Technische Dokumentation

Im Folgenden wird der technische Aufbau des Programms erläutert. Hierbei hilfreich sein kann das simultane Betrachten des Technischen Flowcharts, der die Prozesse des Programms verbildlicht, sowie die Klassen- und Datenbankstrukturen, die Sie ebenfalls im Github finden können.

Das Programm ist in drei Schichten unterteilt: Datenbank, Datenmodellierung und -verarbeitung und Anzeige. Als Build-Management-Tool benutzen wir Maven.

Datenbank

Die Datenbank ist eine PostgreSQL Datenbank. Sie besteht aus 9 Tabellen: Processes, CahngesOfAddresses, StagesOfCOA, Persons, Identifications, Assignees, HouseOwners, HouseProviders und Addresses.
Die Tabelle Processes ist dabei eine Art Sammelstelle aller Prozesse, die mit dem Programm verwirklicht werden können, und beinhaltet alle die ProcessIDs alles Prozesse. In unserem Fall ist dies nur die Ummeldung über ChangesOfAddresses, eine Erweiterung mit Tabellen für weitere Prozesse ist allerdings leicht möglich: Die Processes Tabelle wird über eine JOIN Anweisung mit der Tabelle des jeweiligen Programms verbunden, damit neben der angesprochenen ProzessID auch eine Referenz auf die Tabelle z.B. der Ummeldung gespeichert werden kann. Eine Zuweisung der ProzessID zum Prozesstyps ist somit gewährleistet.
Die Tabelle ChangesOfAddresses speichert alle wichtigen Daten, die für eine Ummeldung notwendig ist. Zum Großteil wird dabei auf eine weitere Tabelle verwiesen. Somit können die Daten sinnvoll kategorisiert werden und sind leicht überschaubar.

Die Schnittstelle zwischen der Datenbankebene und den höher liegenden Ebenen findet dabei über Java Persistence API (JPA) statt. Somit ist es leicht möglich den Datenbanktypen auszutauschen ohne Änderungen am Quellcode vornehmen zu müssen. Lediglich die Anpassung von JPA in einer persistence.xml Datei ist notwendig.

Datenmodellierung und -verarbeitung

Die aus der Datenbank gelesenen Daten werden nicht direkt im Programm verwendet. Sie werden zuvor von einer ModelFactory in entsprechende Models umgewandelt. Nur diese Models werden im weiteren Verlauf zum Anzeigen, Speichern oder Ändern von Daten verwendet. Die Unabhängigkeit zur Datenbank ist also in dem Sinn gegeben, dass eine veränderte Ausgabeform der Daten aus der Datenbank lediglich die Umstellung dieser ModelFactory zur folge hat. Der restliche Quellcode bliebt unbeeinflusst und funktioniert nach umgeschriebener Modellerstellung gleich.

Für die Hauptfunktion des Programms, das Akquirieren von Daten auf Basis von Fragestellungen, benötigt die Modelle im JSON-Format. Dafür wird JSON.simple verwendet. In einem rekursiven verfahren wird sich die Klasse des Modells angeschaut und die Variablen dieser als Keys im JSON-Objekt verwendet, jeweils mit dem initialen Wert null. Sollte eine Variable selbst ein Model sein, greift die Rekursion und ein dem Variablen-Key wird die eigene JSON-Struktur als Wert übergeben.

Nach dem Befüllen der JSON Struktur mit den Daten des Nutzers, wird das Objekt auf selben, rekursiven weg wieder zurück in das Modell gewandelt.

Ebenso wie das Speichermedium der Antworten werden auch die Fragen als JSON-Objekt benötigt. Hierbei wird dem Programm jede Frage als ein Key-Value Paar mit folgender Struktur mitgegeben:

Name der Frage: {

„type“: Wie soll die Antwort gegeben werden (Innput, Auswahl, Datum),
Sprache1: Die eigentliche Frage in Sprache1,
Sprache2: Die eigentliche Frage in Sprache2,
…
„options“: [Array mit den Antwortmöglichkeiten bei einer Auswahl-Antwort],
„case“: {JSON-Objekt, das zu bestimmten Antworten jeweilige Fragen referenziert},
„number“: Die interne Nummer der Frage,
„model“: Gibt eine Referenz darauf, wo im JSON-Objekt des Models die Antwort gespiechert werden soll,
„allowedEmpty“: Darf die Frage leer bleiben,
„previous“: Name der Frage, die vor dieser kommt,
„next“: Name der Frage, die nach dieser kommt

}

Des Weiteren werden Meta-Daten mitgegeben:

„baseModel“: Modellklasse, die durch den Fragenkatalog befüllt werden soll,
„initial“: Name der Frage, die zuerst aufgerufen werden soll,
„questionCount“: Gesamtanzahl aller Fragen (inkl. derer, die nur unter bestimmten Umständen gestellt werden)

Diese Struktur kann sehr leicht durch ein Programm erzeugt werden, mit dem man eigene Fragenkataloge erzeugen kann. Es wird einfach ein Basismodell ausgewählt, welches befüllt werden soll und für jeden Wert des Modells wird eine Frage gestellt. Durch das automatische Generieren wäre die Fehleranfälligkeit im Vergleich zum manuellen Erstellen der Struktur deutlich niedriger und externe, nicht programmier-affine Personen könnten ihren eigenen Katalog erstellen, um beispielsweise die Anmeldung bei einer Fahrschule zu digitalisieren.
Wir hätten gerne auch dieses Teilprogramm in unser Projekt implementiert, konnten dies allerdings aus Zeitgründen nicht mehr umsetzen.

Auch weitere Ressourcen, die nicht im Zusammenhang mit der Datenbank stehen werden nie direkt in das restliche System geladen. Für jeden Typ Medium gibt es einen eigenen Handler: I18nHandler, ColorHandler, FontHandler, InternalPathsHandler, ImageHandler. Die Ressourcen selbst (bzw. der Verweis auf diese) befinden sich jeweils in einer .properties Datei.
Das Programm referenziert nur die Handler, niemals aber die Ressource selbst. Die vom Handler geladene Datei kann einfach und auch im laufenden Programm ausgetauscht werden. Beim neu Laden der jeweiligen Ansicht wird diese dann mit dem gleichen Handler geladen, der nun allerdings auf ein anderes Dokument als Quelle zurückgreift. Ein Wechsel der Sprache oder das Hinzufügen weiterer ist somit problemlos umsetzbar.
Ebenso kann so die Darstellung des GUI durch veränderte Farbschemas und Schriftarten unkompliziert geändert werden und neue Designs sind ohne großen Aufwand testbar.

Anzeige

Die Basis des GUI bildet ein JFrame. Dieser wird beim Start des Programms erzeugt und bleibt bis zum Beenden des Programms bestehen. Der Frame selbst hat zu jedem Zeitpunkt nur einen Inhalt: eine View. Eine View ist ein JPanel, welches als Container für die eigentlichen Elemente dient.
Beim Wechseln der View wird diese aus dem Frame gelöscht und vergessen und die neue View wird geladen. Der Frame aktualisiert sich daraufhin selbst, um die Änderungen anzuzeigen.

Eine View ist immer nach demselben Prinzip aufgebaut: Es gibt einen Header, der für alle Views gleich ist, um eine Webseite vorzutäuschen, und einem Body, welche eine interne Klasse der View darstellt. Der Body wird mit GUI-Elementen befüllt, die teilweise in der jeweiligen View erstellt werden, in den meisten Fällen ist ein GUI-Element allerdings in einer eigenen Klasse ausgelagert. Das hat drei Effekte: Erstens wird somit der Code in der View übersichtlicher und es kann sich auf die Positionierung des Elements in der View konzentriert werden, zweitens entsteht somit eine weitere Trennung und eine Vereinheitlichung der gezeigten Elemente. Was in View A und View B gezeigt wird kann von derselben Stelle aus verändert werden. Somit wirkt das Interface konsistent und einheitlich. Drittens ist eine Erweiterung des Programms um weitere Views um einiges leichter, da die notwenigen Elemente mit hoher Wahrscheinlichkeit bereits programmiert wurden und nur noch in die jeweilige View eingesetzt werden müssen.