



TECHNISCHE UNIVERSITÄT CHEMNITZ

Softwarepraktikum 2007

Gruppe 12

HVS - Hotelverwaltungssystem

TEILBELEG 2

Betreuer Ulrich Halfter

Projektleiter Ken Schmidt
Tolleiv Nietsch
Tobias Doerffel
Norman Paschke
Heiner Reinhardt
Marco Rose

Chemnitz, 4. Juli 2008



Inhaltsverzeichnis

1	Systemarchitektur des Projekts	5
2	Klassenspezifikation des Projekts	6
2.1	Nutzerverwaltung	6
2.1.1	Klasse: appNutzer	6
2.2	Gäste	8
2.2.1	Klasse: appHotelgast	8
2.2.2	MySQL STORED PROCEDURES	8
2.3	Buchungen	9
2.3.1	Klasse: appBuchung	9
2.4	Zusatzleistungen	11
2.4.1	Klasse: appLeistung	11
2.5	Tarife	12
2.5.1	Klasse: appTarif	12
2.5.2	MySQL STORED PROCEDURES	13
2.6	Kategorien	15
2.6.1	Klasse: appKategorie	15
2.7	Zimmer	16
2.7.1	Klasse: appZimmer	16
2.8	Grundeinstellungen	18
2.8.1	Klasse: appSettings	18
2.9	Logging	19
2.9.1	Klasse appLog	19
3	Allgemeine Vereinbarungen	20
3.1	Datenintegrität	20
3.2	Graphical User Interface	21
3.2.1	Klasse: Ui_<dialogName>	21
3.2.2	Template-Klasse: oneClickEditView	21
3.2.3	Template-Klasse: modelHelper	22

1 Systemarchitektur des Projekts

Abbildung 1 auf Seite 5 zeigt den Entwurf der Systemarchitektur.

Aus Gründen der Übersichtlichkeit wurden die Module für Zugriff und Verwaltung von Systemeinstellungen, für Logging und Fehlerbehandlung sowie notwendige Qt-Abstraktionsebenen nicht im Diagramm dargestellt.

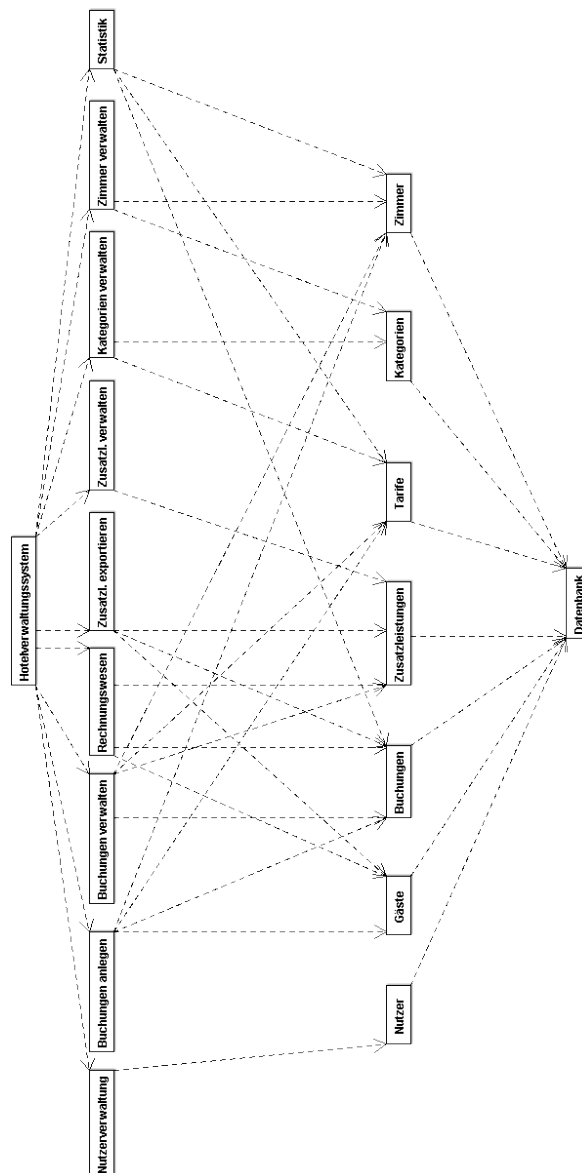


Abbildung 1: Systemarchitektur

2 Klassenspezifikation des Projekts

In der folgenden Klassenspezifikation wird auf Methoden eingegangen, die vor allem in Bezug auf den Informationsaustausch zwischen den Modulen relevant sind. Funktionalitäten zum Erstellen, Modifizieren oder Löschen von Datensätzen werden als Modul-interne Aufgaben behandelt.

2.1 Nutzerverwaltung

2.1.1 Klasse: appNutzer

Bearbeiter:	Marco
Kurzbeschreibung:	stellt Methoden für den Umgang mit Systemnutzern zur Verfügung, sichert das stets ein gültiger Systemnutzer eingeloggt ist
Export:	tryLogonUser, isUserLoggedOn, isAdmin, getUsername
Hinweis:	Klasse steht als Singleton im System zur Verfügung.

tryLogonUser

Typ:	Methode
Parameter:	username - const QString & - Nutzerkennzeichen des anzumeldenden Nutzers
	password - const QString & - Passwort des anzumeldenden Nutzers
Rückgabotyp:	bool
Beschreibung:	Prüft, ob die Kombination aus Nutzerkennzeichen und Passwort gültig ist. Im Falle des Erfolgs werden intern entsprechende Daten manipuliert und <i>true</i> zurückgegeben, ansonsten <i>false</i> .

isUserLoggedIn

Typ:	Methode
Parameter:	void
Rückgabetyt:	bool
Beschreibung:	Überprüft, ob ein Nutzer angemeldet ist und gibt entsprechend <i>true</i> oder <i>false</i> zurück.

isAdmin

Typ:	Methode
Parameter:	void
Rückgabetyt:	bool
Beschreibung:	Überprüft, ob ein Nutzer angemeldet ist und wenn ja, ob er Administratorrechte besitzt. Wenn beides der Fall ist, wird <i>true</i> zurückgegeben, sonst <i>false</i> .

getUsername

Typ:	Methode
Parameter:	void
Rückgabetyt:	QString
Beschreibung:	Gibt das Nutzerkennzeichen des angemeldeten Nutzers zurück, wird vor allem für evt. Log-Funktionen bereitgestellt.

2.2 Gäste

2.2.1 Klasse: appHotelgast

Bearbeiter:	Ken
Kurzbeschreibung:	Gültigkeit/ Prüfung von „Stammkunden“
Export:	search

search

Typ:	Methode
Parameter:	knr - QString - Kundennummer
Rückgabetyt:	int
Beschreibung:	Prüft eine Kundennummer und gibt bei Erfolg die Datensatz-ID (sonst 0) zurück.

2.2.2 MySQL STORED PROCEDURES

gaststat

Typ:	Funktion
Parameter:	paramgast - int - Id des Gastes
Rückgabetyt:	int
Beschreibung:	Gibt die Anzahl der Besuche eines Gastes zurück.
Anwendungsbeispiel:	SELECT *,gaststat(id) FROM gaeste; – der Datenbank zu Liebe bitte so nicht ausführen –

2.3 Buchungen

2.3.1 Klasse: appBuchung

Bearbeiter:	Ken
Kurzbeschreibung:	gibt Auskunft über vorhandene Buchungen
Export:	getStatus, getPreisTotal, getGast, getZimmer, getLeistungen
Import:	appHotelgast, appZimmer, appTarif, appLeistung

getStatus

Typ:	Methode
Parameter:	void
Rückgabetyt:	QVariant (numerisch)
Beschreibung:	Gibt den Status (siehe Datenbankentwurf) einer Buchung zurück.

getPreisTotal

Typ:	Methode
Parameter:	void
Rückgabetyt:	QVariant (numerisch)
Beschreibung:	Gibt den Gesamtpreis einer Buchung zurück.

getGast

Typ:	Methode
Parameter:	void
Rückgabetyt:	int
Beschreibung:	Gibt die Referenz-ID eines Gastes zurück.

2.3 Buchungen

getZimmer

Typ:	Methode
Parameter:	void
Rückgabetyt:	QList<int>
Beschreibung:	Gibt eine Liste der referenzierten Zimmer (bzw. die ID's) zurück.

getLeistungen

Typ:	Methode
Parameter:	void
Rückgabetyt:	QList<int>
Beschreibung:	Gibt eine Liste der referenzierten Zusatzleistungen zurück.

2.4 Zusatzleistungen

2.4.1 Klasse: appLeistung

Bearbeiter:	Tolleiv
Kurzbeschreibung:	gibt Auskunft über vorhandene Zusatzleistungen.
Export:	getName, getPreis

getName

Typ:	Methode
Parameter:	void
Rückgabetyt:	QString
Beschreibung:	Gibt die Bezeichnung einer Zusatzleistung zurück.

getPreis

Typ:	Methode
Parameter:	void
Rückgabetyt:	QVariant (double)
Beschreibung:	Gibt den Preis einer Zusatzleistung zurück.

getPreis

Typ:	Methode
Parameter:	count - int - Anzahl
Rückgabetyt:	QVariant (double)
Beschreibung:	Gibt den Preis einer Zusatzleistung unter Berücksichtigung der Mengenangabe zurück - vor allem notwendig wenn (derzeit nicht zu implementierende) Mengenrabatte gewährt werden.

2.5 Tarife

2.5.1 Klasse: appTarif

Bearbeiter:	Tolleiv
Kurzbeschreibung:	Tarifhandling
Export:	isValid, getKategorie
Import:	appKategorie

isValid

Typ:	Methode
Parameter:	now - QDateTime
Rückgabetyt:	bool
Beschreibung:	Gibt die Gültigkeit eines Tarifes für den übergebenen Zeitpunkt zurück.

isValid

Typ:	Methode
Parameter:	void
Rückgabetyt:	bool
Beschreibung:	siehe isValid(QDateTime) - es wird für die aktuelle Systemzeit geprüft.

getKategorie

Typ:	Methode
Parameter:	void
Rückgabetyt:	int
Beschreibung:	Gibt die Referenz-ID der Kategorie eines Tarifes zurück bzw. 0 bei Standard-Tarif.

2.5.2 MySQL STORED PROCEDURES

gettarif

Typ:	Funktion
Parameter:	paramkat - int - Id der Kategorie
	paramnow - datetime - Abfragezeitpunkt / Gültigkeitszeitpunkt
Rückgabetyt:	int
Beschreibung:	Gibt die ID des für die Kategorie gültigen Tarifes für den Zeitpunkt unter Berücksichtigung eines eventuellen Fallbacks auf einen Standard-Tarif zurück.
Anwendungsbeispiel:	SELECT *,gettarif(id,now()) FROM kategorie;

getpreis_pro_tag

Typ:	Funktion
Parameter:	paramkat - int - Id der Kategorie
	paramnow - datetime - Abfragezeitpunkt / Gültigkeitszeitpunkt
Rückgabetyt:	float
Beschreibung:	Gibt den Tagespreis des für die Kategorie gültigen Tarifes für den Zeitpunkt unter Berücksichtigung eines eventuellen Fallbacks auf einen Standard-Tarif zurück.
Anwendungsbeispiel:	SELECT *,getpreis_pro_tag(id,now()) FROM kategorie;

getbrutto

Typ:	Funktion
Parameter:	paramnetto - float - umzuwandelnder Nettopreis
Rückgabetyt:	float
Beschreibung:	Berechnet den Brutto- Preis mit Hilfe des globalen MwSt-Satzes.
Anwendungsbeispiel:	SELECT *,getbrutto(getpreis_pro_tag(id,now())) FROM kategorie;

convertnetto

Typ:	Funktion
Parameter:	parambrutto - float - umzuwandelnder Bruttopreis
Rückgabetyt:	float
Beschreibung:	Berechnet den Netto- Preis mit Hilfe des globalem MwSt-Satzes.
Anwendungsbeispiel:	SELECT convertbrutto(<preis>);

2.6 Kategorien

2.6.1 Klasse: appKategorie

Bearbeiter:	Tolleiv
Kurzbeschreibung:	gibt Auskunft über vorhandene Kategorien und deren Eigenschaften
Export:	getName, getFeaturelist

getName

Typ:	Methode
Parameter:	void
Rückgabetyt:	QString
Beschreibung:	Gibt die Kategoriebezeichnung zurück.

getFeaturelist

Typ:	Methode
Parameter:	void
Rückgabetyt:	QMap<QString, QVariant>
Beschreibung:	Gibt eine Liste aller Kategorie-Parameter und deren Status als geordnete Liste zurück.

2.7 Zimmer

2.7.1 Klasse: appZimmer

Bearbeiter:	Norman
Kurzbeschreibung:	gibt Auskunft über vorhandene Zimmer und deren Status
Export:	getNummer, getStatus
Import:	appKategorie, appBuchung

getNummer

Typ:	Methode
Parameter:	void
Rückgabetyt:	QString
Beschreibung:	Gibt die „echte“ Zimmernummer als Kombination von Etage und Zimmernummer zurück.

getStatus

Typ:	Methode
Parameter:	now - QDateTime - zu prüfender Zeitpunkt
Rückgabetyt:	QVariant (numerischer Inhalt)
Beschreibung:	Gibt den Status eines Zimmer zu einem durch den Parameter bestimmten Zeitpunkt an. (0^ = frei, 1^ = belegt, 2^ = gesperrt, ...)

getStatus

Typ:	Methode
Parameter:	void
Rückgabetyt:	QVariant (numerischer Inhalt)
Beschreibung:	siehe getStatus (QDateTime) - es wird mit der aktuellen Systemzeit gearbeitet.

getStatus

Typ:	Methode
Parameter:	beginn - QDateTime
	ende - QDateTime
Rückgabetyt:	QVariant (numerischer Inhalt)
Beschreibung:	siehe getStatus (QDateTime) - geprüft wird für den durch die beiden Parameter eingegrenzten Zeitraum.

2.8 Grundeinstellungen

2.8.1 Klasse: appSettings

Bearbeiter:	Tobias
Kurzbeschreibung:	stellt systemübergreifend die Möglichkeit zur dauerhaften Speicherung von Einstellungen zur Verfügung.
Export:	setValue, getValue

setValue

Typ:	Funktion
Parameter:	handle - QString - Schlüssel der Einstellung, die geschrieben werden soll.
	value - QVariant - Wert der Einstellung.
	typ - QString - Grundtyp der Einstellung mögliche Werte sind „Int“, „Float“ und „Text“. Dieser Parameter kann vernachlässigt, wenn numerische Werte keiner erhöhten Genauigkeit bedürfen.
Rückgabetyt:	void
Beschreibung:	Speichert einen Grundeinstellungswert zur dauerhaften Verwendung.

getValue

Typ:	Funktion
Parameter:	handle - QString - Schlüssel der Einstellung, die ausgelesen werden soll.
	defaultValue - QVariant - Standardwert der zurückgegeben werden soll, wenn noch kein gespeicherter Wert vorhanden ist.
Rückgabetyt:	QVariant
Beschreibung:	Liest einen Wert der Grundeinstellungen aus und gibt ihn als QVariant zurück.

2.9 Logging

2.9.1 Klasse appLog

Bearbeiter:	Tobias
Kurzbeschreibung:	stellt systemübergreifend die Möglichkeit zur Aufzeichnung von Anwendungs-Ereignissen zur Verfügung.
Export:	write

write

Typ:	Funktion
Parameter:	id - QString - frei wählbare Rückverfolgungs- oder ModulID
	message - QString - Log-Eintrag
Rückgabetyt:	void
Beschreibung:	Hängt die Meldung an das aktuelle Anwendungs-Log.

3 Allgemeine Vereinbarungen

3.1 Datenintegrität

referenceCount<Tabelle>

Typ:	Funktion
Parameter:	paramid - int - Id des Datensatzes
Rückgabetyt:	int
Beschreibung:	Zählt die Referenzen auf einen Datensatz. Es werden nur Datensätze mit „deleted=0“ gezählt. Diese STORED PROCEDURE sollte für jede Tabelle zur Verfügung stehen.

referenceFullCount<Tabelle>

Typ:	Funktion
Parameter:	paramid - int - Id des Datensatzes
Rückgabetyt:	int
Beschreibung:	siehe „referenceCount“ - zusätzlich werden auch als gelöscht gekennzeichnete Datensätze berücksichtigt.

3.2 Graphical User Interface

3.2.1 Klasse: `Ui_ <dialogName>`

Dialogteilobjekte

Bezeichnung:	<code>setUpUi</code>
Typ:	Methode
Parameter:	* <code><dialogName></code> - <code>QDialog</code> - Elternobjekt
Rückgabetyt:	<code>void</code>
Beschreibung:	Diese Funktion stellt einzelne Teilelemente (durch Vererbung an weitere Klassen) einem Elternelement zur Verfügung. GUI und Funktionalität werden daher getrennt. Eine solche Klasse existiert für fast jeden Dialog im Hotelverwaltungssystem.

3.2.2 Template-Klasse: `oneClickEditView`

Bearbeiter:	Tobias
Kurzbeschreibung:	<code>oneClickEditView</code> ist eine Erweiterung, die mittels Template-Instantiierung auf beliebige Qt-Views angewendet werden kann. Die Datenfelder einer <code>oneClickEditView</code> werden mit nur einem Klick (statt Doppelklick) in den Editor-Modus versetzt. Bei Verwendung eines <code>CheckboxDelegates</code> in der View wird außerdem sofort deren Zustand gewechselt.
Export:	<code>oneClickEditView</code> (Konstruktor)

`oneClickEditView` (Konstruktor)

Typ:	Funktion
Parameter:	<code>view</code> - Referenz auf Zeiger auf Template-Parameter - Zeiger auf zu ersetzende View-Komponente
Rückgabetyt:	-
Beschreibung:	Ersetzt die per Zeiger übergebene View-Komponente durch seine eigene Instanz und löscht die alte View, wobei die Klasse vom Template-Parameter (also der View) abgeleitet ist, um die <code>mousePressEvent</code> -Methode zu überladen.

3.2.3 Template-Klasse: modelHelper

Bearbeiter:	Tobias
Kurzbeschreibung:	modelHelper ist eine Erweiterung, die mittels Template-Instantiierung auf beliebige Qt-Models angewendet werden kann. Mit Hilfe dieser Klasse ist es möglich, noch flexibleren Code zu schreiben, der für die Manipulation von Daten (auf dem Model) keine hart-codierten Zeilen-/Spaltenindizes verwendet sondern stattdessen ausschließlich über Zeichenketten arbeitet. Die Zeichenketten sind hierbei in der Regel Spaltennamen von Datenbank-Tabellen die über QSqlTableModel o.ä. verwendet werde.
Export:	section, updateSectionName

modelHelper (Konstruktor)

Typ:	Funktion
Parameter:	parent - QObject * - Zeiger auf Elternelement
Rückgabetyt:	-
Beschreibung:	Initialisierung der Klasse

section

Typ:	Funktion
Parameter:	header_data - const QVariant & - konstante Referenz auf gesuchtes Datum (i.d.R. eine Zeichenkette)
	orientation - Qt::Orientation - in horizontalen oder vertikalen Kopfdaten suchen?
Rückgabetyt:	int
Beschreibung:	Sucht in den Kopfdaten (Spalten-/Zeilenbeschriftungen) nach gegebenem Datum und gibt den Index (n-te Spalte/Zeile) zurück oder -1, wenn nicht gefunden.

updateSectionName

Typ:	Funktion
Parameter:	<i>old_name</i> - const QString & - konstante Referenz auf alte (zu ersetzende) Spalten-/Zeilenbeschriftung
	<i>new_name</i> - const QString & - konstante Referenz auf neue Spalten-/Zeilenbeschriftung
	<i>orientation</i> - Qt::Orientation - in horizontalen oder vertikalen Kopfdaten suchen?
Rückgabebetyp:	bool
Beschreibung:	Ersetzt den Namen der Spalte/Zeile (je nach <i>orientation</i>), die den bisherigen Namen <i>old_name</i> hatte durch <i>new_name</i>