Adressvalidierung für Kundendaten

POS-Projekt HTL Kaindorf - Schuljahr 2018/19, 2. Semester

Christian Gutmann
Christoph Klampfer
Marina Spari

Projektstart: Montag, 8.April 2019
Projektende: Samstag, 15. Juni 2019

Inhaltsverzeichnis

Facts/Links

IDE
Netbeans

Projecttype
Webproject

Build-Management
Maven

Versionsverwaltung
Github
https://github.com/mspari/AdressvalidierungPOS

Aufgabenverteilung
Trello
https://trello.com/b/pP4ShxrV/adressvalidierung-für-kundendaten

Datenbank
mySQL → remotemysql.com

Technische Schwerpunkte
MySQL, REST Webservice, HTML/CSS5, Java Web

Projektantrag: Adressvalidierung

Projektteam/vorgeschlagene Rollenverteilung

Christian Gutmann: DB, Backend

Christoph Klampfer: Backend, Kommunikation

Marina Spari: Web Oberflächen-Entwicklung (Frontend), Grafik, CSS

Beschreibung des Projektes

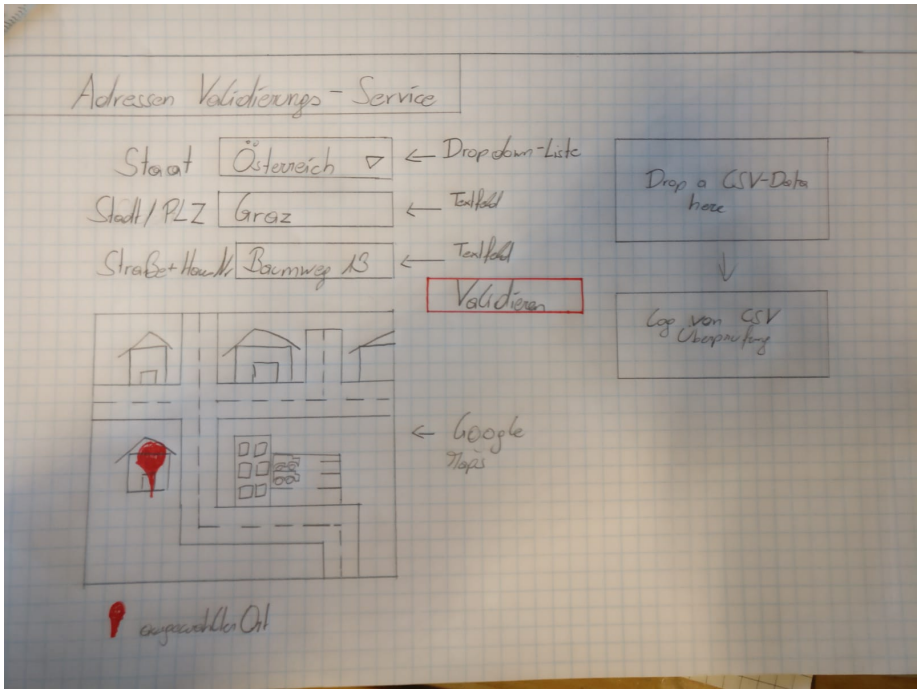
Das Ziel des Projekts ist es, eine Datenbank für Kundenadressdaten (Wohnort) zu entwickeln. Diese Adressdaten werden über eine Web-GUI, die sowohl am Desktop als auch auf Mobile Devices benutzbar ist (zumindest in der Browsersimulation), eingegeben. Auch der Upload von .csv Dateien sollte möglich sein. Bevor diese in der Datenbank gespeichert werden, werden diese über API-Funktionen validiert und bei Bedarf korrigiert bzw. bei ungenauen Eingaben mit mehreren Zuordnungsmöglichkeiten wird eine Auswahl angezeigt. Dies soll auch als Vorprojekt für eine eventuelle Diplomarbeit dienen.

Technische Schwerpunkte

MySQL, REST Webservice, HTML/CSS5, Java

Adressen Validierung API (noch nicht sicher welche): Streetlayer API, SmartyStreets API, SAP DQM Service

Mockup



Projektbeschreibung (engl.)

The idea of this project is to give users the possibility to cleanse their address data and store it into a database for further usage.

Please just clone our Project from https://github.com/mspari/AdressvalidierungPOS and start it.

You will see a startpage with two different options:

- on the left side to validate a single Address by typing it
- on the right site the possibility to upload a csv-File with address data

If you decide to validate a single address, make sure to fill out all fields and then press the button "validate". It will forward you to another page where you get a suggestion for correcting your address. If the address is still incorrect, the webservice couldn't find any improvements for your address.
If you want to adapt you address before storing it into the database, feel free to chance it as you want. Please note that the houseNr mustn't have any letters in it!!
On the right site of the currently displayed page, you can see a map where the corrected address is displayed. If you feel ready to insert your address into the database, just click the button "accept".

In case you want to validate multiple addresses with just one call, you can find a sample file under res/testdata.csv. Please note that it must be a csv file and the format must exactly match in order to get a successful result. Unfortunately, the drag'n'drop doesn't work on all browsers anymore...
After clicking upload file, it can take a few moments (depending on the speed of you internet connection) until you get a table with corrections displayed. Just tick those which you want to store in your database and hit "accept selected".

Sometimes, the database and the webservice don't work as expected - it's only freeware...

Projektphasierung

Wir haben für die Phasierung des Projekts einen SCRUM-ähnlichen Ansatz verwendet. Jeder Sprint war eine Woche (bzw. 2 Doppelstunden) lang – beim DailyScrum am Montag wurde jeweils besprochen, was bis zur nächsten Woche am Montag fertig sein sollte.

Unser „DailyScrum“ fand immer am Anfang der Doppelstunde statt. Dabei wurden unsere Probleme und Fortschritte besprochen. Dies erwies sich sehr oft als hilfreich, da wir dabei oft auf gemeinsame Lösungen gekommen sind.

Als Backlog haben wir Trello verwendet (https://trello.com/b/pP4ShxrV/adressvalidierung-f%C3%BCr-kundendaten) und auch aktiv genutzt. Die Unterteilung dieses Boards haben wir absichtlich so gewählt, da es für uns so am einfachsten war zu erkennen was gemacht ist und in welchen Bereichen wir noch zu tun haben.

Use-Cases/Funktionalitäten

- Eingabe und Validierung einer einzelnen Adresse
- Anzeige der korrigierten Adresse auf GoogleMaps
- Speichern der korrigierten Adresse in einer Remote Datenbank
- Hochladen eines Files über einen Filechooser bzw. Drag'n'Drop
- Korrigierte Adressen anzeigen und Auswahlmöglichkeit bieten
- Ausgewählte Adressen in einer Datenbank speichern

Programmflow

Unser Programmflow lässt sich auch ohne Klassendiagramme sehr einfach erklären:

Die Basis hierbei bildet die Klasse „Address“ welche eine Adresse mit allen dazugehörigen Attributen (Straßenname, Hausnummer, Postleitzahl, Ortschaftsname, Region, Land) abbildet. Eine solche Adresse (oder auch mehrere als File) wird in einem dem User angezeigten jsp File eingeben und diese an das Servlet übermittelt, welches weitere Methoden zur „Bearbeitung“ aufruft.

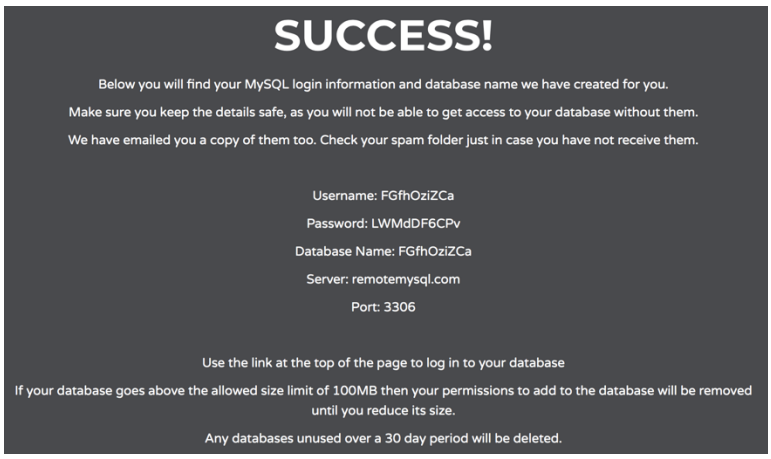
Diese Methoden sind teilweise aufgrund der Übersicht/Verständlichkeit in andere Klassen ausgelagert, jedoch sollte dies mit ein wenig Verständnis zur objektorientierten Programmierung selbsterklärend sein (readCsv Methode statisch in einer IO Klasse). Zur Speicherung statischer Daten wie Zugangsdaten (Username, Passwort) oder URLs werden Servlets verwendet.

Der Aufbau der Datenbankverbindung ist (wie in der Schule gelernt) auf mehrere Klassen aufgeteilt, um eine bessere Modularisierung und auch mehr Verständlichkeit zu schaffen. Weiters bietet dies den Vorteil, dass beispielsweise PreparedStatements einfach wiederverwendet werden können.

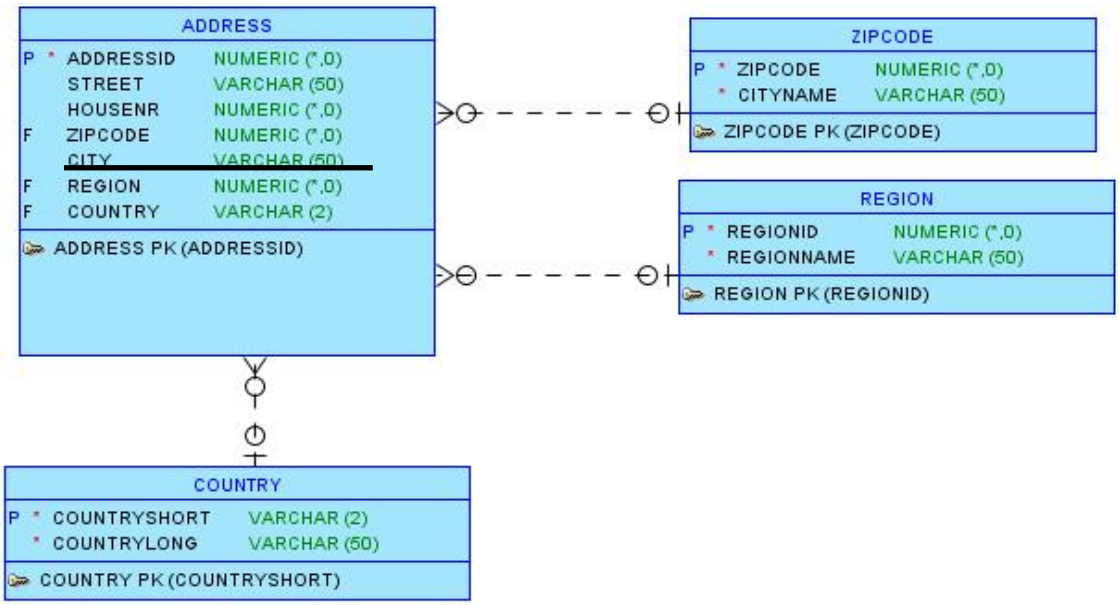
Für die Anzeige der Websites werden verschiedene .jsp Dateien verwendet, welche immer von den Servlets über einen RequestDispatcher aufgerufen werden.

Datenbank

Als Datenbanksystem haben wir uns für eine mySQL Datenbank entschieden. Diese Entscheidung haben wir getroffen, da es am einfachsten war dafür ein kostenlosen remote-hosting zu bekommen. Dabei haben wir die einzige uns sinnvoll erscheinende Möglichkeit, remotemysql.com, benutzt.



Das ER Modell unserer Datenbank sieht hierbei wie folgt aus: Jede Adresse hat eine AddressID, die Street, die HouseNr gespeichert. Der FK ZipCode referenziert die Table ZipCode, die den zugehörigen CityName speichert. Weiters referenziert Country (dies ist das Kürzel, z.B. AT) den Ländernamen und „Region“ über einen numerischen Key die Region (dies wäre z.B. Steiermark).



Verwendete Ressourcen/Herausforderungen

Google Maps

Die Einbindung von Google Maps erwies sich sehr lange als schwierig, da Google ein Paymentsystem hat (man zahlt zwar erst ab 10.000 Aufrufen), welche jedoch unsere Kreditkarten nicht akzeptiert hat. Dadurch war es auch nicht möglich einen API Key dafür zu bekommen. Natürlich gibt es auch andere Möglichkeiten wie OpenStreet Maps aber auch diese haben nicht wie gewünscht funktioniert. Schlussendlich haben wir herausgefunden, dass wir eine Map über einen iFrame einbinden können und die gewünschte Adresse über die URL angeben können.

DQM Webservice

Das DQM (Data Quality Management) Webservice der SAP bereitete sehr lange Probleme. Anfänglich konnten wir es garnicht benutzen, da die Authentifizerung in der Free (bzw. Trial) Version sehr kompliziert ist. Leider gibt es dazu auch keine wirklich gute Doku…. Irgendwann schafften wir es endlich über Postman den ersten Request abzusenden – jedoch funktionierte dies in Java danach nicht… Irgendwann (mit viel Glück) haben wir auch diese Probleme behoben – denn leider waren die Fehlermeldungen von dem Service nicht zu brauchen (z.B. Invalid Parameters – dabei ist es relativ schwer den richtigen von 10 zu finden).

GitHub

Das Aufsetzen von GitHub war anfänglich etwas kompliziert, da wir dies alle noch nie gemacht haben. Dadurch fraß dies auch einiges an Zeit…
Leider zerstörten und auch unterschiedliche Versionen oft unsere Fortschritte. Diese verschiedenen Versionen traten oft sogar ohne uns ersichtlichen Grund auf – schlussendlich haben wir aber alle Merging Issues (wenn auch oft viel Nerven draufgegangen sind) beheben könne.

JSON Parser

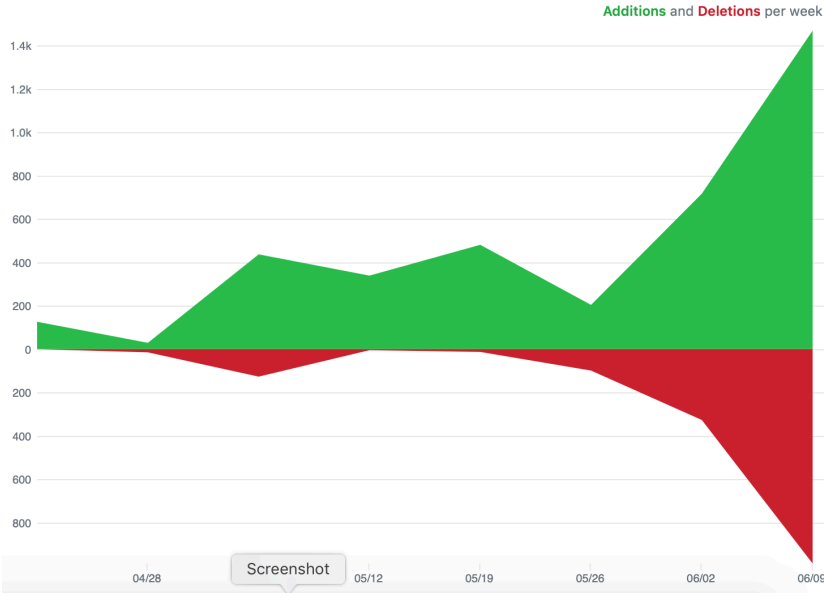
Teilweise verstehen wir bis heute noch nicht warum der JSON Parser manchmal nicht funktioniert (hat), vermutlich liegt dies an verschiedenen Charset-Problemen.

MySQL Connector

Der MySQL Connector bereitete anfänglich manchmal Probleme, bis wir draufgekommen sind, dass durch Maven 2 verschiedene MySQL Connectoren geladen wurden und immer der falsche verwendet wurde (gleiche Klassen-/Methodennamen). Durch ein komplettes neu klonen des Projekts von GitHub hat dies aber danach wieder funktioniert.

Umfänge/Zahlen zum Projekt

- 43 Commits auf GitHub
- Anfänglich viel Code hinzugefügt – danach aufgeräumt



- 9 verschiedene Dependencies
- Gemeinsam rund 100 Arbeitsstunden (davon ca. 75 im Unterricht)

Resümee

Als Abschluss möchte ich anmerken, dass das Projekt meiner Meinung nach uns alles sehr viel gebracht hat. Neu war für uns alle definitiv das Arbeiten mit externen Webservices und vorallem auch mit JSON. Weiters war auch das selbstständige Finden einer Lösung anfänglich sehr herausfordernd.