Technische Dokumentation

Gruppe 3: inFlight System

Aktuelle Themen der IT – SoSe 25

Leonie Bretscher, Jonas Juston, Inàs Mountassir,

Oliver Simon, Vanessa Weber

Inhalt

1. Metadaten

Java-Version:

- Quell- und Zielkompilierung ist auf Java 23 gesetzt.

Abhängigkeiten:

- JavaFX Controls und FXML (für UI-Entwicklung), speziell mit macOS-spezifischem Classifier für javafx-controls.
- JUnit Jupiter für Unit-Tests.
- SQLite JDBC für Datenbankzugriff.
- Gson zur JSON-Verarbeitung.
- Mockito für das Mocking in Tests.
- SLF4J und Logback als Logging-Framework.

Build-Plugins:

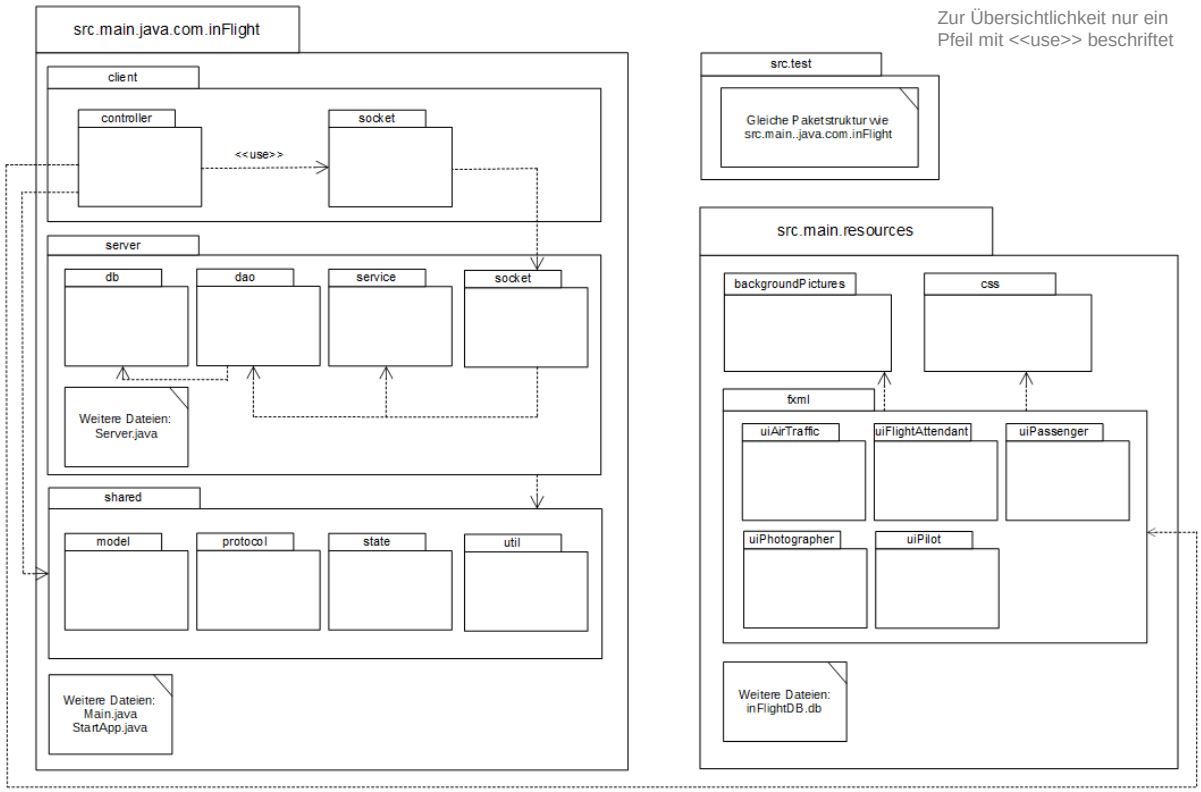
- maven-compiler-plugin steuert den Java-Compiler mit Version 23.
- javafx-maven-plugin ermöglicht das Ausführen der JavaFX-Anwendung direkt über Maven (mit Startklasse com.inFlight.StartApp).
- maven-surefire-plugin führt die Unit-Tests aus.

2. Installation

Clone das Repository *https://github.com/Solimon12/ATdIT2_IBAIT23_G3_InFlight* in IntelliJ und stelle sicher, dass alle Maven Sources korrekt geladen sind.

Starte die Klasse StartApp.java im Paket src.main.java.com.inFlight und starte dort zuerst den Server, indem der Server Button angeklickt wird.
Um den Demozustand zu erhalten, öffne anschließend zwei Passenger-Clients und jeweils einen Flight Attendant-, Photographer-, Pilot- und Air Traffic-Client durch das Betätigen der jeweiligen Buttons.

3. Projektstruktur



Hauptpaket src.main.com.inFlight:

Enthält die Hauptklasse StartApp und zentrale Logik.

Paket client:

controller: Steuerungsschicht für die GUI.

socket: Client-Server-Verbindung auf Client-Seite. Wird vom Controller genutzt, um Anfragen an den Server zu schicken und Antworten zu erhalten.

Paket server:

socket: Beinhaltet den ClientHandler für die Client-Server-Verbindung auf Server-Seite und den ProtocolHandler zur Verarbeitung von Socket-Anfragen.

service: Logikschicht - Der ProtocolHandler delegiert die Anfragen an den jeweiligen Service.

dao: Datenzugriffsschicht - Wird von der Serviceschicht genutzt, um auf die Datenbanktabellen zuzugreifen

db: Stellt die Connection zur SQLite Datenbank dar.

Paket shared:

model: Datenmodelle wie Photographer und InventoryItem.

protocol: Enthält eine Enum, die alle möglichen Aktionen im System definiert, und einen Nachrichtencontainer für den Austausch zwischen Client und Server

state: Wird zur Verwaltung von Zuständen, die global verfügbar sein müssen, verwendet. Sie sind als Singleton-Klassen implementiert, damit der Zustand zentral und konsistent bleibt.

util: Basierend auf der Rolle eines Benutzers wird die entsprechende GUI-Ansicht geladen und angezeigt. Es wird jeder Rolle eine spezifische FXML-Datei zugeordent und öffnet ein neues Fenster mit der entsprechenden Benutzeroberfläche.

Paket src.main.resources:

backgroundPictures: Enthält die Hintergrundbilder der Clients.

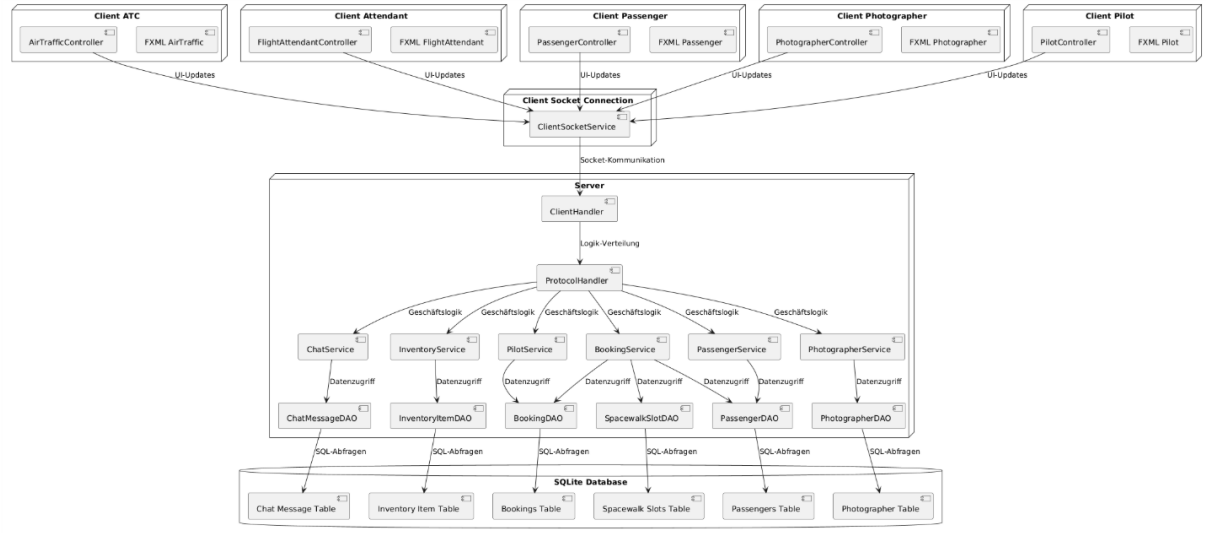
css: Enthält die css-Stylesheets, um das Hintergrundbild des jeweiligen Clients zu setzen.

fxml (view): Enthält die FXML-Dateien der Clients.

Paket src.test:

Enthält die gleiche Struktur wie das Hauptpaket. Hier befinden sich die Unittests der Klassen.

4. Client-Server Struktur



Die Architektur ist ein Client-Server-System, das über Sockets kommuniziert. Es enthält verschiedene Rollen (Air Traffic Controller (ATC), Flugbegleiter, Passagier, Fotograf und Pilot).

Das System ist in mehrere Schichten aufgeteilt:
• Die Präsentationsschicht besteht aus den FXML-Views und den Controller-Klassen der Clients.
• Die Kommunikationsschicht wird auf Client-Seite vom ClientSocketService und auf Server-Seite vom ClientHandler übernommen.
• Die Geschäftslogikschicht liegt in den Service-Klassen auf dem Server.
• Die Datenzugriffsschicht besteht aus den DAO-Klassen, die mit der SQLite-Datenbank interagieren.

Client und Server tauschen JSON-Nachrichten aus, die über eine TCP-Socketverbindung geschickt werden.

Client-Struktur

Jede Benutzerrolle hat einen eigenen JavaFX-Client, der aus zwei Hauptteilen besteht:
• Controller: Steuerung der UI
• FXML-View: fxml-Dateien,mit denen der User interagiert

Diese Clients tauschen über eine zentrale Komponente (ClientSocketService) Nachrichten mit dem Server aus. Der ClientSocketService ist dafür zuständig, JSON-Nachrichten über eine Socket-Verbindung zu senden und zu empfangen. Alle Clients können Antworten vom Server empfangen, um ihre Benutzeroberflächen dynamisch anzupassen.

Server-Komponente

Der Server nimmt Verbindungen von Clients entgegen und erstellt für jede Verbindung eine ClientHandler-Instanz. Diese ist für die Kommunikation mit dem jeweiligen Client zuständig und leitet empfangene Nachrichten an den ProtocolHandler weiter.

Der ProtocolHandler sorgt dafür, dass die Nachrichten an die richtigen Services weitergeleitet werden. Die spezialisierten Service-Klassen enthalten die Geschäftslogik:
• ChatService
• InventoryService
• PilotService
• BookingService
• PassengerService
• PhotographerService

Jeder dieser Services ist für einen bestimmten Funktionsbereich der Anwendung verantwortlich.

Datenzugriff und Persistenz

Für den Zugriff auf die Datenbank wird das Data Access Object (DAO) Pattern verwendet. In den DAO-Klassen sollen nur CRUD-Methoden stehen. Zur Demo wurde sich jedoch für eine Vereinfachung davon entschieden. Jede Service-Klasse nutzt eine passende DAO-Klasse, um Daten aus der Datenbank abzufragen oder zu speichern. Die DAO-Klassen interagieren mit einer SQLite-Datenbank und enthalten alle nötigen SQL-Abfragen.

Die Datenbank besteht aus folgenden Tabellen:
• Chat Message Table: genutzt von ChatMessageDAO
• Inventory Item Table: genutzt von InventoryItemDAO
• Booking Table: genutzt von BookingDAO
• Spacewalk Slots Table: genutzt von SpacewalkSlotDAO
• Passengers Table: genutzt von PassengerDAO
• Photographer Table: genutzt von PhotographerDAO

In der Datenbank werden alle dauerhaften Daten gespeichert, die die Anwendung benötigt.

5. Klassendiagramme

5.1. Client

5.1.1. Controller

Die Controller-Klassen im Projekt sind dafür verantwortlich, die Logik zwischen der Benutzeroberfläche (FXML-Ansichten) und der zugrunde liegenden Geschäftslogik zu verwalten. Sie reagieren auf Benutzerinteraktionen, steuern die Navigation zwischen verschiedenen Ansichten und kommunizieren mit Diensten wie dem ClientSocketService, um Daten zu senden oder zu empfangen:

RoleSelectorController:

- Öffnet Fenster für verschiedene Benutzerrollen.
- Startet den Server und deaktiviert den Button, um Mehrfachklicks zu verhindern.

AirTrafficController:

- Kommunikation mit Piloten, regelmäßige Aktualisierung und Anzeige neuer Nachrichten.

AttendantController:

- Anzeigen, Verwalten und Check-In/Check-Out von Passagieren.
- Anzeigen, Ausleihen, Rückgabe und Wartung von Ausrüstung.
- Chat mit dem Piloten und Anzeigen neuer Nachrichten.

PassengerController:

- Spacewalks buchen, stornieren und Status prüfen.
- Guthaben anzeigen, aufladen und Zahlungen verwalten.
- Anzeigen von Nachrichten vom Piloten.
- Überprüfen und Weiterleiten zu Umfragen.

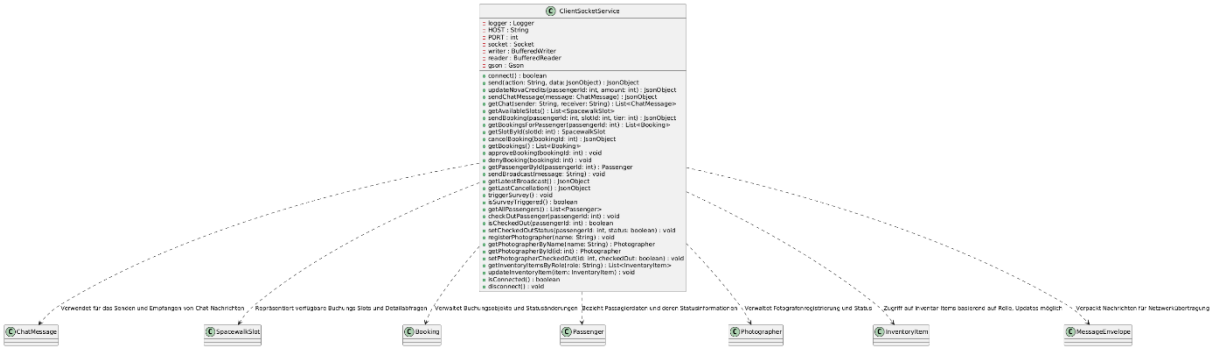
PilotController:

- Anzeigen und Verwalten von Buchungen.
- Kommunikation mit der Flugverkehrskontrolle (ATC) und Flugbegleitern über Chats.
- Senden von Broadcast-Nachrichten an Passagiere.
- Überwachung von Stornierungen und neuen Buchungen.

PhotographerController:

- Anzeigen und Verwalten von Inventar und Wartungsaufgaben.
- Überprüfen und Verwalten von Buchungen für Spacewalks.
- Initialisierung von Dashboards und Bildschirmen basierend auf dem Status des Fotografen.

5.1.2. Socket



Die Klasse ClientSocketService ist eine Service-Klasse, die die Kommunikation zwischen dem Client und dem Server über Sockets verwaltet. Sie stellt Methoden bereit, um Anfragen an den Server zu senden, Antworten zu empfangen und Daten zu verarbeiten. Die Klasse verwendet JSON-Serialisierung und -Deserialisierung, um die Kommunikation zu standardisieren.

Verbindungsmanagement:

- Die Methode connect() baut eine Verbindung zum Server auf, indem sie einen Socket erstellt und die Ein- und Ausgabeströme initialisiert.
- Die Methode disconnect() schließt die Verbindung zum Server und gibt die Ressourcen frei.

Datenübertragung:

- Die Methode send(String action, JsonObject data) sendet eine Anfrage an den Server. Die Anfrage wird in ein JSON-Format serialisiert, und die Antwort wird ebenfalls als JSON-Objekt verarbeitet.

Chat-Funktionalität:

- Nachrichten können mit der Methode sendChatMessage(ChatMessage message) gesendet werden.
- Die Methode getChat(String sender, String receiver) ruft alle Nachrichten zwischen zwei Benutzern ab.

Spacewalk-Buchungen:

- Methoden wie sendBooking(int passengerId, int slotId, int tier) ermöglichen das Buchen von Spacewalk-Slots.
- getAvailableSlots() ruft verfügbare Slots ab.
- cancelBooking(int bookingId) storniert eine Buchung

Broadcasts und Umfragen:

- Mit sendBroadcast(String message) können Nachrichten an alle Passagiere gesendet werden.
- Die Methode triggerSurvey() löst eine Umfrage aus.
- isSurveyTriggered() prüft, ob eine Umfrage aktiv ist.

Inventar- und Passagiermanagement:

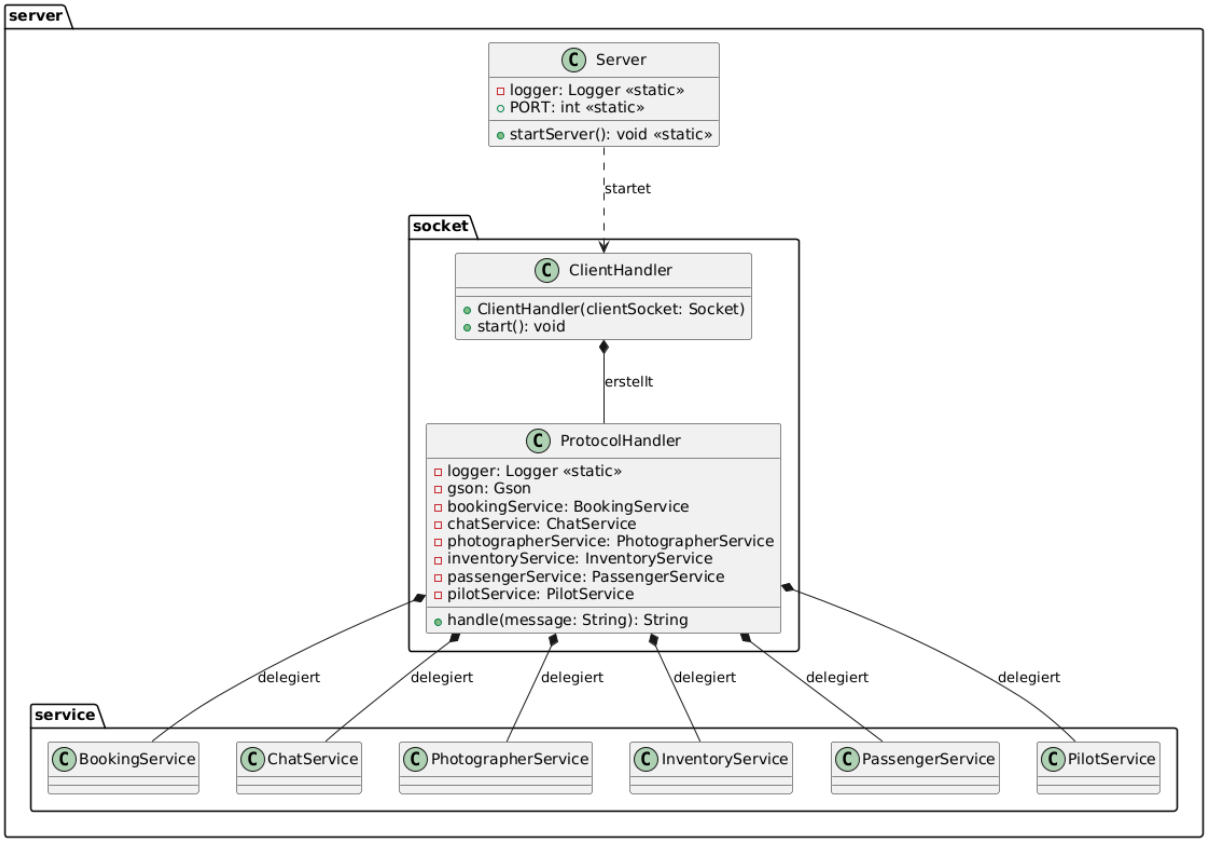
- getAllPassengers() ruft eine Liste aller Passagiere ab.
- getInventoryItemsByRole(String role) liefert Inventargegenstände basierend auf der Rolle.

Fehlerbehandlung:

- Bei Verbindungsproblemen oder fehlerhaften Anfragen werden Fehlerprotokolle erstellt und JSON-Fehlermeldungen zurückgegeben.

5.2. Server

5.2.1. Socket



Server:

Die Klasse Server ist für das Starten des Servers und das Akzeptieren eingehender Client-Verbindungen verantwortlich. Sie erstellt für jede Verbindung einen neuen Thread, um die Kommunikation mit dem Client zu handhaben.

ClientHandler:

Die Klasse ClientHandler verwaltet die Kommunikation mit einem verbundenen Client. Sie liest Anfragen vom Client, verarbeitet diese mit einem ProtocolHandler und sendet Antworten zurück.

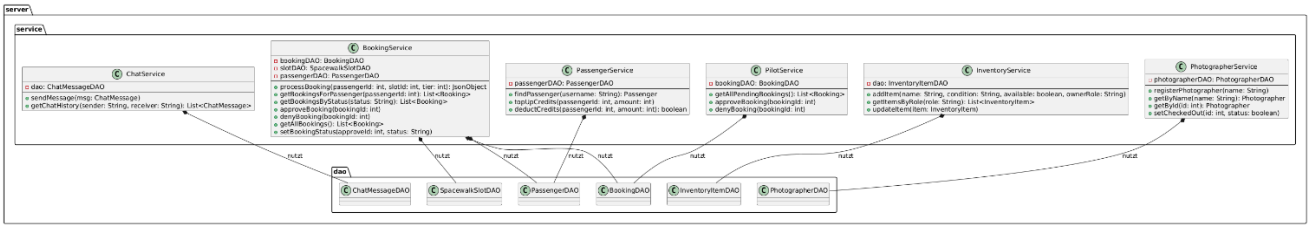
ProtocolHandler:

Die Klasse ProtocolHandler verarbeitet eingehende Nachrichten basierend auf dem ActionType (Enum in shared.protocol). Sie verwendet verschiedene Services, um die Anfragen zu bearbeiten und Antworten zu generieren.

- Deserialisiert eingehende Nachrichten und identifiziert die Aktion.

- Führt die entsprechende Logik basierend auf dem ActionType aus (z. B. Login, Buchungen, Chat-Nachrichten).
- Serialisiert die Antwort in JSON-Format und gibt sie zurück.

5.2.2. Service



Service-Klassen kapseln die Geschäftslogik der Anwendung und dienen als Vermittler zwischen den Datenzugriffsschichten (DAOs) und der Anwendungsschicht. Sie stellen Methoden bereit, um spezifische Operationen auszuführen und werden vom ProtocolHandler aufgerufen.

BookingService:

Verwaltet Spacewalk-Buchungen, einschließlich der Erstellung, Genehmigung, Ablehnung und Abfrage von Buchungen. Er prüft auch die Verfügbarkeit von Slots und aktualisiert den NovaCredits-Stand der Passagiere.

ChatService:

Verarbeitet chatbezogene Operationen, wie das Senden von Nachrichten und das Abrufen des Chatverlaufs zwischen zwei Benutzern.

InventoryService:

Verwaltet inventarbezogene Operationen, wie das Hinzufügen, Abrufen und Aktualisieren von Inventargegenständen basierend auf der Rolle des Besitzers.

PassengerService:

Verwaltet passagierbezogene Operationen, wie das Finden von Passagieren, das Aufladen oder Abziehen von NovaCredits und die Verwaltung von Passagierdaten.

PhotographerService:

Verwaltet Fotografen, einschließlich Registrierung, Abruf von Fotografen nach Name oder ID und Verwaltung ihres Check-out-Status.

PilotService

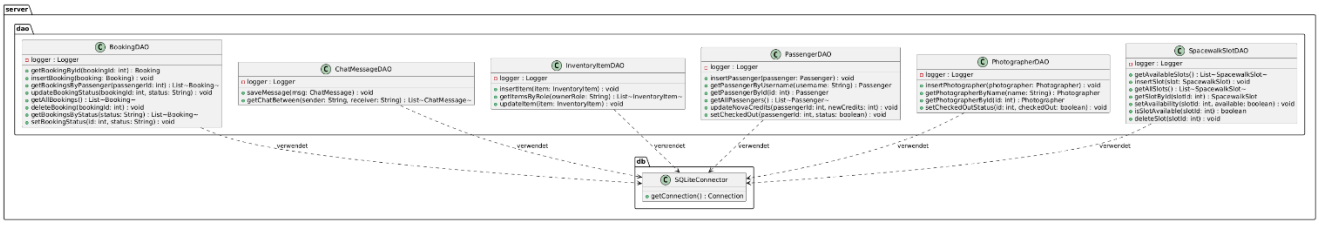
Verwaltet pilotbezogene Aufgaben, wie das Abrufen ausstehender Buchungen und das Genehmigen oder Ablehnen von Buchungen.

Weitere Dependencies zu shared.model:

- BookingService nutzt Booking, SpacewalkSlot und Passenger.
- ChatService nutzt ChatMessage.
- InventoryService nutzt InventoryItem.
- PassengerService nutzt Passenger.

- PhotographerService nutzt Photographer.
- PilotService nutzt Booking.

5.2.3. DAO & DB



Die Klasse SQLiteConnector ist eine Hilfsklasse, die eine Verbindung zur SQLite-Datenbank herstellt. Sie verwendet den JDBC-Treiber, um eine Verbindung zu einer Datenbankdatei herzustellen, deren Pfad in der Klasse definiert ist.

DAO-Klassen (Data Access Objects) kapseln den Zugriff auf die Datenbank. Sie bieten Methoden, um CRUD-Operationen (Create, Read, Update, Delete) auf spezifischen Tabellen auszuführen (zur Demo hier vereinfacht).

BookingDAO:

Verwaltet den Zugriff auf die bookings-Tabelle. Bietet Methoden zum Einfügen, Abrufen, Aktualisieren und Löschen von Buchungen sowie zum Filtern nach Status oder Passagier.

InventoryItemDAO:

Verwaltete den Zugriff auf die inventory_item-Tabelle. Bietet Methoden zum Hinzufügen, Abrufen und Aktualisieren von Inventargegenständen basierend auf deren Besitzerrolle.

ChatMessageDAO:

Verwaltete den Zugriff auf die chat_message-Tabelle. Bietet Methoden zum Speichern von Chat-Nachrichten und Abrufen des Chatverlaufs zwischen zwei Benutzern.

PassengerDAO:

Verwaltet den Zugriff auf die passengers-Tabelle. Bietet Methoden zum Abrufen, Aktualisieren und Verwalten von Passagierdaten.

PhotographerDAO:

Verwaltete den Zugriff auf die photographer-Tabelle. Bietet Methoden zum Hinzufügen, Abrufen und Aktualisieren von Fotografen, einschließlich ihres Check-out-Status.

SpacewalkSlotDAO:

Verwaltete den Zugriff auf die spacewalk_slots-Tabelle. Bietet Methoden zum Verwalten von Spacewalk-Slots, wie das Abrufen verfügbarer Slots, das Aktualisieren ihrer Verfügbarkeit und das Löschen von Slots.

Weitere Dependencies in shared.model:

BookingDAO nutzt Booking

ChatMessageDAO nutzt ChatMessage

InventoryItemDAO nutzt InventoryItem

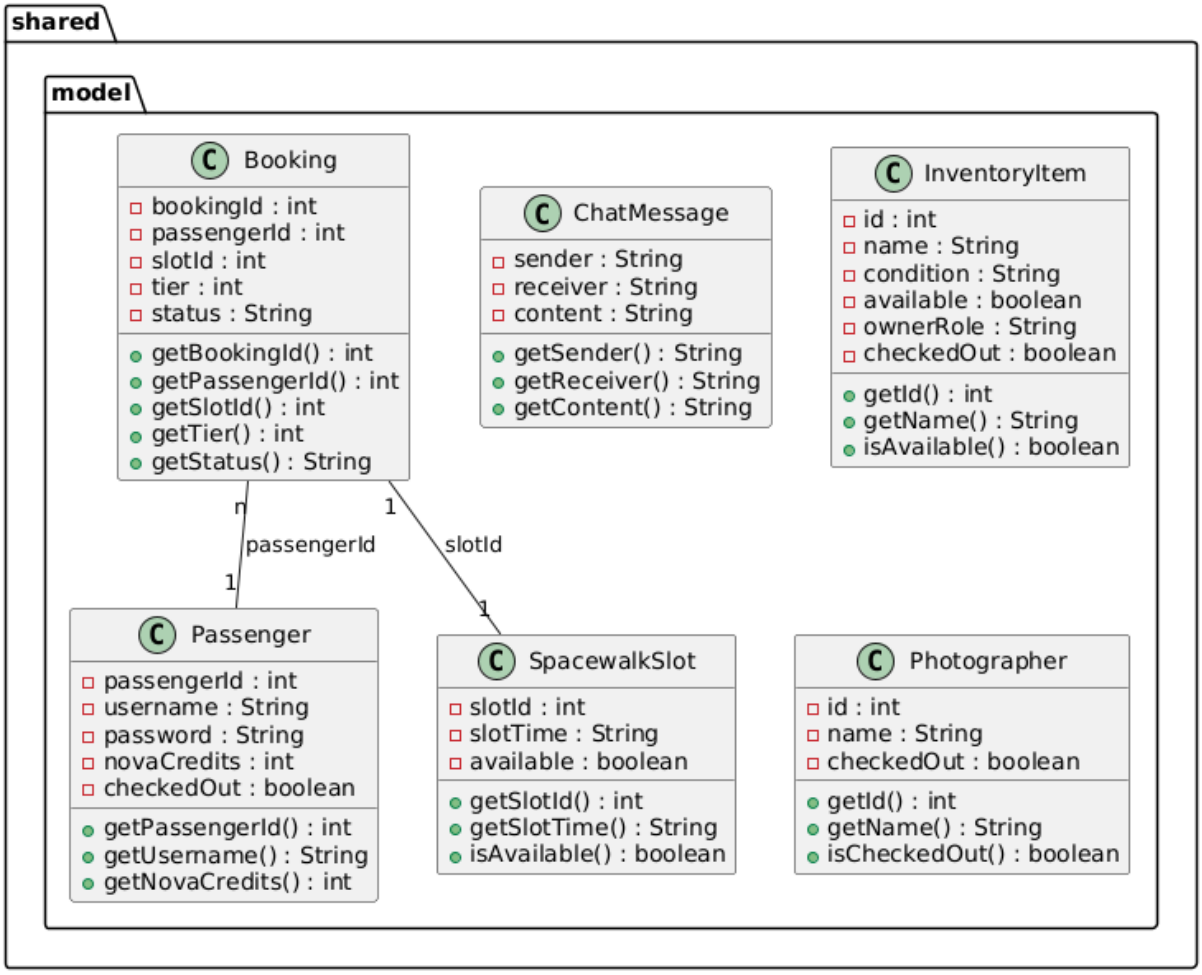
PassengerDAO nutzt Passenger

PhotographerDAO nutzt Photographer

SpacewalkSlotDAO nutzt SpacewalkSlot

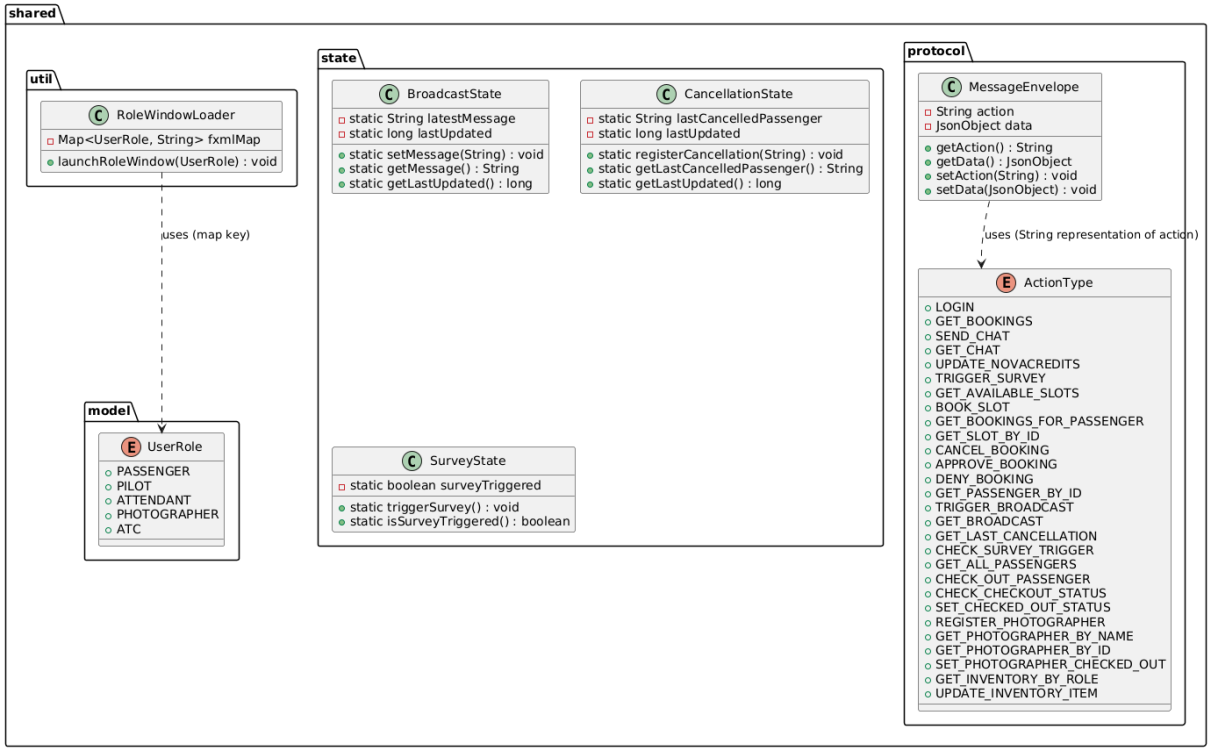
5.3. Shared

5.3.1. Model



Die Shared-Model-Klassen dienen als POJOs (Plain Old Java Objects) und enthalten Attribute sowie Getter- und Setter-Methoden, um die Daten zu persistieren. Diese Klassen werden verwendet, um Daten zwischen verschiedenen Schichten der Anwendung zu übertragen.

5.3.2. State, util & protocol



State:

Die drei Klassen im Paket shared.state (BroadcastState, CancellationState, SurveyState) werden verwendet, um globale Zustände innerhalb der Anwendung zu verwalten und sind als Singleton-Klassen implementiert.

BroadcastState:

Verwalten der letzten Broadcast-Nachricht und ihres Zeitstempels.

CancellationState:

Speichern des Namens des zuletzt stornierten Passagiers und des Zeitpunkts der Stornierung.

SurveyState:

Verwalten des Zustands, ob eine Umfrage ausgelöst wurde.

Util:

RoleWindowLoader:

Die Klasse ist dafür verantwortlich, basierend auf der Rolle eines Benutzers (UserRole), die entsprechende FXML-Datei zu laden und das passende Fenster zu starten.

UserRole:

Die Enumeration definiert die verschiedenen Benutzerrollen in der Anwendung.

Protocol:

MessageEnvelope:

Die Klasse repräsentiert einen Nachrichten-Container, der in der Anwendung verwendet wird. Sie enthält eine Aktion (action) und die zugehörigen Daten (data) im JSON-Format.

ActionType:

Die Enumeration definiert die verschiedenen Aktionen, die in der Anwendung ausgeführt werden können.

6. ActionTypes

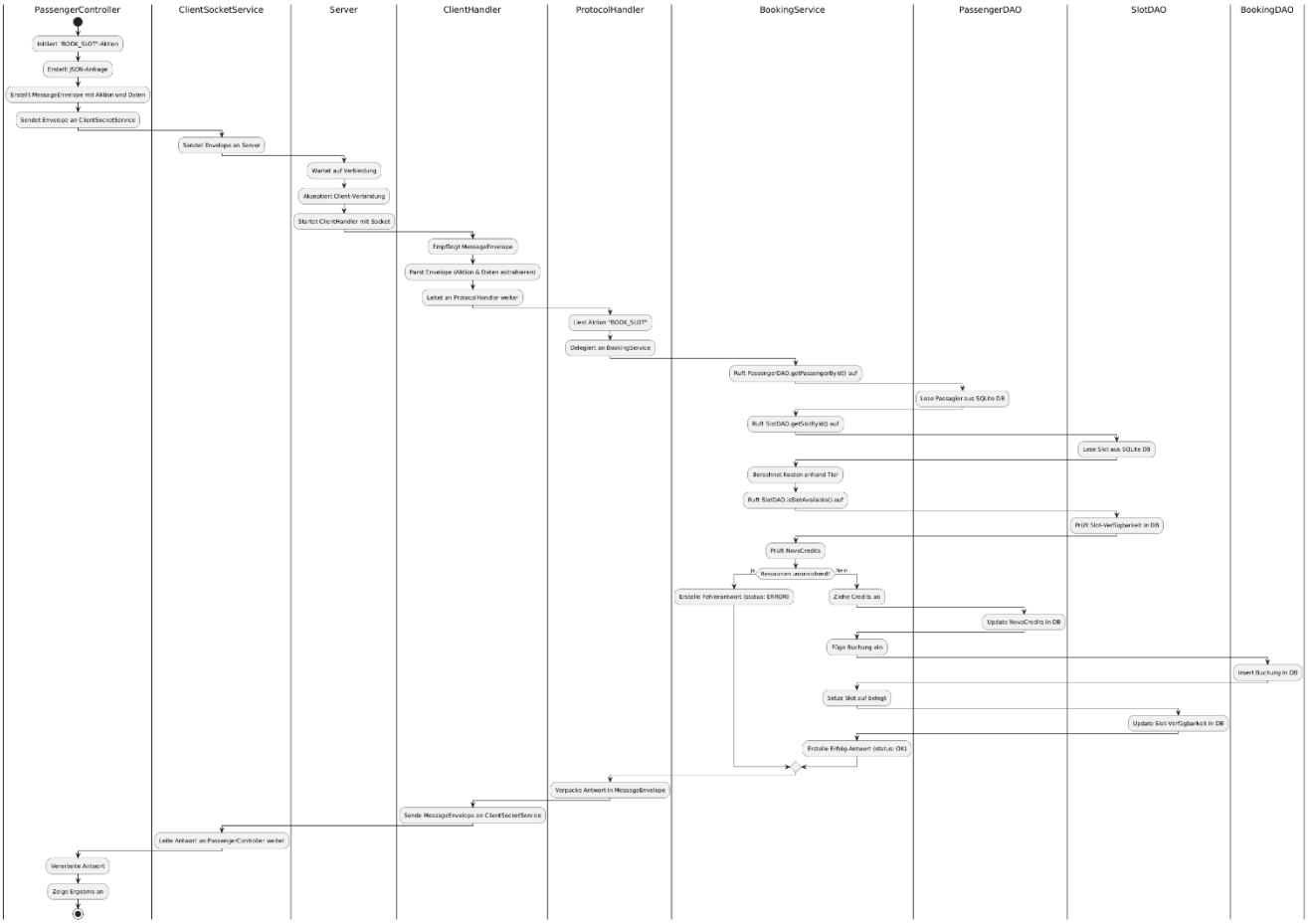
Die ActionType-Enum wird im Projekt verwendet, um verschiedene Aktionen festzulegen, die zwischen Client und Server ausgetauscht werden können. Sie ist eine zentrale Liste von Befehlen, die der Server versteht, und sorgt für eine klare und strukturierte Kommunikation im Protokoll.

Das Aktivitätsdiagramm zeigt den Ablauf eines serverseitigen Verarbeitungsprozesses im Client-Server-System. Es veranschaulicht, wie ein Client eine Aktion startet, diese über eine Socketverbindung an den Server schickt und wie der Server sie verarbeitet.
Der Server liest die Nachricht, leitet sie an die passende Geschäftslogik (Service-Klasse) weiter, die dann mithilfe von DAOs auf die SQLite-Datenbank zugreift. Am Ende wird das Ergebnis wieder an den Client geschickt.

Der Ablauf durchläuft folgende Hauptkomponenten:

- **ClientController & ClientSocketService**:
  Erstellung und Versand einer JSON-Nachricht.
- **Server & ClientHandler**:
  Verbindungsannahme, Entgegennahme der Nachricht, Weiterleitung an die Geschäftslogik.
- **ProtocolHandler**:
  Verteilung der Nachricht auf die zuständige Service-Klasse.
- **Service-Klasse**:
  Verarbeitet die Anfrage und ruft DAOs zur Datenmanipulation auf.
- **DAOs & Datenbank**:
  Datenbankzugriffe (CRUD).
- **Antwortversand an den Client**:
  Rückgabe des Ergebnisses an den Client.

6.1. BOOK_SLOT



Der Passagier startet die Aktion BOOK_SLOT im Client, woraufhin eine JSON-Nachricht über den ClientSocketService an den Server gesendet wird. Der Server nimmt die Verbindung entgegen, erstellt einen ClientHandler, der die Nachricht an den ProtocolHandler weiterleitet.

Der ProtocolHandler erkennt die Aktion und delegiert sie an den BookingService. Dieser verifiziert den Passagier, prüft die Slot-Verfügbarkeit, berechnet den Preis und das Guthaben. Ist genug Guthaben vorhanden, wird der Betrag abgezogen, der Slot reserviert, die Buchung gespeichert und eine Bestätigung erzeugt. Andernfalls wird eine Fehlermeldung erstellt.

Die Antwort wird an den Client zurückgesendet, wo sie im PassengerController verarbeitet und dem Benutzer angezeigt wird.

6.2. LOGIN

- Ruft den PassengerService auf
- Verifiziert, ob der Passagier existiert
- Genehmigt die Anmeldung oder zeigt eine Fehlermeldung an

6.3. GET_BOOKINGS

- Ruft den BookingService auf
- Liest alle Buchungen aus der Datenbank aus

6.4. SEND_CHAT

- Ruft den ChatService auf
- Schreibt die neue Nachricht in die Datenbank

6.5. GET_CHAT

- Ruft den ChatService auf
- Liest alle Nachrichten zwischen zwei Rollen aus

6.6. UPDATE_NOVACREDITS

- Ruft den PassengerService auf
- Liest den Passagier anhand der Id aus der Datenbank aus
- Passt den NovaCredits-Kontostand an

6.7. TRIGGER_SURVEY

- Setzt den SurveyState auf true

6.8. GET_AVAILABLE_SLOTS

- Ruft den BookingService auf
- Liest alle verfügbaren Zeitslot aus der Datenbank aus

6.9. GET_BOOKINGS_FOR_PASSENGER

- Ruft den BookingService auf
- Liest alle Buchungen eines Passagiers anhand seiner Id aus

6.10. GET_SLOT_BY_ID

- Ruft den BookingService auf
- Liest alle Zeitslots für eine Slot-Id aus

6.11. CANCEL_BOOKING

- Ruft den BookingService auf
- Setzt den stornierten Zeitslot auf verfügbar
- Updated die NovaCredits des Passagiers
- Löscht die Buchung
- Trägt die Stornierung in CancellationState ein

6.12. APPROVE_BOOKING

- Ruft den BookingService auf
- Setzt den Status der Buchung auf genehmigt

6.13. DENY_BOOKING

- Ruft den BookingService auf
- Setzt den abgelehnten Zeitslot auf verfügbar
- Updated die NovaCredits des Passagiers
- Löscht die Buchung

6.14. GET_PASSENGER_BY_ID

- Ruft den PassengerService auf
- Liest den Passagier anhand der Id aus der Datenbank aus

6.15. TRIGGER_BROADCAST

- Trägt die Broadcast-Nachricht in den BroadcastState ein

6.16. GET_BROADCAST

- Liest die aktuelle Broadcast-Nachricht aus BroadcastState aus

6.17. GET_LAST_CANCELLATION

- Liest die aktuelle Stornierung aus CancellationState aus

6.18. CHECK_SURVEY_TRIGGER

- Liest aus SurveyState aus, ob die Umfrage für die Passagiere gestartet wurde

6.19. GET_ALL_PASSENGERS

- Ruft den PassengerService auf
- Liest alle Passagiere aus der Datenbank aus

6.20. CHECK_OUT_PASSENGER

- Ruft den PassengerService auf
- Ändert den Check-Out-Status des Passagiers in der Datenbank

6.21. CHECK_CHECKOUT_STATUS

- Ruft den PassengerService auf
- Liest den Check-Out-Status des Passagiers aus der Datenbank

6.22. SET_CHECKED_OUT_STATUS

- Ruft den PassengerService auf
- Ändert den Check-Out-Status des Passagiers in der Datenbank

6.23. REGISTER_PHOTOGRAPHER

- Ruft den PhotographerService auf
- Schreibt den neuen Photographen in die Datenbank

6.24. GET_PHOTOGRAPHER_BY_NAME

- Ruft den PhotographerService auf
- Liest den Photographen anhand des Namens aus der Datenbank

6.25. GET_PHOTOGRAPHER_BY_ID

- Ruft den PhotographerService auf
- Liest den Photographen anhand der Id aus der Datenbank

6.26. SET_PHOTOGRAPHER_CHECKED_OUT

- Ruft den PhotographerService auf
- Ändert den Check-Out-Status des Photographers in der Datenbank

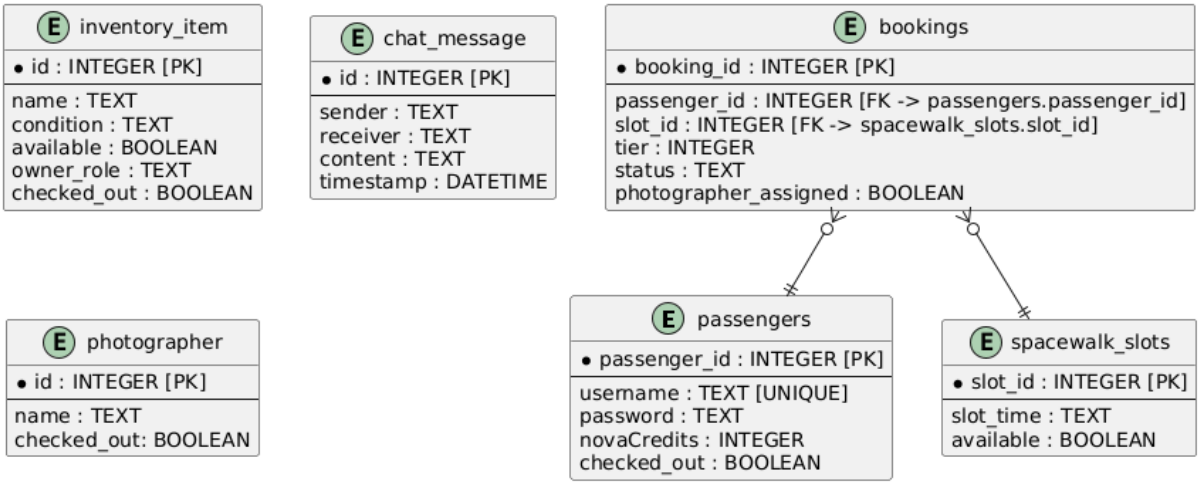
6.27. GET_INVENTORY_BY_ROLE

- Ruft den InventoryService auf
- Liest die InventoryItems aus der Datenbank, die der jeweiligen Rolle zugeordnet sind

6.28. UPDATE_INVENTORY_ITEM

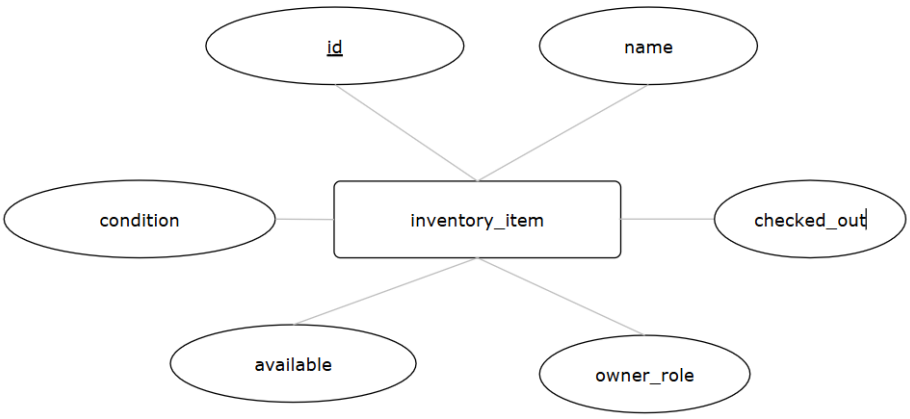
- Ruft den InventoryService auf
- Passt den Inventory-Eintrag in der Datenbank an

7. Datenbank



Die Datenbank besteht aus 6 Tabellen zur Verwaltung der dauerhaften Daten, die in der Anwendung genutzt werden.
Inventory_item, chat_message und photographer sind alleinstehende Tabellen ohne Beziehungen zu anderen Tabellen.
Bookings stellt die SpaceWalk-Buchungen dar, die jeweils aus einem passenger und einem spacewalk_slot bestehen.

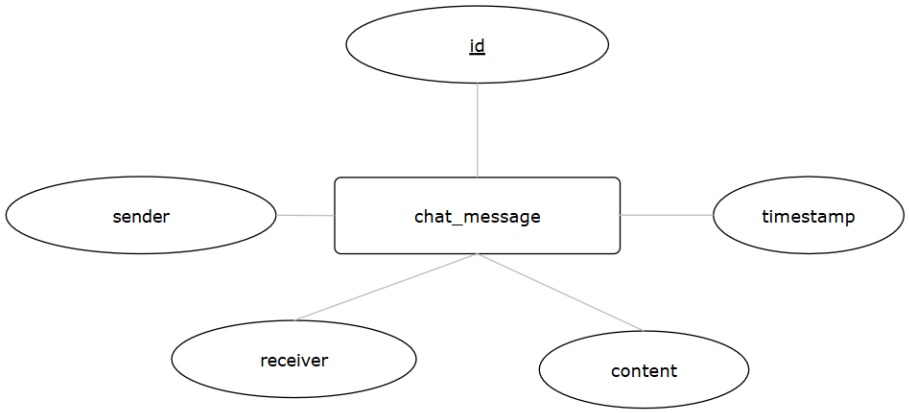
7.1. Inventory_item



In dieser Tabelle wird die Ausrüstung, die in den Inventaren des Photographen und Flugbegleiters angezeigt werden sollen, gespeichert. Ein Inventareintrag besteht aus einer eindeutigen ID, der Bezeichnung der Ausrüstung (name), dem Zustand der Ausrüstung (condition, z.B. „Defekt“, „Überprüft“), einer Verfügbarkeit (available, ist das Ausrüstungsstück gerade für die Nutzung vorhanden), einem Check-Out-Status (checked_out, wurde das Ausrüstungsstück für die Nutzung aus dem Inventar entnommen) und einer Rolle, der es zugeordnet ist (owner_role, z.B. Photographer, Attendant).

So können alle Ausrüstungsgegenstädne gespeichert werden und klar, dem richtigen Inhaber / Verwalter zugeordnet werden.

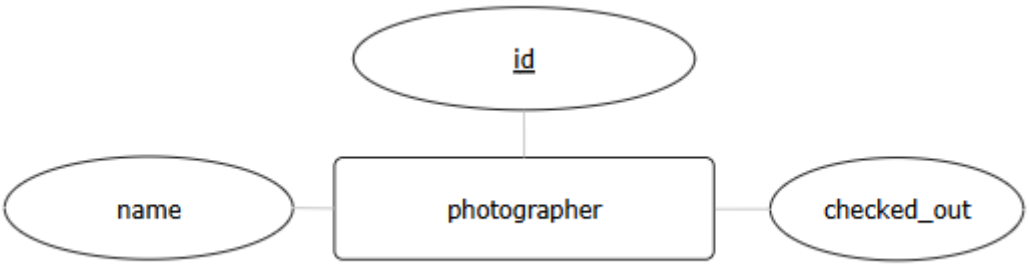
7.2. chat_message



In dieser Tabelle werden die Chat-Nachrichten, die in der Anwendung versendet wurden, gespeichert. Ein Chat-Eintrag besteht aus einer eindeutigen ID, dem Absender der Nachricht (sender), dem Empfänger der Nachricht (receiver), einem Zeitstempel (timestamp, wann die Nachricht versandt wurde) und dem Nachrichteninhalt, der übermittelt werden soll (content).

So können alle Chat-Nachrichten gespeichert werden und klar, dem richtigen Absender und Empfänger zugeordnet werden.

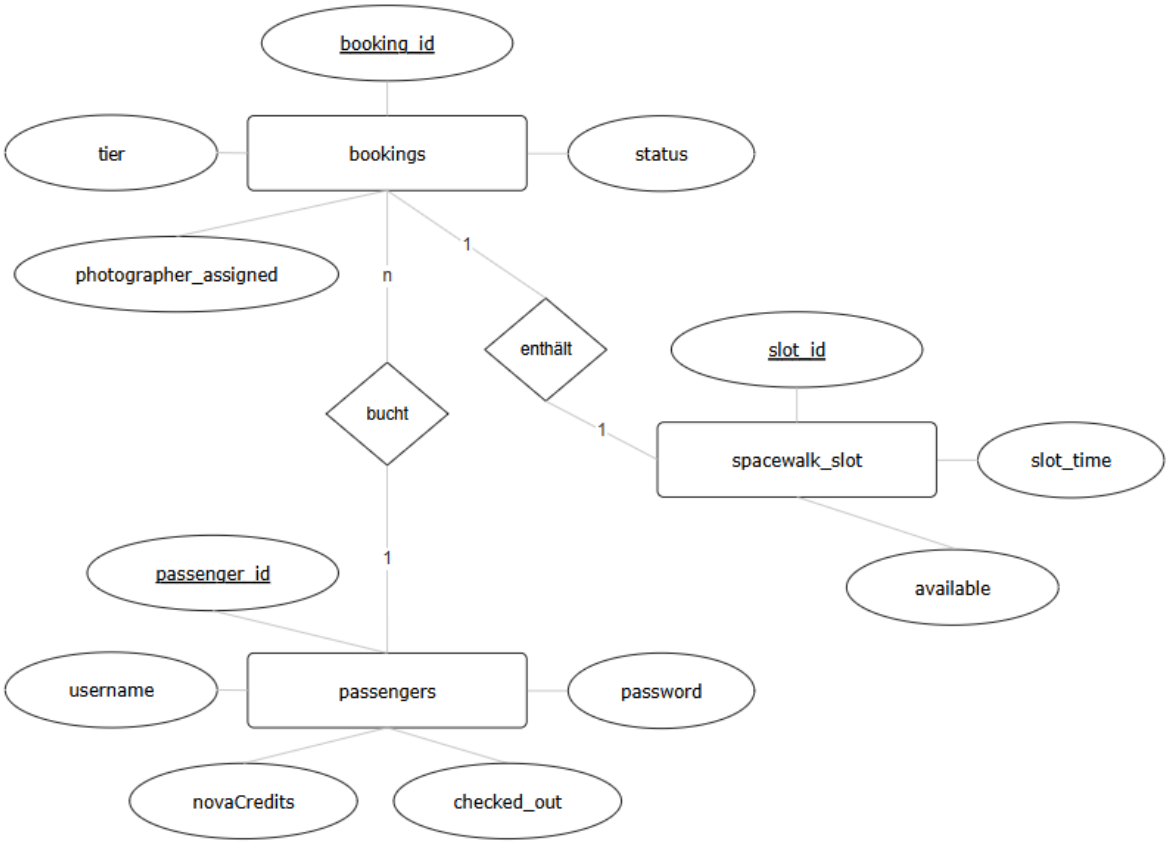
7.3. photographer



In dieser Tabelle werden die Photographer, die am Flug beteiligt sind, gespeichert. Ein Photograph-Eintrag besteht aus einer eindeutigen ID, dem Namen des Photographen (name) und seinem Check-Out-Status (checked_out, befindet er sich gerade im Raumschiff oder außerhalb).

So können alle Photographen und ihr jeweiliger Status gespeichert werden.

7.4. bookings, passengers & spacewalk_slots



In der Tabelle passengers werden die Passagiere, die am Flug teilnehmen, gespeichert. Ein Passagier-Eintrag besteht aus einer eindeutigen ID (passenger_id), dem Namen des Passagiers, mit dem er sich auch für das Aufladen der NovaCredits anmeldet (username), dem Passwort, mit dem er sich auch für das Aufladen der NovaCredits anmeldet (password), dem aktuellen Kontostand der In-Flight-Currency (novaCredits) und seinem Check-Out-Status (checked_out, befindet er sich gerade im Raumschiff oder außerhalb).

In der Tabelle spacewalk_slot werden die Zeitslots für den SpaceWalk, die ein Passagier als Aktivität buchen kann, gespeichert. Ein Spacewalk-Slot-Eintrag besteht aus einer eindeutigen ID (slot_id), der Zeit, zu der der SpaceWalk stattfinden soll (slot_time) und der Verfügbarkeit des Zeitslots (available, wurde der Zeitslot bereits von einem Passagier gebucht).

In der Tabelle bookings werden die Buchungen für den SpaceWalk, die ein Passagier als Aktivität buchen kann, gespeichert. Ein Buchungs -Eintrag besteht aus einer eindeutigen ID (booking_id), dem Pakettyp, den der Kunde gebucht hat (tier, entspricht Essential, Comfort & Prestige), dem Status (status, z.B. „bestätigt“, „abgelehnt“) und der Information, ob ein Photograph benötigt wird (photographer_assigned, bei Prestige-Buchungen nötig).

Eine Buchung enthält einen Passagier, der die Buchung getätigt hat, und einen Zeitslot, zu dem die Aktivität ausgeführt werden soll. Ein Zeitslot kann nur Teil einer Buchung sein, da er danach vergeben ist, ein Passagier kann aber mehrere Buchungen durchführen.

So können alle Buchungen mit all ihren Details klar strukturiert und gespeichert werden.