|  |  |
| --- | --- |
|  | tud_logo |
|  | Dokumentation Übung 2 |
|  |  |
|  |  |
|  | 25. Juni 2016  Informatik im Bauwesen II – Gruppe 01  Florian Saumweber, 2354534  Benjamin Krauß, 2388173 |
|  |  |

# Anforderungen an die Software

Die Webanwendung dient als webbasiertes Monitoring-System für Gebäudesanierungen.

Auf einem BIM-Server werden alle Gebäudemodelle abgespeichert und über das webbasierte System zugänglich gemacht. Auf diese Weise wird eine Überwachung des Sanierungsfortschrittes erlaubt. Allerdings werden der Einfachheit halber die Gebäudemodelle lokal auf dem Rechner gespeichert und somit vereinfacht simuliert.

Als Nutzergruppen sind Handwerker und Angestellte des Baudezernats (Bauplaner) vorgesehen, welche nach der Anmeldung vom System spezifische Interaktionsmöglichkeiten bzw. einen unterschiedlichen Funktionsumfang erhalten.

Folgende Daten werden hierbei in der Datenbank verwaltet:

* Sanierungsaufträge
* Handwerker und Bauplaner
* Login-Informationen

Die Realisierung findet in der Programmiersprache Java statt. Mittels der Datenbanksprache SQL ist für die persistente Datenhaltung eine MySQL-Datenbank vorgesehen. Die Datenbankschnittstelle soll über JavaBeans unter der Zuhilfenahme von JDBC erfolgen.

Die Präsentation und Abfrage der Daten wird mit Java Server Pages erstellt. Die Sanierungsfortschritte bzw. die Bauteile eines Gebäudes werden in ifc-Files abgebildet.

# Use-Cases

Die Anwendungsfälle zeigen die Aufgaben, welche vom System übernommen werden sollen.

In *Abbildung 1* wird das Use-Case-Diagramm nach UML-Notation gezeigt. Darin sind z. B. die zwei Nutzergruppen Bauplaner und Handwerker visualisiert.

Einige Use-Cases werden über eine extend-Beziehung erweitert, welche als optionale Erweiterung verstanden wird. Dies bedeutet beispielsweise, dass nach dem Aufruf von Use‑Case *eigene Aufträge einsehen* der Use-Case *Auftragsstatus ändern* ausgeführt werden kann, aber nicht ausgeführt werden muss.



Abbildung 1: Use-Case-Diagramm

# Entity-Relationship-Modell

Die Datenmodellierung innerhalb des Entity-Relationship-Modells (ERM) dient dazu die Anforderungen aufzugreifen und einheitlich zu visualisieren. Sie bildet die Grundlage des Datenbankentwurfes.

Die Entitäten in diesem Modell sind *Beruf, Nutzer* und *Sanierungsauftrag*. Die Verfeinerung durch weitere Attribute ist der *Abbildung 2* zu entnehmen. Die Relationen setzen die Entitäten zueinander in Beziehung, werden durch die Kardinalitäten in der Semantik erweitert und sind in der *Abbildung 2* grün gekennzeichnet.

In der Entität Nutzer sind sowohl die Nutzergruppe Handwerker als auch die Nutzergruppe Bauplaner hinterlegt, da diese sich lediglich durch den Beruf unterscheiden.

Zwischen Nutzer und Sanierungsauftrag besteht eine n:m Beziehung, da z. B. ein Handwerker mehrere Sanierungsaufträge haben kann. Zudem kann ein Sanierungsauftrag von verschiedenen Nutzern betrachtet werden.



Abbildung 2: ERM

# Relationaler Datenbankentwurf

Der relationale Datenbankentwurf leitet sich aus dem ERM ab (siehe *Abbildung 2*).

Die entsprechenden Entitäten mit den zugehörigen Attributen wurden in Beziehung gesetzt und spezifiziert. In *Abbildung 3* ist zu sehen, dass die n:m Beziehung des ERMs mittels der Link‑Tabelle *LnSiehtAn* aufgelöst wurde.

Aus dem in MySQL umgesetzten Modell ergeben sich die konkreten Befehle und Anweisungslisten zur Erstellung der Datenbank. Der SQL-Code zur Generierung der Datenbank inklusive der eingefügten Beispieldaten findet sich in der geforderten .sql‑Datei.

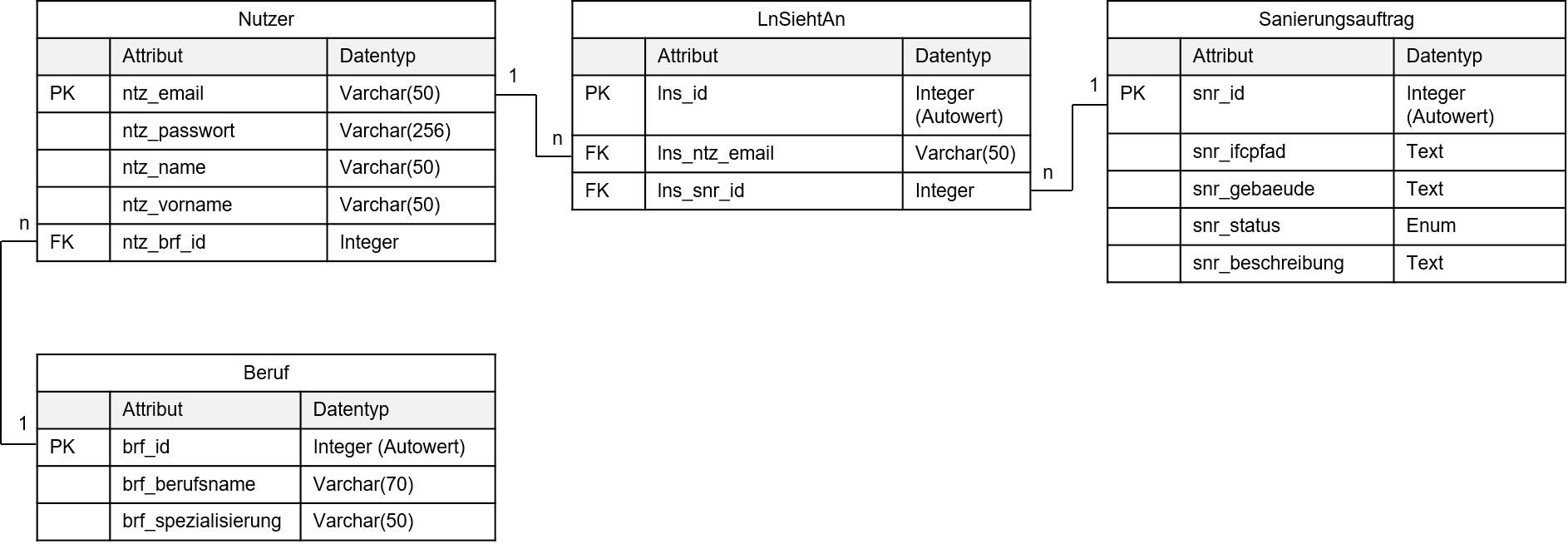


Abbildung 3: Relationaler Datenbankentwurf

# IfcPropertySets zur Annotation von Schadensinformationen

IFC-Files sind der Qusistandard zum Austausch von Daten zwischen BIM-Servern. Das komplexe Austauschformat kann ebenfalls mit Java erstellt werden. Dieser Weg ist jedoch sehr mühsam, weswegen die IfcJavaToolbox in das Projekt eingebunden wurde. Die Library stellt Methoden zur Verfügung, um einfacher ein komplexes Format erstellen zu können. Die CRUD-Operationen wurden im Default-Package im File ManipulateIfcFile.java implementiert. Jedoch existiert für den Update-Befehl nur ein Workaround, das Löschen und Neuhinzufügen. Da das Ifc-Format seine Informationen hierarchisch ablegt, bietet sich eine modulare Implementierung an. Ausschließlich Kernoperationen wurden hierbei öffentlich gekennzeichnet, Hilfsfunktionen verbleiben mit ihrer Sichtbarkeit ausschließlich innerhalb der Klasse. Sie alle sind Objektmethoden. Einzige Ausnahme sind die Delete-Methoden, sie sind static. Um die Funktionalität der Methoden zu verifizieren wurde ein Testfile TestManipulateIfcFile.java geschrieben. Bei positiver Rückmeldung muss das Vorgehen nur in JSP-Seiten implementiert werden. Um die Elementauswahl einfacher zu handhaben, wurden überladene Methoden erstellt welche mit Switch-Afragen arbeiten. Auch wurde insofern möglich mit generics gearbeitet, um Copy-Paste-Code zu vermeiden.

# XML Schema und JAXB Binding für Schadensinformationen

Das XML Schema findet mittels JAXB Binding Anwendung im Projekt. Hierbei wurde ein einfach strukturiertes und schlankes Schema definiert. Siehe nachfolgend:

<?xml version="1.0" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

xmlns="generated"

targetNamespace="generated"

attributeFormDefault="unqualified"

elementFormDefault="qualified">

<xs:element name="auftrag">

<xs:complexType>

<xs:sequence>

<xs:element type="xs:string" name="beschreibung" minOccurs="1" maxOccurs="1"/>

</xs:sequence>

<xs:attribute type="xs:unsignedInt" name="id" use="required"/>

<xs:attribute type="xs:string" name="globallyUniqueId" use="required"/>

<xs:attribute type="xs:string" name="status" use="required"/>

</xs:complexType>

</xs:element>

</xs:schema>

Dieses Schema offeriert für einen Auftrag eine Struktur wie in dargestellt. Die Id ist der Auftragsnummer gleichzusehen. Das Metadatum GloballyUniqueId ist diejenige eindeutige Id aus dem Ifc-Modell. Dadurch kann ein Auftrag nicht nur auf Raumebene zugeordnet, sondern sogar auf Elementebene angelegt werden. Der Status des Auftrags wird über das gleichnamige Attribut gepflegt. Nach diesen drei Metadaten kommt der Hauptteil des Files. Er repräsentiert die Auftragspositionen. Der Einfachheitshalbe wurde lediglich ein Text-Attribut Beschreibung verwendet, welches anstatt strukturierten Positionsdaten, diese in einem Fließtext ausgibt. Damit das XML-File Informationen enthält, wurden die Daten allesamt als verpflichtend gekennzeichnet. Die jeweiligen Datentypen wurden denjenigen des RDM (siehe ) angepasst.

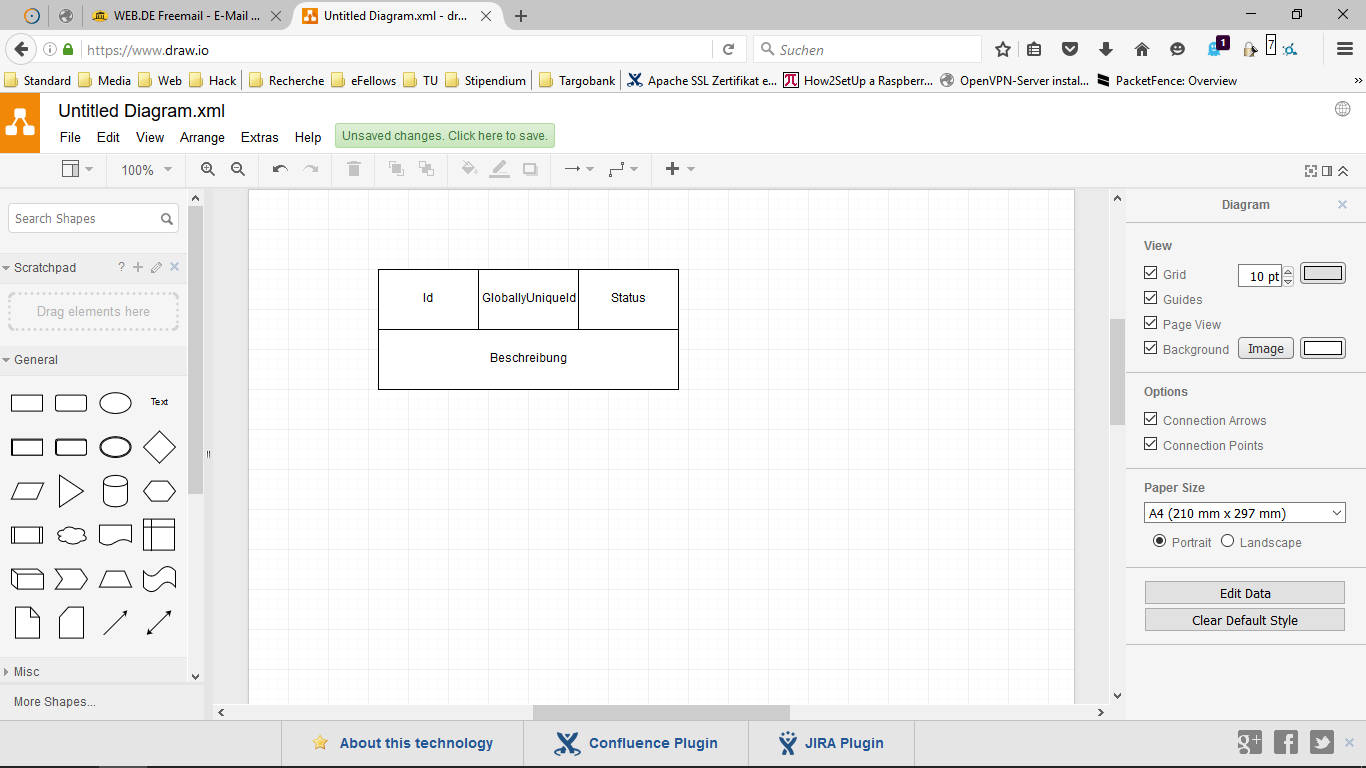


Abbildung : Struktur eines XML-Files mittels entworfenen Schema

Da Netbeans die Generierung von JAXB Bindings unterstützt, wurde mittels dem Schema ein Binding mit getter- und setter-Methoden extrahiert. Sie befinden sich im Project-Package <<generated>>. Allerdings erfordern CRUD-Operationen ein tiefgründigeres Zusammenspiel einzelner Operationen, wie auch Standardroutinen zum Laden und Speichern. Diese Logik wurde im Default-Package der Klasse ManipulateXmlFile einverleibt. Wurde ein Auftrag in Java manipuliert, so wurde dieser im alten File gespeichert. War allerdings die Id noch nicht vorhanden, so wird ein neues File erstellt. Denn die AuftragsId ist gleichzeitig der Dateiname.

# Konzept und Umsetzung der Webanwendung

Konzeptionell wird das Projekt in verschieden Aufgabenbereiche unterteilt:

Es ist eine Logik zur Erstellung von dynamischen Webseiten vorgesehen. Java Server Pages (JSP) sind HTML Seiten mit eingebettetem Java Code, die die grafische Benutzeroberfläche erzeugen. Der Nutzer hat im Webbrowser die Möglichkeit zur Interaktion und kommt daher mit dieser Logik direkt in Berührung.

Der Datenbank-spezifische Code regelt die Zugriffe auf die Datenquelle, um die in der Datenbank hinterlegten Beispieldaten für den Nutzer zugänglich zu machen. Dabei handelt es sich z. B. um Methoden, die SQL Befehle ausführen können oder auch die Verbindungen aufbauen können. Die JSP-Seiten greifen daher auf Data Access Objects zurück, in denen der Datenbank-spezifische Code integriert ist.

Eine dritte Komponente stellen Container dar, die zur Datenübertragung genutzt werden und eine korrekte Datenkapselung sicherstellen sollen. Diese nennen sich JavaBeans.

Die Zugangsdaten zum MySQL Server sind in der Klasse iib2\_gruppe1.DbConnection.java hinterlegt. Ein Ausschnitt ist im Folgenden zu sehen:

public class DbConnection {

public static Connection connection;

private DbConnection dbConnection;

private final String driverName;

private final String serverName;

private final String portNumber;

private final String database;

private final String uri;

private final String username;

private final String password;

public DbConnection() {

this.connection = null;

this.driverName = "com.mysql.jdbc.Driver";

this.serverName = "localhost";

this.portNumber = "3306";

this.database = "iib2\_ue2\_gruppe01";

this.uri = "jdbc:mysql://" + serverName + ":" + portNumber + "/" + database;

this.username = "gruppe1";

this.password = "iib2";

this.connect();

}

Die Data Access Objects im Package beanDao erben von DbConnection, damit alle nötigen Daten für die Verbindung mit der Datenbank verfügbar sind. Dort sind die für die Abfragen nötigen SQL-Befehle hinterlegt, was am Beispiel der BerufDao.java Klasse zu sehen ist:

public class BerufDao extends DbConnection {

public List<Beruf> select() {

PreparedStatement preparedStatement = null;

ResultSet resultSet = null;

List<Beruf> returnList = new ArrayList<Beruf>();

String statement = "select \* from beruf;";

try {

preparedStatement = this.connection.prepareStatement(statement);

resultSet = preparedStatement.executeQuery();

while (resultSet.next()) {

Beruf beruf = new Beruf();

beruf.setBrfId(resultSet.getInt("brf\_id"));

beruf.setBrfBerufname(resultSet.getString("brf\_berufname"));

beruf.setBrfSpezialisierung(resultSet.getString("brf\_spezialisierung"));

returnList.add(beruf);

}

} catch (SQLException e) {

System.out.println("SQLException: " + e.getMessage());

}

return returnList;

}

Die Daten werden bei den Abfragen in Instanzen der entsprechenden Klassen übertragen, die innerhalb des Packages bean zu finden sind. Im obigen Beispiel wurde in der BerufDao.java‑Klasse auf die Beruf.java‑Klasse als Bean zurückgegriffen.

Die grafische Benutzeroberfläche entsteht durch ein Zusammenspiel von HTML‑ und Java‑ Anweisungen innerhalb der JSPs, die untereinander agieren.

Ein Ausschnitt von dashboard.jsp verdeutlicht dies. Hier werden zum einen Variablenwerte in Java‑Code innerhalb einer if-Abfrage überprüft und zum anderen entsprechende Verlinkungen zu anderen JSP-Seiten in HTML umgesetzt:

33 <%

34 if(session.getAttribute("nutzerArt") == "Handwerker"){

35 %>

36

37 <h1>Einsehen</h1>

38 <ul>

39

40 <li><a href="./seeEntity.jsp?entity=eigeneAuftraege" title="eigene Aufträge einsehen">Eigene Aufträge</a></li>

41 <li><a href="./searchGebaeude.jsp" title="Aufträge pro Gebäude einsehen">Aufträge eines Gebäudes</a></li>

42

43 </ul>

44

45 <%

46 }

47 %>

# Erläuterungen zur graphischen Benutzeroberfläche

Im Beispieldatendatensatz wurden verschiedene Benutzergruppen hinterlegt, die sich mit ihrer E‑Mailadresse und ihrem Kennwort in der Applikation anmelden können. An dieser Stelle seien einige Anmeldedaten aufgeführt:

Handwerker:

|  |  |
| --- | --- |
| E-Mailadresse: | Passwort: |
| karl.hofmann@maler.de | karl |
| fritz.fischer@wassertechnik.de | fritz |

Bauplaner:

|  |  |
| --- | --- |
| E-Mailadresse: | Passwort: |
| julia.meier@baudezernat.de | julia |
| max.schulz@baudezernat.de | max |

Der User trifft zu Beginn auf folgende grafische Benutzerfläche, bei der er sich mit seiner E‑Mailadresse und seinem Passwort anmelden kann (*siehe Abbildung 4*).



Abbildung 5: Login

Das Dashboard gibt je nach Benutzergruppe einen Überblick über die verfügbaren Interaktionsmöglichkeiten (*siehe Abbildung 5 und 6*):



Abbildung 6: Dashboard Handwerker



Abbildung 7: Dashboard Bauplaner

Im Dashboard des Bauplaners wurde eine weitere Funktionalität hinterlegt: Das Einsehen aller Nutzer. So hat der Bauplaner die Möglichkeit einen Überblick über die Nutzer zu erlangen. Dabei werden auch neu angelegte Handwerker visualisiert.

In *Abbildung 7* sehen Sie Maske zum Anlegen eines neuen Handwerkers. Unter Beruf erscheint ein Dropdown-Menü mit den in der Datenbank angelegten Berufen der Handwerker. Daraus sollte die passende Tätigkeit ausgesucht werden. Ob das Einfügen des neuen Datensatzes erfolgreich war, wird anschließend in der ersten Zeile eingeblendet. In *Abbildung 7* ist zu sehen, dass ein neuer Nutzer Handwerker erfolgreich angelegt wurde.

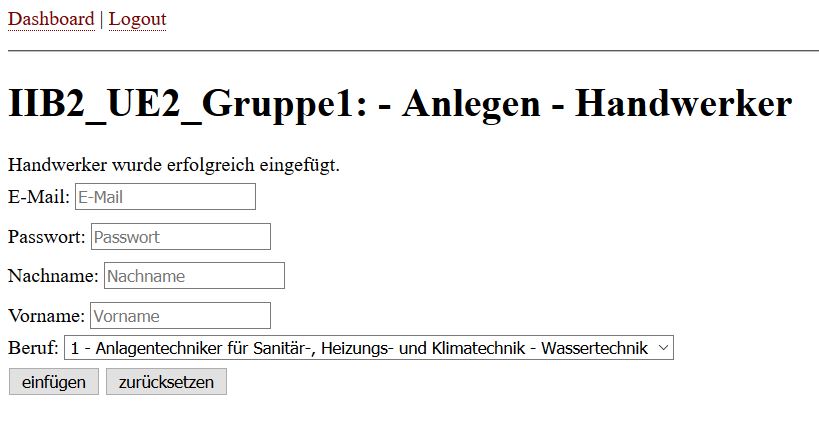


Abbildung 8: Anlegen eines neuen Handwerkers

Die Nutzergruppe Handwerker kann über das Dashboard alle eigenen Aufträge einsehen. Mittels des Bearbeiten-Buttons kann der Status eines eigenen Auftrags geändert werden (siehe *Abbildung 8*). In der MySQL‑Datenbank wurde dieser Parameter in einer Enumeration hinterlegt, welche nur die Werte eingegangen, handwerker\_beauftragt, erledigt und reklamiert annimmt. Falls es einen Fehlerfall geben sollte, wird der Nutzer darauf hingewiesen.



Abbildung 9: Status eines Auftrags ändern

Der Handwerker kann ebenfalls nach Aufträgen suchen, die alle zum gleichen Gebäude gehören. Die Suchmaske mit einer Beispieleingabe Wohnhaus spiegelt sich in *Abbildung 9* wider.



Abbildung 10: Suchmaske für Sanierungsaufträge eines Gebäudes