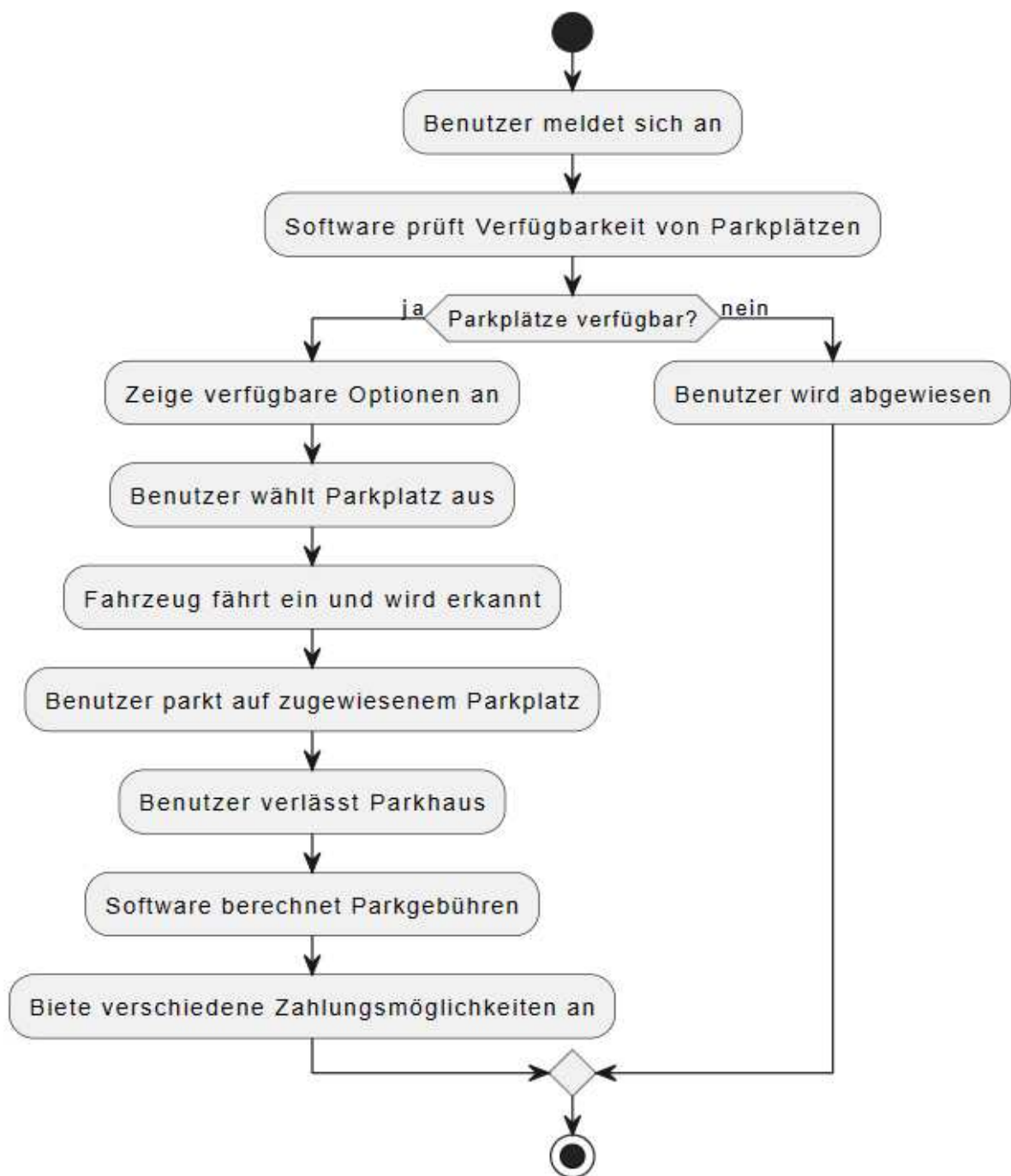
zugeordnet werden, aber jeder Parkplatz kann mehrere Fahrzeuge aufnehmen

(1:n-Beziehung).

Eine Instanz der Tabelle "User" kann mehrere Instanzen der Tabelle "Reservation" haben, aber jede Reservierung ist nur einem Benutzer zugeordnet (1:n-Beziehung).

Eine Instanz der Tabelle "Reservation" ist einem bestimmten Parkplatz in der Tabelle

"ParkingSpot" zugeordnet, aber jeder Parkplatz kann mehrere Reservierungen haben (1:n-Beziehung).



13

Abbildung 2: Programmablaufplan (PAP)

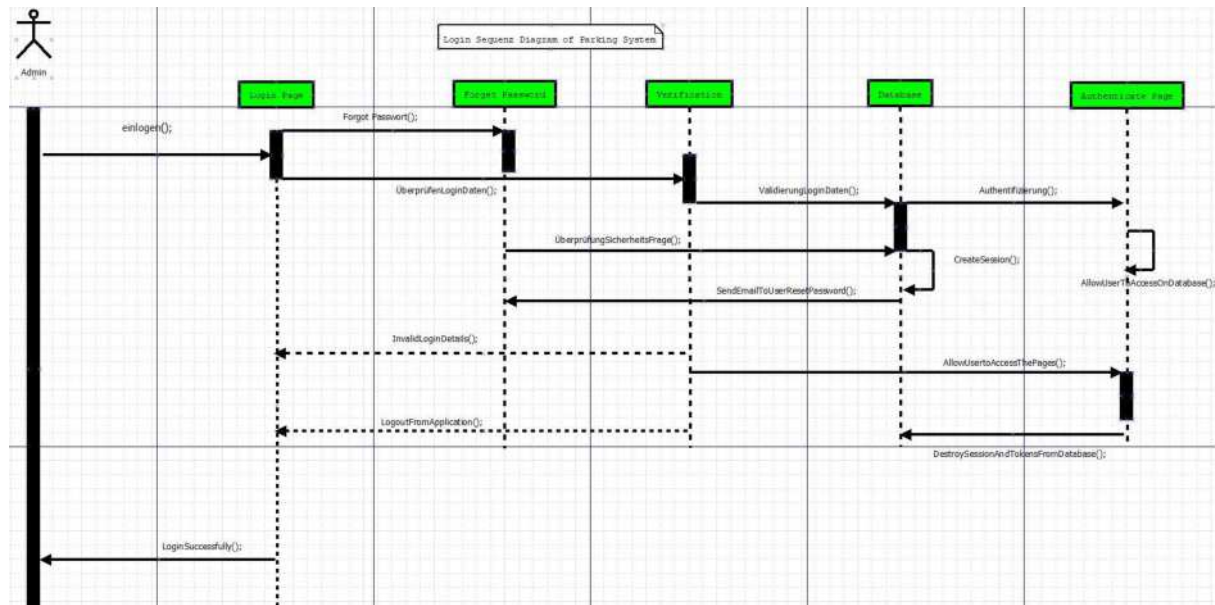


Abbildung 3: Login Sequenzdiagramm¹⁴

A16 Ereignisgesteuerte Prozesskette

Eine ereignisgesteuerte Prozesskette (EPK) ist eine Darstellungsmethode zur Analyse und Optimierung von Geschäftsprozessen. Hierbei werden die verschiedenen Prozessschritte als Ereignisse und Funktionen dargestellt und durch Pfeile verbunden, um den Ablauf des Prozesses zu visualisieren. Die EPK kann helfen, Schwachstellen und Engpässe im Prozess zu identifizieren und Maßnahmen zur Verbesserung zu planen.

Für die Parkhaus-Simulationssoftware soll ein EPK folgendes enthalten:

1. Start: Das Diagramm beginnt mit dem Start der Software.
2. Anmeldung: Der Benutzer meldet sich an und gibt seine Anmeldeinformationen ein.
3. Verfügbarkeit prüfen: Die Software prüft die Verfügbarkeit von Parkplätzen und zeigt dem Benutzer die verfügbaren Optionen an.
4. Parkplatzauswahl: Der Benutzer wählt einen verfügbaren Parkplatz aus.
5. Einfahrt: Das Fahrzeug fährt in das Parkhaus ein und wird von der Software erkannt.
6. Parken: Der Benutzer parkt das Fahrzeug auf dem zugewiesenen Parkplatz.
7. Ausfahrt: Der Benutzer verlässt das Parkhaus.
8. Zahlungsabwicklung: Die Software berechnet die Parkgebühren und bietet dem Benutzer verschiedene Zahlungsmöglichkeiten an.
9. Ende: Das Diagramm endet mit dem Ende der Software.

Durch die Darstellung des Prozesses in einer EPK können mögliche Probleme und Engpässe identifiziert werden, wie z.B. lange Wartezeiten beim Ein- oder Ausfahren oder Probleme bei der Zahlungsabwicklung. Dadurch können entsprechende Verbesserungsmaßnahmen geplant und umgesetzt werden.

¹⁴ Erstellt mit dem Programm Dia

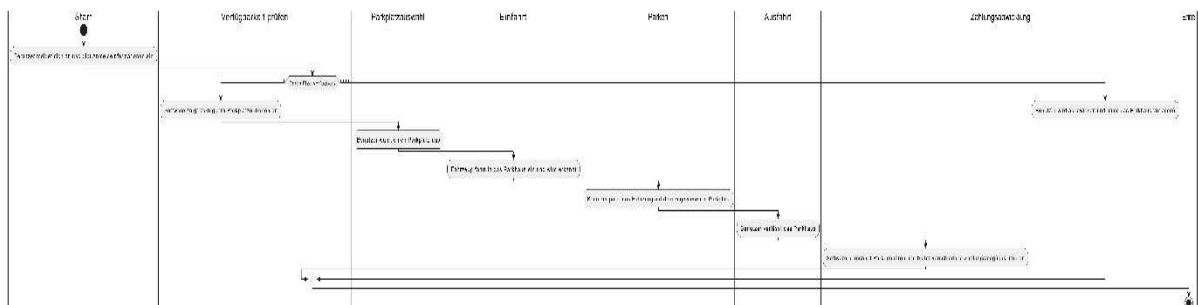
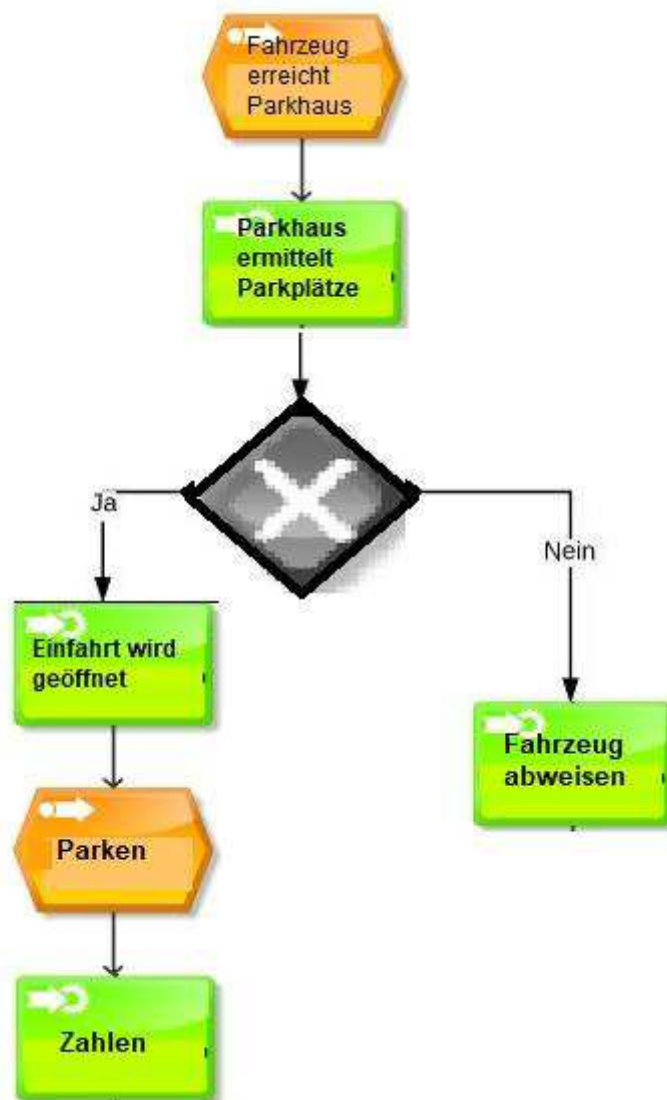
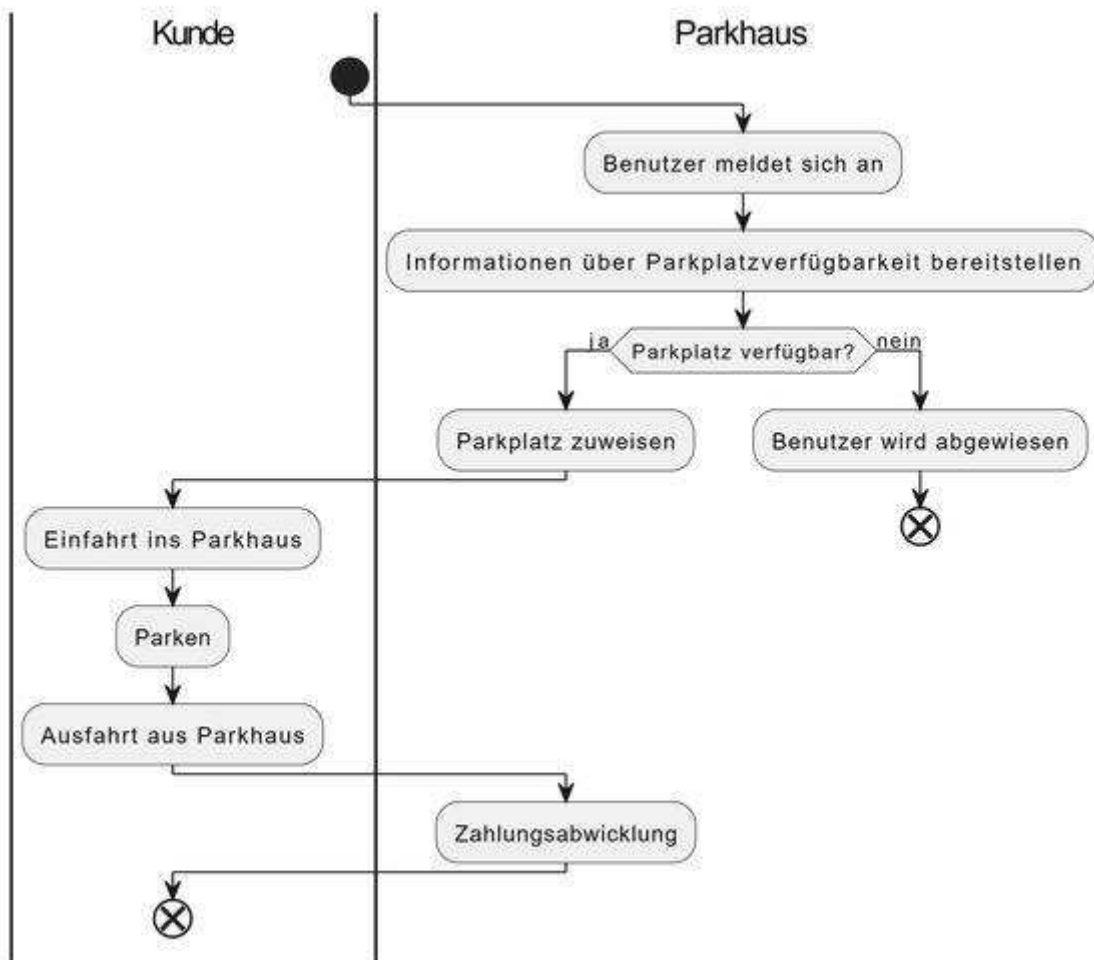


Abbildung 4: Prozess des Einparkens

A17 Aktivitätsdiagramm



Hier sind Schritte, die im Aktivitätsdiagramm für die Parkhaus-Simulationssoftware dargestellt werden

Ein Aktivitätsdiagramm für die Parkhaus-Simulationssoftware kann den Ablauf der verschiedenen Aktivitäten innerhalb der Software darstellen:

1. Start: Der Prozess beginnt mit einem Startknoten, der den Beginn des Fahrzeugparks darstellt.
2. Anmeldung: Der nächste Schritt ist die Anmeldung des Benutzers. Hierbei erfolgen die Benutzerauthentifizierung und die Ermittlung des Parkplatzes.
3. Informationsbereitstellung: Hierbei wird der Benutzer über die Verfügbarkeit von Parkplätzen informiert.
4. Wenn kein Parkplatz verfügbar ist, wird der Benutzer abgewiesen und das Programm beendet.
5. Wenn jedoch ein Parkplatz verfügbar ist, startet der Prozess der Parkplatzzuweisung.
6. Dieser Prozess wählt einen geeigneten Parkplatz für das Fahrzeug aus und informiert den Benutzer entsprechend.
7. Einfahrt: Nachdem der Benutzer seinen Parkplatz zugewiesen bekommen hat, erfolgt der Prozess der Einfahrt in das Parkhaus.
8. Die Software erkennt das Fahrzeug und öffnet die Schranke.
9. Parken: Sobald das Fahrzeug im Parkhaus ist, wird der Prozess des Parkens gestartet. Hierbei gibt die Software dem Benutzer Anweisungen, wie er sein Fahrzeug am besten parken kann.
10. Zahlungsabwicklung: Wenn der Benutzer fertig ist und das Parkhaus verlässt, wird der Prozess der Zahlungsabwicklung gestartet. Hierbei teilt die Software dem Benutzer mit, wie viel er bezahlen muss, und bietet ihm verschiedene Zahlungsmöglichkeiten an.
11. Beenden: Schließlich endet der Prozess mit einem Endknoten, der das Ende der Software darstellt.

15

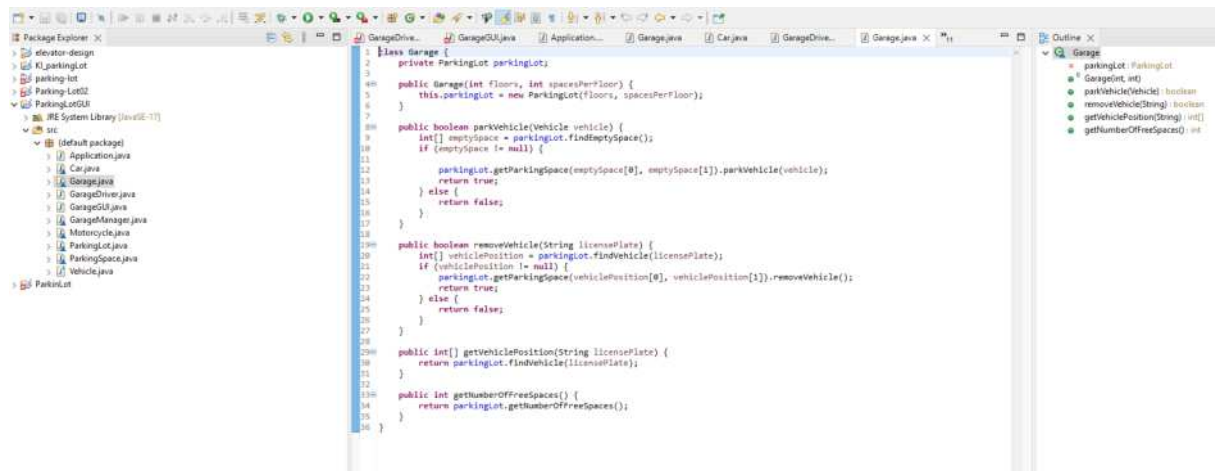


Abbildung 5: Liste der einzelnen Klassen in der IDE Eclipse

15

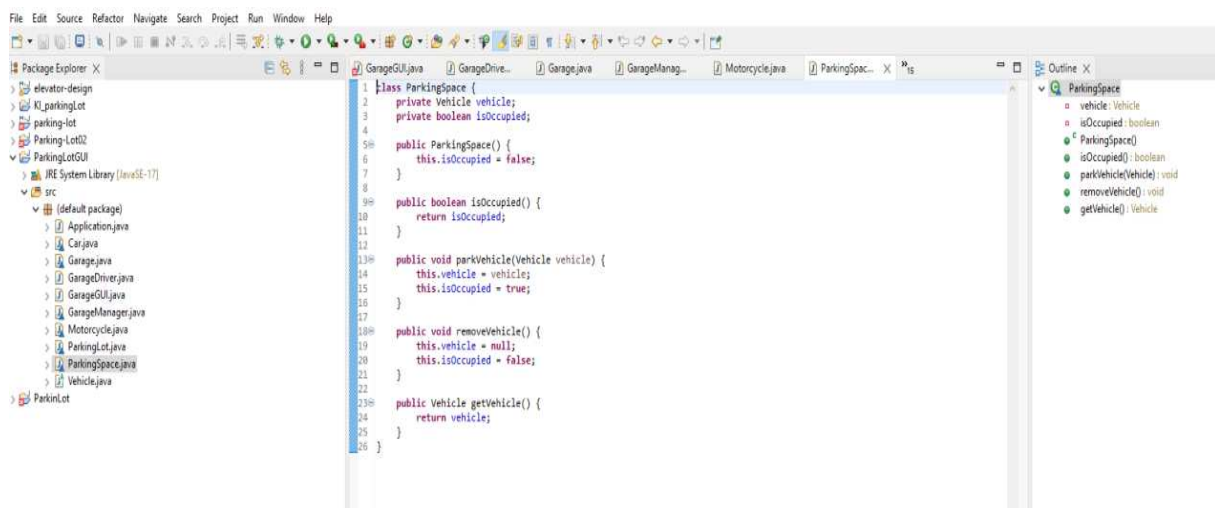


Abbildung 6: ParkingSpace.java

15 Java DIE Eclipse

A18 Screenshots der Anwendung



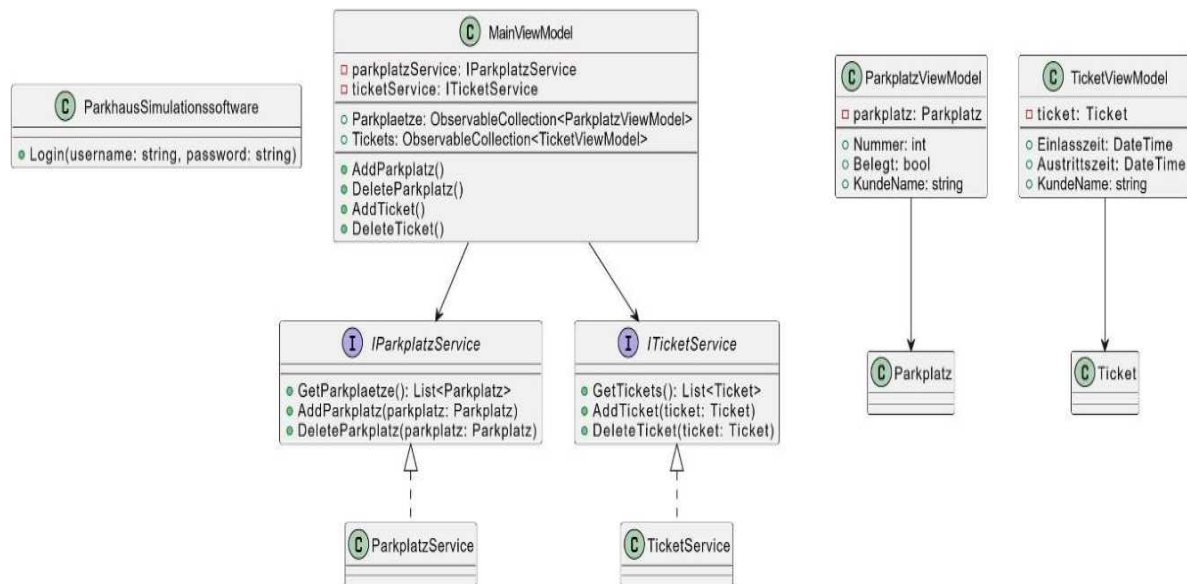
Abbildung 7: Funktion Ausparken ausführen ¹⁶



Abbildung 8: Mitarbeiter/Admin Login¹⁷

¹⁶Benutzeroberfläche erstellt mit JavaFX, (<https://de.wikipedia.org/wiki/JavaFX>) ¹⁷Mit XAMPP und Note++ erstellt

A19 Das Domain-Modell der Parkhaus-Simulationssoftware



Das Domain-Modell beschreibt die grundlegenden Objekte und ihre Beziehungen zueinander, die in der Parkhaus-Simulationssoftware verwendet werden. Das Modell zeigt, dass es Kunden gibt, die Tickets erhalten, um einen Parkplatz zu nutzen. Jeder Parkplatz hat eine eindeutige Nummer und kann nur von einem Kunden genutzt werden. Ein Ticket ist an einen Parkplatz gebunden und kann nur von dem Kunden verwendet werden, dem der Parkplatz zugewiesen wurde.

Das Datenbankmodell ist eine Darstellung der Daten, die in der Datenbank der Parkhaus-Simulationssoftware gespeichert werden. Das Modell zeigt, welche Tabellen erstellt werden müssen, um die Informationen zu speichern, die im Domain-Modell beschrieben wurden. Die Tabellen sind so strukturiert, dass sie effizient abgefragt werden können und die Integrität der Daten gewahrt bleibt.

Die MVVM-Architektur ist eine Methode, um eine Softwareanwendung zu strukturieren. Dabei wird die Anwendung in drei Teile unterteilt: das Model, das die Daten und die Geschäftslogik enthält, die View, die die Benutzeroberfläche darstellt, und das ViewModel, das als Bindeglied zwischen dem Model und der View fungiert. Die Trennung dieser Komponenten erleichtert die Wartung und Erweiterung der Anwendung.

A20 Entwicklerdokumentation (Auszug)

Die Entwicklerdokumentation im Anhang beschreibt die technischen Aspekte der Parkhaus Simulationssoftware, wie z.B. die verwendeten Technologien, die Softwarearchitektur und die Implementierungsdetails.

Titel: Parkhaus-Simulationsprogramm

1. Einleitung

Das Parkhaus-Simulationsprogramm ist eine Java-basierte Anwendung, die entwickelt wurde, um das Parken und Entfernen von Fahrzeugen in einem mehrstöckigen Parkhaus zu simulieren. Die Anwendung verfügt über eine einfache grafische Benutzeroberfläche (GUI) ³, die den Benutzern ermöglicht, Autos und Motorräder zu parken oder zu entfernen, und zeigt den aktuellen Status der Parkplätze an.

³ Benutzeroberfläche(GUI) erstellt mit Java FX

2. Anforderungen

Die Hauptanforderungen des Parkhaus-Simulationsprogramms sind:

2.1. Unterstützung für unterschiedliche Arten von Fahrzeugen

- Autos
- Motorräder

2.2. Einzigartige ID (Kennzeichen) für jedes Fahrzeug

2.3. Flexibilität bei der Anzahl der Etagen und Parkplätze pro Etage

2.4. Zuteilung eines freien Parkplatzes oder Ablehnung, wenn keine freien Parkplätze vorhanden sind

2.5. Verwaltungsfunktionen, wie die Position eines bestimmten Fahrzeugs und die Anzahl der freien Parkplätze abzufragen

3. Systemarchitektur

Die Anwendung besteht aus mehreren Klassen, die die verschiedenen Komponenten des Systems repräsentieren:

- Vehicle: Abstrakte Basisklasse für Fahrzeuge
- Car: Klasse, die Autos repräsentiert und von der Vehicle-Klasse erbt
- Motorcycle: Klasse, die Motorräder repräsentiert und von der Vehicle-Klasse erbt
- ParkingSpace: Klasse, die einen Parkplatz repräsentiert
- ParkingLot: Klasse, die das Parkhaus repräsentiert
- Garage: Klasse, die die Hauptfunktionen für das Parken und Entfernen von Fahrzeugen bereitstellt
- GarageGUI: Klasse, die die grafische Benutzeroberfläche für das Parkhaus-Simulationsprogramm erstellt und verwaltet

4. Benutzerhandbuch

Um das Parkhaus-Simulationsprogramm zu verwenden, starten Sie die Anwendung, indem Sie die oben beschriebenen Schritte zum Kompilieren und Ausführen des Programms befolgen.

Die grafische Benutzeroberfläche besteht aus einem Textfeld, in das Sie das Kennzeichen des Fahrzeugs eingeben, und drei Schaltflächen zum Parken eines Autos, Parken eines Motorrads und Entfernen eines Fahrzeugs. Es gibt auch ein Textfeld, das den aktuellen Status der Parkplätze anzeigt.

Um ein Fahrzeug zu parken oder zu entfernen, führen Sie die folgenden Schritte aus:

1. Geben Sie das Kennzeichen des Fahrzeugs in das Textfeld ein.
2. Klicken Sie auf die entsprechende Schaltfläche, um ein Auto zu parken, ein Motorrad zu parken oder ein Fahrzeug zu entfernen.
3. Überprüfen Sie das Textfeld, um den aktuellen Status der Parkplätze und etwaige Meldungen über den Erfolg oder Misserfolg der Aktion zu sehen.

5. Fazit

Das Parkhaus-Simulationsprogramm bietet eine einfache und intuitive Möglichkeit, das Parken und Entfernen von Fahrzeugen in einem mehrstöckigen Parkhaus zu simulieren. Die Anwendung erfüllt die gestellten Anforderungen und ermöglicht den Benutzern, die Hauptfunktionen auf einfache Weise zu verwenden. Durch die modulare Architektur des Programms können in Zukunft weitere Verbesserungen und Funktionen hinzugefügt werden, um den Funktionsumfang zu erweitern oder auf spezifische Anforderungen zuzuschneiden.

6. Zukünftige Erweiterungen

Einige mögliche Erweiterungen für das Parkhaus-Simulationsprogramm sind:

6.1. Erweiterte Fahrzeugtypen

Die Unterstützung für zusätzliche Fahrzeugtypen wie Elektrofahrzeuge, Nutzfahrzeuge oder Fahrräder könnte hinzugefügt werden.

6.2. Reservierungssystem

Ein Reservierungssystem könnte implementiert werden, um Benutzern zu ermöglichen, Parkplätze im Voraus zu reservieren.

6.3. Preismodell

Ein Preismodell könnte hinzugefügt werden, um die Parkgebühren auf der Grundlage der Parkdauer, des Fahrzeugtyps oder der Tageszeit und des Wochentags zu berechnen.

6.4. Statistiken und Berichte

Erweiterte Statistiken und Berichtsfunktionen könnten hinzugefügt werden, um detaillierte Informationen über die Nutzung des Parkhauses, Einnahmen und Auslastung zu liefern.

6.5. Integration mit externen Systemen

Das Parkhaus-Simulationsprogramm könnte mit externen Systemen wie Zahlungsanbietern, Navigationssystemen oder Parkhausmanagement-Software integriert werden.

7. Anhang

Im Anhang finden Sie den Quellcode für das Parkhaus-Simulationsprogramm und die zugehörige grafische Benutzeroberfläche. Stellen Sie sicher, dass alle Klassen in separaten Java-Dateien gespeichert sind und im selben Verzeichnis gespeichert sind, bevor Sie versuchen, das Programm zu kompilieren und auszuführen.

8. Installation und Systemvoraussetzungen

Um das Parkhaus-Simulationsprogramm auszuführen, stellen Sie sicher, dass Ihr System die folgenden Voraussetzungen erfüllt:

8.1. Java Development Kit (JDK)

Stellen Sie sicher, dass Sie das Java Development Kit (JDK) auf Ihrem System installiert haben. Das JDK enthält die notwendigen Tools, um Java-Anwendungen zu kompilieren und auszuführen. Sie können das JDK von der offiziellen Oracle-Website herunterladen: <https://www.oracle.com/java/technologies/javase-downloads.html>

8.2. Betriebssystem

Das Parkhaus-Simulationsprogramm ist plattformunabhängig und sollte auf jedem Betriebssystem ausgeführt werden können, das Java unterstützt, einschließlich Windows, macOS und Linux.

9. Support und Wartung

Für Fragen, Fehlerberichte oder Anregungen zur Verbesserung des Parkhaus-Simulationsprogramms wenden Sie sich bitte an den Entwickler oder das zuständige Support-Team. Regelmäßige Updates und Wartungsarbeiten werden durchgeführt, um sicherzustellen, dass das Programm fehlerfrei und auf dem neuesten Stand bleibt.

10. Lizenz und Urheberrecht

Das Parkhaus-Simulationsprogramm ist urheberrechtlich geschützt. Die Nutzung, Vervielfältigung oder Weitergabe des Programms oder eines Teils davon ist nur mit ausdrücklicher schriftlicher Genehmigung des Urheberrechtsinhabers gestattet. Alle Rechte vorbehalten.

11. Haftungsausschluss

Das Parkhaus-Simulationsprogramm wird "wie besehen" und ohne Gewährleistung jeglicher Art, weder ausdrücklich noch stillschweigend, zur Verfügung gestellt. Der Entwickler und der Urheberrechtsinhaber

DIE PARKHAUS SIMULATIONSSOFTWARE

Implementierung eines neuen Parkhauses mithilfe einer Simulationssoftware

übernehmen keine Haftung für Schäden, die durch die Verwendung des Programms entstehen könnten. Die Verwendung des Programms erfolgt auf eigenes Risiko des Benutzers.

12. Schlussbemerkungen

Das Parkhaus-Simulationsprogramm wurde entwickelt, um den Betrieb eines Parkhauses in einer einfachen und benutzerfreundlichen Umgebung zu simulieren. Es ermöglicht Benutzern, Fahrzeuge zu parken, freie Parkplätze zu finden und Fahrzeugpositionen abzufragen. Die modulare Architektur des Programms ermöglicht es, weitere Funktionen und Verbesserungen hinzuzufügen, um den Anforderungen verschiedener Parkhausbetreiber gerecht zu werden.

Wir hoffen, dass dieses Programm dazu beiträgt, das Parkhausmanagement effizienter und benutzerfreundlicher zu gestalten und wertvolle Informationen zur Verfügung zu stellen, die bei der Planung und Optimierung von Parkhäusern helfen können.

Bei Fragen oder Anregungen zögern Sie bitte nicht, uns zu kontaktieren. Wir freuen uns darauf, Ihre Erfahrungen mit dem Parkhaus-Simulationsprogramm zu hören und Möglichkeiten zur Verbesserung des Programms zu erforschen.

Vielen Dank für Ihr Interesse an unserem Parkhaus-Simulationsprogramm.

Mit freundlichen Grüßen,

Daniel Massa



[Software Entwickler]

[donmassa@proton.me]

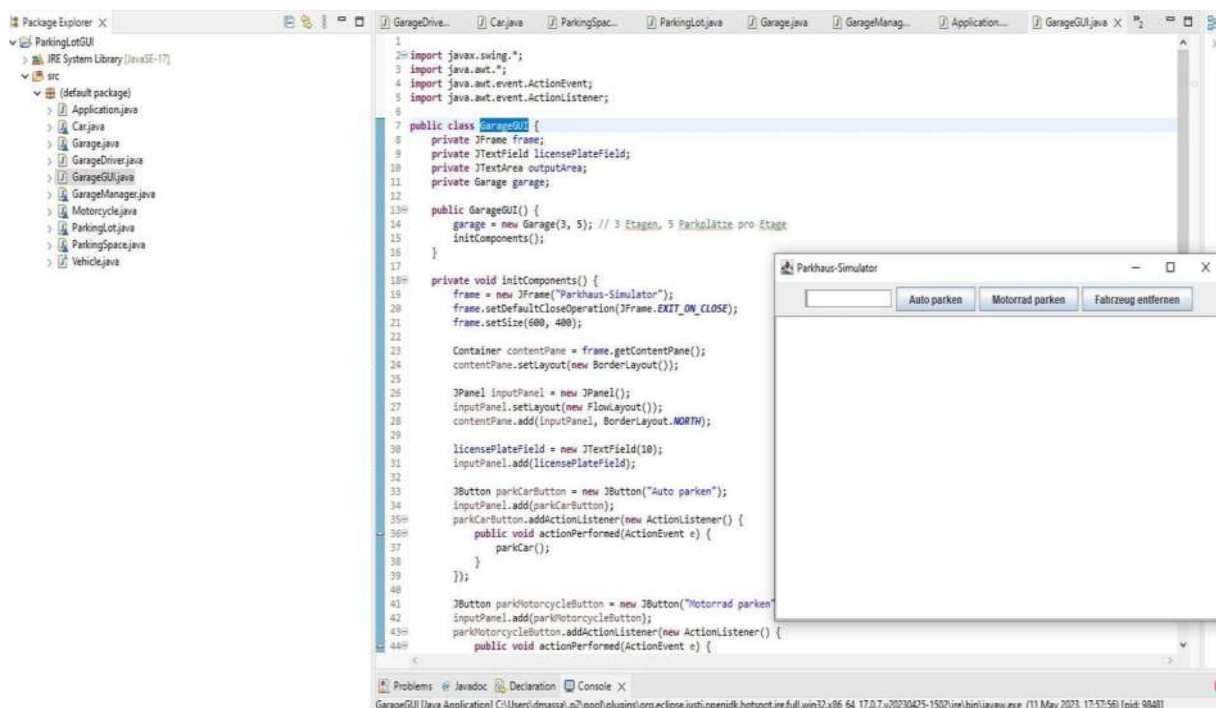
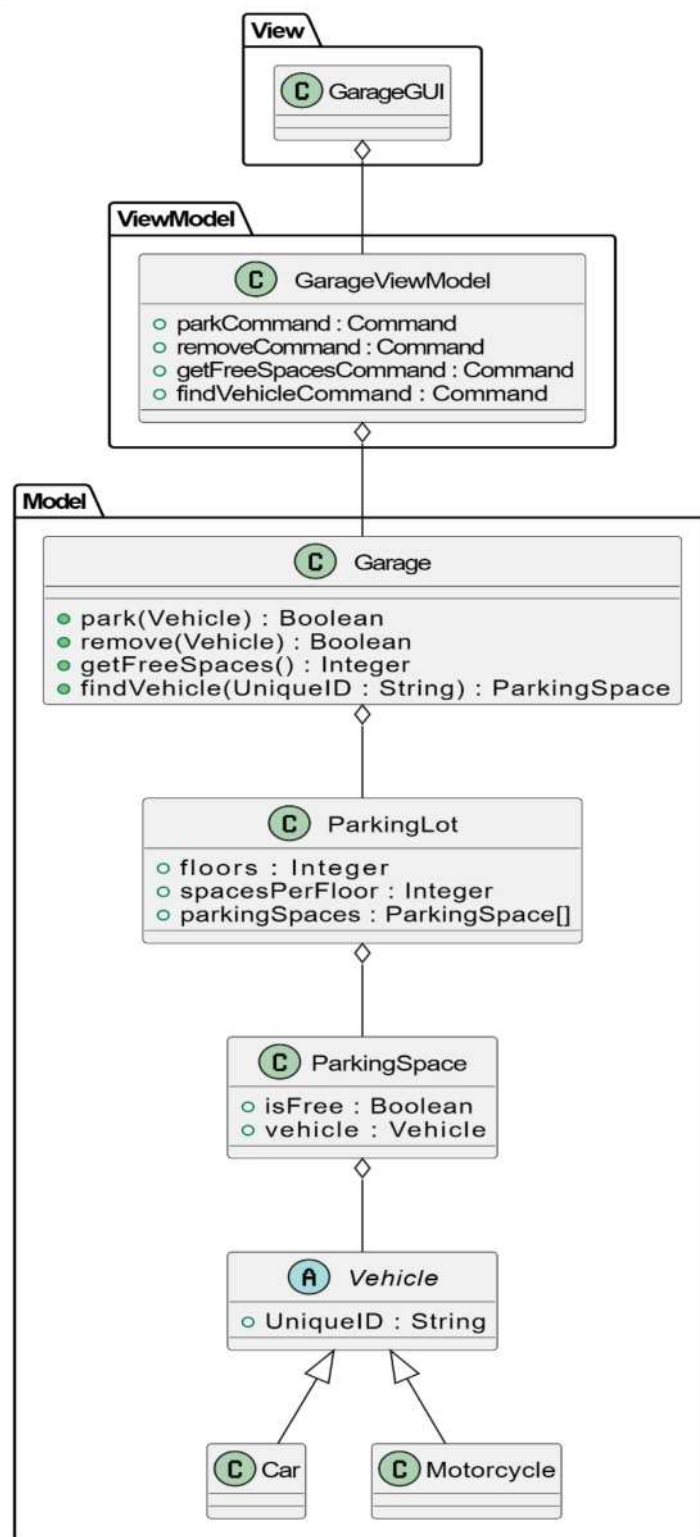
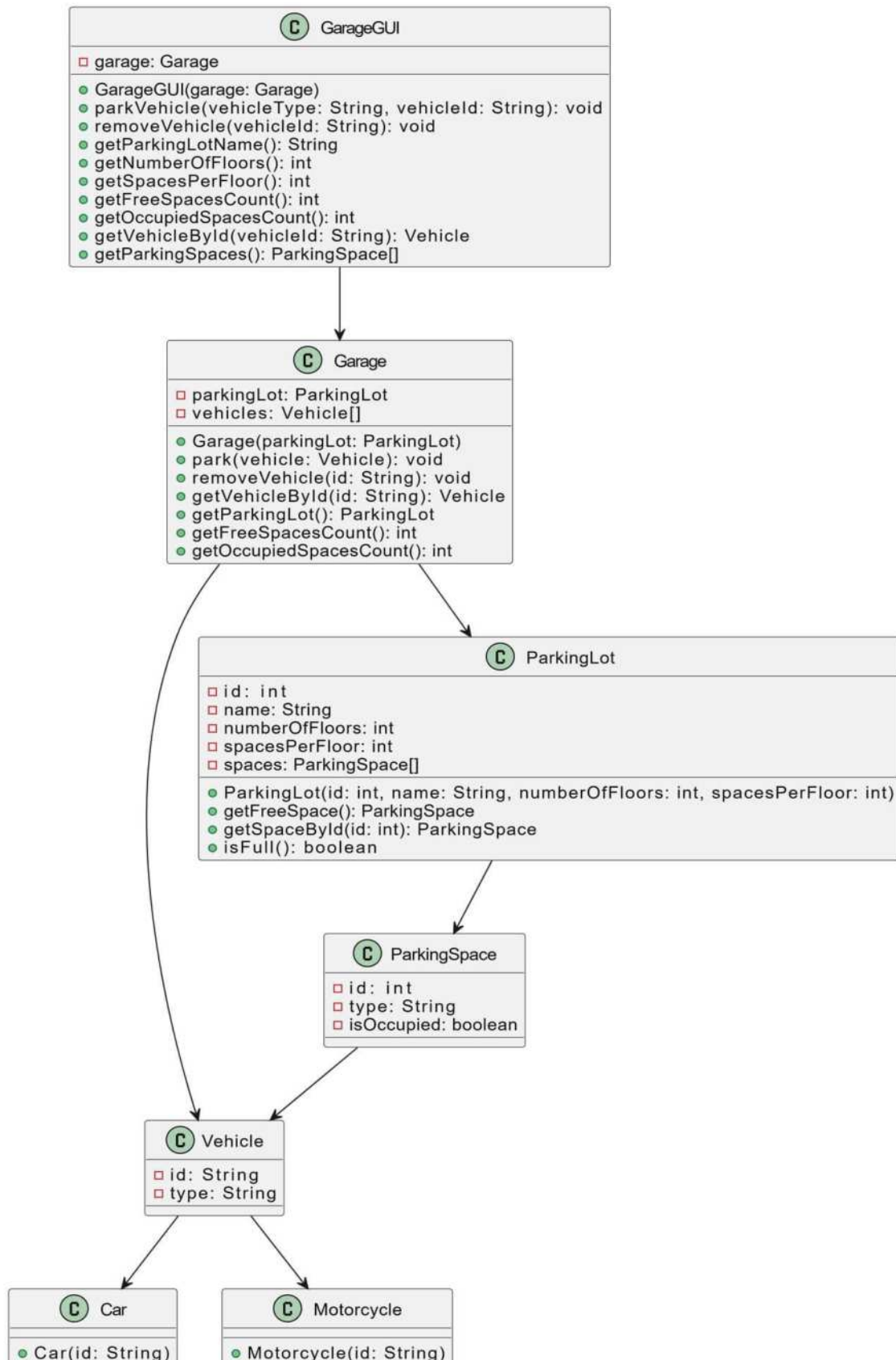


Abbildung 9: Parkhaus-Simulator

A21 Klassenmodell von der Parkhaus Simulationssoftware strukturiert im MVVC-Model.



A22 Entity-Relationship-Model



A23 Meilensteinplan

Ein Meilensteinplan für die Parkhaus Simulationssoftware:

1. Anforderungsanalyse - 1 Woche
2. Erstellung des Pflichtenhefts
3. Definition von Use Cases und User Stories
4. Systemdesign - 2 Wochen
5. Erstellung des Klassendiagramms
6. Entwurf des Datenbankmodells
7. Festlegung der Architektur und Technologien
8. Implementierung - 6 Wochen
9. Umsetzung der Software nach den Vorgaben des Pflichtenhefts
10. Programmierung der Datenbankanbindung
11. Implementierung der Benutzeroberfläche
12. Tests - 2 Wochen
13. Durchführung von Unit-Tests und Integrationstests
14. Behebung von Fehlern und Bugs
15. Dokumentation - 1 Woche
16. Erstellung der Entwickler- und Benutzerdokumentation
17. Zusammenstellung des Projektreports
18. Rollout - 1 Woche
19. Deployment der Software auf dem Zielsystem
20. Schulung der Endbenutzer
21. Wartung - 4 Wochen
22. Überwachung und Pflege der Software
23. Behebung von auftretenden Fehlern und Problemen
24. Die genauen Zeitangaben können je nach Projektumfang und Teamgröße variieren und sollten immer in enger Absprache mit allen Projektbeteiligten festgelegt werden.

A24 Klasse: ParkhausSimulationssoftwareTest

Im Anhang ist der Codeausschnitt für die Klasse Vehicle enthalten, die ein grundlegendes Element der Parkhaus Simulationssoftware darstellt und die Eigenschaften und Methoden für Fahrzeuge im Parkhaus bereitstellt.

```
import static org.junit.Assert.*;
import org.junit.Test;

public class ParkhausSimulationssoftwareTest {

    @Test
    public void testKonfiguration() {
        ParkhausSimulationssoftware parkhaus = new
        ParkhausSimulationssoftware();

        parkhaus.setAnzahlParkplaetze(100);
        parkhaus.setParkdauer(60);
        parkhaus.setGebuehr(1.5);

        assertEquals(100, parkhaus.getAnzahlParkplaetze());
    }
}
```



```
        assertEquals(60, parkhaus.getParkdauer());
        assertEquals(1.5, parkhaus.getGebuehr(), 0.01);
    }

    @Test
    public void testSimulation() {
        ParkhausSimulationssoftware parkhaus = new
        ParkhausSimulationssoftware();
        parkhaus.setAnzahlParkplaetze(100);
        parkhaus.setParkdauer(60);
        parkhaus.setGebuehr(1.5);

        parkhaus.startSimulation();
        assertTrue(parkhaus.isSimulationGestartet());

        parkhaus.stopSimulation();
        assertFalse(parkhaus.isSimulationGestartet());
    }
}
```

Listing 1: Testklasse

A25 Klasse "ParkingSpace"

```
class ParkingSpace {
    private Vehicle vehicle;
    private boolean isOccupied;

    public ParkingSpace() {
        this.isOccupied = false;
    }

    public boolean isOccupied() {
        return isOccupied;
    }
}
```

```
public void parkVehicle(Vehicle vehicle) {
    this.vehicle = vehicle;
    this.isOccupied = true;
}

public void removeVehicle() {
    this.vehicle = null;
    this.isOccupied = false;
}

public Vehicle getVehicle() {
    return vehicle;
}
}
```

Listing 2: Klasse "ParkhausSimulationssoftwareTest"

```
class ParkingLot {
    private int floors;
    private int spacesPerFloor;
    private ParkingSpace[][] parkingSpaces;

    public ParkingLot(int floors, int spacesPerFloor) {
        this.floors = floors;
        this.spacesPerFloor = spacesPerFloor;
        this.parkingSpaces = new
ParkingSpace[floors][spacesPerFloor];

        for (int i = 0; i < floors; i++) {
            for (int j = 0; j < spacesPerFloor; j++) {
                parkingSpaces[i][j] = new ParkingSpace();
            }
        }
    }

    public int getFloors() {
        return floors;
    }
}
```

```
public int getSpacesPerFloor() {
    return spacesPerFloor;
}

public ParkingSpace getParkingSpace(int floor, int space) {
    return parkingSpaces[floor][space];
}

public int[] findEmptySpace() {
    for (int i = 0; i < floors; i++) {
        for (int j = 0; j < spacesPerFloor; j++) {
            if (!parkingSpaces[i][j].isOccupied()) {
                return new int[]{i, j};
            }
        }
    }
    return null;
}

public int getNumberOfFreeSpaces() {
    int count = 0;
    for (int i = 0; i < floors; i++) {
        for (int j = 0; j < spacesPerFloor; j++) {
            if (!parkingSpaces[i][j].isOccupied()) {
                count++;
            }
        }
    }
    return count;
}

public int[] findVehicle(String licensePlate) {
    for (int i = 0; i < floors; i++) {
        for (int j = 0; j < spacesPerFloor; j++) {
            if (parkingSpaces[i][j].isOccupied() &&
                parkingSpaces[i][j].getVehicle().getLicensePlate().equals(licensePlate)) {
```

```
        return new int[]{i, j};
    }
}
return null;
}
```

Listing 3: Klasse "ParkingSpot"

A26 Klasse "GarageGUI"

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class GarageGUI {
    private JFrame frame;
    private JTextField licensePlateField;
    private JTextArea outputArea;
    private Garage garage;

    public GarageGUI() {
        garage = new Garage(3, 5); // 3 Etagen, 5 Parkplätze pro
        Etage
        initComponents();
    }

    private void initComponents() {
        frame = new JFrame("Parkhaus-Simulator");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(600, 400);

        Container contentPane = frame.getContentPane();
        contentPane.setLayout(new BorderLayout());

        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new FlowLayout());
    }
}
```

```
        contentPane.add(inputPanel, BorderLayout.NORTH);

        licensePlateField = new JTextField(10);
        inputPanel.add(licensePlateField);

        JButton parkCarButton = new JButton("Auto parken");
        inputPanel.add(parkCarButton);
        parkCarButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                parkCar();
            }
        });

        JButton parkMotorcycleButton = new JButton("Motorrad
parken");
        inputPanel.add(parkMotorcycleButton);
        parkMotorcycleButton.addActionListener(new ActionListener()
{
            public void actionPerformed(ActionEvent e) {
                parkMotorcycle();
            }
        });

        JButton removeVehicleButton = new JButton("Fahrzeug
entfernen");
        inputPanel.add(removeVehicleButton);
        removeVehicleButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                removeVehicle();
            }
        });

        outputArea = new JTextArea();
        JScrollPane scrollPane = new JScrollPane(outputArea);
        contentPane.add(scrollPane, BorderLayout.CENTER);

        frame.setVisible(true);
    }
```

```
private void parkCar() {
    String licensePlate = licensePlateField.getText();
    Car car = new Car(licensePlate);
    boolean success = garage.parkVehicle(car);
    if (success) {
        outputArea.append("Auto (" + licensePlate + ")
erfolgreich geparkt.\n");
    } else {
        outputArea.append("Auto (" + licensePlate + ") konnte
nicht geparkt werden. Kein freier Parkplatz.\n");
    }
}

private void parkMotorcycle() {
    String licensePlate = licensePlateField.getText();
    Motorcycle motorcycle = new Motorcycle(licensePlate);
    boolean success = garage.parkVehicle(motorcycle);
    if (success) {
        outputArea.append("Motorrad (" + licensePlate + ")
erfolgreich geparkt.\n");
    } else {
        outputArea.append("Motorrad (" + licensePlate + ")
konnte nicht geparkt werden. Kein freier Parkplatz.\n");
    }
}

private void removeVehicle() {
    String licensePlate = licensePlateField.getText();
    boolean success = garage.removeVehicle(licensePlate);
    if (success) {
        outputArea.append("Fahrzeug (" + licensePlate + ")
erfolgreich entfernt.\n");
    } else {
        outputArea.append("Fahrzeug (" + licensePlate + ")
konnte nicht gefunden und entfernt werden.\n");
    }
}
```

```
public static void main(String[] args) {  
    SwingUtilities.invokeLater(new Runnable() {  
        public void run() {  
            new GarageGUI();  
        }  
    });  
}
```

Java(FX) Code der Benutzeroberfläche

A27 Klasse Garage

```
class Garage {  
    private ParkingLot parkingLot;  
  
    public Garage(int floors, int spacesPerFloor) {  
        this.parkingLot = new ParkingLot(floors, spacesPerFloor);  
    }  
  
    public boolean parkVehicle(Vehicle vehicle) {  
        int[] emptySpace = parkingLot.findEmptySpace();  
        if (emptySpace != null) {  
            parkingLot.getParkingSpace(emptySpace[0],  
emptySpace[1]).parkVehicle(vehicle);  
            return true;  
        } else {  
            return false;  
        }  
    }  
  
    public boolean removeVehicle(String licensePlate) {  
        int[] vehiclePosition =  
parkingLot.findVehicle(licensePlate);  
        if (vehiclePosition != null) {  
            parkingLot.getParkingSpace(vehiclePosition[0],  
vehiclePosition[1]).removeVehicle();  
            return true;  
        }  
    }  
}
```

```
        } else {  
            return false;  
        }  
    }  
  
    public int[] getVehiclePosition(String licensePlate) {  
        return parkingLot.findVehicle(licensePlate);  
    }  
  
    public int getNumberOfFreeSpaces() {  
        return parkingLot.getNumberOfFreeSpaces();  
    }  
}
```

Listing 3: Klasse Hauptklasse

Dieser Beispielcode erstellt eine einfache Parkhaus-Simulation mit Fahrzeugen, Parkplätzen und einer Parkgarage. Sie können diese Codeausschnitte erweitern und an Ihre spezifischen Anforderungen anpassen, z. B. durch Hinzufügen weiterer Funktionen wie Reservierungen, unterschiedliche Parkgebühren oder zusätzliche Fahrzeugtypen.

A28 Klasse "Reservation"

```
import java.time.LocalDateTime;  
  
public class Reservation {  
    private Vehicle vehicle;  
    private LocalDateTime startTime;  
    private LocalDateTime endTime;  
  
    public Reservation(Vehicle vehicle, LocalDateTime startTime,  
        LocalDateTime endTime) {  
        this.vehicle = vehicle;  
        this.startTime = startTime;  
        this.endTime = endTime;  
    }  
  
    public Vehicle getVehicle() {  
        return vehicle;  
    }  
  
    public LocalDateTime getStartTime() {
```



```
        return startTime;
    }

    public LocalDateTime getEndTime() {
        return endTime;
    }
}
```

A29 Erweiterung der Klasse "ParkingGarage" mit Reservierungsfunktionalität

```
import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Map;

public class ParkingGarage {
    // ...
    private Map<Integer, Reservation> reservations;

    public ParkingGarage(int capacity) {
        // ...
        this.reservations = new HashMap<>();
    }

    public boolean reserveParkingSpot(Vehicle vehicle, LocalDateTime
        startTime, LocalDateTime endTime) {
        for (ParkingSpot spot : parkingSpots) {
            if (!spot.isOccupied() &&
                !reservations.containsKey(spot.getId())) {
                Reservation reservation = new Reservation(vehicle,
                    startTime, endTime);
                reservations.put(spot.getId(), reservation);
                return true;
            }
        }
        return false;
    }

    public boolean cancelReservation(int spotId) {
        if (reservations.containsKey(spotId)) {
            reservations.remove(spotId);
        }
    }
}
```

```

        return true;
    }

    return false;
}
}

```

Mit diesen Erweiterungen können Sie Reservierungen für Parkplätze vornehmen und stornieren. Sie können auch weitere Funktionen hinzufügen, z. B. die Berechnung von Parkgebühren basierend auf der Parkdauer oder unterschiedlichen Fahrzeugtypen.

Bitte beachten Sie, dass der oben angegebene Code nur als Beispiel dient und in einem realen Projekt weiter angepasst und optimiert werden sollte, um sicherzustellen, dass er den Anforderungen und Best Practices entspricht.

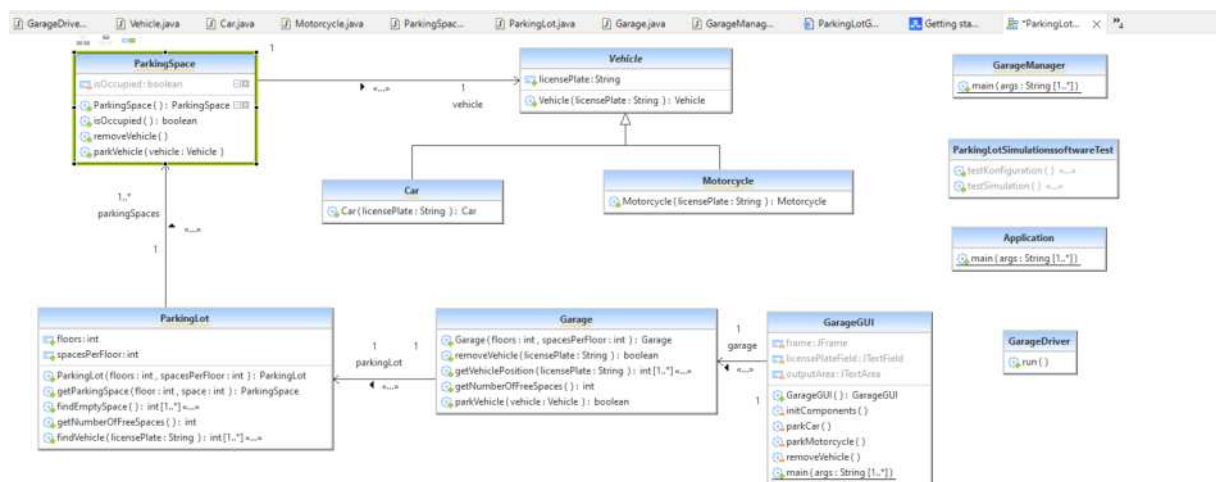
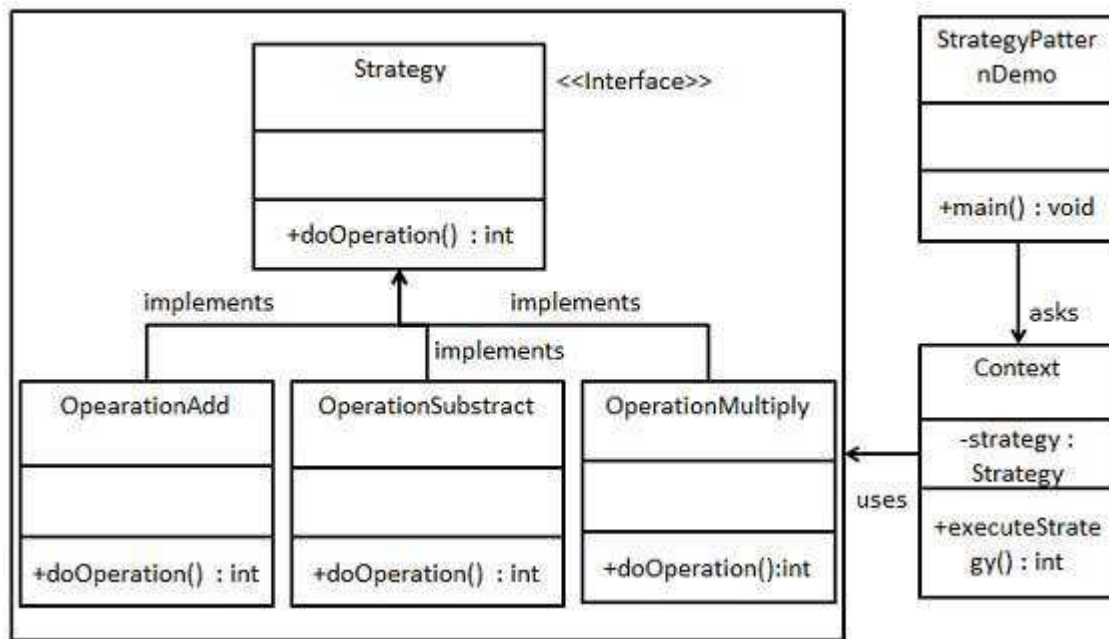


Abbildung 13: Klassendiagramm der Parkhaus Simulationssoftware

A30 Das Strategy Pattern

Das Strategy Pattern Design im Anhang zeigt, wie verschiedene Parkstrategien in der Parkhaus Simulationssoftware implementiert und angewendet werden können, um die Effizienz der Parkplatzbelegung zu optimieren.

Das Strategy Pattern ist ein Designmuster in der objektorientierten Programmierung, dass es ermöglicht, eine Familie von Algorithmen zu definieren, diese zu kapseln und untereinander austauschbar zu machen. Das Pattern besteht aus drei Komponenten: dem Context, den Strategien und der abstrakten Strategy-Klasse



Step 1

Erstellen Sie ein interface.

```

Strategy.java
public interface Strategy {
    public int doOperation(int num1, int num2);
}
    
```

Step 2

Erstelle eine Konkrete Klasse mit der Implementierung des oben genannten Interface.

OperationAdd.java

```

public class OperationAdd implements Strategy{
    @Override
    public int doOperation(int num1, int num2) {
        return num1 + num2;
    }
}
    
```

OperationSubtract.java

```

public class OperationSubtract implements Strategy{
    @Override
    public int doOperation(int num1, int num2) {
        return num1 - num2;
    }
}
    
```

OperationMultiply.java

```

public class OperationMultiply implements Strategy{
    
```

```
@Override
public int doOperation(int num1, int num2) {
    return num1 * num2;
}
}
```

Step 3

Erstellen Sie eine Kontextklasse.

Context.java

```
public class Context {
    private Strategy strategy;

    public Context(Strategy strategy){
        this.strategy = strategy;
    }

    public int executeStrategy(int num1, int num2){
        return strategy.doOperation(num1, num2);
    }
}
```

Step 4

Verwenden Sie den Kontext, um Verhaltensänderungen zu erkennen, wenn die Strategie geändert wird.

.

StrategyPatternDemo.java

```
public class StrategyPatternDemo {
    public static void main(String[] args) {
        Context context = new Context(new OperationAdd());
        System.out.println("10 + 5 = " + context.executeStrategy(10, 5));

        context = new Context(new OperationSubtract());
        System.out.println("10 - 5 = " + context.executeStrategy(10, 5));

        context = new Context(new OperationMultiply());
        System.out.println("10 * 5 = " + context.executeStrategy(10, 5));
    }
}
```

Step 5

Überprüfen Sie die Ausgabe.

$10 + 5 = 15$

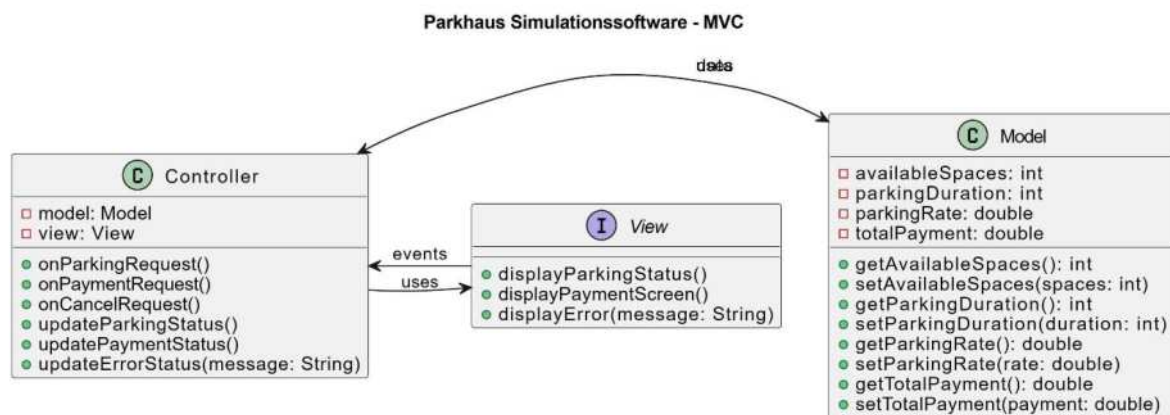
$10 - 5 = 5$

$10 * 5 = 50$

An diesem Beispiel ist zu sehen wie alle 3 verschiedene Verhaltensweisen(Algorithmen)angewandt werden.

Quelle: (https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm)

A31 Model View Control Pattern angewandt am Parkhaus Projekt



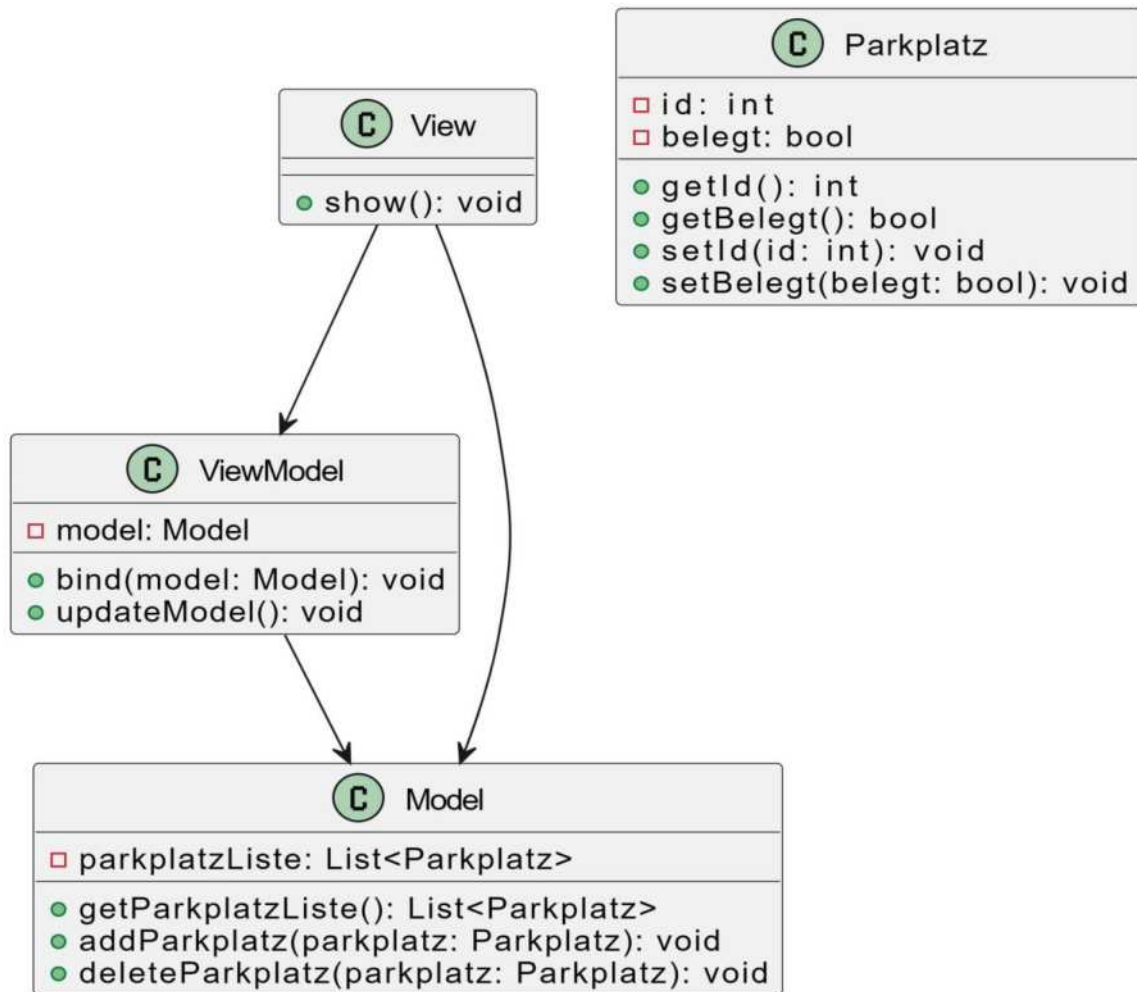
A32 Das Model-View-ViewModel (MVVM) Design Pattern

Das Model-View-ViewModel (MVVM) Design Pattern ist eine Abwandlung des MVC Patterns und ermöglicht eine bessere Trennung von Daten und Logik in der Anwendungsarchitektur. Das MVVM Pattern besteht aus den folgenden drei Komponenten:

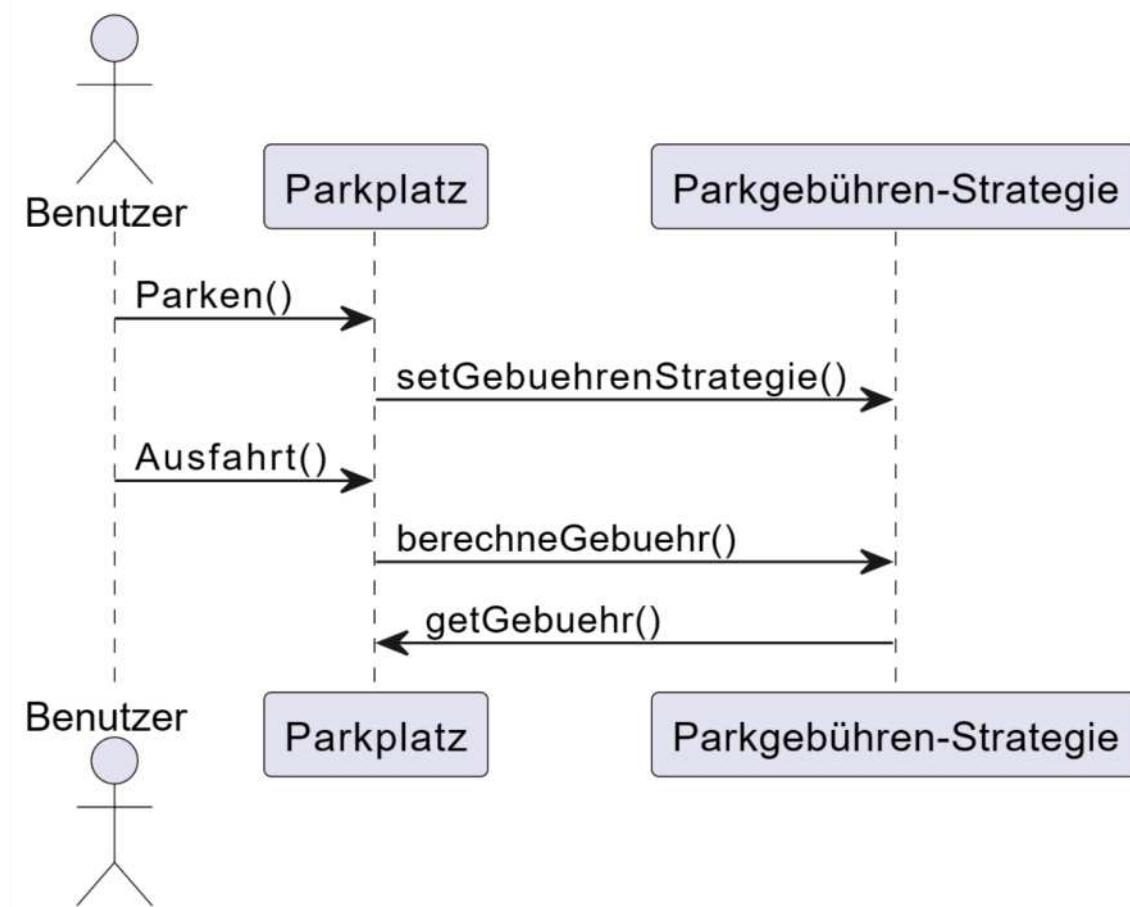
1. Model: Das Model repräsentiert die Daten und enthält die Business-Logik der Anwendung. Es ist für die Verarbeitung der Daten und die Durchführung von Berechnungen zuständig.
2. View: Die View ist für die Darstellung der Daten und die Interaktion mit dem Benutzer zuständig. Sie zeigt die Daten aus dem Model an und leitet Benutzerinteraktionen an den ViewModel weiter.
3. ViewModel: Das ViewModel verbindet das Model mit der View und enthält alle Präsentationslogik. Es stellt den View-Komponenten Daten aus dem Model zur Verfügung und nimmt Benutzereingaben entgegen.

Im MVVM Pattern werden Änderungen am Model automatisch an die View weitergeleitet, ohne dass die View direkt mit dem Model interagiert. Das ViewModel fungiert dabei als Vermittler zwischen Model und View und ermöglicht eine lose Kopplung der Komponenten.

Das MVVM Pattern eignet sich besonders für komplexe Anwendungen mit umfangreicher Logik und einer Vielzahl von Datenquellen. Es erleichtert die Wartung und Erweiterung der Anwendung, da die einzelnen Komponenten separat entwickelt und getestet werden können.



Quelle: (https://de.wikipedia.org/wiki/Model_View_ViewMode)

A33 Strategy Pattern Design angewendet am Parkhaus Projekt:

Der Context ('ParkhausSimulationssoftware') besitzt eine Referenz auf die abstrakte Strategy-Klasse (ParkplatzStrategy), die eine Schnittstelle für die konkreten Strategien (EinfachParkplatzStrategy, MittelParkplatzStrategy, LuxusParkplatzStrategy) definiert. Jede konkrete Strategie implementiert die Methoden der abstrakten Strategy-Klasse auf ihre eigene Art und Weise. Der Context ruft die Methoden der Strategy-Klasse auf und delegiert damit die Ausführung an die jeweilige konkrete Strategie.

Dieses Design ermöglicht es, verschiedene Algorithmen für die Parkplatzzuweisung zu implementieren, ohne dass der Context-Klasse verändert werden muss. Dadurch wird die Flexibilität und Erweiterbarkeit der Parkhaus-Simulationssoftware erhöht.

A34 Zusammenfassung

In dieser Dokumentation haben wir die Implementierung einer Parkhaus-Simulationssoftware vorgestellt. Wir haben den Entwicklungsprozess beschrieben und detaillierte Einblicke in die Analyse-, Entwurfs-, Implementierungs- und Abnahmephase gegeben. Wir haben auch die Funktionalitäten der Software beschrieben, einschließlich der Verwaltung von Parkplätzen, der Verwaltung von Fahrzeugen und der Möglichkeit, Reservierungen vorzunehmen und zu stornieren.

Die Implementierung der Parkhaus-Simulationssoftware erfordert ein gründliches Verständnis der Anforderungen sowie eine systematische Planung und Umsetzung. Die Verwendung von objektorientierter Programmierung, MVC-Architektur und einer umfassenden Teststrategie ist für eine erfolgreiche Implementierung entscheidend.

Abschließend lässt sich sagen, dass die Implementierung einer Parkhaus-Simulationssoftware ein herausforderndes, aber lohnendes Projekt ist, das dazu beitragen kann, den Betrieb von Parkhäusern zu optimieren und die Kundenerfahrung zu verbessern.

A35 Klassendiagramm

Das Klassendiagramm zeigt die wichtigsten Klassen der Parkhaus-Simulationssoftware sowie deren Beziehungen zueinander. Die Klasse "ParkingGarage" repräsentiert das Parkhaus selbst und enthält Funktionen zum Parken und Abstellen von Fahrzeugen sowie zur Verwaltung von Parkplätzen und Reservierungen. Die Klasse "Vehicle" ist die Basisklasse für alle Fahrzeugtypen und enthält eine gemeinsame Eigenschaft - die Kennzeichennummer. Die Klasse "ParkingSpot" repräsentiert einen einzelnen Parkplatz im Parkhaus und enthält Funktionen zum Überprüfen, ob ein Parkplatz belegt ist, sowie zum Belegen und Freigeben des Parkplatzes.

Die Klasse "Reservation" ermöglicht es dem Benutzer, einen Parkplatz zu reservieren und enthält Informationen über das Fahrzeug, den Start- und Endzeitpunkt der Reservierung. Die Klasse "ParkingFeeCalculator" ist eine abstrakte Basisklasse für verschiedene Parkgebühren Berechnungsalgorithmen, wie z.B. eine Flatrate oder stündliche Berechnung. Die Klasse "FlatRateCalculator" und "HourlyRateCalculator" implementieren jeweils einen Algorithmus zur Berechnung der Parkgebühren basierend auf der Art der Gebührenberechnung.

Zusätzlich gibt es die Klassen "Car" und "Motorbike", die von der Klasse "Vehicle" erben und spezifische Eigenschaften für Autos bzw. Motorräder enthalten.

Das Klassendiagramm verdeutlicht die Verwendung von Vererbung, Komposition und Assoziationen zur Umsetzung der Funktionalitäten der Parkhaus-Simulationssoftware. Die Klasse "ParkingGarage" hält eine Liste von Parkplätzen ("ParkingSpot") und eine Liste von Fahrzeugen ("Vehicle"). Die Parkplätze sind in einem Array gespeichert, während die Fahrzeuge in einer Liste ("List<Vehicle>") gehalten werden. Es gibt auch eine Map ("Map<Integer, Reservation>") für Reservierungen, wobei der Schlüssel der ID des Parkplatzes ist und der Wert eine Reservierung ist.

Die Klasse "Vehicle" enthält lediglich eine Eigenschaft, das Kennzeichen ("plateNumber"), und wird von den spezifischeren Fahrzeugtypen wie "Car" und "Motorbike" erweitert.

Die Klasse "ParkingSpot" enthält eine Eigenschaft "id" und eine Eigenschaft "occupied", die angibt, ob der Parkplatz belegt ist oder nicht. Es gibt auch Funktionen zum Überprüfen des Status des Parkplatzes und zum Belegen und Freigeben des Parkplatzes.

Die Klasse "Reservation" enthält Informationen über einem reservierten Parkplatz ("ParkingSpot"), das reservierte Fahrzeug ("Vehicle") sowie den Start- und Endzeitpunkt der Reservierung.

Die abstrakte Klasse "ParkingFeeCalculator" definiert eine Funktion zur Berechnung der Parkgebühren ("calculateFee ()"). Die konkreten Implementierungen, "FlatRateCalculator" und "HourlyRateCalculator", überschreiben diese Funktion mit unterschiedlichen Algorithmen zur Berechnung der Gebühren.

Das Klassendiagramm zeigt auch die Verwendung von Vererbung, da die spezifischeren Fahrzeugtypen wie "Car" und "Motorbike" von der Basisklasse "Vehicle" erben. Die Klasse "Reservation" verwendet Assoziationen, um Beziehungen zwischen einem reservierten Parkplatz und dem reservierten Fahrzeug herzustellen.

DIE PARKHAUS SIMULATIONSSOFTWARE

Implementierung eines neuen Parkhauses mithilfe einer Simulationssoftware

Insgesamt verdeutlicht das Klassendiagramm die grundlegenden Entitäten und Beziehungen innerhalb der Parkhaus-Simulationssoftware. Um das Klassendiagramm weiter zu verfeinern, könnten noch zusätzliche Klassen und Beziehungen hinzugefügt werden, um bestimmte Funktionalitäten der Parkhaus-Simulationssoftware abzubilden. Beispielsweise könnten Klassen für verschiedene Typen von Parkplätzen hinzugefügt werden, z.B. "NormalParkingSpot" und "HandicappedParkingSpot", um spezielle Anforderungen von bestimmten Kunden zu erfüllen. Ebenfalls könnten weitere Klassen für bestimmte Parkhausfunktionen wie Ein- und Ausfahrt, Bezahlung, Ticket-Validierung, usw. hinzugefügt werden.

Außerdem könnte eine Verbindung mit einem Datenbanksystem hergestellt werden, um die Verwaltung von Parkplatzreservierungen, Fahrzeugen und Parkplätzen zu optimieren. Dies würde eine zusätzliche Klasse zur Datenbankkommunikation erfordern, sowie möglicherweise weitere Klassen für Datenbanktabellen und -schemata.

Das Klassendiagramm könnte auch um zusätzliche Assoziationen und Abhängigkeiten erweitert werden, um eine detailliertere Darstellung der Softwarefunktionen und -beziehungen zu ermöglichen. Beispielsweise könnte eine Assoziation zwischen den Klassen "ParkingSpot" und "ParkingFeeCalculator" hinzugefügt werden, um spezifische Berechnungsalgorithmen für bestimmte Parkplätze zu ermöglichen.

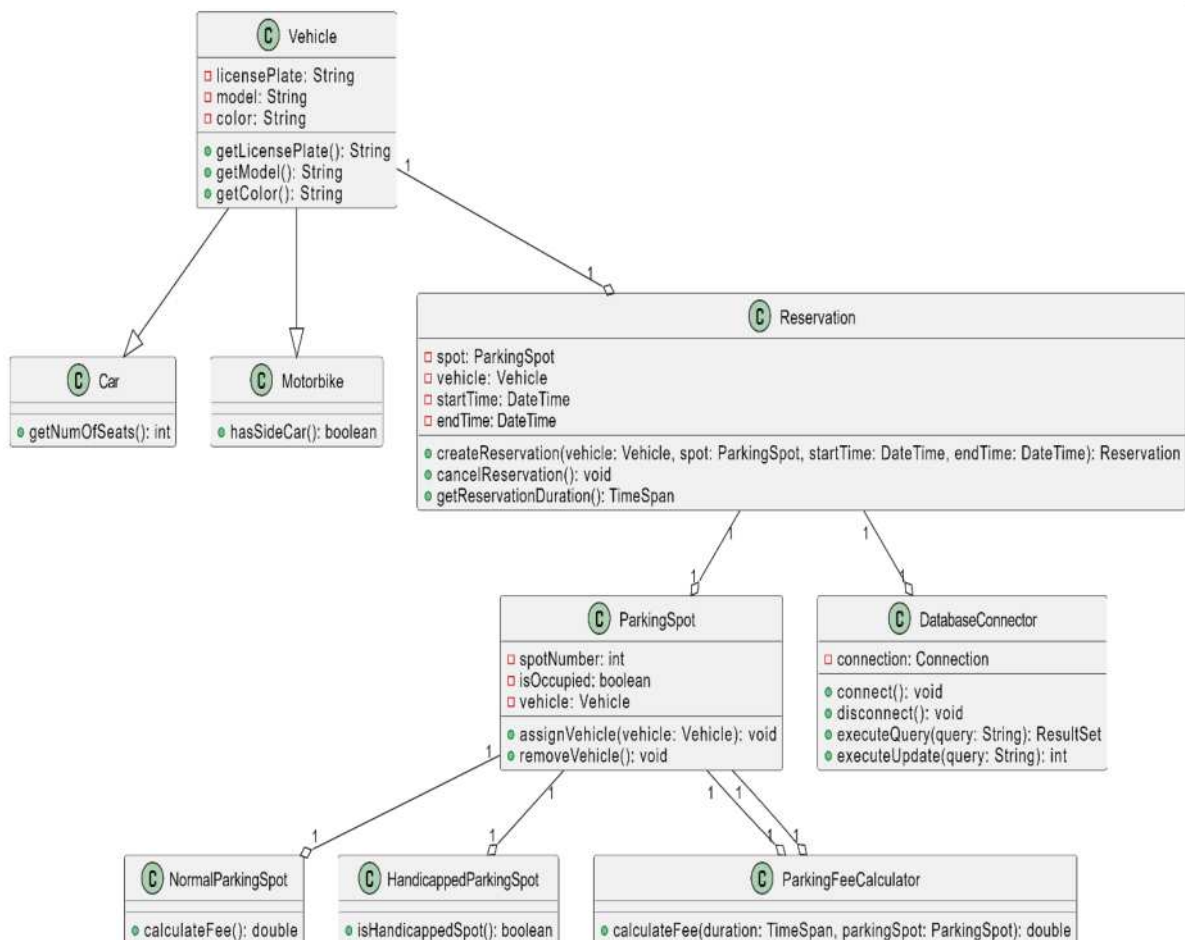


Abbildung 10: Das Klassendiagramm der Parkhaus-Simulationssoftware könnte wie folgt aussehen

A36 Benutzerdokumentation (Auszug)

Die Benutzerdokumentation im Anhang bietet eine Anleitung für die Bedienung der Parkhaus Simulationssoftware und erklärt, wie Benutzer die verschiedenen Funktionen nutzen können.

Im Folgenden finden Sie einen Auszug aus der Benutzerdokumentation der Parkhaus-Simulationssoftware:

1. Einleitung

Die Parkhaus-Simulationssoftware ist eine Anwendung, die es Ihnen ermöglicht, die Prozesse in einem Parkhaus zu simulieren und zu optimieren. Sie können verschiedene Parameter wie die Anzahl der Parkplätze, die Art der Parkplätze, die Parkdauer und die Gebühren einstellen, um das ideale Parkhaus-Layout zu finden.

2. Installation

Die Installation der Parkhaus-Simulationssoftware ist einfach und intuitiv. Sie müssen lediglich die Installationsdatei herunterladen und auf Ihrem Computer ausführen. Die Software ist plattformunabhängig und läuft auf Windows, MacOS und Linux.

3. Anmeldung

Um die Software nutzen zu können, müssen Sie sich mit einem Benutzerkonto anmelden. Wenn Sie noch kein Konto haben, können Sie sich einfach registrieren. Nach erfolgreicher Anmeldung haben Sie Zugriff auf alle Funktionen der Software.

4. Benutzeroberfläche

Die Benutzeroberfläche der Parkhaus-Simulationssoftware ist übersichtlich und benutzerfreundlich gestaltet. Sie können alle Einstellungen und Funktionen über das Hauptmenü erreichen. Die Parkhaus-Simulation wird in Echtzeit ausgeführt, so dass Sie die Auswirkungen Ihrer Einstellungen sofort sehen können.

5. Konfiguration

Die Konfiguration der Parkhaus-Simulation umfasst verschiedene Parameter, die Sie anpassen können. Sie können die Anzahl und Art der Parkplätze, die Parkdauer, die Gebühren und andere Faktoren einstellen, um das ideale Parkhaus-Layout zu finden.

6. Simulation

Nachdem Sie die Konfiguration abgeschlossen haben, können Sie die Parkhaus-Simulation starten. Sie können den Fortschritt der Simulation in Echtzeit überwachen und die Ergebnisse analysieren, um die Leistung Ihres Parkhauses zu optimieren.

7. Berichte

Die Parkhaus-Simulationssoftware generiert detaillierte Berichte über den Verlauf der Simulation und die Leistung des Parkhauses. Sie können diese Berichte verwenden, um Verbesserungen und Optimierungen vorzunehmen.

8. Support

Wenn Sie Probleme oder Fragen zur Parkhaus-Simulationssoftware haben, können Sie sich jederzeit an unseren Support wenden. Wir sind stets bemüht, Ihnen schnell und effizient zu helfen.

Unser Support-Team ist jederzeit für Sie da, um Ihnen bei Fragen oder Problemen bezüglich der Parkhaus-Simulationssoftware weiterzuhelfen. Sie können uns telefonisch unter 040 812000 von 8:00 bis 17:00 Uhr erreichen, im Notfall sind wir rund um die Uhr für Sie erreichbar. Falls Sie eine Support-

DIE PARKHAUS SIMULATIONSSOFTWARE

Implementierung eines neuen Parkhauses mithilfe einer Simulationssoftware

Anfrage haben, können Sie uns auch eine E-Mail an hilfe@grassau.com senden oder unser Ticketsystem nutzen.

Als IT-Dienstleister bieten wir eine breite Palette an Dienstleistungen sowohl für Privat- als auch Firmenkunden an. Dazu gehören unter anderem individuelle Software-Deployment-Lösungen, Virtualisierung, IT-Security, VoIP-Telefonie und Azure Cloud und Microsoft 365. Wir sind stolz darauf, dass wir keine Callcenter verwenden und Sie immer direkt persönlich erreichen können. Wir bieten eine zuverlässige und flexible Unterstützung und beraten Sie ehrlich mit Weitsicht, um Ihre Bedürfnisse optimal zu lösen. Mit unserer langjährigen Erfahrung, zielorientiertem Arbeiten und festen Terminzusagen sind wir immer für Sie da, wenn Sie uns brauchen.



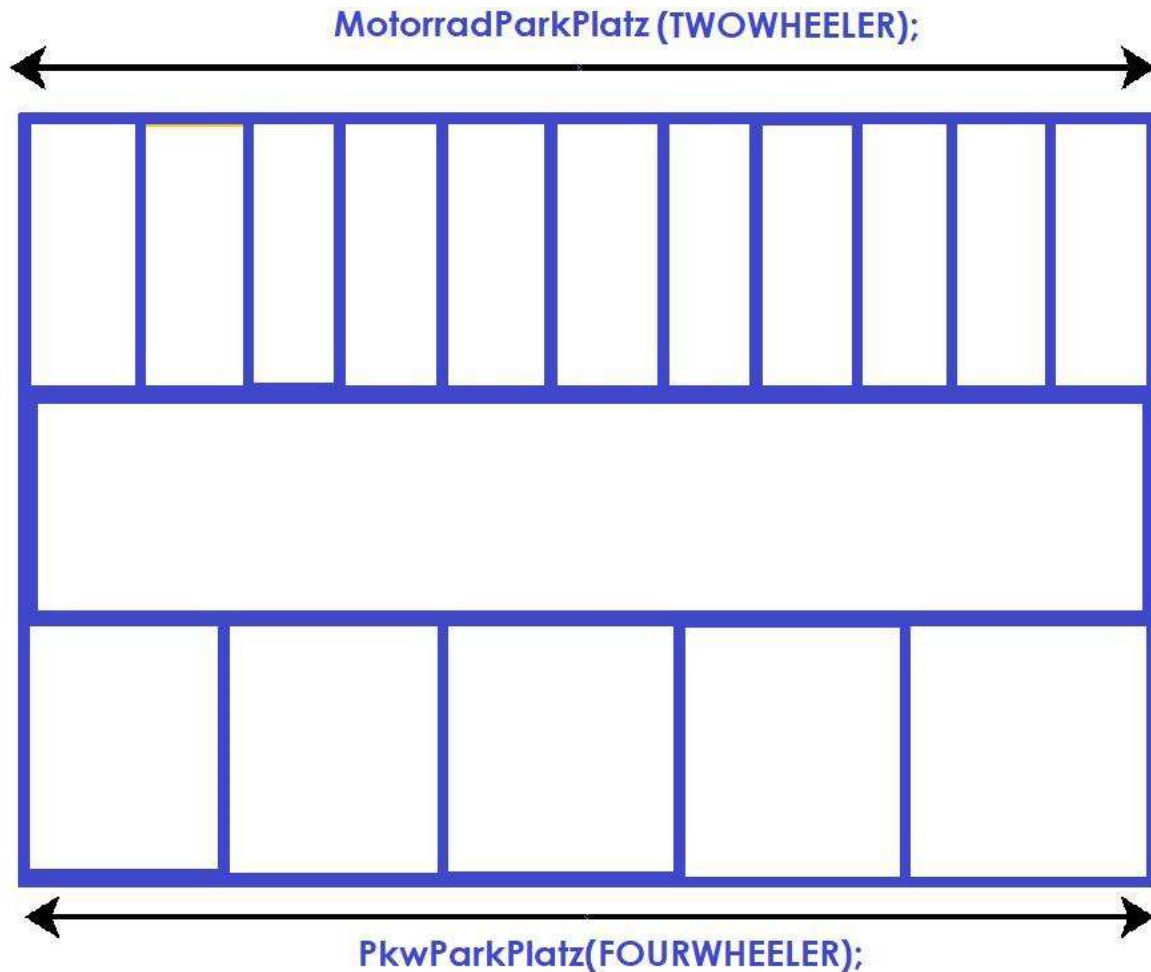
Abbildung 11:

Dies war lediglich ein kurzer Auszug aus der Webpräsenz.

Falls Sie weitere Informationen benötigen, empfehlen wir Ihnen, die vollständige Dokumentation zu lesen oder sich direkt an unseren Support zu wenden. Unsere IT-Experten stehen Ihnen gerne zur Verfügung und helfen Ihnen schnell und effizient bei Fragen oder Problemen rund um die Parkhaus-Simulationssoftware weiter.

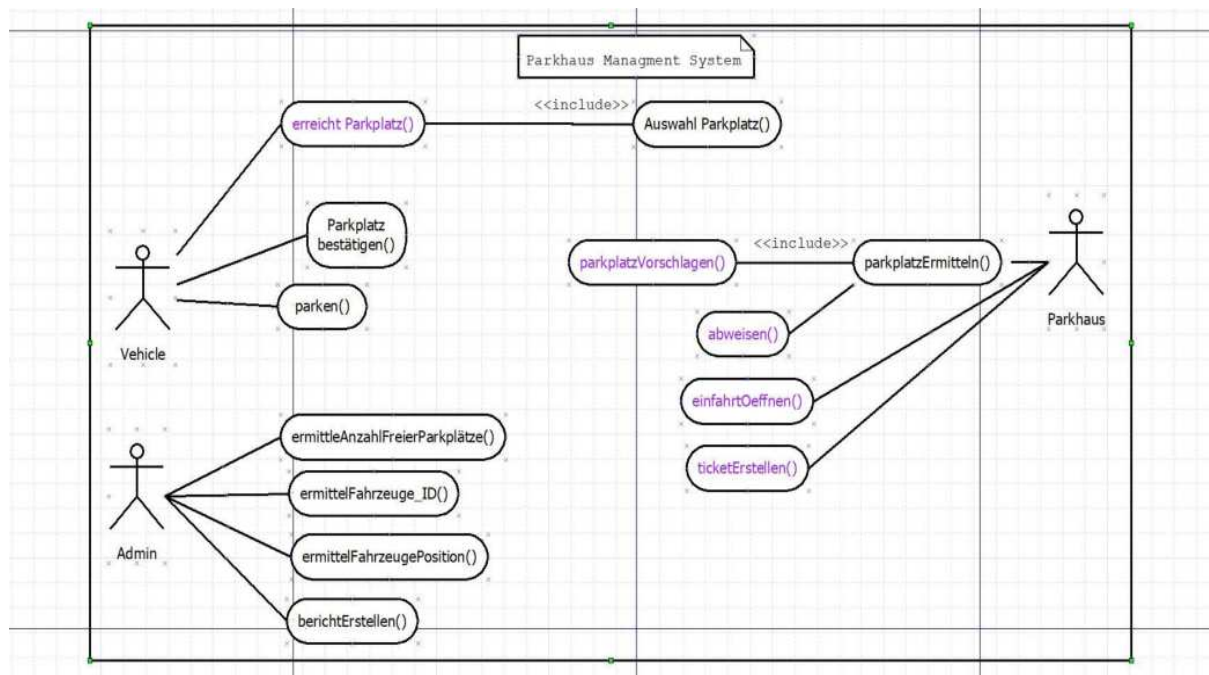
A37 Parkplatz Design

Der Anhang enthält das Parkplatz Design, das die Struktur und Organisation der Parkplätze im Parkhaus darstellt und die Grundlage für die Berechnung und Optimierung der Parkplatzbelegung bildet.



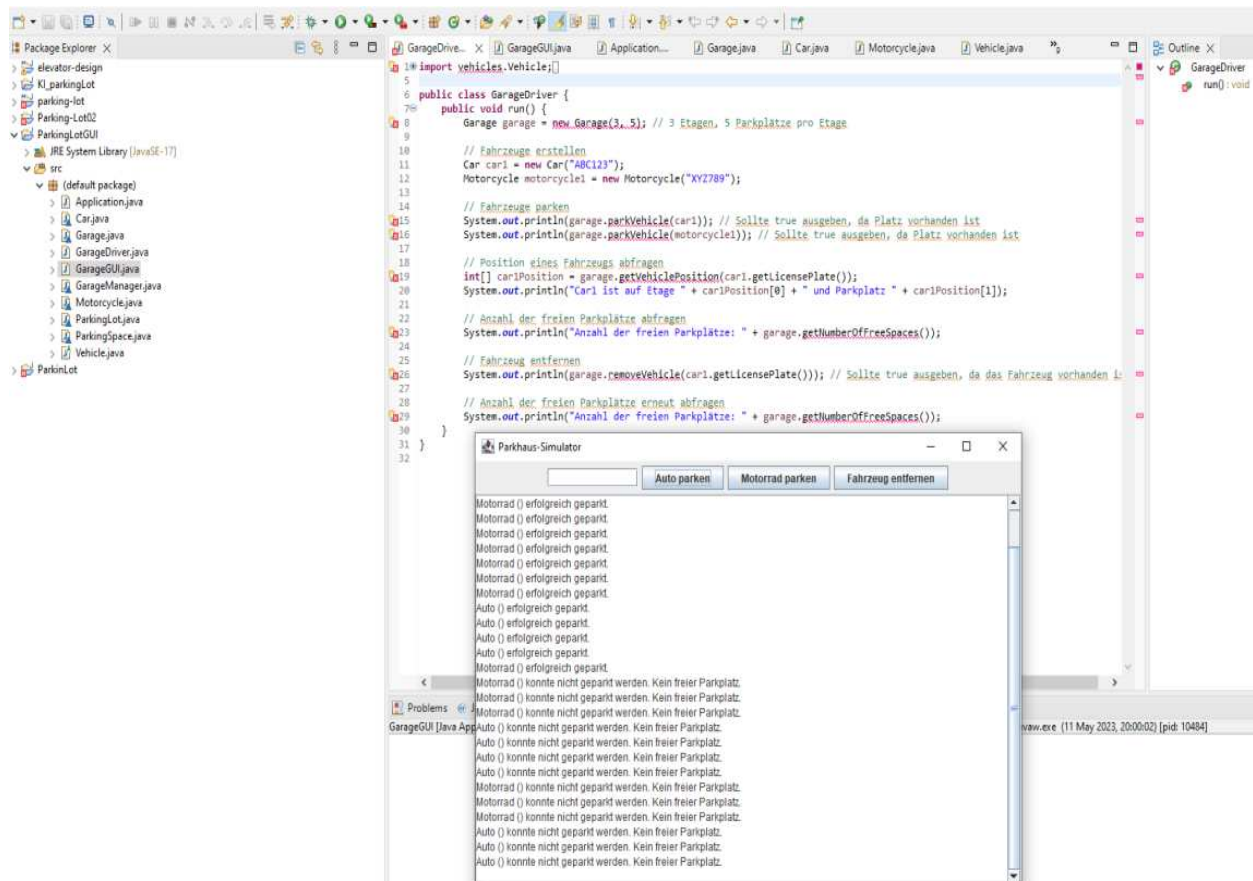
DIE PARKHAUS SIMULATIONSSOFTWARE

Implementierung eines neuen Parkhauses mithilfe einer Simulationssoftware



A38 Use-Case-Diagramm mit Methoden/Prozeduren

Das Use-Case-Diagramm im Anhang zeigt die verschiedenen Anwendungsfälle der Parkhaus Simulationssoftware und die zugehörigen Methoden und Prozeduren, die für die Umsetzung benötigt werden.



A39 Codeschnipsel, der die Einfahrt eines Parkhauses simuliert

Der Codeausschnitt im Anhang zeigt, wie die Einfahrt eines Fahrzeugs in das Parkhaus simuliert wird und welche Methoden und Algorithmen dabei verwendet werden.

Here is a sample code snippet that simulates the entry of a vehicle in a parking lot:

```
public class ParkingLot {
    private int capacity;
    private int freeSpaces;

    // Constructor
    public ParkingLot(int capacity) {
        this.capacity = capacity;
        this.freeSpaces = capacity;
    }

    // Method to simulate vehicle entry
    public void vehicleEnters() {
        if (freeSpaces > 0) {
            freeSpaces--;
            System.out.println("Vehicle entered the parking lot.");
        } else {
            System.out.println("Parking lot is full. Vehicle cannot enter.");
        }
    }
}
```

This is just a simple example, but it shows how you can start building a parking lot simulation in Java.

A40 Das Parkplatzreservierungssystem

Der Anhang enthält Informationen und Materialien zum Parkplatzreservierungssystem der Parkhaus Simulationssoftware, das die Verwaltung von Parkplatzbuchungen und Reservierungen ermöglicht.

Das Parkplatzreservierungssystem ist eine wichtige Komponente der Parkhaus Simulationssoftware, da es den Benutzern die Möglichkeit gibt, einen Parkplatz im Voraus zu buchen und sicherzustellen, dass sie einen Platz haben, wenn sie ankommen.

Das System besteht aus einer Datenbank, die die verfügbaren Parkplätze und Buchungen verwaltet, sowie einer Benutzeroberfläche, über die Benutzer Parkplätze suchen, buchen und stornieren können.

Im Anhang finden Sie das Entity-Relationship-Modell (ERM) des Parkplatzreservierungssystems, das die Beziehungen zwischen den verschiedenen Tabellen der Datenbank zeigt. Außerdem finden Sie Screenshots der Benutzeroberfläche des Systems, die während der Entwicklungsphase erstellt wurden.

Das Parkplatzreservierungssystem ist eine wichtige Funktion der Parkhaus Simulationssoftware und trägt dazu bei, dass die Benutzer eine reibungslose und stressfreie Erfahrung haben, wenn sie das Parkhaus nutzen.

A41 Flowchart Diagramme

Die Flowchart Diagramme im Anhang zeigen die Abläufe und Prozesse der Parkhaus Simulationssoftware in einer visuellen

Darstellung visualisieren

Flowchart-Diagramme können verwendet werden, um die verschiedenen Abläufe und Prozesse der Parkhaus Simulationssoftware in einer visuellen Darstellung zu visualisieren. Hier sind einige Beispiele für mögliche Flowchart-Diagramme:

1. Parkplatz-Reservierungssystem

Dieses Diagramm könnte den Prozess der Parkplatz-Reservierung im Parkhaus darstellen. Es könnte den Benutzerfluss von der Auswahl des Parkhauses bis zur Reservierung eines Parkplatzes und der Zahlungsabwicklung darstellen.

2. Fahrzeug-Einfahrtsprozess

Dieses Diagramm könnte den Prozess der Fahrzeugeinfahrt im Parkhaus darstellen. Es könnte den Benutzerfluss von der Erkennung des Fahrzeugs bis zur Schrankenöffnung und dem Parkplatz-Belegungsprozess darstellen.

3. Fahrzeug-Ausfahrtsprozess

Dieses Diagramm könnte den Prozess der Fahrzeugausfahrt aus dem Parkhaus darstellen. Es könnte den Benutzerfluss von der Bezahlung bis zur Schrankenöffnung und der Ausfahrt des Fahrzeugs darstellen.

4. Datenfluss-Diagramm

Dieses Diagramm könnte den Datenfluss zwischen den verschiedenen Systemkomponenten der Parkhaus Simulationssoftware darstellen. Es könnte die Datenflüsse zwischen den Tabellen User, Vehicle, ParkingSpot und Reservation darstellen.

Durch die Verwendung von Flowchart-Diagrammen können die verschiedenen Abläufe und Prozesse der Parkhaus Simulationssoftware leichter verständlich gemacht werden.

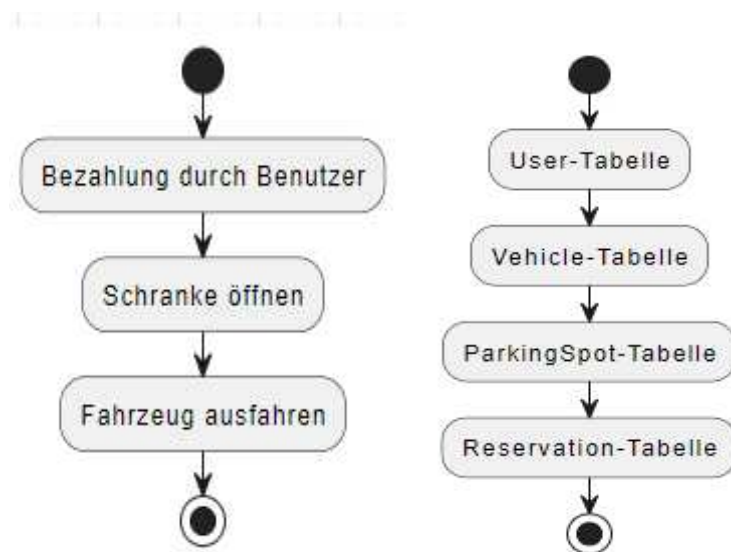
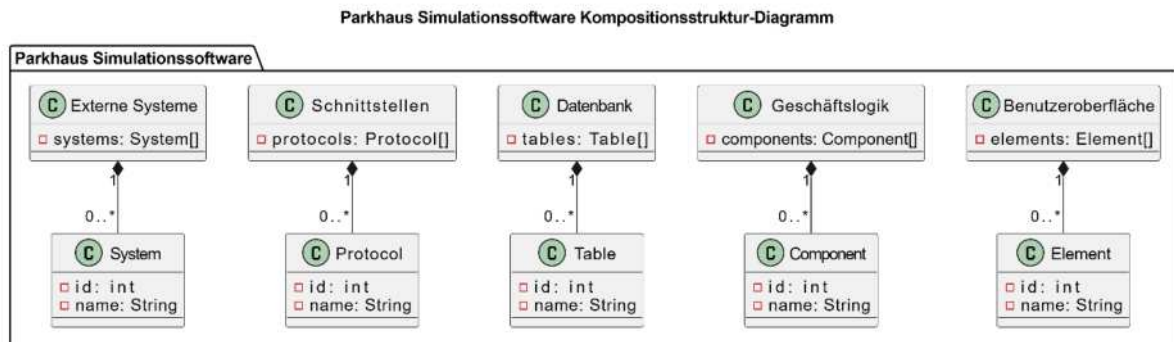
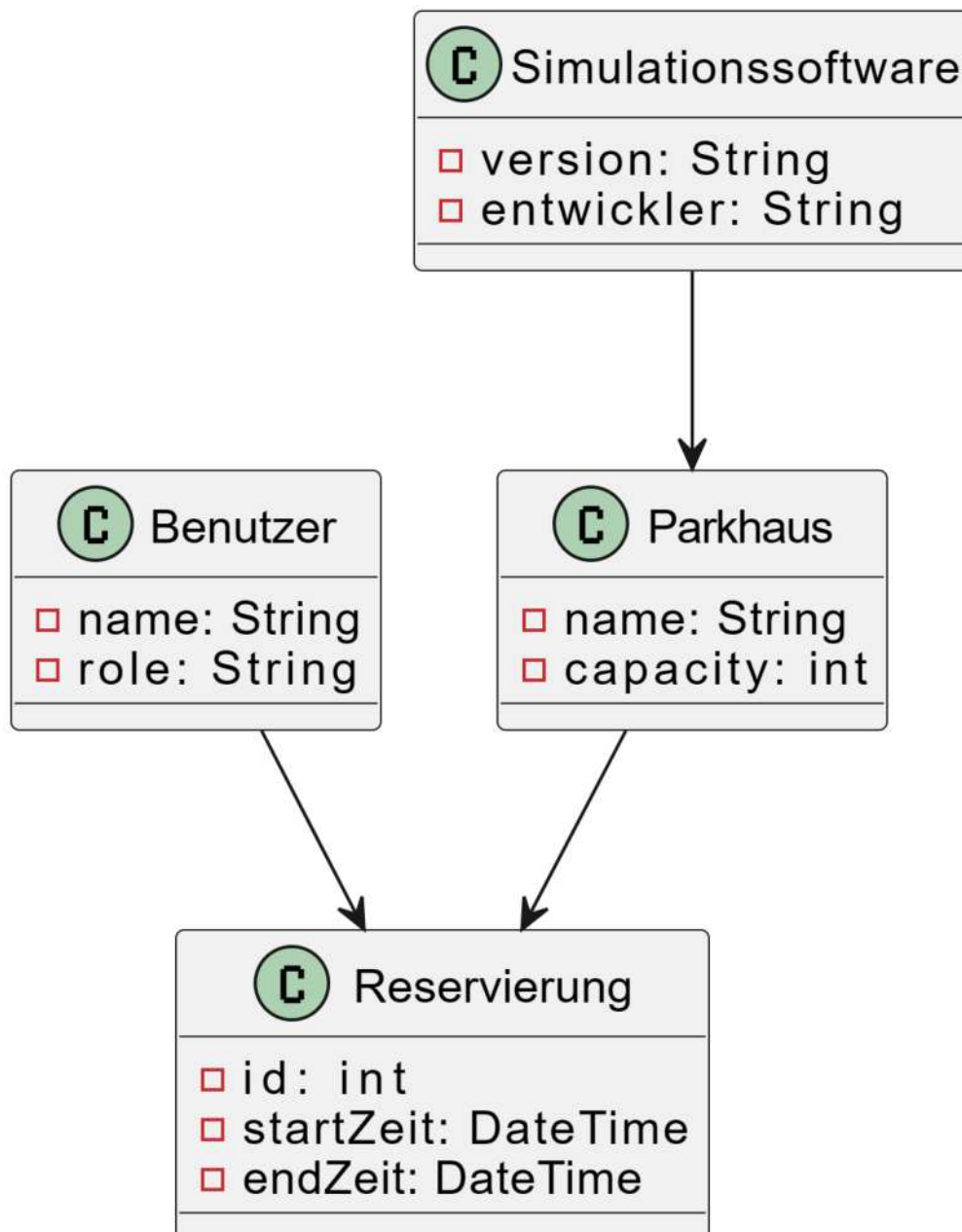


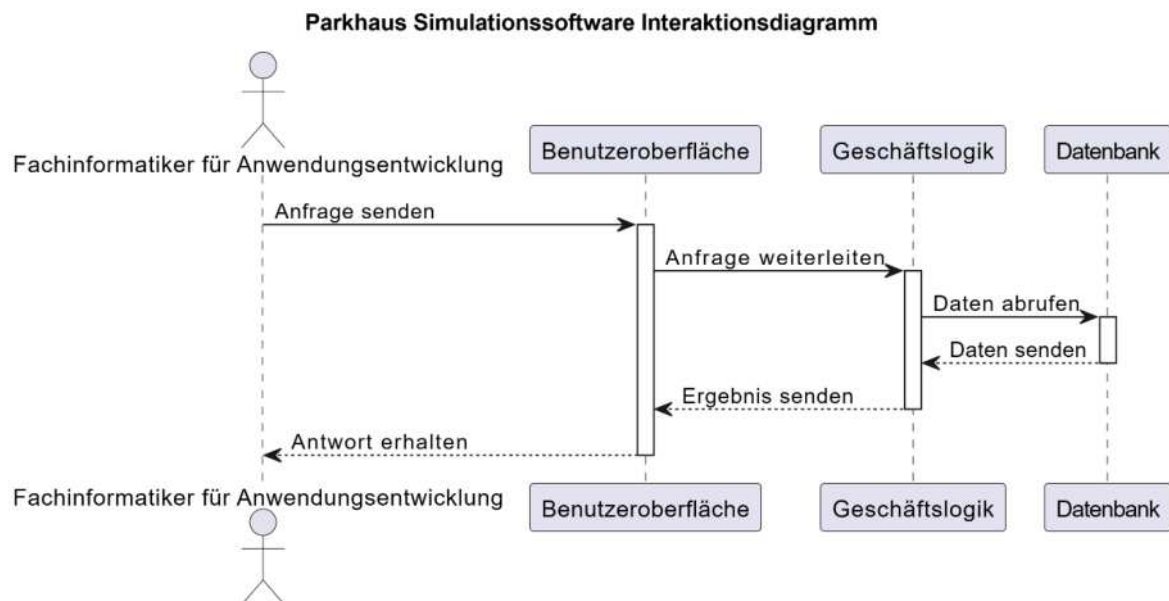
Abbildung 12: Flowchart

A42 Kompositionsstruktur-Diagramm

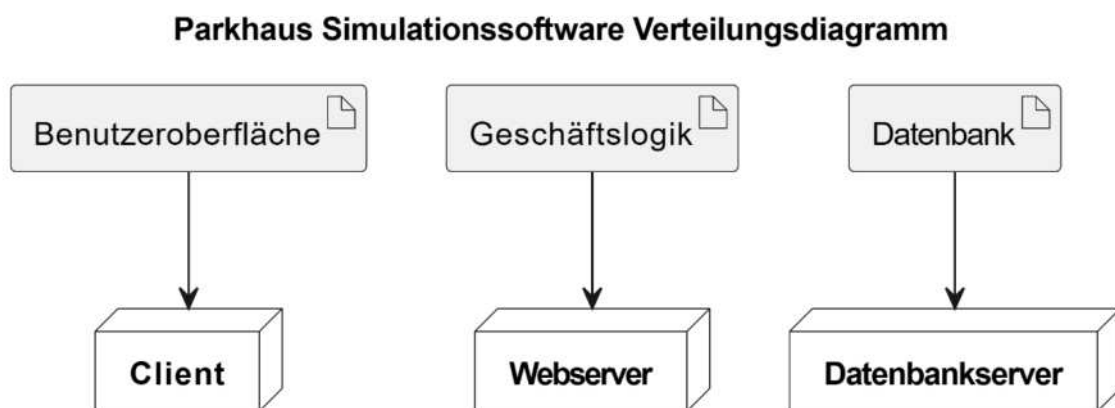


A43 Objektdiagramm**Parkhaus Simulationssoftware Objektdiagramm**

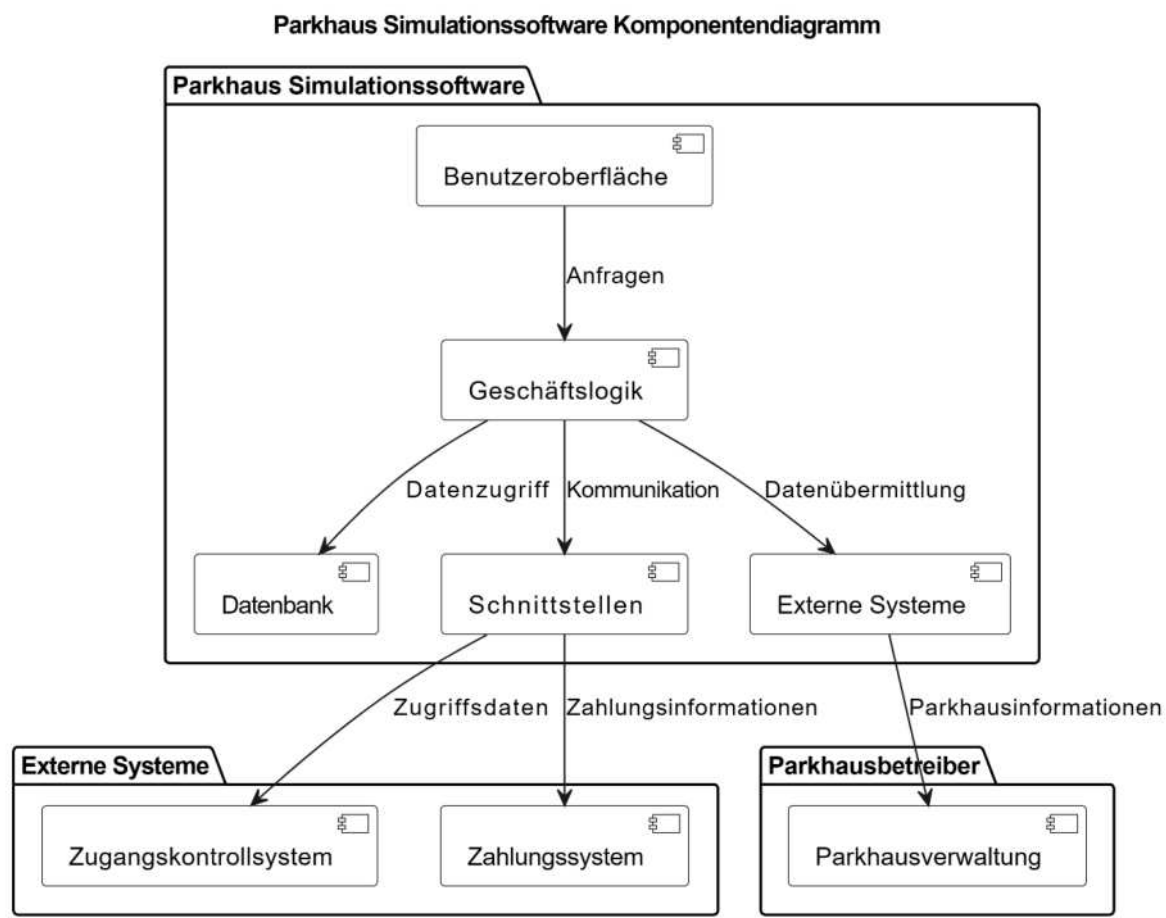
A44 Interaktionsdiagramm



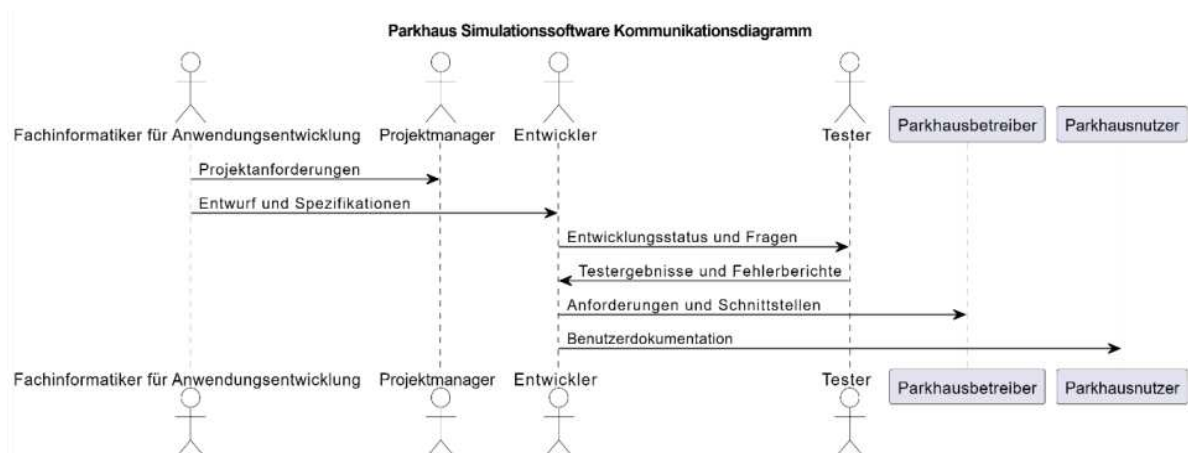
A45 Verteilungsdiagramm



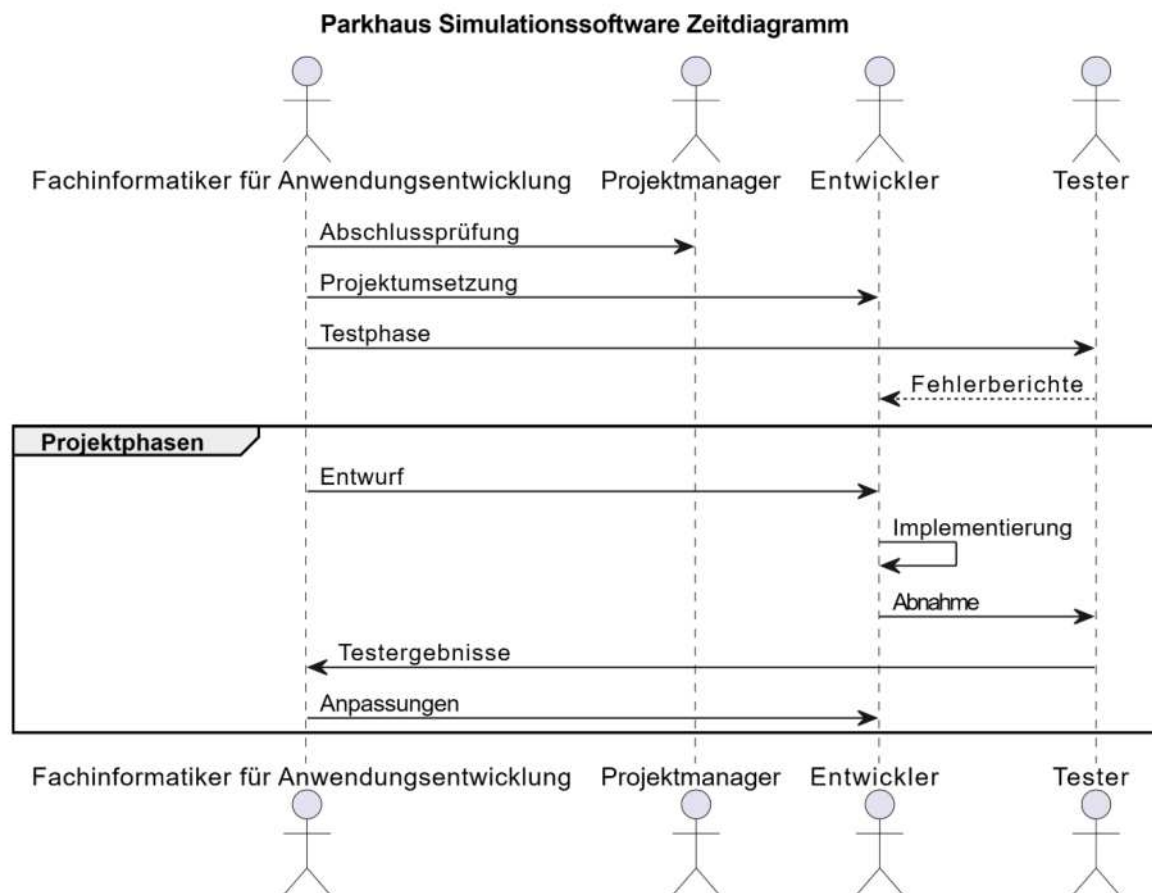
A46 Komponentendiagramm



A47 Kommunikationsdiagramm



A48 Zeitdiagramm



A49 Abnahmeprotokoll

Das Abnahmeprotokoll im Anhang dokumentiert die Ergebnisse der Abnahmetests und die Freigabe der Parkhaus Simulationssoftware durch die Stakeholder.

Projektabschlussbericht

Der Projektabschlussbericht im Anhang gibt einen Überblick über den gesamten Projektverlauf, die erreichten Ziele, die Lessons Learned und den Ausblick auf mögliche zukünftige Erweiterungen und Verbesserungen der Parkhaus Simulationssoftware.

Projektinformationen: Projektname: Parkhaus Simulationssoftware Projektstart: 01.09.2022
Projektende: 22.05.2023 Projektleiter: Daniel-Alfonsin Massa Ausbildungsbetrieb: Grassau GmbH
Ausbildungsberuf: Fachinformatiker für Anwendungsentwicklung

Projektbeschreibung: Im Rahmen meiner Ausbildung zum Fachinformatiker für Anwendungsentwicklung habe ich im Unternehmen Grassau GmbH die Parkhaus Simulationssoftware entwickelt. Die Software simuliert ein Parkhaus und ermöglicht es dem Benutzer, das Parken und die Bezahlung in einem virtuellen Parkhaus zu üben. Das Ziel war es, eine benutzerfreundliche Software zu entwickeln, die den Benutzer in die Lage versetzt, das Parken und die Bezahlung in einem Parkhaus effektiv zu erlernen.

Projektziele:

Entwicklung einer benutzerfreundlichen Parkhaus Simulationssoftware

Umsetzung der Anforderungen des Lastenhefts

Implementierung einer sicheren Anmeldung und Zahlungsabwicklung

Integration einer Datenbank zur Speicherung von Benutzer- und Parkhausdaten

Durchführung von Tests zur Qualitätssicherung

Erstellung einer Dokumentation

- Projektphasen:
- Analysephase
- Entwurfsphase
- Implementierungsphase
- Abnahmephase
- Einführungsphase
- Dokumentation
- Ergebnisse:

Entwicklung einer benutzerfreundlichen Parkhaus Simulationssoftware mit einer übersichtlichen Benutzeroberfläche

Umsetzung der Anforderungen des Lastenhefts

Implementierung einer sicheren Anmeldung und Zahlungsabwicklung

Integration einer Datenbank zur Speicherung von Benutzer- und Parkhausdaten

Durchführung von Tests zur Qualitätssicherung

Erstellung einer umfangreichen Dokumentation, einschließlich einer Entwickler- und Benutzerdokumentation sowie einer Datenbankdokumentation

A50 Lessons Learned:

1. Das Projekt wurde erfolgreich abgeschlossen, wobei einige wertvolle Erfahrungen und Erkenntnisse gewonnen wurden. Eine detaillierte Planung und ein strukturierter Projektverlauf waren von großer Bedeutung für den Erfolg des Projekts. Die klare Kommunikation mit den Projektbeteiligten erwies sich als entscheidend für das reibungslose Funktionieren des Teams und die Erfüllung der Anforderungen.
2. Die Einhaltung von Deadlines und die regelmäßige Überprüfung des Projektfortschritts spielten eine wesentliche Rolle, um das Projekt erfolgreich abzuschließen. Durch eine konsequente Überwachung und Kontrolle konnte sichergestellt werden, dass das Projekt im Zeitplan blieb und eventuelle Verzögerungen rechtzeitig erkannt und behoben werden konnten.
3. Tests und Qualitätssicherungsmaßnahmen wurden von Anfang an in den Entwicklungsprozess einbezogen. Dadurch konnten potenzielle Fehler frühzeitig erkannt und behoben werden, was die Qualität der Parkhaus Simulationssoftware maßgeblich verbesserte. Die kontinuierliche Überprüfung der Funktionalität und die Einbindung von Feedback trugen zur Stabilität und Zuverlässigkeit der Software bei.
4. Ein Ausblick auf die Zukunft zeigt, dass die Parkhaus Simulationssoftware sich als nützliches Instrument zur Schulung von Parkhausbenutzern erwiesen hat. Es besteht das Potenzial, die Software weiterzuentwickeln und zusätzliche Funktionen zu integrieren, wie zum Beispiel die Anbindung an Parkleitsysteme oder die Bereitstellung von Echtzeit-Parkplatzinformationen. Diese Erweiterungen würden die Funktionalität der Software erweitern und zu einer verbesserten Parkraumnutzung beitragen.

5. Insgesamt hat das Projekt zur Entwicklung der Parkhaus Simulationssoftware wichtige Erkenntnisse geliefert. Eine sorgfältige Planung, klare Kommunikation, Einhaltung von Deadlines, umfassende Tests und ein konsequenter Qualitätsanspruch waren ausschlaggebend für den erfolgreichen Abschluss des Projekts. Die gewonnenen Erfahrungen dienen als wertvolle Grundlage für zukünftige Projekte im Bereich der Softwareentwicklung und Simulationstechnologie.
6. Die Nutzung einer Nutzwertanalyse unterstützte die objektive Entscheidungsfindung bei der Auswahl der besten Optionen, sei es bei der Auswahl der Zielplattform, der Evaluierung von Softwarepaketen oder der Bewertung der Projektergebnisse.
7. Die enge Zusammenarbeit mit den Stakeholdern während des gesamten Projekts, einschließlich der Anforderungsanalyse, des Entwurfs und der Abnahme, trug maßgeblich zum Erfolg der Parkhaus Simulationssoftware bei. Regelmäßige Kommunikation und Feedbackschleifen ermöglichten eine kontinuierliche Anpassung an die Bedürfnisse und Erwartungen der Beteiligten.
8. Diese Lessons Learned sind wertvolle Erkenntnisse, die in zukünftigen Projekten berücksichtigt werden können, um eine erfolgreiche Umsetzung und Optimierung ähnlicher Softwarelösungen zu gewährleisten.

A51 Glossar

Das Glossar im Anhang enthält eine Liste der im Projekt verwendeten Fachbegriffe, Abkürzungen und deren Bedeutungen, um das Verständnis der Dokumentation und der Parkhaus Simulationssoftware zu erleichtern.

Glossar

Anwendungsentwicklung: Der Prozess der Erstellung, Implementierung und Wartung von Softwareanwendungen, die auf bestimmte Geschäftsanforderungen zugeschnitten sind.

Benutzeroberfläche (engl. Graphical User Interface - GUI): Ein visuelles Mittel zur Interaktion zwischen dem Benutzer und einem Computerprogramm, das durch grafische Elemente wie Schaltflächen, Menüs und Textfelder dargestellt wird.

Datenmodell: Eine abstrakte Darstellung der Datenstrukturen, die in einer Datenbank oder einer anderen Form von Datenverarbeitungsanwendungen verwendet werden.

Datenbank: Ein System zum Speichern und Verwalten von Daten, auf das über eine Anwendungssoftware zugegriffen werden kann.

Entwurfsphase: Die Phase in der Softwareentwicklung, in der die Anforderungen an die Anwendung in ein konkretes Design umgesetzt werden.

Geschäftslogik: Die Logik und Regeln, die eine Anwendung verwendet, um Geschäftsprozesse und -funktionen zu automatisieren.

Lastenheft: Ein Dokument, das die Anforderungen an ein Softwareprojekt beschreibt, einschließlich der funktionalen und nicht-funktionalen Anforderungen.

Nutzwertanalyse: Eine Methode zur Bewertung der Eignung von Alternativen anhand von Kriterien und ihrer relativen Bedeutung für die Gesamtentscheidung.

Pflichtenheft: Ein Dokument, das die Anforderungen an ein Softwareprojekt detailliert beschreibt, einschließlich der technischen Spezifikationen und der Funktionen, die die Anwendung erfüllen muss.

Simulationssoftware: Eine Software, die es ermöglicht, ein System oder einen Prozess unter simulierten Bedingungen zu testen oder zu optimieren.

Use-Case-Diagramm: Ein Diagramm, das die Interaktionen zwischen dem System und seinen Benutzern darstellt und die verschiedenen Use Cases und ihre Beziehungen untereinander beschreibt.

Wirtschaftlichkeitsanalyse: Eine Methode zur Bewertung der Kosten und des Nutzens einer Investition oder eines Projekts, um zu bestimmen, ob es wirtschaftlich sinnvoll ist.

A52 Projektpräsentation

Die Projektpräsentation im Anhang enthält eine Zusammenfassung der Projektarbeit, der entwickelten Parkhaus Simulationssoftware und der erreichten Ziele. Sie dient als Grundlage für die Präsentation des Projekts vor der Prüfungskommission und den Stakeholdern.





I. Analyse des Ist-Zustands:

- ★ Erfassung und Bewertung der bestehenden Parkhauslösungen und Technologien
- ★ Identifizierung von Verbesserungspotenzialen und Anforderungen

II. "Make or Buy"- Entscheidung und Wirtschaftlichkeitsanalyse:

- ★ Abwägung der Vor- und Nachteile zwischen Eigenentwicklung und Kauf
- ★ Betrachtung der Kosten und des Nutzens für das Unternehmen
- ★ Entscheidung für die Eigenentwicklung der Parkhaus Simulationssoftware



III. Anforderungsanalyse und Lastenheft:




- ★ Ermittlung und Dokumentation der Anforderungen an die Software
- ★ Zusammenarbeit mit Stakeholdern für eine umfassende Erfassung der Anforderungen
- ★ Erstellung des Lastenhefts als Grundlage für die Entwicklung

VI. Einführungsphase und Schulung:

- ★ Installation, Konfiguration und Schulung der Anwender in der Zielumgebung
- ★ Bereitstellung der Benutzerdokumentation und Unterstützung bei der Nutzung der Software





V. Qualitätssicherung und Abnahme:

- ★ Durchführung von Code Reviews und automatisierten sowie manuellen Tests
- ★ Identifizierung und Behebung von Fehlern und Verbesserungspotenzialen
- ★ Abnahme der Software durch Stakeholder mit Unterzeichnung des Abnahmeprotokolls

IV. Design und Implementierung:

- ★ Verwendung des Model-View-Controller (MVC) Entwurfsmusters
- ★ Entwicklung der Benutzeroberfläche mit Fokus auf Benutzerfreundlichkeit und Ästhetik
- ★ Implementierung der Geschäftslogik und Algorithmen für die Datenverarbeitung



VII. Ergebnisse und Ausblick:



- ★ Erfüllung der gesteckten Ziele und Anforderungen des Projekts
- ★ Wertvolle Erfahrungen und erlangte Fähigkeiten während des Projekts
- ★ Potenziale zukünftiger Erweiterungen, wie Integration von Echtzeit-Daten oder Machine-Learning-Algorithme

Schlusswort:

- ★ Zusammenfassung der präsentierten Inhalte und des Projektabschlusses
- ★ Betonung des Mehrwerts der Parkhaus Simulationssoftware für effiziente Parkhausplanung und -optimierung
- ★ Dank an das Team und die Stakeholder für ihre Unterstützung



A53 Danksagung

An dieser Stelle möchte ich, Daniel-Alfonsin Massa, mich bei allen bedanken, die mich während der Projektarbeit unterstützt und begleitet haben. Mein besonderer Dank gilt meinem Ausbildungsleiter, meinen Kollegen und Freunden, die mit Rat und Tat zur Seite standen. Ohne ihre Hilfe und Unterstützung wäre dieses Projekt nicht möglich gewesen.

Urheberrechtsangaben Autor: Daniel-Alfonsin Massa Alle Rechte an dieser Dokumentation und der darin beschriebenen Parkhaus Simulationssoftware liegen bei Daniel-Alfonsin Massa und der Grassau GmbH. Eine Vervielfältigung oder Verwendung der Dokumentation oder der Software oder Teilen davon in anderen elektronischen oder gedruckten Publikationen ist ohne ausdrückliche Zustimmung des Autors und der Grassau GmbH nicht gestattet.

An dieser Stelle möchte ich mich herzlich bei allen Personen bedanken, die mich während meiner Ausbildung zum Fachinformatiker für Anwendungsentwicklung unterstützt haben.

Ein besonderer Dank geht an das Institut Berufliche Bildung Hammerbrookstrasse 47a und die Handelskammer Hamburg für die Möglichkeit, diese Ausbildung zu absolvieren und meine Fähigkeiten zu erweitern.

Ebenso möchte ich mich bei meinen Dozenten, Herrn Meyer, Gisela Neira Ahumada, Matthias Brinkmann und Tomas Krenz, bedanken, die mich durch ihre fachliche Kompetenz und ihre engagierte Betreuung stets motiviert haben.

Nicht zuletzt möchte ich mich bei meinem Ausbildungsbetrieb, der Grassau GmbH, bedanken. Hier konnte ich das Gelernte in der Praxis anwenden und vertiefen.

Vielen Dank an alle, die mich auf meinem Weg begleitet und unterstützt haben.

Ich möchte auch meine aufrichtige Dankbarkeit gegenüber der Praxis Dr. Jahnke und der Praxis Dr. Dr. Yagdiran & Kollegen zum Ausdruck bringen. Sie haben mich während meiner Ausbildung und darüber hinaus mit Rat und Tat unterstützt.

Ein besonderer Dank gebührt Frau Dr. Ristic, die während dieser Zeit der Entbehrung immer an meiner Seite stand und mir geholfen hat. Ihre Unterstützung war unerlässlich und ich bin sehr dankbar dafür.

Zu guter Letzt möchte ich mich auch bei mir selbst bedanken. Seit dem 2. Februar 2020 habe ich durchgehend hart gearbeitet und eine bemerkenswerte Ausdauer gezeigt. Ohne meinen eigenen Antrieb und meine Entschlossenheit hätte ich dieses Projekt nicht erfolgreich abschließen können.

Nochmals danke ich allen, die mich auf diesem Weg begleitet und unterstützt haben. Ihr Beitrag zu meiner Ausbildung und zu diesem Projekt ist unbezahlbar und wird immer geschätzt.

A54 Urheberrechtsangaben

Autor: Daniel-Alfonsin Massa Alle Rechte an dieser Dokumentation und der darin beschriebenen Parkhaus Simulationssoftware liegen bei Daniel-Alfonsin Massa und der Grassau GmbH. Eine Vervielfältigung oder Verwendung der Dokumentation oder der Software oder Teilen davon in anderen elektronischen oder gedruckten Publikationen ist ohne ausdrückliche Zustimmung des Autors und der Grassau GmbH nicht gestattet.

A55 Lizenzierung und Nutzungsbedingungen

Diese Dokumentation und die Parkhaus Simulationssoftware werden unter einer Lizenz veröffentlicht, die die Nutzung und Verbreitung der Software unter bestimmten Bedingungen erlaubt. Die genauen Nutzungsbedingungen und Einschränkungen sind in einem separaten Lizenzvertrag im Anhang enthalten und regeln die Rechte und Pflichten der Nutzer sowie des Autors, Daniel-Alfonsin Massa, und der Grassau GmbH.