Hochschule für Technik, Wirtschaft und Kultur Leipzig (FH)
Fachbereich Informatik, Mathematik und Naturwissenschaften

Softwarepraktikum - Gruppe 3

Entwurfsdokument

Leipzig, 30. April 2007

„Architecture is basically a container of something.
I hope they will enjoy not so much the teacup, but the tea.“
Yoshio Taniguchi

Vorgelegt von:
Yves Adler (05IN)
Marco Aust (05IN)
Franz-Sebastian Lorenz (05IN)
Katharina Schewzow (05IN)
Manuel Unglaub (05IN)
Nico Wagner (05IN)
Unterstützt von:
Matthias Jauernig (06INM)
Lehrveranstaltung: Softwarepraktikum / Projektmanagement-Praktikum
Verantwortlicher Professor: Prof. Dr. rer.nat. Karsten Weicker

Inhaltsverzeichnis

1 Überblick **1**

1.1 Einflussfaktoren 1

1.1.1 Einsatzbedingungen 1

1.1.2 Umgebungs- und Randbedingungen 1

1.1.3 Nichtfunktionale Produkt- und Qualitätsanforderungen 1

1.2 Grundsatzentscheidungen 1

1.2.1 Datenhaltung 2

1.2.2 Verteilung im Netz 2

1.2.3 Benutzungsoberfläche 2

1.2.4 Hilfesystem 2

2 Software-Architektur **3**

2.1 3-Schichten-Architektur 3

2.2 Klassendiagramm 4

2.3 ERM-Diagramm 5

3 Feinstruktur der Komponenten **6**

3.1 wepromato.servlet Paket 6

3.1.1 WepromatoServlet 7

3.1.2 AdminServlet 7

3.1.3 alle anderen Servlets 7

3.2 wepromato.db Paket 8

3.2.1 DBManager 9

3.2.2 DBLogger 10

3.3 wepromato.system Paket 11

3.3.1 «Interface» Logger 12
3.3.2 User 12
3.4 wepromato.system.exception Paket 14
3.4.1 CreateUserException 14
3.4.2 LoginException 14
3.4.3 InvalidLoginException 14
3.4.4 InvalidPasswordException 14
3.4.5 InvalidGroupException 14
3.5 wepromato.date Paket 15
3.5.1 WepromatoDate 16
3.5.2 WepromatoDateImpl 16
3.5.3 ProjectDate 16
3.5.4 Milestone 17
3.5.5 CustomMilestone 18
3.5.6 News 19
3.6 wepromato.year Paket 20
3.6.1 Year 21
3.7 wepromato.account Paket 23
3.7.1 Account 24
3.7.2 Tutor 25
3.7.3 Student 26
3.7.4 ProjectLeader 27
3.7.5 QualityManager 27
3.7.6 TeamMember 28
3.8 wepromato.project Paket 29
3.8.1 Project 30

3.8.2 WorkPackage 32
3.8.3 WorkingHours 34
3.8.4 Problem 35
3.8.5 TeamMood 36
3.8.6 TeamMemberRole 37
3.8.7 Vacation 38
3.9 Exemplarische Sequenzdiagramme 39

4 Schnittstellen **43**
4.1 Import / Export von Jahrgängen 43
4.2 Projektplanungsalgorithmus 45
4.2.1 Methoden der Hilfsklasse 46

A Code-Style-Guide **48**
A.1 Einleitung 48
A.2 Java Code Style 48
A.2.1 Einrückung 48
A.2.2 Klammersetzung 48
A.2.3 Leerzeilen und Leerzeichen 48
A.2.4 Reihenfolge der Elemente einer Klasse 50
A.2.5 Namensgebung 50
A.2.6 Kommentare 51
A.2.7 Beispiele 51
A.3 CSS Code Style 53
A.3.1 Allgemeines 53
A.3.2 Einrückung / Leerzeilen 53
A.3.3 Namensgebung 53

A.3.4 Kommentare 54

A.3.5 Beispiel 54

A.4 Verzeichnisstruktur 55

1 Überblick

1.1 Einflussfaktoren

In diesem Abschnitt werden Einflussfaktoren der zu entwerfenden Software-Architektur dargestellt.

1.1.1 Einsatzbedingungen

Es handelt sich um eine webbasierte Multi-User-Anwendung, welche sequenziell arbeitet. Die daraus entstehenden Forderungen bezüglich der Mehrbenutzerverwaltung sind eingearbeitet.

1.1.2 Umgebungs- und Randbedingungen

Durch den Einsatz von Java (ab Version 5) wird das System Plattform- und Datenbanksystemunabhängig, wobei aus Kostengründen MySQL zu empfehlen ist. Konkret wird das System auf den Rechnern des Fachbereichs zum Einsatz kommen, auf denen Apache Tomcat 6.x und ein MySQL-Server 5.x läuft. Als Benutzerschnittstelle wird ein Webbrowser (Mozilla Firefox 1.5+, Internet Explorer ab 5.0) verwendet.

1.1.3 Nichtfunktionale Produkt- und Qualitätsanforderungen

Für die Authentifizierung der Nutzer sind HTTPS-Verbindungen vorgesehen, welche der Apache Tomcat Server unterstützt. Passwörter werden als SHA-1 Hash verschlüsselt in der Datenbank gespeichert, um die Passwortsicherheit zu gewährleisten. Die Berechnung von SHA-1-Hashes wird durch Java Security-Packages erfolgen. Desweiteren wird die PrepardStatement-Klasse verwendet, um SQL-Injektionen zu verhindern.

1.2 Grundsatzentscheidungen

Hier sind alle wichtigen Entscheidungen aufgeführt, die beim Entwurf der Software berücksichtigt werden sollen.

1.2.1 Datenhaltung

Um Daten langfristig zu speichern, wird eine relationale Datenbank (RDBMS) benutzt. Die Datenhaltung erfolgt auf dem Datenbankserver.

1.2.2 Verteilung im Netz

Wepromota kann in einem lokalen Netzwerk oder über das Internet zum Einsatz kommen, da es die Web-Architektur zur Kommunikation benutzt.

1.2.3 Benutzungsoberfläche

Die Benutzeroberfläche wird durch JavaServer Pages beziehungsweise Servlets (Apache Tomcat) bereitgestellt.

1.2.4 Hilfesystem

Es wird eine Online-Hilfe zur Verfügung gestellt, die für die wichtigsten Seiten die Bedienung erklärt. Desweiteren wird ein Benutzerhandbuch angeboten/bereitgestellt.

2 Software-Architektur

2.1 3-Schichten-Architektur

Das System wird in Form einer Drei - Schichten - Architektur implementiert werden.

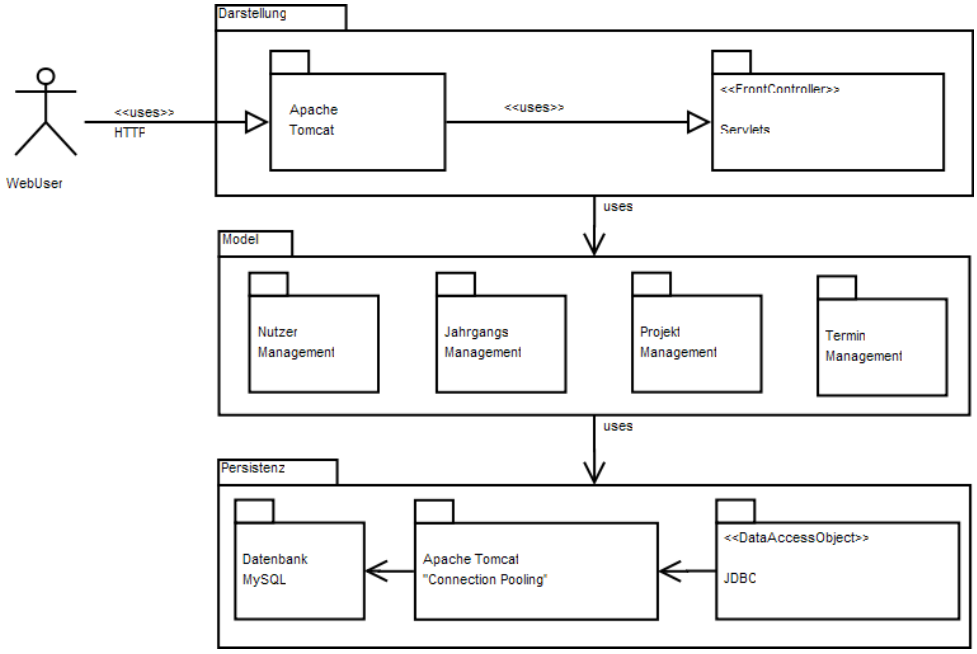

Abbildung 1: Systemüberblick

2.2 Klassendiagramm

Dieses Klassendiagramm zeigt die Grobstruktur der Anwendung.

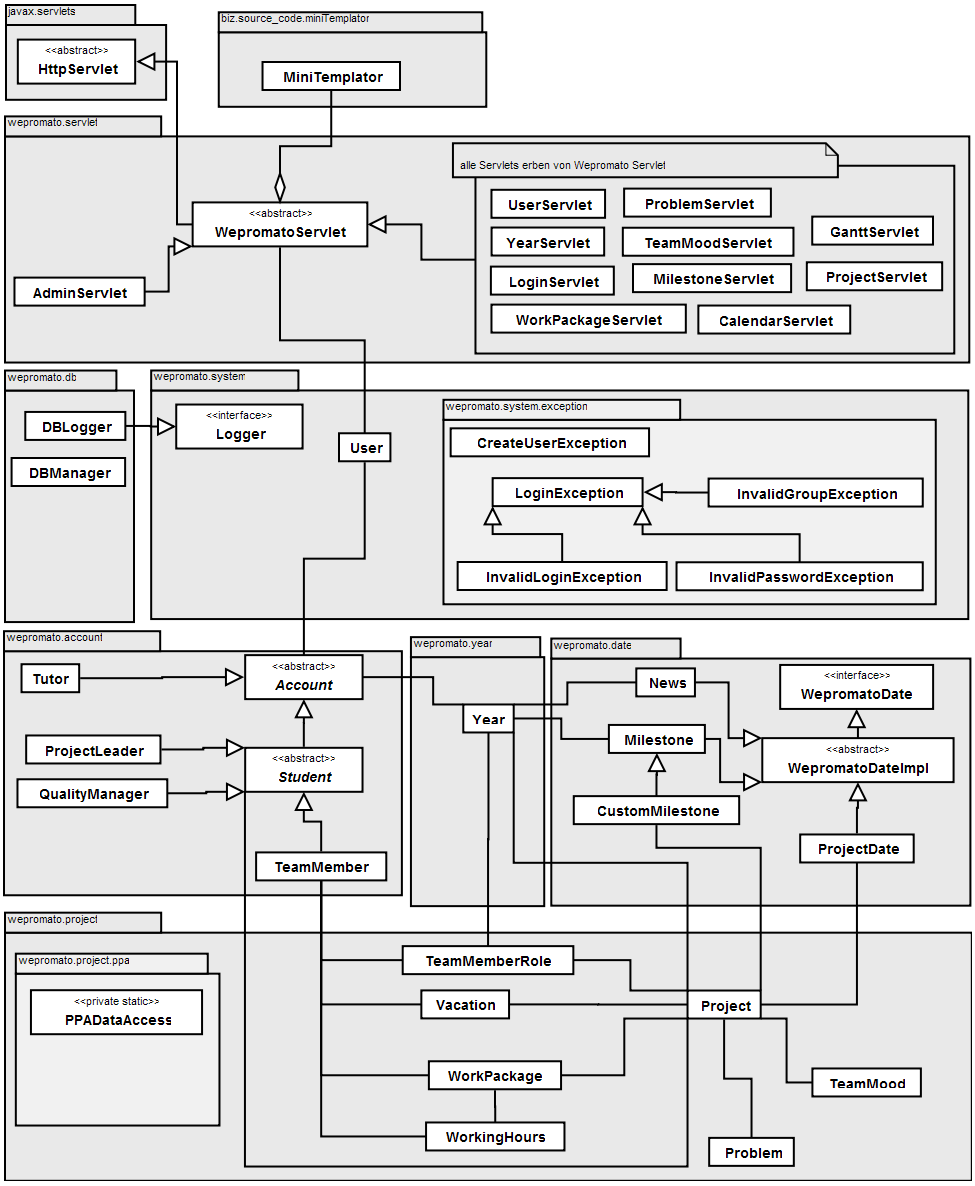

Abbildung 2: Klassendiagramm

2.3 ERM-Diagramm

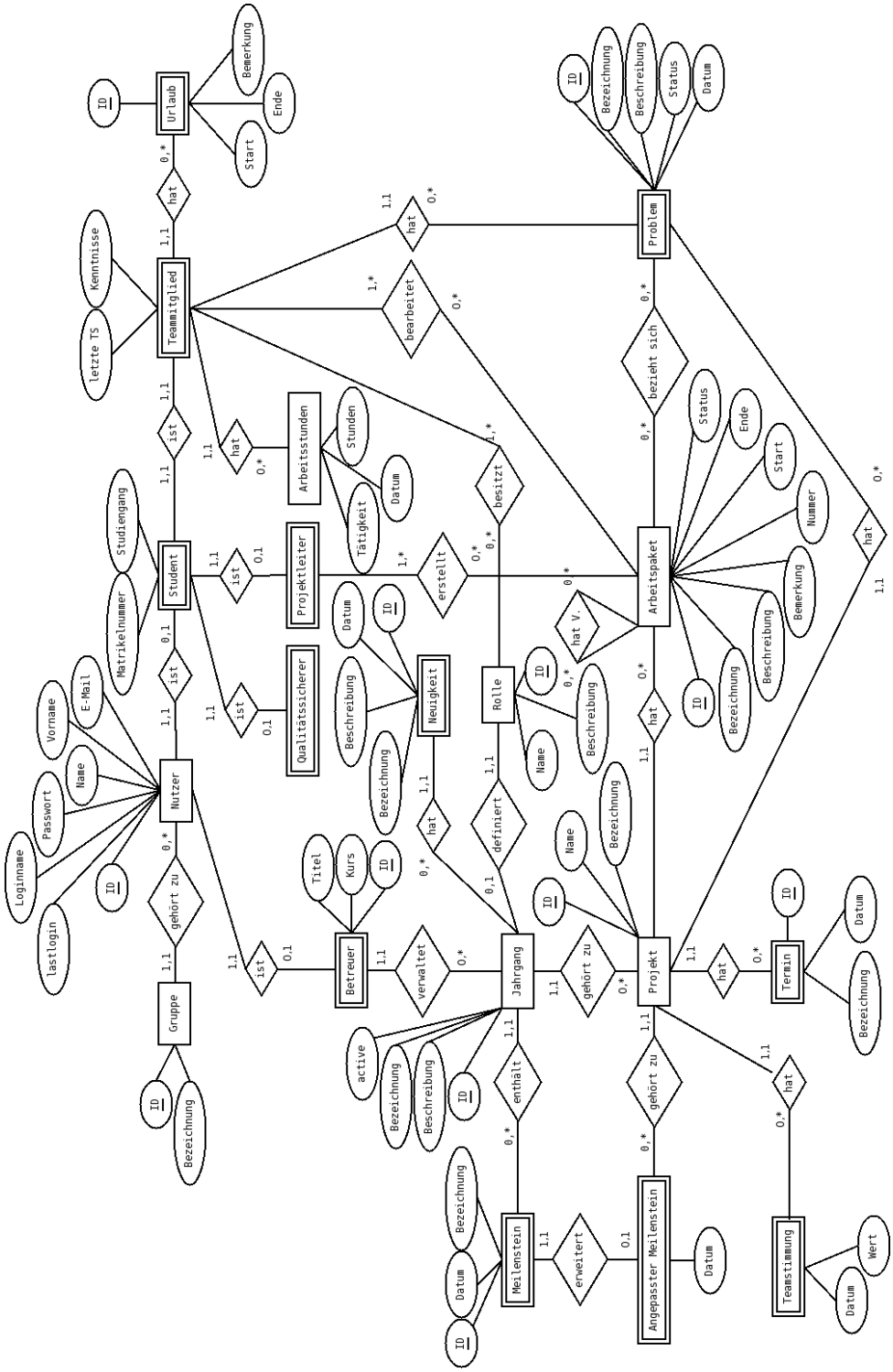

Abbildung 3: ERM-Diagramm

3 Feinstruktur der Komponenten

3.1 wepromato.servlet Paket

Das Servlet Paket stellt im gesamten Projekt das wohl flexibelste dar. Das heißt, das während der Entwicklung nach belieben Servlets erstellt, restruktiriert oder entfernt werden können.

Die folgende Struktur ist nur ein grober Entwurf, da sich genaueres aus der Oberfläche ergibt.

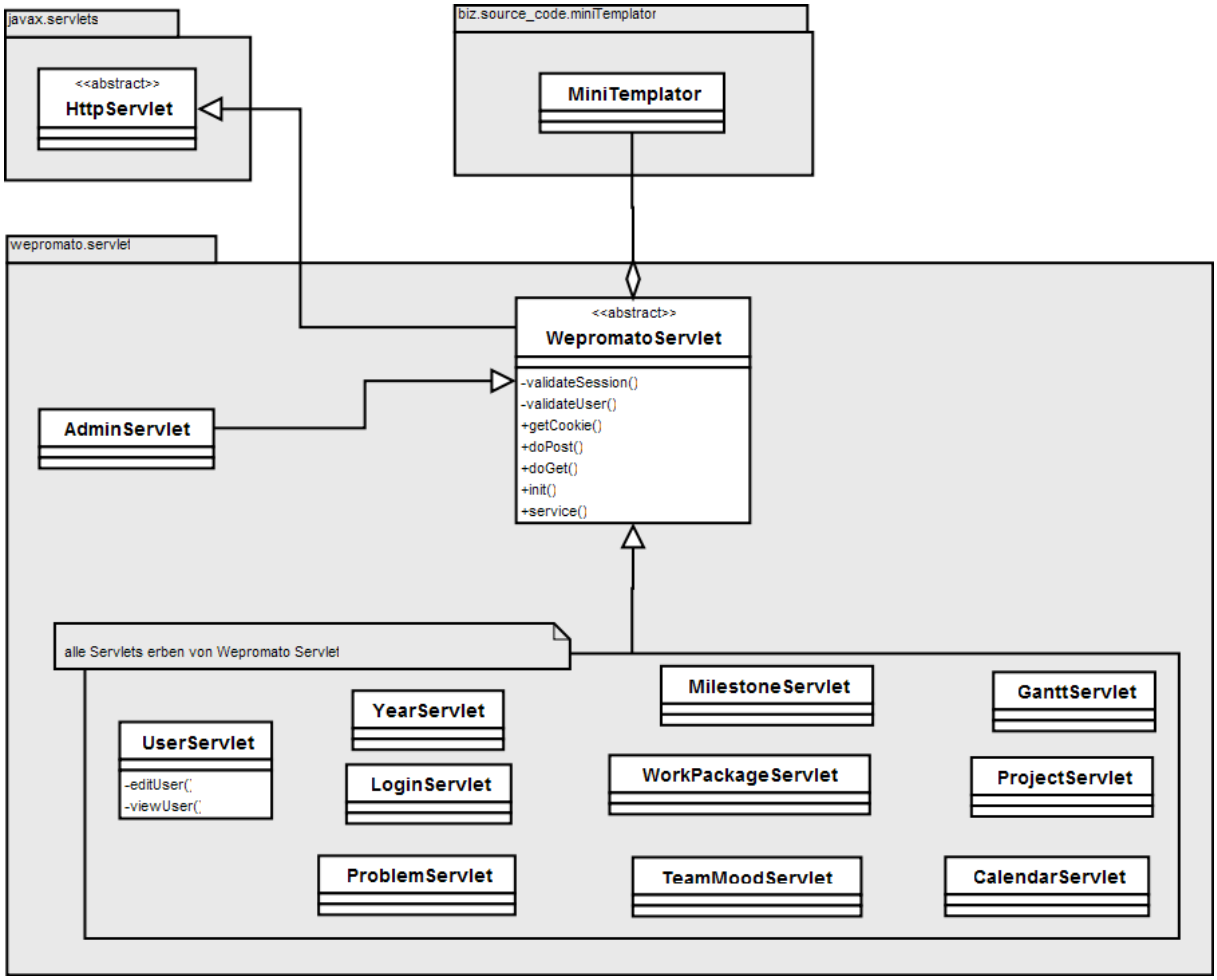

Abbildung 4: wepromato.servlet Paket

3.1.1 WepromatoServlet

Das Basis Servlet für alle Servlets in der Anwendung. Es erbt direkt von HttpServlet und implementiert Grundfunktionen zur Session- und Userverwaltung. Alle anderen Servlets der Anwendung erben von WepromatoServlet und müssen sich somit nicht um diese Standardaufgaben kümmern.

3.1.2 AdminServlet

Dieses Servlet ist völlig losgelöst von dem Rest der Anwendung. Es dient nur der Verwaltung der Betreuer (Tutoren).

3.1.3 alle anderen Servlets

Ergeben sich aus Oberflächenstruktur und werden während der Entwicklung erstellt. Eine fein gegliederte Struktur ist dabei großen, eventuell unübersichtlichen Servlets vorzuziehen.

3.2 wepromato.db Paket

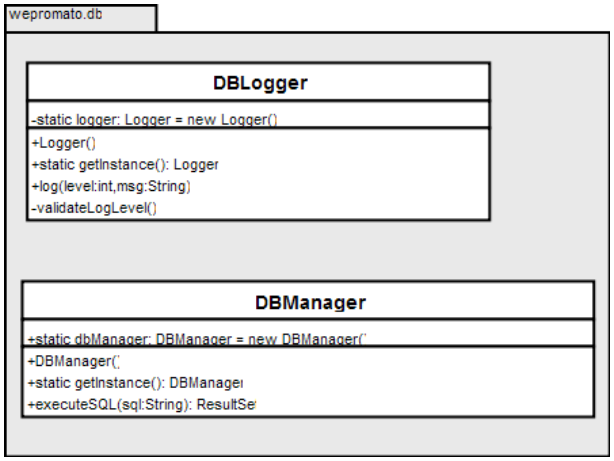

Abbildung 5: wepromato.db Paket

3.2.1 DBManager

Wrapper-Klasse für die Verbindung(en) zur Datenbank, nach dem Singleton Entwurfsmuster. Dabei wird darauf geachtet, dass das Connection Pooling vom Apache Tomcat Server genutzt wird. Das heißt, das für jede SQL Anfrage eine Verbindung aus dem Connection Pool genutzt wird. Nach Abschluß dieser Anfrage wird die Verbindung zur Datenbank wieder geschlossen. Anzumerken ist, dass Tomcat diese Verbindung weiterhin hält. Über die genaue Konfiguration von Tomcat ist noch zu entscheiden (Anzahl der Verbindungen, etc.).

1. Attribute
   dbManager
   statische Variable, so dass es nur eine Instanz dieser Klasse in der ganzen Webanwendung geben kann.

2. Konstruktoren
   nur der Standard Konstruktor

3. Methoden
   getInstance()
   gibt die einzige Instanz dieser Klasse zurück
   executeSQL(sql: String)
   Methode, um ein SQL Kommando auszuführen. Dabei wird auf eine der vom Tomcat verwalteten DB-Verbindungen zurückgegriffen.

3.2.2 DBLogger

Diese Klasse implementiert das wepromato.system.Logger Interface, und stellt eine Log Funktionalität in die Datenbank zur verfügung. Diese Klasse ist nach dem Singleton Entwurfsmuster entworfen.

1. Attribute
   logger
   statische Variable, so das es nur eine Instanz dieser Klasse in der ganzen Webanwendung geben kann.

2. Konstruktoren
   nur der Standard Konstruktor

3. Methoden
   getInstance()
   gibt die einzige Instanz dieser Klasse zurück
   log(level: int, msg: String)
   Methode, um eine Nachricht, mit einem als Konstante in DBLogger definierten Loglevel in die Datenbank zu speichern.
   validateLogLevel()
   Methode, die beim Aufruf von log() aufgerufen wird. Dient zum überprüfen des Loglevels.

3.3 wepromato.system Paket

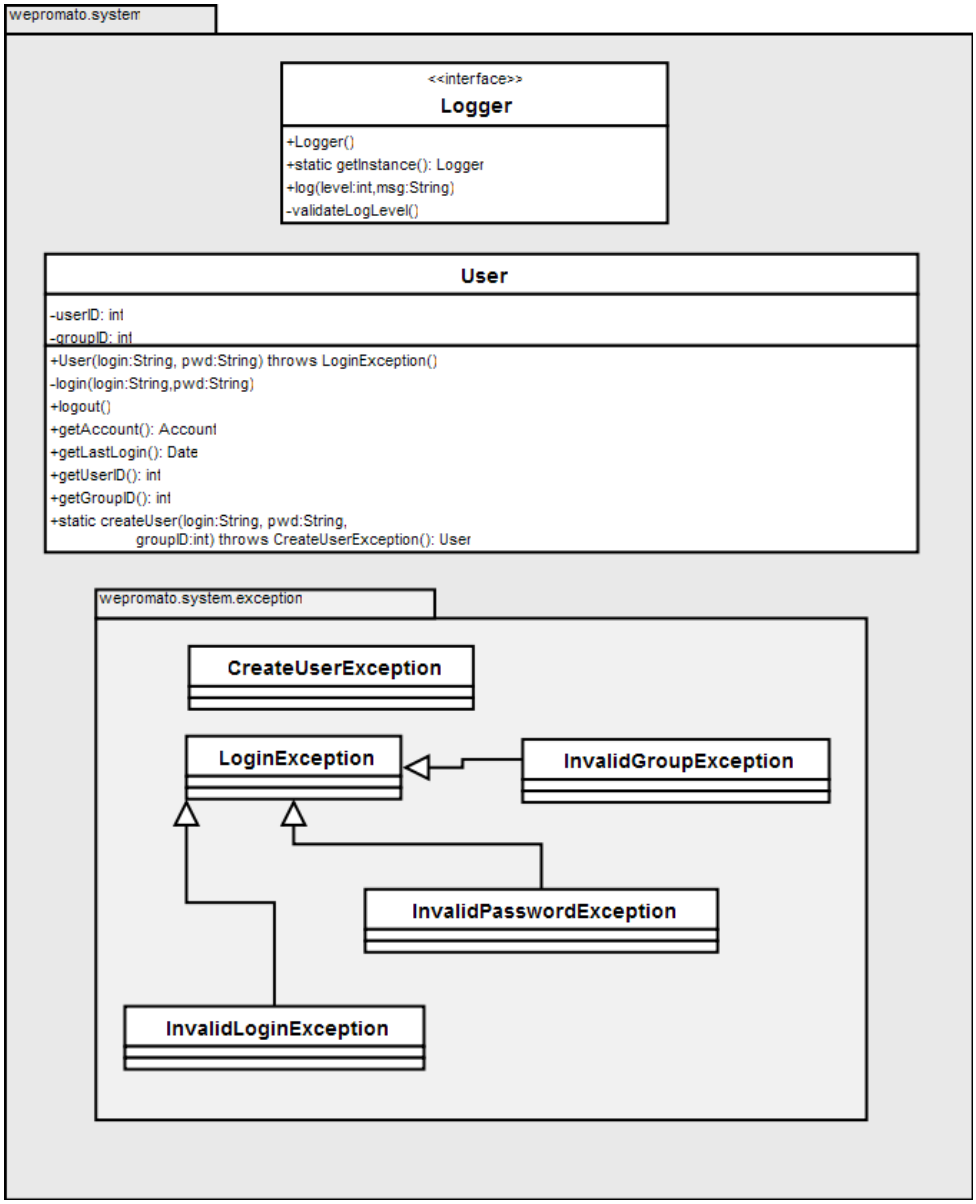

Abbildung 6: wepromato.system Paket

3.3.1 «Interface» Logger

siehe implementierende Klasse DBLogger

3.3.2 User

Hält die Daten der User im System während diese eingeloggt sind, um die Zugriffsrechte zu kontrollieren.

1. Attribute
   userID
   Eindeutige ID für jeden Nutzer
   groupID
   Eindeutige ID für die Gruppe in die der User ist

2. Konstruktoren
   User(login: String, pwd:String)
   wirft im Fehlerfall eine LoginExeption, der User wird mit den übergebenen Parametern eingeloggt indem die Methode login(login:String, pwd:String) aufgerufen wird

3. Methoden
   private login(login:String, pwd:String)
   wird vom Konstruktor aufgerufen um den User einzuloggen

   logout()
   User wird ausgeloggt

   getAccount():Acount
   den Account des Users anhand der groupID zurück

   getLastLogin():Date
   gibt das Datum des letzten Logins zurück

   static createUser(login:String, pwd:String, groupID:int
   wirft im Fehlerfall eine CreateUserException, Legt einen neuen User an

Sonst

Getter- und Setter- Methoden für die Attribute

3.4 wepromato.system.exception Paket

3.4.1 CreateUserException

Wird geworfen, wenn das Anlegen eines Users fehlgeschlagen ist.

3.4.2 LoginException

Wird geworfen, wenn das Einloggen eines Users fehlgeschlagen ist.

3.4.3 InvalidLoginException

Wird geworfen, wenn ein User versucht sich mit falschem Login anzumelden.

3.4.4 InvalidPasswordException

Wird geworfen, wenn ein User versucht sich mit einem falschen Passwort anzumelden.

3.4.5 InvalidGroupException

Wird bei falscher groupID geworfen.

3.5 wepromato.date Paket

Enthält alle Klassen, die der Terminverwaltung dienen.

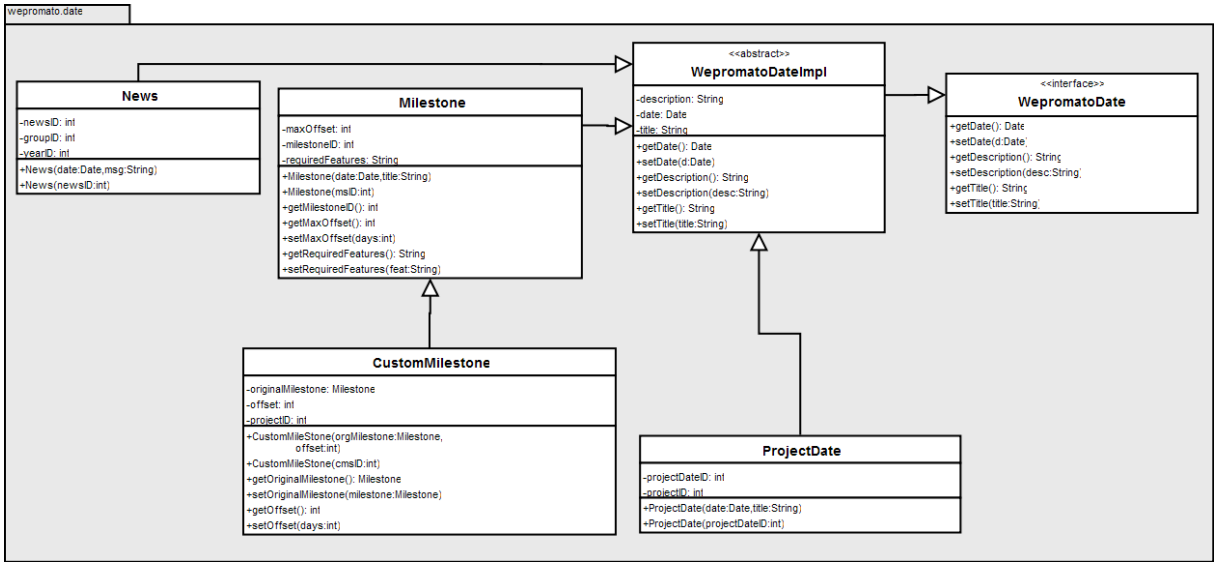

Abbildung 7: wepromato.date Paket

3.5.1 WepromatoDate

Ist ein Interface das von WepromatoDateImpl implementiert wird.

3.5.2 WepromatoDateImpl

Ist eine abstrakte Klasse von der News, Milestone und ProjectDate abgeleitet sind.

3.5.3 ProjectDate

1. Attribute
   projectDateId
   Eindeutige ID für den Termin.
   projectID
   Eindeutige ID für das Projekt

2. Konstruktoren
   projectDate(prjectDateID:int)
   Ruft den ProjectDate(date:Date, title:String) Konstruktor auf.
   ProjectDate(date:Date, title:String)
   Initialisiert das ProjectDate Objekt mit den Übergebenen Parametern.

3.5.4 Milestone

1. Attribute
   maxOffset:int
   Maximale zeitliche Variation in Tagen
   milestoneId: int
   Eindeutige ID für den Meilenstein
   requiredFeatures: String
   Geforderte Produktfunktionen des Projekts zum Meilenstein

2. Konstruktoren
   Milestone(msID:int)
   Ruft den Milestone(date:Date, title:String) Konstruktor auf
   Milestone(date:Date, title:String)
   Initialisiert den Meilenstein mit den Übergebenen Parametern

3. Methoden
   Sonst
   Getter- und Setter- Methoden für die Attribute

3.5.5 CustomMilestone

1. Attribute
   originalMilestone
   der Meilenstein aus dem er erstellt wurde
   offset
   zeitliche Verschiebung in Tagen
   projectID
   Eindeutige ID des Projekts zu dem der Meilenstein gehört

2. Konstruktoren
   CustomMileStone(cmsID:int)
   ruft den CustomMileStone(orgMilestone:Milestone, offset:int) Konstruktor auf
   CustomMileStone(orgMilestone:Milestone, offset:int)
   Initialisiert den Meilenstein mit den übergebenen Parametern

3. Methoden
   Sonst
   Getter- und Setter- Methoden für die Attribute

3.5.6 News

1. Attribute

   newsID

   Eindeutige ID der Neuigkeit

   groupID

   Eindeutige ID der Gruppe zu der die Neuigkeit gehört

   yearID

   Eindeutige ID des Jahrgangs zu dem die Neuigkeit gehört

2. Konstruktoren

   News(newsID:int)

   Ruft den News(date:Date, msg:String) Konstruktor auf

   News(date:Date, msg:String)

   Initialisiert die Neuigkeit mit den übergebenen Parametern

3.6 wepromato.year Paket

Abbildung 8: wepromato.year Paket

3.6.1 Year

Beschreibt einen Jahrgang und ermöglicht Zugriff auf die Projekte die zum Jahrgang gehören.

1. Attribute
   active
   gibt an ob der Jahrgang aktiv ist
   yearID
   eindeutige ID des Jahrgangs
   title
   der Titel des Jahrgangs
   tutorID
   ID des Betreueres der für den Jahrgang zuständig ist

2. Konstruktoren
   Year(yearID:int)
   ruft den Year(bez:String, tutor:Tutor) Konstruktor auf
   Year(bez:String, tutor:Tutor)
   Initialisiert den Jahrgang mit den Übergebenen Parametern

3. Methoden
   addProject(p:Project)
   fügt ein Projekt hinzu
   removeProject(p:Project)
   entfernt ein Projekt
   addStudent(s:Student)
   fügt einen Student hinzu
   removeStudent(s:Student)
   entfernt einen Student
   addRoles(tmr:TeamMemberRole)
   fügt einen neue Rolle für die Teammitglieder hinzu
   removeRoles(tmr:TeamMemberRole)
   entfernt eine Rolle
   addMilestone(ms:Milestone)
   fügt einen Meilenstein hinzu
   removeMilestone(ms:Milestone)
   entfernt einen Meilenstein
   addNews(news:News, groupID:int)

fügt eine Neuigkeit hinzu
removeNews(news:News)
entfernt eine Neuigkeit
export(file:File)
Exportiert einen Jahrgang in eine Datei
«static» import(file:File):Year
Importiert einen Jahrgang aus einer Datei

Sonst
Getter- und Setter- Methoden für die Attribute

3.7 wepromato.account Paket

Enthält alle Klassen, die der Nutzerverwaltung dienen. Diese Klassen dienen später der Unterscheidung des Users, nachdem dieser sich am System angemeldet hat.

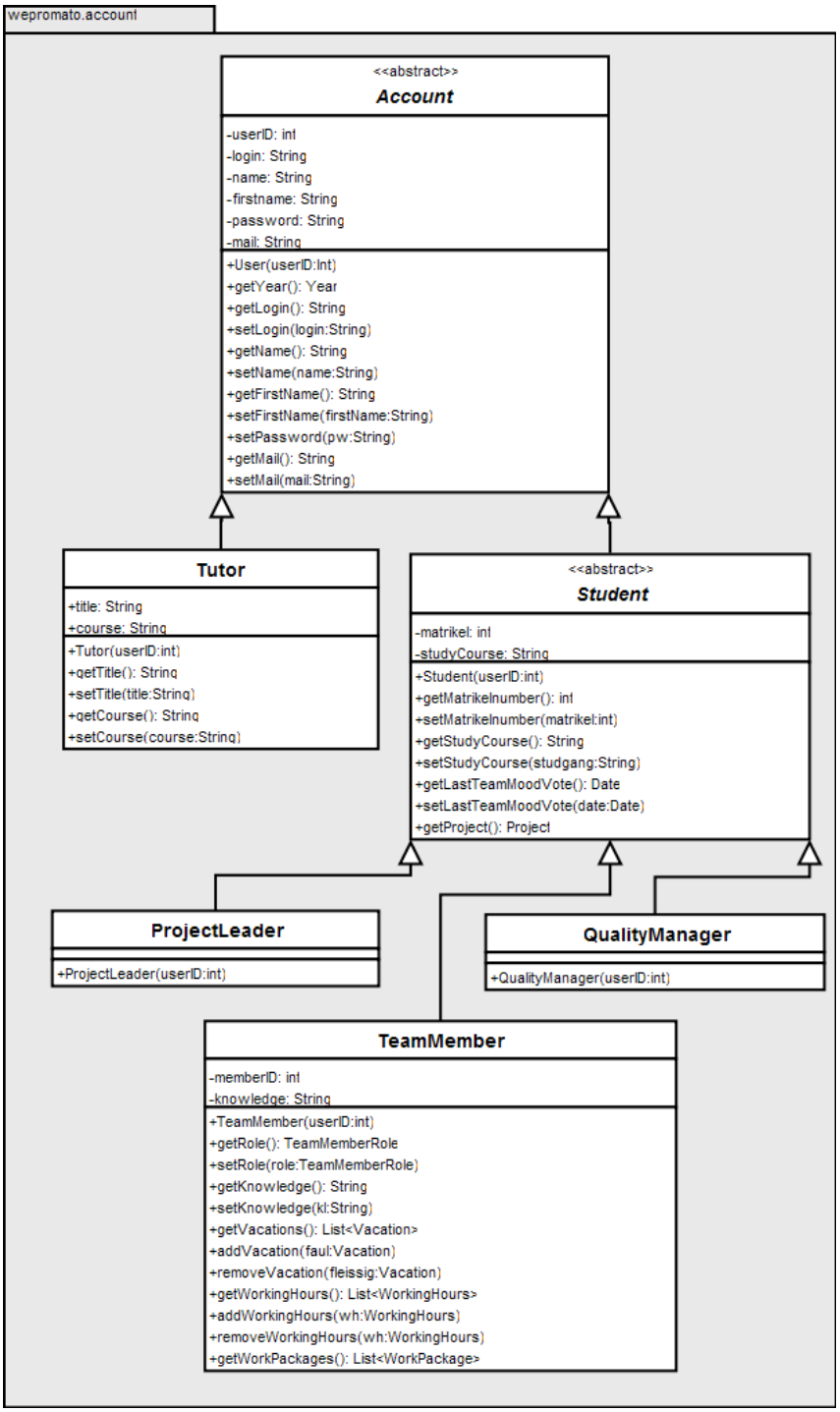

Abbildung 9: wepromato.user Paket

3.7.1 Account

Abstrakte Klasse welche Grundfunktionen, die von alle Nutzer-Klassen geteilt werden implementiert.

1. Attribute
   userID
   Eindeutige ID für jeden Nutzer
   login
   Login Name
   password
   Passwort für das Login
   name
   Nachname des Nutzers
   firstname
   Vorname des Nutzers
   mail
   eMail-Adresse des Nutzers

2. Konstruktoren
   User(userID: int)
   erstellt ein User Objekt mittels der UserID. Allerdings wird diese Klasse nie instanziiert, der Konstruktor ist nur vorhanden, damit erbende Klassen diesen Nutzen können

3. Methoden
   Sonst
   Getter- und Setter- Methoden für die Attribute

3.7.2 Tutor

Erbt von User und ruft den Konstruktor von User auf.

1. Attribute
   title
   Titel des Betreuers (z.B.: Prof. Dr.)
   course
   Name des Kurses, der vom Betreuer geleitet wird

2. Konstruktoren
   Tutor(userID: int)
   erstellt ein Tutor Objekt mittels der UserID. Wird an User (Superklasse) delegiert

3. Methoden
   Sonst
   Getter- und Setter- Methoden für die Attribute

3.7.3 Student

Abstrakte Klasse welche Grundfunktionen, die von allen Studenten bezogenen Klassen geteilt werden implementiert.

1. Attribute
   matrikel
   Matrikelnummer des Studenten
   studyCourse
   Name des Kurses, der vom Studenten belegt wird.

2. Konstruktoren
   Student(id: int, login: String, pwd: String)
   erstellt ein Student Objekt mittels des Logindaten und einer id. Wird an User (Superklasse) delegiert

3. Methoden
   Sonst
   Getter- und Setter- Methoden für die Attribute

3.7.4 ProjectLeader

Erbt von Student und ruft den Konstruktor von Student auf. Ansonsten keine Änderungen.

1. Konstruktoren
   ProjectLeader(userID: int)
   erstellt ein ProjectLeader Objekt mittels der UserID. Wird an Student (Superklasse) delegiert

3.7.5 QualityManager

Erbt von Student und ruft den Konstruktor von Student auf. Ansonsten keine Änderungen.

1. Konstruktoren
   QualityManager(userID: int)
   erstellt ein QualityManager Objekt mittels der UserID. Wird an Student (Superklasse) delegiert

3.7.6 TeamMember

Erbt von Student und ruft den Konstruktor von Student auf. Diese Klasse dient zur Darstellung eines Teammitgliedes.

1. Attribute
   role
   Rolle des Studenten innerhalb des Teams. Vom Typ TeamMemberRole
   vacations
   Liste des Urlaube eines Studenten (alle vom Typ Vacation)
   knowledge
   Das enorme Wissen eines Studenten in Textform
   workingHours
   geleistete Arbeitszeiten in Form einer Liste von WorkingHour Objekten

2. Konstruktoren
   TeamMember(userID: int)
   erstellt ein TeamMember Objekt mittels der UserID. Wird an Student (Superklasse) delegiert

3. Methoden
   public void addVacation(faul: Vacation)
   fügt dem Team Mitglied einen Urlaub hinzu
   public void removeVacation(fleissig: Vacation)
   entfernt den entsprechenden Urlaub aus dem System
   public void addWorkingHours(wh: WorkingHours)
   fügt dem Team Mitglied Arbeitsstunden hinzu
   public void removeWorkingHours(wh: WorkingHours)
   entfernt das entsprechende WorkingHour Objekt
   Nur getter und setter für die oben beschriebenen Attribute.

   Sonst
   Getter- und Setter- Methoden für die Attribute

3.8 wepromato.project Paket

Enthält alle Klassen, die der Projektverwaltung dienen.

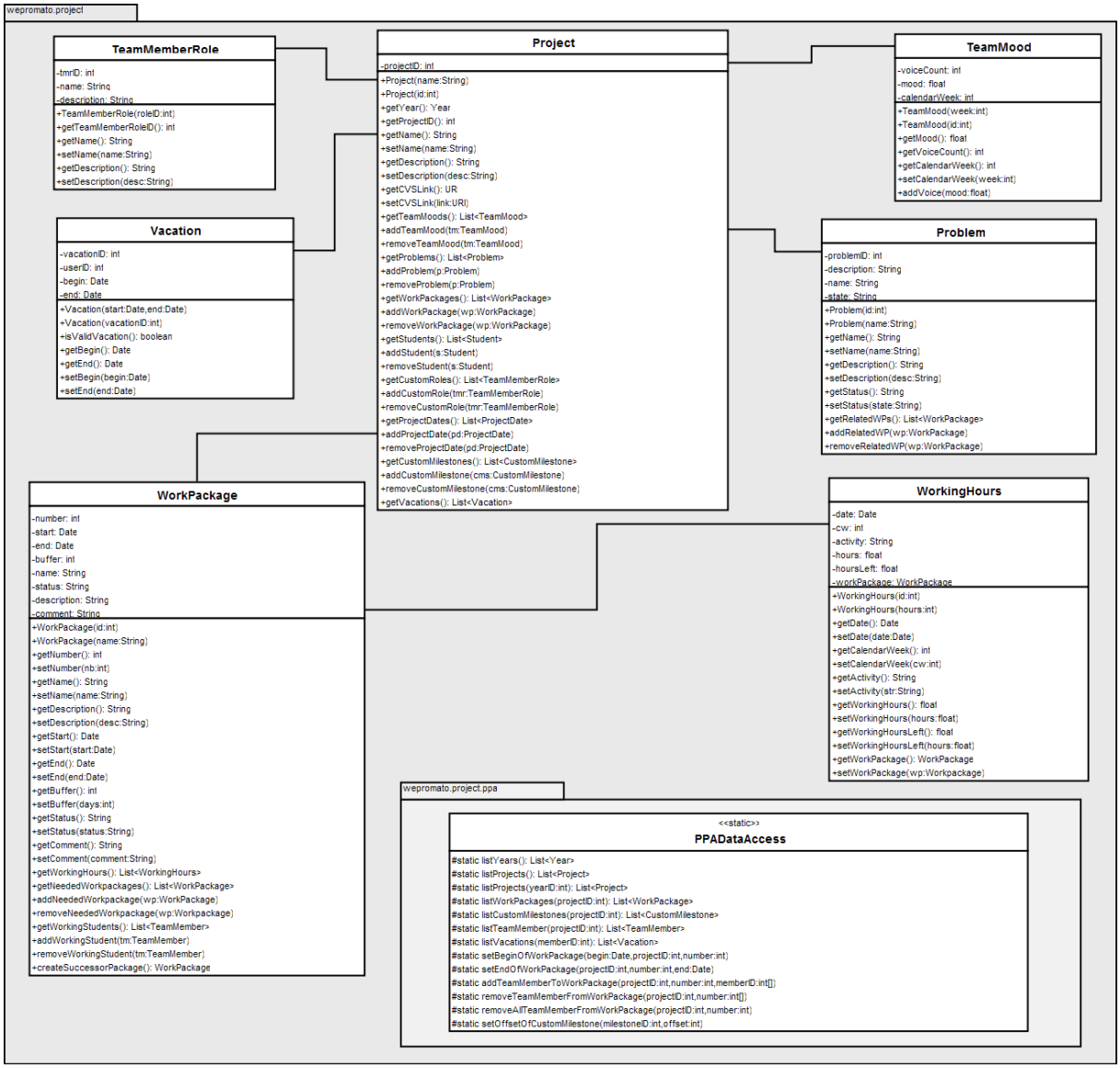

Abbildung 10: wepromato.project Paket

3.8.1 Project

Beschreibt ein Project und ermöglicht den Zugriff auf die dazugehörigen Arbeitspakete und Teammitglieder.

1. Attribute
   projectID
   eindeutige ID für das Projekt

2. Konstruktoren
   Project(id:int)
   ruft den Project(name:String) Konstruktor auf
   Project(name:String)
   Initialisiert das Projekt mit den übergebenen Parametern

3. Methoden
   addTeamMood(tm:TeamMood)
   fügt die Teamstimmung hinzu
   removeTeamMood(tm:TeamMood)
   entfernt die Teamstimmung
   addProblem(p:Problem)
   fügt ein Problem hinzu
   removeProblem(p:Problem)
   entfernt ein Problem
   addWorkPackage(wp:WorkPackage)
   fügt ein Arbeitspaket hinzu
   removeWorkPackage(wp:WorkPackage)
   entfernt ein Arbeitspaket
   addStudent(s:Student)
   fügt einen Student zum Projekt hinzu
   addStudent(s:Student)
   entfernt einen Student aus dem Projekt
   addCustomRole(tmr:TeamMemberole)
   fügt eine Rolle hinzu
   removeCustomRole(tmr:TeamMemberole)
   entfernt eine Rolle
   addProjectDate(pd:ProjectDate)
   fügt einen Termin hinzu

removeProjectDate(pd:ProjectDate)
entfernt einen Termin
addCustomMilestones(cms:CustomMilestone)
fügt einen angepassten Meilenstein hinzu
removeCustomMilestones(cms:CustomMilestone)
entfernt einen angepassten Meilenstein aus dem Projekt
Sonst
Getter- und Setter- Methoden für die Attribute

3.8.2 WorkPackage

Beschreibt ein Arbeitspaket mit zeitlichem Anfang und Ende sowie den Pufferzeiten und den Teammitgliedern die es bearbeiten.

1. Attribute
   number
   Nummer des Arbeitspaketes
   start
   Begin des Arbeitspaketes (Datum)
   end
   Ende des Arbeitspaketes (Datum)
   buffer
   Pufferzeit des Arbeitspaketes in Tagen
   name
   Name des Arbeitspaketes
   status
   Status des Arbeitspaketes
   description
   Beschreibung des Arbeitspaketes
   comment
   Kommentar zum Arbeitspaket

2. Konstruktoren
   WorkPackage(id:int)
   ruft den WorkPackage(name:String) Konstruktor auf
   WorkPackage(name:String)
   Initialisiert das Arbeitspaket mit den übergebenen Parametern

3. Methoden
   addNeededWorkpackage(wp:WorkPackage)
   fügt ein benötigtes Arbeitspaket hinzu
   removeNeededWorkpackage(wp:WorkPackage)
   entfernt ein benötigtes Arbeitspaket hinzu
   addWorkingStudent(tm:TeamMember)
   fügt einen Student hinzu der das Arbeitspaket bearbeitet
   removeWorkingStudent(tm:TeamMember)
   entfernt einen Student der das Arbeitspaket bearbeitet

createSuccessorPackage():WorkPackage
erstellt ein Nachfolgerarbeitspaket
Sonst
Getter- und Setter- Methoden für die Attribute

3.8.3 WorkingHours

Beschreibt die Arbeitsstunden der Teammitglieder die sie für die verschiedenen Arbeitspakete brauchen.

1. Attribute
   date
   Datum an dem die Arbeitsstunden abgeleistet wurden
   cw
   Kalenderwoche in der die Arbeitsstunden abgeleistet wurden
   activity
   Tätigkeit der Arbeitsstunden
   hours
   Anzahl der Stunden die abgeleistet wurden
   hoursLeft
   Anzahl der Stunden die noch benötigt werden um das Arbeitspaket fertig zu stellen
   workPackage
   Arbeitspaket an dem gearbeitet wurde

2. Konstruktoren
   WorkingHours(id:int)
   ruft den WorkingHours(hours:int) Konstruktor auf
   WorkingHours(hours:int)
   Initialisiert das Arbeitspaket mit den übergebenen Parametern

3. Methoden
   Sonst
   Getter- und Setter- Methoden für die Attribute

3.8.4 Problem

Beschreibt die Probleme der Teammitglieder.

1. Attribute
   problemID
   eindeutige ID des Problems
   description
   Beschreibung des Problems
   name
   Name des Problems
   state
   Status des Problems

2. Konstruktoren
   Problem(id:int)
   ruft den Problem(name:String) Konstruktor auf
   Problem(name:String)
   Initialisiert das Problem mit den übergebenen Parametern

3. Methoden
   addRelatedWP(wp:WorkPackage)
   fügt ein betroffenes Arbeitspaket hinzu
   removeRelatedWP(wp:WorkPackage)
   entfernt ein betroffenes Arbeitspaket
   Sonst
   Getter- und Setter- Methoden für die Attribute

3.8.5 TeamMood

Beschreibt die Teamstimmung

1. Attribute
   voiceCount
   Anzahl der abgegebenen Stimmen
   mood
   Teamstimmung in der entprechenden Kalenderwoche
   calendarWeek
   Kalenderwoche

2. Konstruktoren
   TeamMood(id:int)
   Ruft den TeamMood(week:int) Konstruktor auf
   TeamMood(week:int)
   Initialisiert das Teamstimmungsobjekt mit den übergebenen Parametern

3. Methoden
   addVoice(mood:float)
   Hinzufügen einer Stimme
   Sonst
   Getter- und Setter- Methoden für die Attribute

3.8.6 TeamMemberRole

Beschreibt die Rollen der Teammitglieder.

1. Attribute
   tmrID
   eindeutige ID der Rolle
   name
   Name der Rolle
   description
   Beschreibung der Rolle

2. Konstruktoren
   TeamMemberRole(id:int)
   ruft den TeamMemberRole(roleID:int) Konstruktor auf
   TeamMemberRole(roleID:int)
   Initialisiert die Rolle mit den übergebenen Parametern

3. Methoden
   Sonst
   Getter- und Setter- Methoden für die Attribute

3.8.7 Vacation

Erbt von Student und ruft den Konstruktor von Student auf. Diese Klasse dient zur Darstellung eines Teammitgliedes.

1. Attribute
   begin
   Startdatum des Urlaubs
   end
   Enddatum des Urlaubs

2. Konstruktoren
   Vacation(begin: Date, end: Date)
   erstellt ein Vacation Objekt mittels der Start- und Endzeitpunkte

3. Methoden
   Getter- und Setter- Methoden für die Attribute

3.9 Exemplarische Sequenzdiagramme

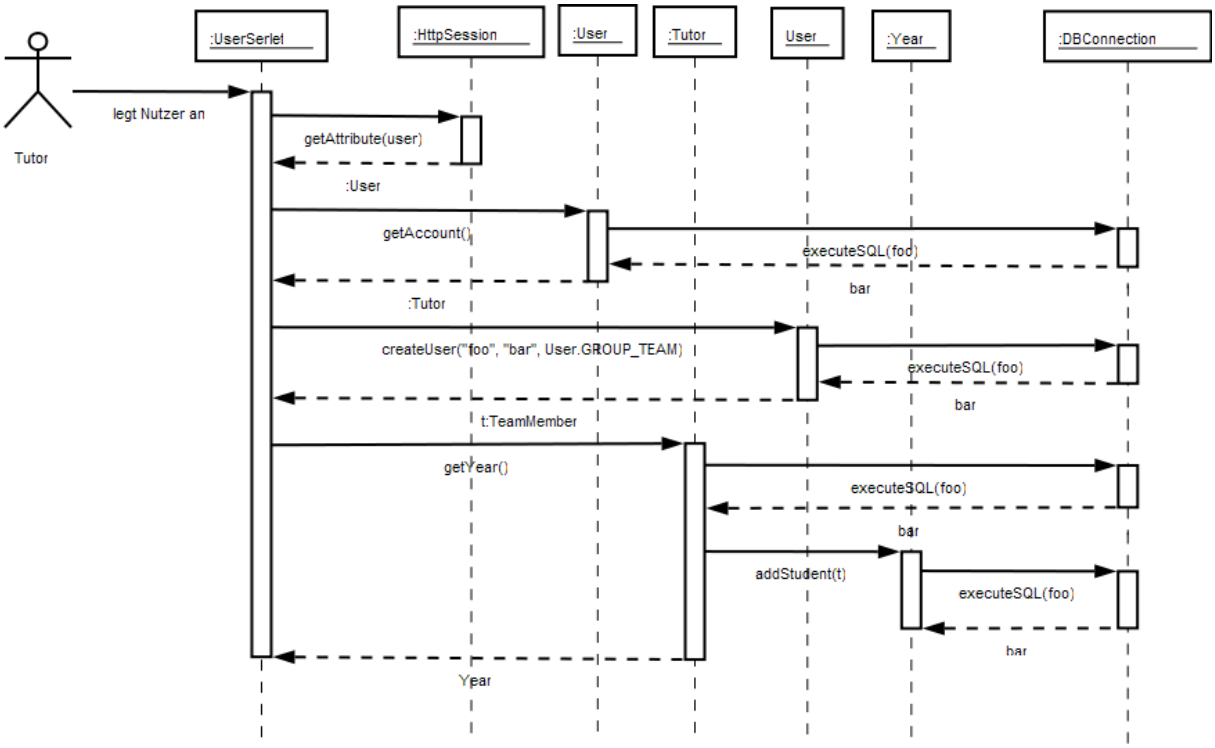

Abbildung 11: Betreuer legt einen neuen Nutzer an

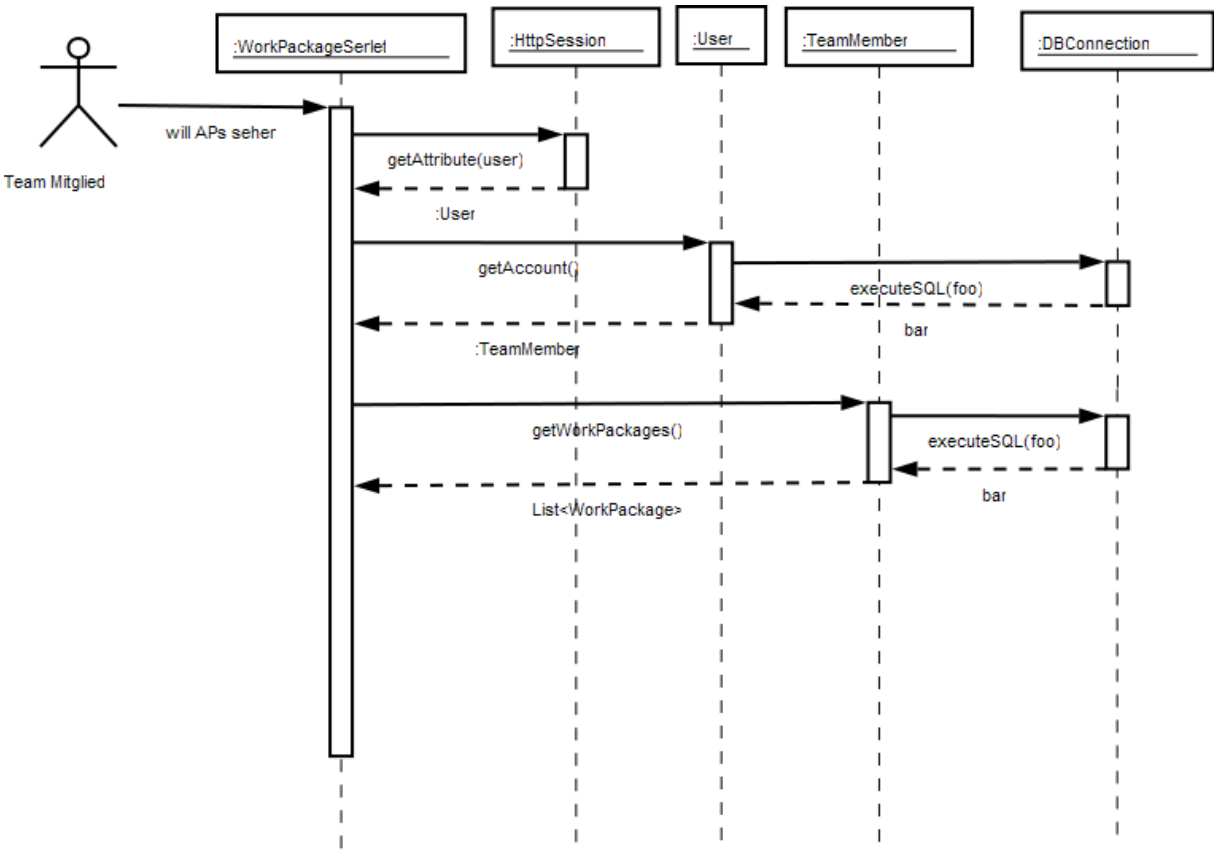

Abbildung 12: Teammitglied lässt alle eigenen Arbeitspakete anzeigen

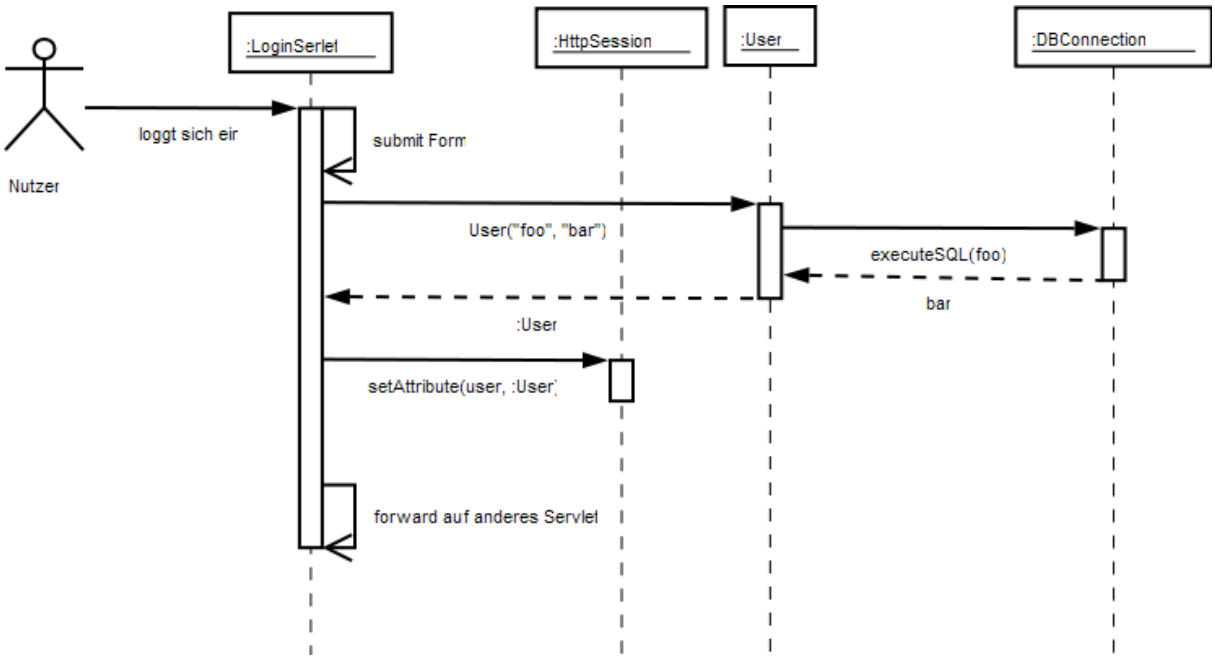

Abbildung 13: Ein beliebiger Nutzer meldet sich am System an

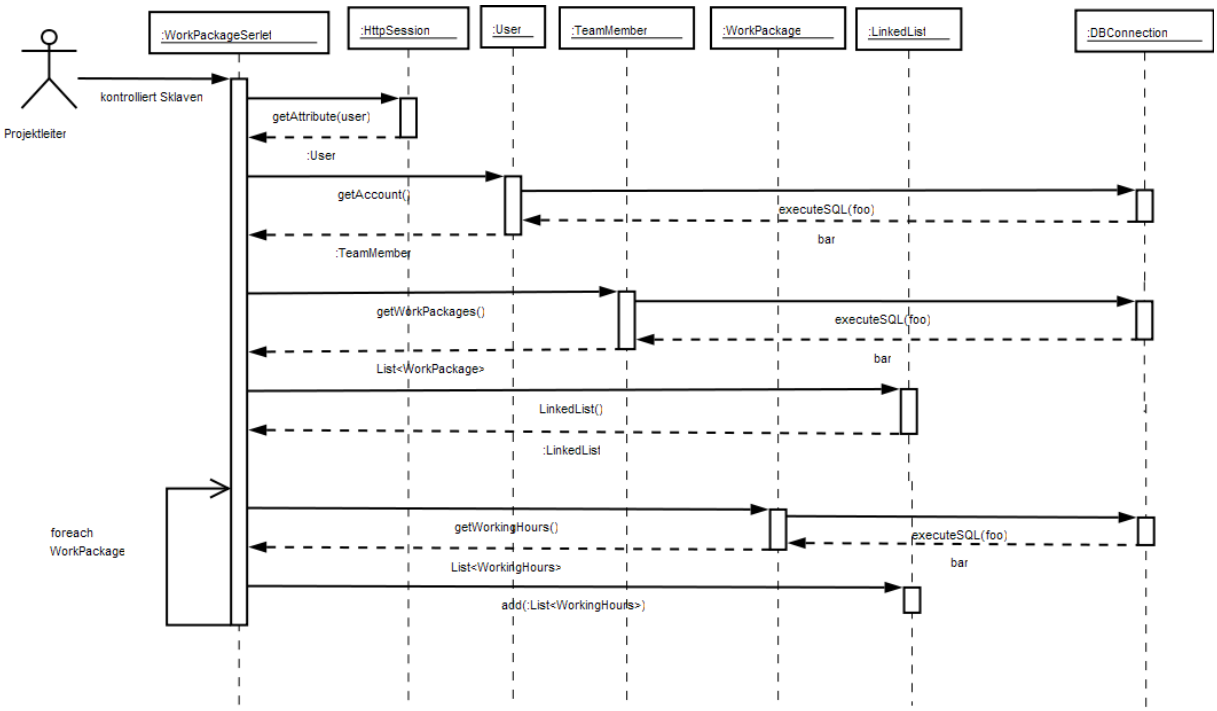

Abbildung 14: Projektleiter lässt alle Arbeitsstunden eines Projektes anzeigen

4 Schnittstellen

4.1 Import / Export von Jahrgängen

Zum Import und Export von Jahrgängen wird XML benutzt, da sich Daten dadurch hierarchisch anordnen und gruppieren lassen, auch stellt Java bereits Möglichkeiten zum effektiven XML-Handling bereit.

In Wepromato werden Funktionen verfügbar sein, die das Importieren und Exportieren von Jahrgängen aus bzw. in lokale XML-Dateien ermöglichen. Dabei wird beim Import besonderer Wert auf die Behandlung von Fehlern gelegt.

Die XML-Struktur ist wie folgt definiert:

```
<Wepromato SchemaVersion="0.1">
  <ProjektInfo Name="Wepromato" Version="0.0" Autor="Benutzer" />
  <Betreuer>
    <Betreuer ID="" Titel="" Kurs="" />
  </Betreuer>
  <Jahrgänge>
    <Jahrgang ID="" BetreuerID="" Beschreibung="" Bezeichnung="">
      <Meilensteine>
        <Meilenstein ID="" Datum="" Bezeichnung="" />
      </Meilensteine>
      <Neuigkeiten>
          <Neuigkeit Datum="" Beschreibung="" />
      </Neuigkeiten>
      <Rollen>
        <Rolle ID="" Name="" Beschreibung="" />
      </Rollen>
      <Projekte>
        <Projekt Name="" Bezeichnung="" CVSLink="">
          <Teamstimmungen>
            <Teamstimmung  Datum="" Wert="" />
          </Teamstimmungen>
          <Arbeitspakete>
            <Arbeitspaket ID="" Nummer="" Bezeichnung="" Beschreibung=""
            Bemerkung="" Start="" Ende="" Status="" />
```

```
          </Arbeitspakete>
          <Termine>
            <Termin Datum="" Bezeichnung="" />
          </Termine>
          <AngepassteMeilensteine>
              <AngepassterMeilenstein MeilensteinID="" Datum="" />
          </AngepassteMeilensteine>
          <Personal>
            <Projektleiter>
              <Nutzerdaten NutzerID="" Loginname="" Name="" Vorname=""
               Passwort="" Mail="" />
              <Studentendaten Matrikelnummer="" Studiengang="" />
            </Projektleiter>
            <Qualitätssicherer>
              <Nutzerdaten NutzerID="" Loginname="" Name="" Vorname=""
               Passwort="" Mail="" />
              <Studentendaten Matrikelnummer="" Studiengang="" />
            </Qualitätssicherer>
            <Teammitglied RollenID="" Kenntnisse="">
              <Nutzerdaten NutzerID="" Loginname="" Name="" Vorname=""
               Passwort="" Mail="" />
              <Studentendaten Matrikelnummer="" Studiengang="" />
              <Probleme>
                <Problem Bezeichnung="" Beschreibung="" Status="" />
              </Probleme>
              <Urlaube>
                <Urlaub Beginn="" Ende="" />
              </Urlaube>
              <Arbeit>
                <Arbeitsstunden ArbeitspaketID="" Datum="" Kalenderwoche=""
                 Stunden="" Tätigkeit="" />
              </Arbeit>
            </Teammitglied>
          </Personal>
        </Projekt>
      </Projekte>
    </Jahrgang>
  </Jahrgänge>
</Webpromato>
```

4.2 Projektplanungsalgorithmus

Der Projektplanungsalgorithmus sollte als Paket (wepromato.project.ppa) innerhalb des wepromato.project Paketes integriert werden. Dazu wurden bereits Vorkehrungen getroffen, unter Anderem existiert das Paket bereits und enthält die Java-Klasse 'PPADataAccess'. Diese dient zur Interaktion zwischen dem zu implementierenden Projektplanungsalgorithmus und des vorhandenen Systems. Die zur Verfügung stehenden statischen Methoden dieser Klasse werden unter 4.2.1 erläutert. Zur Integration in die Oberfläche müsste lediglich das gewünschte Servlet angepasst werden. Die vorhandenen Attribute und deren Methoden der verschiedenen Klassen sind aus dem Klassendiagramm zu entnehmen.

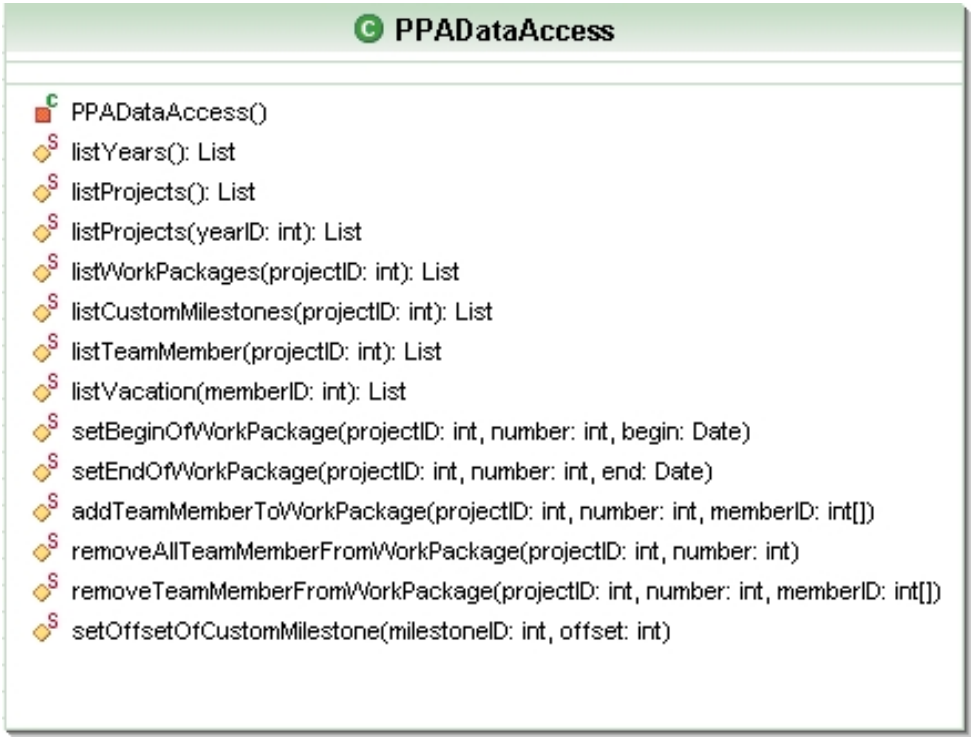

Abbildung 15: Inteface

4.2.1 Methoden der Hilfsklasse

protected static List<Year> listYears()
Funktion: Dient zum Abfragen aller vorhanden Jahrgänge.
Übergabewerte: %
Rückgabewerte: Jahrgänge als Liste von Year-Objekten

protected static List<Project> listProjects()
Funktion: Dient zum Abfragen aller vorhanden Projekte.
Übergabewerte: %
Rückgabewerte: Projekte als Liste von Project-Objekten

protected static List<Project> listProjects(int yearID)
Funktion: Dient zum Abfragen der vorhanden Projekte zu einem Jahrgang.
Übergabewerte: %
Rückgabewerte: Projekte als Liste von Project-Objekten

protected static List<TeamMember> listTeamMember(int projectID)
Funktion: Dient zum Abfragen der Team-Mitglieder eines Projektes.
Übergabewerte: ProjectID als Integer
Rückgabewerte: Team-Mitglieder als Liste von TeamMember-Objekten

protected static List<Vacation> listVacation(int memberID)
Funktion: Dient zum Abfragen des Urlaubes eines Team-Mitgliedes.
Übergabewerte: MemberID als Integer
Rückgabewerte: Urlaub als Liste von Vacation-Objekten

protected static List<WorkPackage> listWorkPackages(int projectID)
Funktion: Dient zum Abfragen der Arbeitspakete eines Projektes.
Übergabewerte: ProjectID als Integer
Rückgabewerte: Arbeitspakete als Liste von WorkPackage-Objekten

protected static List<CustomMilestone> listCustomMilestones(int projectID)
Funktion: Dient zum Abfragen der Meilensteine eines Projektes.
Übergabewerte: ProjectID als Integer
Rückgabewerte: Meilensteine als Liste von CustomMilestone-Objekten

protected static void setBeginOfWorkPackage(int projectID, int number, Date begin)
Funktion: Dient zum Festlegen des Begindatums eines Arbeitspaketes eines Projektes.
Übergabewerte: ProjectID als Integer, Number als Integer, Datum als Date-Objekt
Rückgabewerte: %

protected static void setEndOfWorkPackage(int projectID, int number, Date end)
Funktion: Dient zum Festlegen des Enddatums eines Arbeitspaketes eines Projektes.
Übergabewerte: ProjectID als Integer, Number als Integer, Datum als Date-Objekt
Rückgabewerte: %

protected static void addTeamMemberToWorkPackage(int projectID, int number, int... memberID)
Funktion: Dient zum Hinzufügen von Team-Mitgliedern zu einem Arbeitspaket eines Projektes.
Übergabewerte: ProjectID als Integer, Number als Integer, MemberID als Integer
Rückgabewerte: %

protected static void removeTeamMemberFromWorkPackage(int projectID, int number, int... memberID)
Funktion: Dient zum Entfernen einer oder mehrerer Team-Mitglieder von einem Arbeitspaket eines Projektes.
Übergabewerte: ProjectID als Integer, Number als Integer, MemberID als Integer
Rückgabewerte: %

protected static void removeAllTeamMemberFromWorkPackage(int projectID, int number)
Funktion: Dient zum Entfernen aller Team-Mitglieder eines Arbeitspakets eines Projektes.
Übergabewerte: ProjectID als Integer, Number als Integer
Rückgabewerte: %

protected static void setOffsetOfCustomMilestone(int milestoneID, int offset)
Funktion: Dient zum Festlegen des Puffers in Tagen für einen Meilenstein eines Projektes.
Übergabewerte: MilestoneID als Integer, Offset als Integer
Rückgabewerte: %

A Code-Style-Guide

A.1 Einleitung

'Der Programmierstil dient den Programmierern, nicht umgekehrt.' - Das heißt dieses Dokument gilt als Leitfaden, in begründbaren Fällen kann natürlich von diesen Vorschlägen abgewichen werden.

Es sollte jedoch wenn möglich eingehalten werden, um ein einheitliches Bild des Quelltextes zu erhalten.

Im Wiki steht außerdem ein Codestyle Template für den Eclipse Code-Formatter bereit.

A.2 Java Code Style

A.2.1 Einrückung

- Es sind ausschließlich Tabs zum Einrücken zu verwenden (1 Tab pro Block).
- Werden die Klammern bei Strukturen mit nur einem Befehl nach **if**, **while** oder **for** weggelassen, so muss ein Zeilenumbruch und eine Einrückung erfolgen und danach eine Leerzeile stehen.

A.2.2 Klammersetzung

- Öffnende Klammern auf der nächsten Zeile, vertikal gleichauf mit dem Klassen- / Methodenkopf
- Schließende Klammern stehen auf einer eigenen Zeile.
- Folgt bei Strukturen mit nur einem Befehl nach **if**, **while** oder **for** jedoch ein else Zweig so sind auf jedenfall Klammern zu setzen.

A.2.3 Leerzeilen und Leerzeichen

- Eine Leerzeile nach jedem Block.
- Den Schlüsselwörtern **if**, **while**, **switch**, **catch** oder **for** muss ein Leerzeichen folgen.

- Classcasts folgt kein Leerzeichen.
- Die linke Klammer für die Parameter einer Methode folgt direkt ohne Leerzeichen, ebenso sollten 'Array-Indexklammern' direkt nach dem Array-Variablennamen stehen.
- Vor und nach binären Operatoren steht ein Leerzeichen.
- Sollten mehrere Zuweisungen untereinander stehen, so sind die Gleichheitszeichen, wenn möglich mittels Leerzeichen auf eine Linie zu bringen. Das gleiche kann natürlich auch auf Variablen- / Konstantendeklarationen angewandt werden.

A.2.4 Reihenfolge der Elemente einer Klasse

Java Beispiel 1 Reihenfolge der Elemente

```
class Reihenfolge
{
    // Attributes
    private blubb irgendwas;

    // constructors
    public Reihenfolge() {...}

    // methods
    public void fooBar() {...}
}
```

A.2.5 Namensgebung

- Klassen- und Methodennamen bestehen nur aus 'A' bis 'Z' / 'a' bis 'z' und '0' bis '9', keine non-ASCII Zeichen, Striche und Umlaute.
- Methoden und Klassen haben englische Namen zu tragen, Kommentare sind in Deutsch zu halten.
- Die .java Datei einer Klasse trägt ihren Namen (Foo.java für Klasse Foo).
- Java Klassen und Interfaces fangen mit Großbuchstaben, Methoden und Variablen mit Kleinbuchstaben an. Bei 'Wortwechseln' ist ein Großbuchstabe zu verwenden, sonst wird der Name Kleingeschrieben. Bsp: anotherFourWordMethod
- Konstanten-Namen sollten komplett aus Großbuchstaben bestehen. Hier besteht eine wichige Ausnahme: Unterstriche sind erlaubt, um Wörter zu trennen.
- Package-Namen sollten komplett aus Kleinbuchstaben bestehen.
- vermeide getAnotherVeryLongMethodeName, davon gibts im SDK schon genug
- Generell sollte nur eine Java Klasse in einer .java Datei stehen. Sollten Hilfsklassen mit in der Datei stehen, so stehen diese am Ende der Datei.

A.2.6 Kommentare

- Für jede Methode ist ein Javadoc-Kommentar Pflicht, die Ausführlichkeit dieses Kommentars ist abhängig von der Methode. Der Sinn der Methode muss auf jedenfall erkennbar sein.
- Sollte mehr als eine Java Klasse in einer .java Datei stehen, dann sollte am Anfang ein Hinweis auf weitere Klassen stehen.
- Es wird empfohlen komplexe Konstrukte mit Inline Kommentaren zu erklären, dies ist aber nicht Pflicht.

A.2.7 Beispiele

Java Beispiel 2 Leerzeichen

```
foo (i, j);    // NO!
foo(i, j);     // YES!

args [0];    // NO!
args[0];     // YES!

z = 2*x + 3*y;          // NO!
z = 2 * x + 3 * y;      // YES!

count ++;  // NO!
count++;   // YES!

for(int i = 0;i < 10;i++)    // NO!
for (int i = 0; i < 10; i++) // YES!

getSomething(arg0,arg1);   // NO!
getSomething(arg0, arg1);  // YES!

Object aObject = ( Object ) anotherObject;    // NO!
Object aObject = (Object)anotherObject;       // YES!
```

Java Beispiel 3 Einrückung / Leerzeilen

```
public void foo()
{
    while (bar > 0)
    {
        System.out.println();
        bar--;
    }

    if (blobb == 2356)
         //irgendwas;

    if (blubb == blobb)
    {
        foo    = bar.getFoo();
        foobar = foo.merge(bar);
    }
    else
    {
        // irgendwas
    }

    switch (schalter)
    {
        case 1:
            // irgendwas
            break;
        default:
            // irgendwas
            break;
    }
}
```

A.3 CSS Code Style

A.3.1 Allgemeines

Generell ist eine externe CSS Datei vorzuziehen, bei kurzen Definitionen jedoch nicht Pflicht.

A.3.2 Einrückung / Leerzeilen

- Öffnende Klammer auf einer Zeile mit dem zu definierenden Element (class oder id).
- Schließende Klammer auf einer eigenen Zeile. Eine Leerzeile folgt auf jeden Block.
- Es sind ausschließlich Tabs zum Einrücken zu verwenden (1 Tab pro Block).
- Jede Eigenschaft auf steht in eigener Zeile.
- Ähnliche, bzw. Themenverwandte Eigenschaften sind, wenn möglich zu Gruppieren.

A.3.3 Namensgebung

Auch wenn es sich nur um CSS handelt, sollten sinvolle Namen verwendet werden. Im Allgemeinen sollten Namen ihren Verwendungszweck wiedergeben, nicht Ihre Eigenschaften. Also Anstatt 'aFunkySlashDottedCell' eher 'tableCellStyle' oder sowas in der Art.

A.3.4 Kommentare

Es sind keine Kommentare zu verwenden, es sei denn Kommentare sind wirklich nötig. Kommentare erhöhen nur das Datenaufkommen beim Transfer.

A.3.5 Beispiel

CSS Beispiel 1 (Inline CSS)

```
<style type="text/css"><!--
.tdStyle {
    border-bottom: 1px solid black;
    border-left: 1px solid black;
    text-align: center;
}

.cellStyle {
    ...
}
--></style>
```

A.4 Verzeichnisstruktur

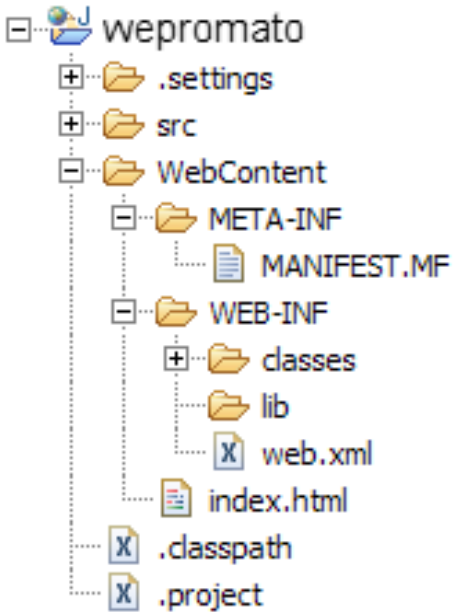

Abbildung 16: Projekt-Verzeichnisstruktur