Architekturdokumentation

Saturday, March 4, 2023 7:11 PM

ARCHITEKTURÜBERBLICK

Splitter

Splitter ist ein minimal viable product (MVP) , welches Menschen, die mit Gruppen unterwegs sind helfen soll,

ihre Ausgaben im Blick zu behalten und diese in der Gruppe am Ende des Tripps auszugleichen.

Dieser Architekturüberblick lässt Euch die maßgeblichen Entwurfsentscheidungen nachvollziehen.

Er zeigt die Struktur der Lösung und das Zusammenspiel zentraler Elemente.

Die Gliederung der Inhalte erfolgt nach der arc42-Vorlage.

1. EINFÜHRUNG UND ZIELE

1.1 Aufgabenstellung:

Was ist Splitter?

- Splitter ist eine vom Funktionsumfang eingeschränkte Software (MVP)
- Sie dient dem Geldausgleich innerhalb einer vom Gruppenleiter erstellten Gruppe, für die gemeinsame Ausgaben getätigt werden.

Wesentliche Features:

- Erstellungen von beliebig großen Gruppen für Ausflüge etc.
- Möglichkeit Ausgaben von Gruppenteilnehmer für diverse Aktivitäten einzutragen
- Berechnung eines mögichen Geldausgleiches für die eingetragenen Ausgaben der Teilnehmer

1.2 Qualitätsziele:

Wir probieren hier eine adäquate Rechnung zum Ausgleich von beliebig vielen Ausgaben durch Aktivitäten bei beliebig hohen Geldmengen innerhalb der erstellten Gruppen zu finden, jedoch ist nicht immer die minimale Anzahl an Transaktionen garantiert.

2. RANDBEDINGUNGEN

2.1 Technische Randbedingungen:

Randbedingung	Erläuterung, Hintergrund
Umsetzung mit Spring Boot als Webanwendung	Im Rahmen der Veranstaltung vorgegebene Randbedingung.
Datenbankbetrieb	Als Datenbank wird PostgreSQL verwendet. Die Datenbank muss in einem Docker-Container laufen
Implementierung in Java	Entwicklung unter Version Java 17
Authentifizierung	Authentifizierung durch OAuth bereitgestellt, OAuth provider ist hier GitHub

2.2 Organisatorische Randbedingungen:

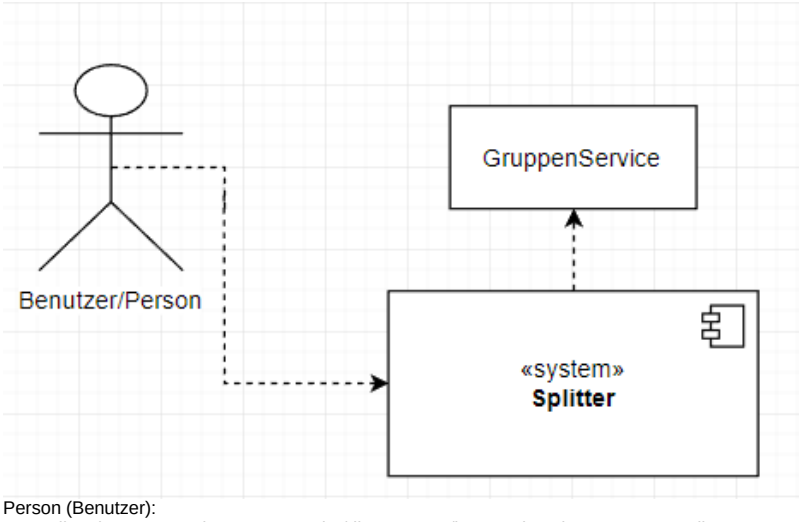
Randbedingung	Erläuterung, Hintergrund
Team	Iraj Masoudian (ixYozora), Jamie Ishmael Gerlach (JamieGFL), Erik Gal (PTamoynesis)
Zeitplan	Beginn der Entwicklung Februar 2023, vorzeigbare Version 14. März 2023 bis auf Persistenz. Fertigstellung 17. März 2023.
Vorgehensmodell	Als Architekturmuster verwenden wir hier die Onion-Architektur. Domänengetriebene Entwicklung (DDD). Zur Dokumentation der Architektur wird arc42 verwendet.
Entwicklungswerkzeuge	Entwurf mit OneNote. Erstellung der Java Quelltexte in IntelliJ.
Testwerkzeuge	JUnit für inhaltliche Korrektheit als auch Integrationstests. Persistenztests werden mit H2 umgesetzt.

2.3 Konventionen

Konvention	Erläuterung, Hintergrund
Architekturdokumentation	Terminologie und Gliederung nach dem deutschen arc42-Template.
Kodierrichtlinien	Produktivcode muss nach den Regeln des Google-Styleguides geschrieben werden. Einsatz von Checkstyle und Spotbugs.
Sprache	Benennung von Domänenobjekten in Deutsch, technische Sprache in Englisch.

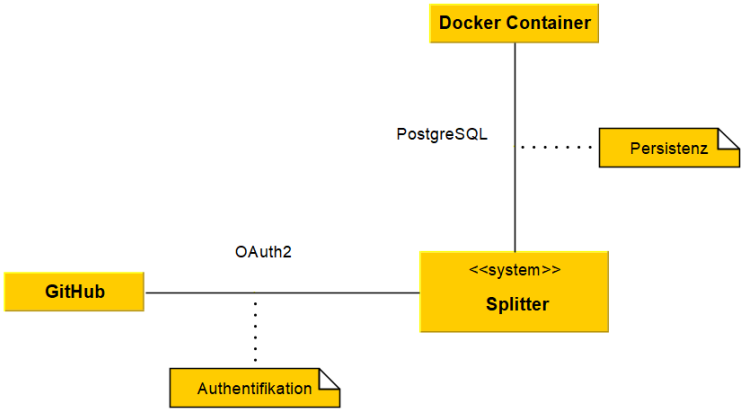
3. KONTEXTABGRENZUNG

3.1 Fachlicher Kontext



Person (Benutzer):
Um Splitter benutzen zu können muss der/die Benutzer/in zunächst eine Gruppe erstellen. Personen können über ihren GitHub Nam en zu der Gruppe hinzugefügt werden. Jede Person in der Gruppe kann innerhalb der Gruppe für andere Personen Ausgaben eintragen. Wenn Ausgaben eingetragen worden sind kann der Nutzer durch das System von Splitter die nötigen Transaktionen berechnen lasse n, um die Ausgaben auszugleichen.

3.2 Technischer Kontext



GitHub (Fremdsystem):
Wir verwenden die GitHub Daten über OAuth2 für Authentifizierungen und Zuordnungen von Gruppen und Ausgaben der angemeldeten Person.
Docker Container (Fremdsystem):
Wir verwenden den Docker Container zur Verwaltung unserer Daten in einer PostgreSQL Datenbank.

4. Komponenten der Anwendung

4.1 Domänenobjekte

Wir haben uns für 1 Aggregat mit dem Aggregate Root, der Gruppe, entschieden, da wir es einfacher fanden eine 1 zu n Beziehun g zwischen der Gruppe und den Personen zu modellieren und implementieren als eine m zu n Beziehung.
Teil des Aggregetates sind eine Reihe von Entitäten/Wertobjekten, die im folgenden beschrieben werden:

Gruppe:	Die Gruppe ist unser Aggregate Root und verfügt über alle zentralen Methoden, um die Funktionalität unser Anwendung zu gewähren. Die Gruppe speichert eine ID und einen Gruppennamen, eine Personenliste und die in der Gruppe getätigten Ausgaben als auch die nötigen Transaktionen, Informationen darüber, ob die Gruppe geschlossen ist oder ob bereits Ausgaben getaetigt wurden und eine Reihe von Hilfswerten für die Berechnung der Transaktionen (Nettobetraege, Ausgleich)
Person:	Personen stellen die User unserer Anwendung dar, welche auf diese zugreifen und sich selber als auch andere Personen mit GitHub-Namen in die Anwendung eintragen können. In der Domäne ist eine Person als ein Wertobjekt modelliert, welches den Namen der Person speichert.
Ausgabe:	Entität, in welchem jeweils eine Ausgabe mit einem Namen, einem Ausleger, Kosten und den Rest der Personen, für die ausgelegt wird, gespeichert wird
Aktivitaet:	Wertobjekt, in welchem die Aktivität der Ausgabe gespeichert wird
Transaktion:	Entität, in welchem eine nötige Transaktion mit den beteiligten Personen und dem Betrag gespeichert ist

4.2 Bausteinsicht

Service
Splitter ist ein Monolith, weswegen es keine Subsysteme gibt. Die primäre Aufgabe vom Splittersystem selbst ist wie vorher beschrieben die Verwaltung von Gruppen, in welchen Personen, Ausgaben und nötige Transaktionen zum Ausgleich gespeichert werden können.

Das System realisiert diese Verwaltung primär mit Hilfe der Schnittstelle des GruppenService, durch welchen folgende Funktionalitäten verfügbar sind:

addGruppe	Fügt Gruppe für aktuellen Nutzer hinzu
closeGruppe	Schließt aktuelle Gruppe
getSingleGruppe	Gibt Gruppe mit angegebener ID zurück
addPersonToGruppe	Fügt der Gruppe eine Person hinzu
addAusgabeToGruppe	Fügt der Gruppe eine Ausgabe hinzu
transaktionBerechnen/getTransaktionen	Berechnet die nötigen Transaktionen/gibt diese zurück
personToGruppeMatch	Gibt alle Gruppen für eine bestimme Person zurück

Es gibt zusätzlich noch einen RestGruppenService, welcher dafür verantwortlich ist den Zugriff von nativen Clients für Android und IOS zu ermöglichen und für diese als eine REST-Schnittstelle agiert.
Dieser hat die selben Funktionalitäten wie der GruppenService, ist jedoch darauf ausgerichtet Daten aus JSON-Dokumenten entgegenzunehmen als auch Daten aus dem Backend entsprechend zu mappen, sodass sie als Antwort wieder als JSON-Dokument an die Clients geschickt werden können.

Validierung

Es werden drei verschiedene Objekte benutzt, um Eingaben in die Anwendung zu validieren:

GruppenForm	validiert den eingegebenen Gruppennamen auf Länge und darauf, dass überhaupt einer eingegeben wurde
AusgabenForm	validiert, dass alle Eingaben für eine Ausgabe eingegeben wurden und dass kein negativer Betrag eingetragen wird
LoginForm	validiert eingegeben GitHub-Namen für Personen auf GitHub-Namenskonvention

Datenbankschnittstelle

Die Anwendung implementiert das GruppenRepository Interface, welches vom GruppenService als Schnittstelle zur Datenbank genutzt wird. Dieses wird vom GruppenRepositoryImpl implementiert und verfügt über folgende Methoden:

findAll()	gibt alle persistierten Gruppen zurück
findById(Integer Id)	gibt die persistierte Gruppe mit angegebener ID zurück
save(Gruppe)	Persistiert eine Gruppe
toGruppe	mapped Datenbankobjekt der Gruppe (GruppeDTO) auf Gruppe und gibt diese zurück
fromGruppe	mapped Gruppe auf Datenobjekt der Gruppe (GruppeDTO) und gibt dieses zurück

Es werden eine Reihe von DTOs benutzt, um das Gruppen-Aggregat zu persistieren.
Wir haben die Entscheidung getroffen, für das Wertobjekt der Person verschiedene DTOs zu benutzen, da eine Person in verschiedenen Kontexten existiert und somit Probleme verursacht, wenn sie überall als das selbe Objekt persistiert wird.
Es gibt den Gruppenkontext, in welchem PersonDTOs gespeichert werden, den AusgabenKontext, in welchem ein AuslegerDTO als auch TeilnehmerDTOs gespeichert werden und den Transaktionskontext, in welchem zwei Personen ZahlerDTO und ZahlungsempfaengerDTO gespeichert werden.