Designdokumentation – Team Cloudia

Um den Kontaktsplitter umzusetzen, benötigt die zu erstellende Software ein Design. Der erstellte Entwurf gliedert sich in einen Grobentwurf, der aufzeigt, wie die Software und ihre Komponenten miteinander agieren und einen Feinentwurf, der zeigt, wie die Komponenten der Software selbst aufgebaut sind und funktionieren.

Grobentwurf

Der Kontaktsplitter wird mittels einer Client-Server-Architektur umgesetzt. Der Client (Frontend) ist eine VueJS-Anwendung, die im Browser ausgeführt werden kann. Sie stellt eine Oberfläche zur Verfügung und hält für die Sitzung die eingetragenen Kontakte. Innerhalb der Oberfläche wird sowohl die Freitexteingabe als auch eine rein manuelle Eingabe unterstützt. Außerdem ist es möglich neue Titel hinzuzufügen (zu erlernen).

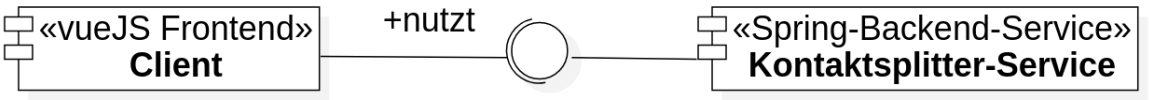
Zur Aufsplittung der Freitexteingaben sendet das Frontend einen POST-Request an einen Spring-REST-Web-Service (Subroute: /contact), der sich um die eigentliche Aufsplittung kümmert. Als Body wird dabei der Kontakt-String innerhalb eines Transfer-Objekts gesendet. Abschließend kann der Request vier unterschiedliche Antworten erhalten:

- HTTP-Code 200: Aufsplittung erfolgreich und aufgesplitteter Kontakt ist als JSON-Objekt in Antwort-Body enthalten.
- HTTP-Code 400:
    - Aufsplittung nicht möglich. Weiteres Handling im Frontend. Kein Objekt im Body
    - Aufsplittung nur partiell möglich. Partiell aufgesplittete Adresse als Objekt im Body der Antwort
- HTTP-Code 404: Die angefragte Route im Service existiert nicht.
- HTTP-Code 500: Ein Server-Fehler ist aufgetreten.

Bei der Schnittstelle für die Erlernung neuer Titel, wird auch der Titel-String in einem Transferobjekt mit einem POST-Request an den Service geschickt (Subroute: /title-learning). Folgende Antworten sind auf den Request möglich:

- HTTP-Code 200: Erlernen des neuen Titels erfolgreich.
- HTTP-Code 404: Die angefragte Route im Service existiert nicht.
- HTTP-Code 500: Ein Server-Fehler ist aufgetreten.

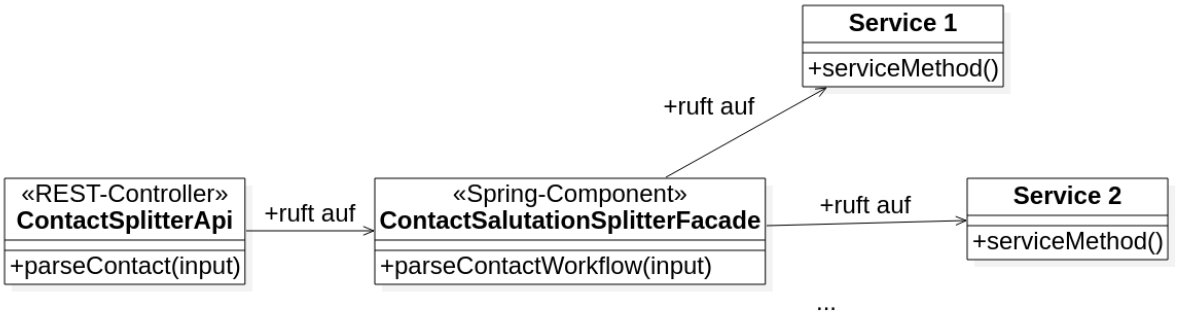
Folgendes Komponentendiagramm fasst den Grobentwurf zusammen und zeigt, wie Client und Server zusammenhängen. Die gezeigte Schnittstelle vereint beide beschriebenen Requests.



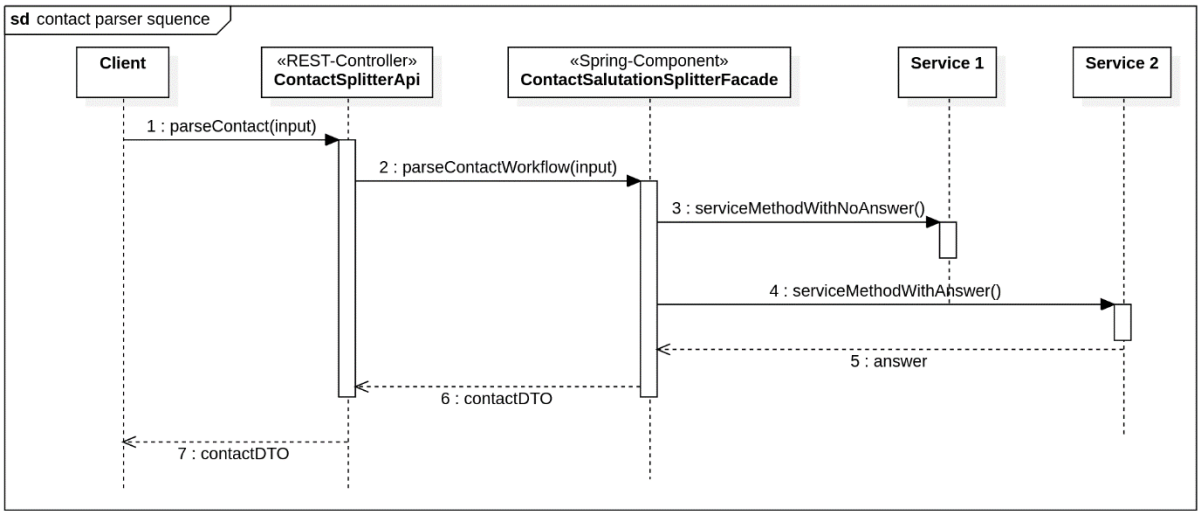
Feinentwurf

Der innere Aufbau des “Kontaktsplitter-Service” folgt einer linearen Schichtenarchitektur bestehend aus einer “boundary"-Schicht, die die Anfragen entgegennimmt und infolgedessen die Schnittstelle definiert und einer “control”-Schicht, die die Businesslogik der Anwendung enthält.

Innerhalb der “control”-Schicht wird ein Fassaden-Pattern angewendet, um den Workflow zu steuern. Die Fassadenklasse kennt alle relevanten Services, spricht diese an und nimmt deren Antworten entgegen. Am Schluss gibt die Fassade das fertige Kontaktobjekt zurück. Folgendes Klassendiagramm zeigt schemahaft die innere statische Struktur des Service (unvollständig).



Weiterhin kann in diesem Sequenzdiagramm gesehen werden, wie zur Laufzeit ein erfolgreicher Request abläuft. Das gezeigte Diagramm zeigt einen Aufruf schemahaft. Die reale Lösung hat mehr Services.



Um Fehler innerhalb des Programmes zu behandeln, werden Exceptions geworfen, die innerhalb eines Exception-Handels behandelt werden. Der ExceptionHandler selbst ist ein “@RestControllerAdvice” innerhalb des Spring-Frameworks. Die Exception, die geworfen wird, falls ein Fehler beim Splitting aufkommt, ist die “ContactParsingException”.

Für die Erlernung neuer Titel wird ein einfacherer Workflow verwendet. Von der “boundary”-Schicht aus wird der Request wieder vollständig an die “control”-Schicht weitergegeben. Hierbei wird er nicht an eine Fassaden-Klasse weitergegeben, sondern direkt vom zuständigen Spring-Service bearbeitet (*TitleLearningService*). Dieser Bearbeitet die Anfrage vollständig und hat keinen Rückgabewert. Wird keine Exception geworfen, wird dem Client der HTTP-Code 200 zurückgegeben.