# Aufsetzen des Projektes

Aus dem Dokument CRM – Analyse & Design wissen wir, dass wir eine Webserveranwendung programmieren müssen. Dazu brauchen wir einen Server mit Unterstützung für das http-Protokoll (Web) und einen Weg, passende HTML-Seiten generieren zu können (noch offen).

Weiterhin ist klar, dass wir die Daten in einer DB persistent halten müssen(H2). Da unsere Applikation objektorientiert arbeitet, die Datenhaltung aber in einer SQL-Datenbank erfolgen soll, brauchen wir Unterstützung für die Umwandlung von Tabellenresultaten zu Objekten und umgekehrt (JPA).

Damit wir möglichst komfortabel arbeiten können, brauchen wir ein zusätzliches Tool für die Entwicklerunterstützung (DevTools).

Um möglichst schnell Arbeiten zu können, lassen wir uns das Projektgerüst generieren.

## Spring Boot

Gehen Sie auf die Seite https://start.spring.io/

- Wir wollen ein Javaprojekt mit Maven und Spring Boot **Version 1.4.7**
- Als Group wählen Sie ch.zli.m223
- Als Artifact wählen Sie CRM
- Unter Dependencies schreiben und wählen Sie der Reihe nach Web, Thymeleaf, JPA, H2 und DevTools

Ihr Bildschirm sollte etwa so aussehen:

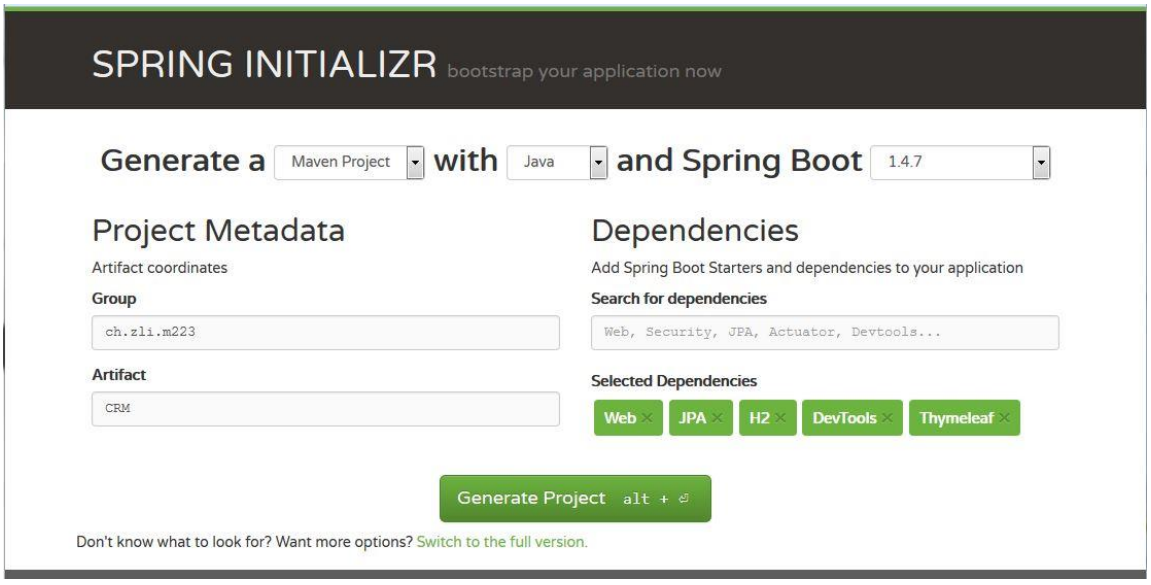


Abbildung 1 Website SpringInitializer

Anschliessend klicken Sie «Generate Project». Die Datei «CRM.zip» wird automatisch heruntergeladen.

Stellen Sie erst sicher, dass die Proxy-Einstellungen in Eclipse richtig sind und Sie Verbindung mit dem Netz haben.

Kopieren Sie dann den Inhalt von «CRM.zip» in Ihren Workspace, öffnen Sie Eclipse und importieren Sie das Projekt aus dem CRM-Ordner als Maven-Project.

Zum Testen der Funktionalität erstellen Sie die Datei «index.html» im Ordner «src/main/resources/static» und füllen sie mit folgendem Inhalt:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Getting Started: Serving Web Content</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  </head>
  <body>
    <h1>Greetings from holiday</h1>
  </body>
</html>
```

Anschliessend starten Sie den Server mit «run as Spring Boot App» und geben im Browser «localhost:8080» ein zum Testen.

Beenden Sie anschliessend die Serverapplikation.

## Projektaufbau

Aus dem Dokument «Serverapplikationen» ist klar, dass wir unsere Applikation mittels Layer (Schichten) aufbauen wollen.

Wir brauchen also je ein Java-Package für

- die Business Objects («model»)
- die Serviceschicht («service»)
- für die Repositoryschicht («repository»)
- für die Web-Controllerschicht («web»)

Erstellen Sie diese Packages als Subpackages von» ch.zli.m223.CRM».

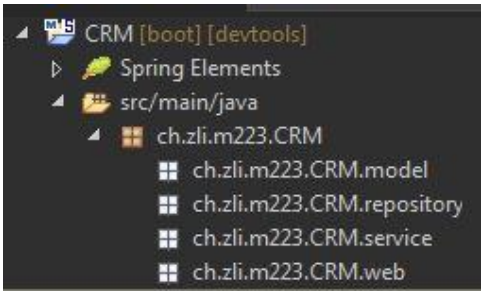


Abbildung 2 Ordnerstruktur CRM

## Definieren der Serverfunktionalität

Im Dokument «Analyse & Design» haben wir im Abschnitt Design festgelegt, welche Funktionalität unser Server und unsere Business Objects anbieten. Implementieren Sie entsprechenden **Interfaces** im jeweils passenden Package.

Versehen Sie alle Interface und alle Methoden mit einem ausführlichen Java-Doc Kommentar. Dokumentieren Sie alle Parameter und insbesondere die Return-Werte (wann ist etwas null)?

Soweit sind wir fertig. Die restliche Arbeit besteht ☺ nur noch ☺ aus der Implementierung der vorhandenen Funktionalität, den Tests und der Implementierung des Web-Controllers.

# Anhang: Interfaces «Customer», «Memo», «CustomerService»

```
/** Business object abstraction for a customer */
public interface Customer {
	/** @return the customer id or null if not yet persisted on the datastore */
	Long getId();

	/** @return the customers name */
	String getName();

	/** @return the customers street */
	String getStreet();

	/** @return the customers location */
	String getCity();

	/** @return a list of memos */
	List<Memo> getMemos();
}


/** Business object abstraction for a memo */
public interface Memo {
	/** @return the memo id or null if not yet persisted on the data store */
	Long getId();

	/** @return the date the Memo was created. */
	Date getCoverageDate();

	/** @ return the memos content */
	String getNote();
}


/** The customer services */
public interface CustomerService {
	/** @return the list of all customers*/
	List<Customer> getCustomerList();

	/**
	 * @param customerId the customer id
	 * @return a customer object or null if not found
	 */
	Customer getCustomer(long customerId);

	/**
	 * Add a new customer
	 * @param name its name
	 * @param street its street
	 * @param city its city
	 * @return the newly created customer object
	 */
	Customer addCustomer(String name, String street, String city);

	/**
	 * Add a new Memo to a customers memo list
	 * @param customerId the customers id
	 * @param memotext the text for the new memo
	 * @return the newly created memo object
	 */
	Memo addMemoToCustomer(long customerId, String memotext);
	}
```