Entwurf: Crayons 2.0

Julius Busch, Marc Jepsen, Natalia Krylova,
Levin Schickle, Ian Fitzgerald, Ali Akil

16. Januar 2017

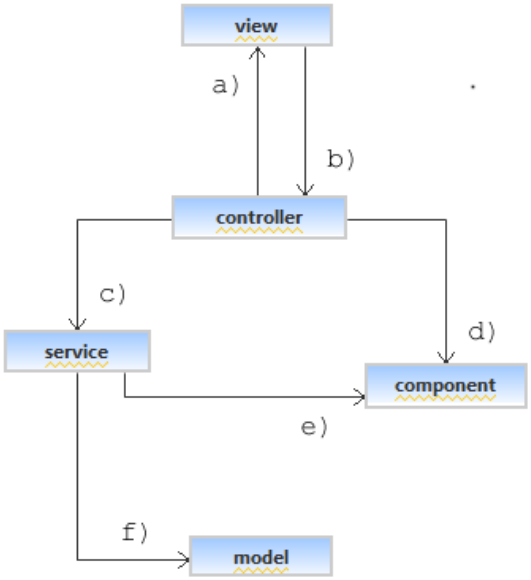
Inhaltsverzeichnis

1 Einleitung

In diesem Entwurf werden Architektur und Paketaufteilung des Softwareprojekts Crayons 2.0 erläutert. Es handelt sich um eine Vaadin/Spring basierte WebApp, die im Rahmen des Moduls Praxis der Softwareentwicklung [PSE] am Karlsruher Institut für Technologie [KIT] entwickelt wird.
Der Entwurf klärt den strukturellen Aufbau von Crayons 2.0 und soll als Grundlage der folgenden Implementierung dienen. Zunächst wird die gewählte Architektur erläutert und im Anschluss durch Darlegung der Pakete und Klassen detailierter beschrieben. Die Darstellung des Datenbank-Schemas bildet den Abschluss des Entwurfs.
Die verwendeten Klassen und ihre Funktion der im Pflichtenheft gelisteten Tools werden nicht erläutert und als grob bekannt vorrausgesetzt.

2 Architektur

Die verwendete Architektur ist eine, auf kleinere Vaadin/Spring webapps zugeschnittene Schichtenarchitektur. Das System wird in view, controller, service, component und model gegliedert. Der Login und Authentifizierung sind für erleichterte Sicherheitsimplementierung nochmals separiert.
Eine genauere Beschreibung ihrer Funktion ist im Kapitel des jeweiligen Pakets zu finden.



a) Der Controller sendet der View die benötigten Anzeigedetails und Benachrichtigungen über Aktualisierungen.
b) Die View sendet dem Controller per Listener die Benutzerinteraktionen.
c) Der Controller verwendet die vom Service angebotenen Operationen zur Umsetzung der jeweiligen Anfrage.
d)+e) Component enthält etwaige separierte Klassen, die vom Programm benötigt werden.
f) Service führt die Steuerungslogik auf dem Model aus.

3 Hauptklassen

4 Paket view

Situierung:

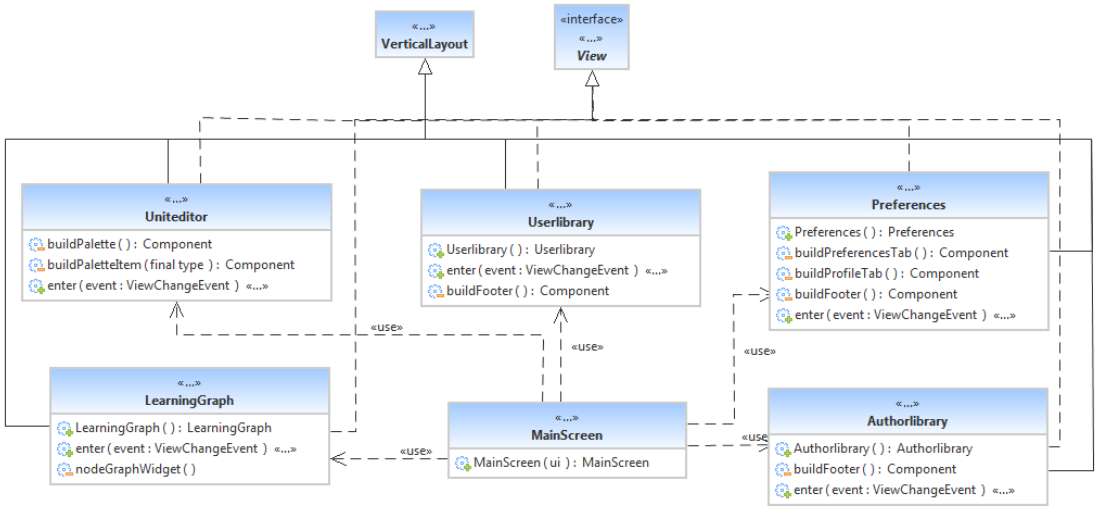
Diese Klasse ist für die Darstellung der Benutzeroberfläche und Entgegennahme der Benutzerinteraktion zuständig.

Blackbox-Beschreibung

Das Paket besitzt für jede Benutzeransicht eine eigene Klasse, die die Darstellung von Hauptansicht, Autorenbibliothek, Benutzerbibliothek, Einstellungen, Lernkurseditor, Lerneinheitseditor und Fehlermeldungen festlegt und entsprechende Benutzereingaben entgegen nimmt.

Interner Aufbau

Die Hauptansicht MainScreennimmt im Konstruktor die Vaadin UI entgegen und vereint alle weiteren Ansichtsklasse, zwischen denen per Navigator gewechselt werden kann. Diese erweitern das Vaadin VerticalLayout und implementiert View. Die grundlegende Darstellung wird im Konstruktor festgelegt und die benötigten Listener hinzugefügt.



Feinentwurf

MainScreen
Der MainScreen enthält das Navigationsmenü und einen Navigator, der das Wechseln zwischen allen Ansichten ermöglicht.

- **+ MainScreen(ui : MyUI) : MainScreen**
  Hier wird das Menü und alle Ansichten hinzugefügt, um dem Controller-Paket die Navigation durch die UI zu ermöglichen.

Authorlibrary
Diese Klasse regelt die Darstellung der Autorenbibliothek

- **+ AuthorLibrary() : AuthorLibrary**
  Fügt alle benötigten Anzeigeelemente und Komponenten zusammen

- **- buildFooter() : Component**
  builFooter erzeugt ein HorizontalLayout-Component, welches der Ansicht im Konstruktor hinzugefügt wird und die Darstellung der Liste, sowie Details der anzuzeigenden Lernkurse ermöglicht.

Userlibrary
Diese Klasse regelt die Darstellung der Benutzerbibliothek

- **+ Userlibrary() : Userlibrary**
  Fügt alle benötigten Anzeigeelemente und Komponenten zusammen

- **- buildFooter() : Component**
  builFooter erzeugt ein HorizontalLayout-Component, welches der Ansicht im Konstruktor hinzugefügt wird und die Darstellung der Liste, sowie Details der anzuzeigenden Lernkurse ermöglicht.

Preferences
Diese Klasse ist die Anzeige der Benutzereinstellungen und persönlicher Daten.

- **+ Preferences() : Preferences**
  Fügt alle benötigten Anzeigeelemente und Komponenten zusammen

- **- buildPreferencesTab() : Component**
  Stellt die benötigten Elemente der Einstellungsanzeige als VerticalLayout zusammen

- **- buildProfileTab() : Component**

Stellt die benötigten Elemente der Profilanzeige als VerticalLayout zusammen

- **- buildfooter() : Component**
  Alle verbliebenen Elemente, wie die Speichern-Schaltfläche, können der Ansicht über den footer hinzugefügt werden.

LearningGraph
Diese Klasse beschreibt die Ansicht des Lernkurseditors.

- **+ LearningGraph() : LearningGraph**
  Fügt alle Komponenten zur Darstellung des Editors in einem VerticalLayout zusammen.

- **- nodeGraphWidget() : void**
  Die Erstellung des Graphen wird hier durch die Integration von JointJS realisiert.

Uniteditor Diese Klasse regelt die Anzeige des Lerneinheitseditors.

- **+ Uniteditor() : Uniteditor**
  Stellt die benötigten Elemente des Lerneinheitseditors als VerticalLayout zusammen

- **- buildPalette() : Component**
  Hier wird die Toolbar für den Editor aus einzelnen Tools zusammengestellt.

- **- buildPaletteItem(type : PaletteItemType) : Component**
  Hier können die Tools für den Editor erstellt werden.

5 Paket controller

Situierung

In dem paket Controller soll die Logik der UI ausgelagert werden und die Navigation des MainScreens gehandhabt werden.

Blackbox-Beschreibung

In einem LoginFormListener werden die eingegeben Userdaten mittels eines AuthenticationManagers überprüft. Im Erfolgsfall wechselt die UI zum MainScreen. Hier wird mittels der Menu Klasse die Navigation zwischen den Reitern ermöglicht.

Interner Aufbau

Feinentwurf

LoginFormListener
Die Klasse erbt von der Vaadin UI Klasse Button.Clicklistener. Es wird ein privater authManager deklariert. In der Methode buttonClick, welche als Parameter ein Button.ClickEvent übergeben wird, wird zunächst der gefeuerte Button, das zugehörige Login Form und anschließend die darin eingegeben Strings als Felder initialisiert. Anschließend werden dem Authentificationmanager die Zugansdaten übergeben und das Ergebnis in der von Spring bereitgestellten Klasse SecurityContextHolder abgespeichert. Dies alles passiert in einem try Block sodass eine AuthenticationException gefangen werden kann. Sollte es dazu kommen wird diese angezeigt indem der Methode show der von Vaadin bereitgestellten Notification Klasse die Nachricht der Exception übergeben wird.

- + **buttonClick(Button.ClickEvent)**

textbf Menu
Die Menu Klasse erbt von der Vaadin CssLayout Komponente, welche es ermöglicht Komponenten und ihre Captions in das gleiche div Element zu rendern.

- + **Menu(navigator : Navigator) : Menu**
  Dem Konstruktor wird ein Navigator übergeben. Navigator ist eine von Vaadin bereitgestellte Klasse, die es ermöglicht das Views registriert oder dynamisch generiert werden können. Außerdem kann sie auch als Listener für View Veränderungen

fungieren und vor allem switchen zwischen den Views ermöglichen.

- **+ addView(view : View, name : String, caption String, icon Resource) : void**
  In dieser Methode soll die Regestrierung der als Parameter übergebenen Views im Navigator bereitgestellt werden.

- **- createViewButton(name : String, caption : String, icon : Resource) :void**
  Hier werden die Menüreiter generiert. Falls ein Button click event gefeuert wird, wechselt der Navigator zu der von dem Button repräsentierten View.

6 Paket service

Situierung

Das Service-Paket enthält die Steuerungslogik der Daten. Während der "Controller" die Benutzerinteraktion verarbeitet, ist der "Service" für die Betriebslogik und Daten verantwortlich.

Blackbox-Beschreibung

Interner Aufbau

Feinentwurf

DatabaseException

- **+ DatabaseException(reason : String) : DatabaseException**
- **+ getReason() : String**
- **+ setReason(reason : String) : void**

JDBCConnection
Die Klasse JDBCConnection ist als Singleton realisiert und dient als Kommunikationsschnittstelle zwischen der Datenbank und dem Programm. Sie Baut die Verbindung zur Datenbank auf, kann diverse Exceptions werfen und liefert ein Statement zurück.

- **+ getInstance() : JDBCConnection**
  Initialisiert eine Instanz, falls zuvor noch keine generiert wurde und liefert diese zurück (Singleton).
- **- JDBCConnection() : JDBCConnection**
  Privater Konstruktor, da nur eine Instanz existieren soll, auf die per getInstance zugegriffen werden kann.
- **- initConnection() : void**
  Initialisiert die Verbindung zur Datenbank

- **- openConnection() : void**
  Öffnet die Verbindung zur Datenbank

- **+ getStatement() : Statement**
  Liefert das Statement zurück.

- **+ closeConnection() : void**
  Schließt die Verbindung.

AddNewUserListener
Implementiert einen Listener für das hinzufügen eines Users

- **+ getInstance() : LanguageControl**
  Liefert die Instanz, da Singleton.

UserDAO
DataAcessObjekt - Datenzugriffsobjekt für User

- **+ createDBTable()**
  erstellt einen DbTable für User.

- **+ findAll : List<User>**
  Liefert eine Liste aller Benutzer.

- **+ save(User user) : void**
  Speichert einen Benutzer

Language
Enum, welches den umgang mit den Sprachen erleichtern und sichern soll.

LanguageControl
Diese Klasse ist als Singleton realisiert und dient als schnittstelle zwischen Programm und den Sprachpaketen.

- **+ getInstance() : LanguageControl**
  Liefert die Instanz, da Singleton.

- **+ getRes() : ResourceBundle**
  Liefert die Ressourcedatei mit dem Sprachpaket

- **+ getCurrentLocale() : Locale**
  Liefert das aktuell gesetzte Locale

- **+ setCurrentLocale(currentLocale : Locale) : void**

Setzt das Locale

- + **setCurrentLocale(newLanguage : Language) : void**
  Setzt ds Locale mithilfe eines LanguageEnums

LanguageControl
Diese Klasse ist als Singleton realisiert und dient als schnittstelle zwischen Programm und den Sprachpaketen.

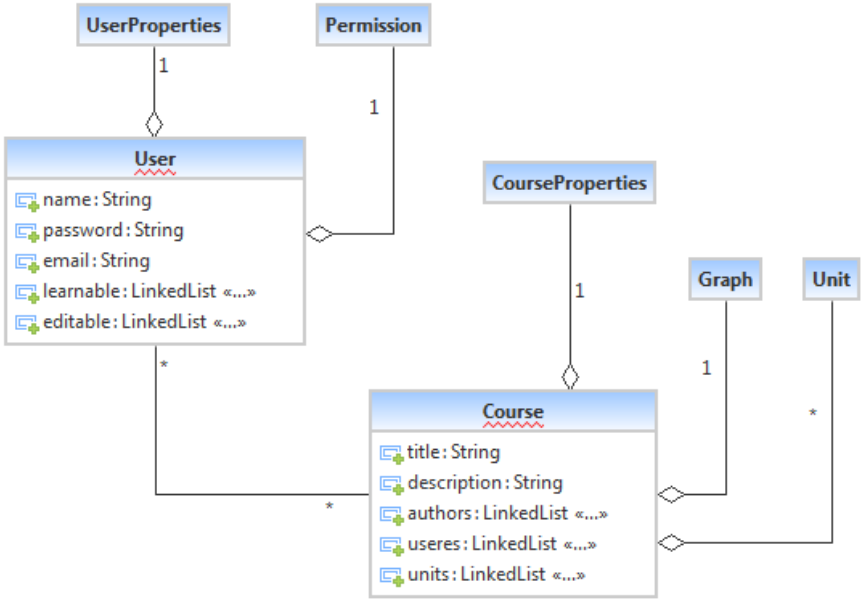
- + **getInstance() : LanguageControl**
  Liefert die Instanz, da Singleton.

7 Paket model

Situierung

Dieses Paket definiert alle Datenbankeinheiten die gespeichert werden müssen und stellt sie dem Service-Paket zur verfügung.

Interner Aufbau



Feinentwurf

Course

Einheit eines Lernkurses, mit allen folgenden relevanten Kursdaten.

- **String title**
  Der Name des Kurses

- **String description**
  Die Beschreibung des Kurses

- **LinkedList<User> authors**
  Die Liste aller zugeteilten Benutzer, die den Kurs bearbeiten können

- **LinkedList<User> useres**
  Die Liste aller zugeteilten Benutzer, die den Kurs einsehen können.

- **LinkedList<Unit> Units**
  Die Liste aller enthaltenen Lerneinheiten

- **Graph graph**
  Die Graphenstruktur, die die Einheiten verbindet

- **CourseProperties properties**
  Die vom Autor eingestellten Optionen des Kurses

User
Einheit eines Benutzers, mit allen folgenden relevanten Daten.

- **String name**
  Name des Benutzers

- **String email**
  Email des Benutzers

- **String password**
  Passwort der Benutzers

- **LinkedList<Course> learnable**
  Liste der zugeteilten Kurse

- **LinkedList<Course> editable**
  Liste der bearbeitbaren Kurse

- **Permissions permissions**
  Rechte des Benutzers

- **UserProperties properties**
  Accounteinstellungen des Benutzers

Unit
Dateneinheit der Lerneinheit bestehend aus einer oder mehreren HTML Seiten

Test

Die Testeinheiten bilden eine eigene Dateneinheit, da andere Elemente als bei der Lerneinheit (Unit) gespeichert werden müssen.

Graph
Dateneinheit, welche die Verlinkung von Lerneinheiten (Unit) darstellen kann.

UserProperties
Dateneinheit der Accounteinstellungen eines Benutzers, wie z.B. Sprache.

CourseProperties
Dateneinheit der Kurseinstellungen, z.B. bezüglich Sichbarkeit der Lerneinheiten.

Permissions
Dateneinheit, die ein Rechteset definieren kann.
Folgende rechte sollten als boolean enthalten sein:

- Lernkurse zugeteilt bekommen, verlassen
- zugeteilte Lernkurse bearbeiten
- eigene Lernkurse erstellen, löschen
- eigene Lernkurse bearbeiten (u.a. Lerneinheiten erstellen, ändern, löschen)
- eigene Lernkurse mit den anderen Benutzern teilen
- neue Benutzerkonten erstellen, löschen
- Rechte eines Benutzers ansehen, zuteilen, zurückrufen
- alle bereits erstellten Benutzerkonten ansehen

8 Paket component

9 Paket authentification

10 Datenbank