Kundenliste anzeigen

Für die weitere Entwicklung unserer Applikation gehen wir Anwendungsfall-Orientiert vor.

Ziel

Dem Benutzer soll auf seinem Client (im Browser) eine sortierte Liste (Tabelle) der vorhandenen Kunden angezeigt werden.

Kundenübersicht

Kunde	Kundennummer
Bucher Michael	2
Näf Claudia	1
Weidmann Sarah	3

Abbildung 1 Kundenübersicht CRM

Vorgehen

Als Erstes brauchen wir einen Web-Kontroller, welcher die Anfrage entgegennimmt, die verlangten Daten organisiert und dafür sorgt, dass eine passende HTML-Datei ausgeliefert wird.

Dazu erstellen wir die Klasse «CustomerController» im Package «web».

Der «CustomerController» besorgt sich seine Daten aus der darunterliegenden Serviceschicht. Das Interface «CustomerService» existiert bereits und bietet auch die passende Funktion: «getCustomerList()»

Dummerweise ist der «CustomerService» nur ein Interface, wir brauchen also als nächstes eine konkrete Implementierung, nämlich die Klasse «CustomerServiceImpl». Diese Klasse erstellen wir in einem neuen Package: «service.impl».

Wie wir aus dem Dokument Serverapplikationen wissen, ist das Service-Layer nur für die Applikationslogik zuständig, der Datenzugriff findet im Repository-Layer statt.

Konkret heisst das, dass wir ein neues Interface «CustomerRepository» benötigen welches mindestens die Funktionalität «List<Customer>findAll ()», am besten bereits sortiert nach dem Namen, enthält. Dieses Interface erstellen wir im Package «repository». (Magic again: Erstellen Sie das Interface «CustomerRepository» und leiten Sie es von «JpaRepository<CustomerImpl, Long>» ab. Damit existiert die Funktion» findAll(Sort sort)» bereits).

Soweit so gut. Leider sind unsere Business Objects momentan nur als Idee (Interface) vorhanden. Wir implementieren deshalb die Klassen «CustomerImpl» und «MemoImpl» in einem neuen Package «model.impl» gemäss der Designresultate aus dem Dokument «CRM - Analyse & Design».

Um unseren Code lauffähig zu machen, müssen wir noch festlegen, wie unsere Daten im Datastore landen und wie wir es schaffen, im Code immer nur von Interfaces zu reden, wo wir zu Laufzeit doch konkrete Objekte brauchen.

Beachten Sie dazu die Dokumente:

- Übersicht - Was sind Annotations
- Übersicht - Was ist Dependency Injection
- Übersicht - Was ist JPA

Nachdem wir unseren Code nun mit den nötigen Annotations versehen haben, bleibt nur noch die Aufgabe, die DB beim Start des Severs mit einigen Daten zu Befüllen, so dass wir etwas zum Anzeigen haben. Dazu erstellen wir in einem neuen Package «data» die Klasse «DataInitializer» gemäss dem Code im Anhang A.

Damit das wirklich läuft, müssen Sie in «CustomerServiceImpl» «addCustomer(…)» implementieren und in «CustomerRepository» die Funktion «create(…)» hinzufügen und implementieren.

Im «CustomerController» haben wir angegeben, dass die Datei «showCustomerList» verantwortlich ist für die Erzeugung des HTML-Codes. Wir erstellen «showCustomerList.html» im «Resourceordner» Templates als HTML-Code gemischt mit Thymeleaf-Anweisungen. Den Code dazu finden Sie im Anhang 2.

Beachten Sie dazu auch das Dokument:

- Übersicht – Spring Controller
- Übersicht - Was ist Thymeleaf

Starten des Servers

Starten Sie jetzt den Server und geben Sie im Browser «localhost:8080» ein. Nach der Behebung allfälliger Fehler (meistens fehlende Annotations) sollte die Kundenübersicht im Browser erscheinen.

Anhang: Class «DataInitializer»

```
@Component
public class DataInitializer implements ApplicationRunner {

	@Autowired CustomerService customerService;

	@Override
	public void run(ApplicationArguments args) throws Exception {

		// Create some application data
		Customer c0, c1, c2;
		c0 = customerService.addCustomer(
			"Werner Max", "Tobelrainli 6", "5416 Kirchdorf");
		c1 = customerService.addCustomer(
			"Bohli Armin", "Dorfgasse 2", "8853 Lachen");
		c2 = customerService.addCustomer(
			"Ehrensberger Susi", "Heimstrasse 41", "5417 Untersiggenthal");
	}
}
```