Projektkonzeption: Panda Spa Management System

Das Projekt Panda Spa ist eine Python-basierte Anwendung zur Verwaltung eines fiktiven Thermalwasser-Wellnesszentrums im Bambuswald, das von einem Panda betrieben wird. Ziel der Anwendung ist es, typische Abläufe eines Spa-Betriebs softwareseitig abzubilden und dabei moderne Konzepte der Softwareentwicklung sowie fortgeschrittene Python-Techniken einzusetzen. Der Fokus liegt auf der strukturierten Modellierung von Spa-Dienstleistungen, der Terminverwaltung für Kunden sowie der Erfassung von Einnahmen und Ausgaben.

Die Anwendung ermöglicht es, verschiedene Spa-Dienstleistungen wie Thermalbäder, Massagen oder Tee-Therapien zu definieren und zu verwalten. Jede Dienstleistung besitzt spezifische Eigenschaften, beispielsweise eine Behandlungsdauer, eine Temperatur oder einen Preis. Um sicherzustellen, dass sicherheitsrelevante und organisatorische Vorgaben eingehalten werden, werden diese Eigenschaften zur Laufzeit automatisch validiert. Hierbei kommen Python-Deskriptoren zum Einsatz, die es erlauben, Geschäftsregeln wie maximale Wassertemperaturen oder zulässige Behandlungszeiten deklarativ festzulegen, ohne die Hauptlogik der Anwendung mit expliziten Prüfungen zu überladen.

Ein weiterer zentraler Bestandteil des Projekts ist das Termin- und Buchungssystem. Kunden – dargestellt durch verschiedene Waldtiere – können Spa-Dienstleistungen zu bestimmten Zeiten buchen. Das System stellt sicher, dass Termine korrekt angelegt werden, keine unzulässigen Überschneidungen auftreten und die gewählten Dienstleistungen den definierten Rahmenbedingungen entsprechen. Die Buchungslogik ist klar strukturiert und wird durch Unit- und Integrationstests abgesichert, um die Zuverlässigkeit der Anwendung zu gewährleisten.

Zusätzlich erfasst das System finanzielle Aspekte des Spa-Betriebs. Einnahmen aus gebuchten Dienstleistungen sowie laufende Ausgaben, beispielsweise für Wasser, Tee oder andere Betriebsmittel, werden dokumentiert und ausgewertet. Auf dieser Basis kann der wirtschaftliche Erfolg des Spa-Betriebs nachvollzogen werden. Die Finanzlogik ist modular aufgebaut und wird ebenfalls durch Tests validiert.

Ein besonderer Schwerpunkt des Projekts liegt auf dem wissenschaftlichen Thema Metaprogrammierung und Introspection in Python. Durch den Einsatz von Deskriptoren und optionalen Metaklassen wird eine deklarative Systemarchitektur realisiert, die Geschäftsregeln automatisiert durchsetzt und die Erweiterbarkeit der Anwendung verbessert. Dieser Ansatz fördert sauberen, wartbaren Code und demonstriert den praktischen Nutzen fortgeschrittener Python-Sprachkonzepte.

Das Projekt wird gemäß gängiger Software-Engineering-Prinzipien umgesetzt und umfasst eine vollständige Dokumentation, Software Requirements, UML-Diagramme (z. B. Use-Case- und Klassendiagramme), eine ausführliche Readme-Datei sowie eine definierte CI-Pipeline. Die Anwendung wird schrittweise entwickelt, beginnend mit einer minimal lauffähigen Version bis hin zu einer vollständigen Implementierung mit Tests, Dokumentation und Reflexion der Projektdurchführung.

Software Requirements (MoSCoW-Prinzip)

Priorität 1 – MUST (Kernfunktionalität)

SR-01: Verwaltung von Spa-Dienstleistungen

- **Beschreibung:** Das System muss die Definition und Verwaltung verschiedener Spa-Dienstleistungen (z. B. Thermalbad, Massage, Tee-Therapie) ermöglichen.
- **Akzeptanzkriterien:**
    - Jede Dienstleistung besitzt eindeutige Attribute (z. B. Dauer, Temperatur, Preis).
    - Dienstleistungen können instanziiert und im System verwendet werden.

SR-02: Automatisierte Validierung von Dienstleistungsparametern

- **Beschreibung:** Das System muss sicherstellen, dass dienstleistungsspezifische Grenzwerte (z. B. maximale Temperatur oder Behandlungsdauer) automatisch zur Laufzeit validiert werden.
- **Akzeptanzkriterien:**
    - Grenzwertverletzungen führen zu einer Fehlermeldung oder Exception.
    - Die Validierung erfolgt über Python-Deskriptoren.
    - Im Anwendungscode sind keine expliziten if-Prüfungen für diese Regeln notwendig.

SR-03: Termin- und Buchungssystem

- **Beschreibung:** Das System muss es ermöglichen, Kunden Termine für Spa-Dienstleistungen zu buchen.
- **Akzeptanzkriterien:**
    - Ein Termin besteht aus Kunde, Dienstleistung, Datum und Uhrzeit.
    - Termine können erfolgreich angelegt werden.
    - Ungültige Buchungen werden abgelehnt.

Priorität 2 – SHOULD (Wesentliche Erweiterungen)

SR-04: Vermeidung von Terminüberschneidungen

- **Beschreibung:** Das System sollte sicherstellen, dass sich Termine für dieselbe Ressource nicht überschneiden.
- **Akzeptanzkriterien:**
    - Überschneidende Termine werden erkannt.
    - Buchungen mit Konflikten werden abgelehnt.

SR-05: Erfassung von Einnahmen

- **Beschreibung:** Das System sollte Einnahmen aus gebuchten Spa-Dienstleistungen erfassen.
- **Akzeptanzkriterien:**
    - Jede bestätigte Buchung erzeugt eine Einnahme.
    - Einnahmen können summiert werden.

SR-06: Erfassung von Ausgaben

- **Beschreibung:** Das System sollte laufende Betriebsausgaben erfassen können.
- **Akzeptanzkriterien:**
    - Ausgaben können mit Betrag und Kategorie gespeichert werden.
    - Gesamtausgaben können berechnet werden.

Priorität 3 – COULD (Optional / Erweiterbar)

SR-07: Gewinnberechnung

- **Beschreibung:** Das System könnte den Gewinn als Differenz zwischen Einnahmen und Ausgaben berechnen.
- **Akzeptanzkriterien:**
    - Gewinn = Summe Einnahmen − Summe Ausgaben.

SR-08: Automatische Registrierung von Dienstleistungen

- **Beschreibung:** Das System könnte neue Spa-Dienstleistungen automatisch registrieren.
- **Akzeptanzkriterien:**
    - Alle Dienstleistungsklassen sind zentral abrufbar.
    - Implementierung erfolgt über eine Metaklasse oder Introspection.

SR-09: Kundenpräferenzen und Buchungsvorschläge

- **Beschreibung:** Das System könnte aus der Historie der Kundenpräferenzen lernen und automatisierte Buchungsvorschläge unterbreiten.
- **Akzeptanzkriterien:**
    - Analyse der Buchungshistorie zur Identifikation bevorzugter Dienstleistungen.
    - Generierung von personalisierten Vorschlägen basierend auf den erlernten Datenmustern.

Ressourcen & Technische Umsetzung

Bibliotheken und Packages

ZWECK	BIBLIOTHEK / MODUL	BEMERKUNG
ZEITMANAGEMENT	datetime	Verwaltung von Terminen und Zeitfenstern.
TESTING	pytest	Durchführung von Unit- und Integrationstests.
DATENHALTUNG	json, sqlite3	Persistierung der Kunden- und Buchungsdaten.
FINANZANALYSE	numpy, pandas	Effiziente Berechnung finanzieller Kennzahlen.
PRÄFERENZANALYSE	scikit-learn	Inferenz-Logik für Buchungsvorschläge.
BERICHTSWESEN	rich, tabulate	Strukturierte Ausgabe von Berichten in der Konsole.

Entwicklungsumgebung & Struktur

Bevor die Implementierung beginnt, wird eine klare Systemarchitektur definiert, um Wartbarkeit und Testbarkeit sicherzustellen. Die Struktur orientiert sich an modernen Software-Engineering-Prinzipien:

- **Ordnerstruktur:**
    - core/: Enthält die Basis-Logik, wie Deskriptoren für die Validierung und Metaklassen.
    - models/: Beinhaltet die Datenklassen für Customer, Service (und dessen Unterklassen wie Sauna, ThermalBath) sowie Appointment.
    - logic/: Hier werden der BookingManager für die Terminprüfung und der FinanceManager für die NumPy-basierten Analysen implementiert.
    - data/: Speicherort für die JSON- oder SQLite-Datensätze.
    - tests/: Ein dedizierter Bereich für Unit- und Integrationstests, getrennt nach Modulen.
- **Module und Klassen:** Die Anwendung wird streng modular aufgebaut. Die Trennung der Verantwortlichkeiten (*Separation of Concerns*) stellt sicher, dass die Validierungslogik (Deskriptoren) unabhängig von der Benutzeroberfläche oder der Finanzberechnung existiert.
- **Dokumentation:** Das Projekt umfasst eine technische Dokumentation bestehend aus einer README.md zur Installation und Bedienung sowie grafischen UML-Modellen (Use-Case- und Klassendiagramme im Mermaid-Format), um die Systembeziehungen visuell darzustellen.
- **Testing-Framework:** Es wird eine Test-Suite auf Basis von pytest erstellt. Dabei werden insbesondere Randfälle (z. B. Überschreitung der Maximaltemperatur im Thermalbad oder Terminüberschneidungen) durch automatisierte Tests abgesichert.