Hochschule für Technik und Wirtschaft Berlin
- Campus Wilhelminenhof -
Belegarbeit
Ortsbasierte Informationssysteme

Thema: Importer von GeoJSON-, GML- und KML-Formaten

Autor: David Linke, Sophie Schauer

Betreuer: Prof. Dr. Thomas Schwotzer

1 Aufgabe der Komponenten

In der Open Historical Data Map werden Karten, anders als bei OpenStreetMap oder GoogleMaps, aus verschiedenen Zeiträumen zur Verfügung gestellt. Um die Datenbank mit einer Vielzahl von historischen Karten füllen zu können, und dies auch für Außenstehende möglich zu machen, ist eine Schnittstelle zum schnellen und einfachen Importieren notwendig. In dieser sollen Kartendaten in den gängigen Formaten, wie GeoJSON, KML, GML, eingefügt, benannt und zeitlich eingeordnet werden können. Die Daten sollen dann direkt in die Datenbank gespeichert werden und so zur Bearbeitung nutzbar sein.

Da PostGIS Geometrien nicht in den Formaten KML, GML oder GeoJSON erfasst, müssen die Dateien zunächst umgewandelt werden. Nach Evaluierung des bestehenden Codes einer Vorgängerversion des Importers und der PostGIS-Dokumentation konnten wir feststellen, dass PostGIS über Funktionen zur einfachen Umwandlung von KML-, GML- und GeoJSON-Formaten in PostGIS-Geometrie-Objekte verfügt. Als Schnittstelle zwischen Benutzer und Datenbank wurde ein Servlet erstellt. Der Benutzer kann auf der Webseite das Format der Datei auswählen, einen Namen hinterlegen, und angeben in welchem Datumsbereich die von ihm zu importierende Datei gültig ist. In einem Textfeld können die eigentlichen Geometriedaten erfasst werden.

Um die Geometriedaten in die OHDM-Datenbank zu importieren wird eine Klasse benötigt, die eine Verbindung zur Datenbank und dem richtigen Schema erstellt. Das Servlet nimmt die eingegebenen Daten entgegen und verbindet sich mit der Datenbank. Die Geometriedaten werden, je nach ausgewählten Format, zur Umwandlung weitergeleitet. Daraufhin muss eine SQL-Query erstellt werden, um die Daten letztlich in die richtige Tabelle der Datenbank einzufügen.

2 Formate

2.1 GML - Geography Markup Language

GML ist eine auf XML basierende Auszeichnungssprache, welche die Übermittlung von Geometriedaten auch unter Einbeziehung von Sensordaten oder ähnlichen Attributen ermöglicht. Es enthält Primitiven unter Anderem wie Objekt, Geometrie, Koordinatenreferenzsystem und Zeit. Die wichtigsten Objekttypen in GML sind Point, LineString und Polygon, und damit ein identisches Modell wie KML auf. GML wird hauptsächlich genutzt um ein Spektrum von Anwendungsobjekten und deren Eigenschaften zu beschreiben. [1] [3]

2.2 KML - Keyhole Markup Language

KML befolgt, wie GML, die XML-Syntax und wird vorangig zur Visualisierung geographischer Informationen verwendet. KML-Daten können genutzt werden, um GML-Inhalte darzustellen. Die Geometriedaten können sowoh inl Vektor- als auch Rasterform angegeben sein, also bestehend aus grafischen Primitiven wie Linien, Kreisen, Polygonen, oder aus einer rasterförmigen Anordnung von Bildpunkten. [3] [4] [5]

2.3 GeoJSON

GeoJSON wird verwendet, um geografische Daten nach der Simple-Feature-Access-Spezifikation zu repräsentieren und wird in der JavaScript Object Notation angegeben. Zu den unterstützten Geometrien zählen auch hier Punkte, Linien und Polygone. Darüber hinaus können auch mehrteilige Typen der genannten Geometrien angezeigt werden. [2]

3 Architektur

3.1 Modellübersicht

Das Projekt besteht aus folgenden Klassen:

- *GisConn.java*- Erstellung der Verbindung zur OHDM-Datenbank
- *ImportServlet.java*- Entgegennahme der Geomteriedaten des Benutzers über die Webseite
- *DatabaseController*- Trägt Geometriedaten in die einzelnen Tabellen der Datenbank ein
- *GeoJSONController.java*- Zur Umwandlung von GeoJSON-Daten in Post GIS-fähige Geometrien
- *GMLController.java*- Zur Umwandlung von GML-Daten in PostGIS-fähige Geometrien
- *KMLController.java*- Zur Umwandlung von KML-Daten in PostGIS-fähige Geometrien
- *Index.html, Styles.css*- Benutzeroberfläche zur Eingabe der Geometriedaten

3.2 Schnittstellendefinitionen

3.2.1 GisConn.java

Diese Klasse wird genutzt um eine Verbindung zur OHDM-Datenbank zu schaffen. Der Hostname, die Datenbank, das Schema, der User und das Passwort müssen zunächst gesetzt werden. Es gibt eine Funktion setConn() um die

Verbindung zu initialisieren, und eine Funktion closeConn() um die Verbindung wieder zu schließen.

3.2.2 ImportServlet.java

Durch Drücken des Submit-Buttons auf der HTML-Seite werden die eingegebenen Daten an das Servlet weitergeleitet. Das Servlet zieht aus diesem Request den Namen, die Klassifikation, den Gültigkeitszeitraum, das Format und die eigentlichen Geometriedaten heraus. Anschließend wird mittels einer Switch-Case-Anweisung unterschieden welches Format und welche Klassifikation ausgewählt wurde. Abhängig davon wird dann die jeweilige Format-Klasse und -Funktion aufgerufen. Wurden die Geometriedaten am Ende erfolgreich eingefügt, erhält das Servlet die ID der neu eingetragenen Geometrie.

3.3 DatabaseController.java

Diese Klasse enthält die Methoden zur Erstellung des Namens der neuen Geo-Daten und ist verantwortlich für den eigentlichen Tabelleneintrag in der Datenbank. Die Methode addGeoObject() erstellt den Name in der passenden Tabelle für die neue Geometrie und addGeoObjectGeometry() sorgt dafür das ein Eintrag in der Tabelle geoObjectGeometrie gemacht wird.

3.3.1 KMLController.java

Wurden vom Servlet Geometriedaten im Format KML empfangen, werden diese an die Klasse KMLController weitergeleitet. Zur Speicherung in der Datenbank müssen einige Parameter gesetzt werden, dazu gehört die UserID, das IDTarget, das TypeTarget, die IDGeoObjectSource und die ClassificationID. Zunächst wird das TypeTarget mit Hilfe der Funktion setTypeTarget() gesetzt. Die Zahl 1 steht hierbei für einen Point, 2 für einen LineString und 3 für ein Polygon. Das Servlet erhält die eingetragenen Daten des Users, und überprüft diese nun auf das Format. Je nach Format wird ein Objekt der entsprechenden Controller-Klasse erstellt. Das IDTarget wird durch Aufrufen der addKMLGeoObject(), addGMLGeoObject(), addGeoJSONGeoObject() zugewiesen. Anschließend wird ein Objekt der DatabaseController-Klasse initialisiert. Die IDGeoObjectSource wird dem Ergbnis des Aufrufs der addGeoObject() zugeteilt. Zuletzt wird die ID der neu erstellten Geometriedatei gebildet.

3.3.2 GMLController.java

Der Ablauf innerhalb der GML-Controller-Klasse ist gleich zu der des eben beschriebenen KML-Controller. Der Aufbau von GML-Daten ist ähnlich zu der von KML-Daten. Die PostGIS-Funktion ST_GeomFromGML() wird genutzt um die Daten umzuwandeln. Die Daten werden zunächst in der Funktion addGMLObject() auf die Tags "Polygon", "Point" und "Linestring" überprüft und dann an die Funktion addGeometry() weitergeleit, wo sie dann der Datenbank hinzugefügt werden.

3.3.3 GeoJSONController.java

Auch in dieser Klasse wird das gleiche Ablaufschema verwendet, wie in den vorgehenden.

4 Probleme

Zur Umwandlung der Geometriedaten aus den einzelnen Formaten wird ein PostGis-JDBC-Treiber genutzt. Ein häufig aufgetretener Fehler war eine Exception, die geworfen wurde, da die Funktionen ST_GeomFromGeoJSON, ST_GeomFromGML und ST_GeomFromKML nicht existieren. Die ist auf eine fehlerhafte Version des JDBC-Treibers zurück zu führen. Bei Auftreten dieses Fehlers muss sichergestellt werden, dass die Version 1.3.3 genutzt wird.

Desweiteren wird ein PostGreSQL-JDBC-Treiber genutzt. Damit zusammenhängend trat vermehrt ein NoSuchMethodError unter dem Pfad target/Imprts-1.0-SNAPSHOT/WEB-INF/lib/ auf. Dieses Problem konnte, durch Löschung der PostGreSQL-Dependency und des entsprechenden Eintrags in der pom.xml-Datei, behoben werden.

Bei der Nutzung der Funktionen ST_GeomFromGeoJSON, ST_GeomFromGML und ST_GeomFromKML sind nur eine Tags wie "Point", "Coordinates" erlaubt, andere Tags können nicht genutzt werden, da sonst eine Exception geworfen wird. Wichtig ist, dass in das Textfeld der Importer-Webseite nur korrekte Daten, in den zur Verfügung stehenden Formaten, eingetragen werden. Das Servlet kann diese sonst nicht an die Datenbank weiterleiten und richtig einordnen.

5 Nutzung

5.1 Code

Der Code unseres Projekts lässt sich unter dem Master-Branch des folgenden Repositories finden:
https://github.com/OpenHistoricalDataMap/Importer.

Zur Programmierung wurde Java genutzt, und die Webseite wurde mit HTML erstellt.

5.2 Deployment / Runtime

Eine Verbindung zum Geoserver kann nur erstellt werden, wenn sich der PC des Benutzers im Netzwerk der HTW befindet. Sollte das Servlet in einem anderen

Netzwerk installiert werden, muss zunächst eine VPN-Verbindung zu SSL-VPN-HTW erzeugt werden. Es muss die persönliche Nutzer-ID in der ImportServlet-Klasse unter der Variable userID eingetragen werden. Mit Hilfe von Maven muss eine .war-Datei im Target-Ordner des Projekts erstellt werden. Sofern Tomcat genutzt wird, muss nun diese .war-Datei in den webapp-Ordner des Tomcat-Verzeichnisses geschoben werden. Der Server wird jetzt gestartet, dieser startet automatisch das Servlet. Der Importer ist damit bereit zur Nutzung.

6 Qualitätssicherung

Die Qualität der Komponenten wird nicht gesichert. Auf das Erstellen von Tests musste verzichtet werden.

7 Vorschläge / Ausblick

Der von uns entwickelte Importer kann nun noch um einen Exporter erweitert werden. Dateien könnten aus einem bestimmten Kartenabschnitt extrahiert und in der Formaten KML, GML und GeoJSON ausgegeben werden. PostGIS stellt hierfür ähnliche Funktionen zur Verfügung wie die von uns genutzten. Auf der jetzigen Import-Webseite könnte dann in diesem Fall angegeben werden in welchem Format und in welchem Datumsbereich die Daten exportiert werden sollen.

8 Literaturverzeichnis

1. Geography Markup Language. https://de.wikipedia.org/wiki/Geography_Markup_Language (27.09.2018)

2. GeoJSON. https://de.wikipedia.org/wiki/GeoJSON (27.09.2018)

3. Keyhole Markup Language. https://de.wikipedia.org/wiki/Keyhole_Markup_Language(27.09.2018)

4. Rastergraik. https://de.wikipedia.org/wiki/Rastergrafik (27.09.2018)

5. Vektorgrafik. https://de.wikipedia.org/wiki/Vektorgrafik (27.09.2018)