

MODUL 133

ZUSATAUFTRAG M133: ERWEITERUNG DER APPLI- KATION BOOKS-SUBJECTS

Michael Abplanalp



Erweiterung der Applikation Books-Subjects

In einem ersten Schritt soll die Applikation etwas modifiziert und Rails konform erstellt werden. In einem zweiten Schritt wird die Benutzeroberfläche zum Bearbeiten der Fachgebiete erstellt. Schliesslich wollen wir ein Model hinzufügen, das mit den Büchern eine n:n-Beziehung hat: Autoren. Jeder Autor kann n Bücher haben, jedes Buch n Autoren.

Die einzelnen Aufgaben:

- Wir arbeiten mit zwei Controllern: `Subjects` und `Books`. Falls Sie nicht bereits mit zwei Controllern arbeiten, müssen diese zuerst erstellt werden.
- Routen erstellen: Wir arbeiten mit `resources`, weil damit die Arbeit mit Resources und Links stark vereinfacht wird. Die sieben notwendigen Routen plus Helpers werden damit automatisch erstellt.
- Im Controller `Subjects` werden die Methoden erfasst, um alle CRUD-Operationen ausführen zu können.
- Zuletzt werden die notwendigen Views erstellt und es werden Funktionstests durchgeführt.

Zusatzauftrag:

- Erstellen des Models `Author` und definieren einer n:n-Beziehung zu den Büchern.
- Erstellen des Controllers `Authors` und erfassen von allen notwendigen Methoden für die CRUD-Operationen.
- Erstellen der notwendigen Views und Testen der Applikation.

Model erstellen

Die Models `Subject` und `Book` sollten bereits vorhanden sein. Ist dies nicht der Fall, holen Sie es nach (siehe Arbeitsblatt 4).

Controller erstellen

Falls Sie bisher nur mit einem Controller gearbeitet haben (für die Bücher und Fachgebiete), dann erstellen Sie nun zwei neue Controller:

```
$ rails g controller Subjects  
$ rails g controller Books
```



Der Name des Controllers sollte in der Mehrzahl stehen.

Routen erstellen

Rails hat ein REST API, d.h. alle Seiten und Funktionen einer Applikation werden via URL aufgerufen und ausgeführt. Der Rails Router erkennt URLs und führt die entsprechende Aktion (Methode) im Controller aus.

Die Routen und das Mapping sind in der Datei `config/routes.rb` festgehalten. Wir löschen die vorhandenen Routen `/books/...` und fügen die folgenden zwei Einträge in diese Datei ein:

```
Rails.application.routes.draw do
  resources :subjects
  resources :books
end
```

Damit erstellt Rails automatisch sieben Routen mit entsprechenden Mappings zum Controller. Für den Subjects-Controller sind das die folgenden Einträge:

HTTP Verb	Pfad	Controller#Action	Beschreibung
GET	/subjects	subjects#index	Liste mit allen Fachgebieten anzeigen
GET	/subjects/new	subjects#new	HTML-Formular anzeigen zum Erfassen eines neuen Fachgebietes
POST	/subjects	subjects#create	Neues Fachgebiet in die DB speichern
GET	/subjects/:id	subjects#show	Details eines Fachgebietes anzeigen
GET	/subjects/:id/edit	subjects#edit	HTML-Formular anzeigen zum Bearbeiten eines Fachgebietes
PATCH/PUT	/subjects/:id	subjects#update	Änderungen in die DB speichern
DELETE	/subjects/:id	subjects#destroy	Ein Fachgebiet löschen

Entsprechend werden natürlich auch die Routen für den Books-Controller generiert.



Für jede Route muss im Controller eine entsprechende Methode eingetragen werden, also die Methoden: `index`, `new`, `create`, `show`, `edit`, `update`, `destroy`

Neben den sieben Routen werden ein paar "Helpers" (Programmierhilfen) zur Verfügung gestellt, damit die Routen einfacher erstellt werden können:

Helper	Gibt zurück
<code>subjects_path</code>	<code>/subjects</code>
<code>new_subject_path</code>	<code>/subjects/new</code>
<code>edit_subject_path(:id)</code>	<code>/subjects/:id/edit</code>
<code>subject_path(:id)</code>	<code>/subjects/:id</code>

Die Routen und Helpers werden in den Views verwendet, um die verschiedenen CRUD-Operationen auszuführen.

Beispiel:

```
<table>
  <% Subject.all.each do |s| %>
    <td><%= s.name</td>
    <td><%= link_to "Ändern", edit_subject_path(s) %></td>
    <td><%= link_to "Löschen", subject_path(s), method: :delete%></td>
  <% end %>
</table>
```

Der Ändern-Link wird in HTML wie folgt ausgegeben:

`Ändern`, wobei 1 = die ID des ersten Fachgebietes

Der Löschen-Link wird in HTML wie folgt ausgegeben:

`Löschen`, wobei 2 = die ID des zweiten Fachgebietes

Book-Views anpassen

Passen Sie in allen Book-Views die Links gemäss vorherigem Kapitel und nachfolgender Tabelle an:

View/Seite	Rails Link
index: Bücherliste	link_to "Index", books_path
new: Neues Buch	link_to "Neues Buch", new_books_path
show: Buch anzeigen	link_to "[Buchtitel]", book_path(@book)
edit: Buch ändern	link_to "Ändern", edit_book_path(@book)
destroy: Buch löschen	link_to "Löschen", book_path(@book), method: :delete

`@book` bezeichnet ein Buchobjekt, in einer Schleife `Book.all.each do |b|` ist das Buchobjekt in der Variablen `b`, also wird anstatt `@book` einfach `b` geschrieben.

Methoden im Controller erfassen

Wir arbeiten mit zwei Controllern: `subjects_controller.rb` und `books_controller.rb`. Die Methoden im Books-Controller sollten bereits vorhanden sein. Informationen dazu finden Sie im Arbeitsblatt 4.

Die Methoden im Subjects-Controller sehen wie folgt aus:

```
class SubjectsController < ApplicationController
  def index
    @subjects = Subject.all
  end

  def new
  end

  def create
```

```
subject = Subject.new(params.require(:subject).permit(:name))
if subject.save
  redirect_to action: "index"
else
  render action: "new"
end
end

def show
  @subject = Subject.find(params[:id])
end

def edit
  @subject = Subject.find(params[:id])
end

def update
  @subject = Subject.find(params[:id])
  if @subject.update_attributes(params.require(:subject).permit(:name))
    redirect_to action: "index"
  else
    render action: "edit"
  end
end

def destroy
  if Book.where(subject_id: params[:id]).count > 0
    redirect_to action: "index", error: true
  else
    Subject.find(params[:id]).destroy
    redirect_to action: "index"
  end
end
end
```

Die Methoden sind praktisch identisch mit denjenigen des Books-Controllers. Eine Ausnahme betrifft die Methode/View `destroy`:

In einem ersten Schritt wird geprüft, ob ein oder mehrere Bücher diesem Fachgebiet zugeordnet sind. Falls ja, kann das Fachgebiet nicht gelöscht werden. In der Methode `index` könnte eine entsprechende Fehlermeldung erstellt werden, die darauf in der View angezeigt wird.

Views erstellen

Die Views für die Bücher sollten wiederum bereits vorhanden sein. Ist dies nicht der Fall, dann schlagen Sie die Informationen dazu im Arbeitsblatt 5 nach.

Die in der Folge vorgestellten Views sind als Vorschläge zu betrachten. Sie sollten möglichst alle Funktionen erstellen. Sie können jedoch nach Belieben zusätzliche Funktionen einbauen, in der Wahl des Designs und der Benutzerführung sind Sie frei.



Der Code für die Views ist nicht vollständig. Der Hauptteil des Codes ist vorhanden, teilweise vorhanden oder es werden zumindest Hinweise geliefert (in **violetter** Farbe). Das Design wurde mit Bootstrap erstellt und fehlt vollständig. Sie sind in der Wahl des Designs natürlich frei.

Für die Subjects erstellen wir die folgenden 4 Views im Verzeichnis `app/views/subjects`:

- `index.html.erb`
- `new.html.erb`
- `show.html.erb`
- `edit.html.erb`

View subjects#index

Die View mit allen Fachgebieten sieht wie folgt aus:

Fachgebiete		
Fachgebiet		
Linux	Ändern	Löschen
Python	Ändern	Löschen
Rails	Ändern	Löschen
SQL	Ändern	Löschen
Web Programming	Ändern	Löschen
Neues Fachgebiet		

`app/views/subjects/index.html.erb`:

```
<h3>Fachgebiete</h3>
<table>
  <tr>
    <th>Name</th>
    <th colspan="2"> </th>
  </tr>
  <% @subjects.each do |s| %>
    <tr>
      <td><%= s.name %></td> als Link auf Fachgebiet zu ergänzen...
      <td><%= link_to "Ändern", edit_subject_path(s.id) %></td>
      <td><%= link_to "Löschen", subject_path(s.id), method: :delete %></td>
    </tr>
  <% end %>
</table>
Link auf "Neues Fachgebiet" zu ergänzen...
```

View subjects#new

Die View sieht wie folgt aus:

Fachgebiet erfassen

Name:

[Fachgebiet speichern](#) [Abbrechen](#)

app/views/subjects/new.html.erb:

```
<h3>Fachgebiet erfassen</h3>
<%= form_for :subject, url: {action: "create"} do |s| %>
  <%= s.label :name, "Name:" %>
  <%= s.text_field :name %>
  <%= s.submit "Fachgebiet speichern" %>
<% end %>
Abbrechen-Button zu ergänzen...
```

View subjects#show

Da die Fachgebiete nur das Attribut `name` besitzen, macht eine Detailview für ein einzelnes Fachgebiet keinen Sinn.

In der View `show` sollen deshalb alle Bücher dieses Fachgebiets aufgeführt werden:

Fachgebiet: Linux

Bücher im Fachgebiet

- [Linux Administration](#)
- [Linux Server](#)

[Fachgebiete](#)

Die Bücher sind jeweils als Link hinterlegt, mit Klick darauf gelangt man in die Detailview des entsprechenden Buches.

app/views/subjects/show.html.erb:

```
<h3>Bücher im Fachgebiet</h3>
<% @subject.books.each do |b| %>
  <%= b.title %> als Link auf Buch zu ergänzen...
<% end %>
Link auf "Fachgebiete" zu ergänzen...
```

View subjects#edit

Fachgebiet ändern

Name:

[Änderungen speichern](#) [Abbrechen](#)

app/views/subjects/edit.html.erb:

```
<h3>Fachgebiet ändern</h3>  
<%= form_for @subject, url: {action: "update"} do |s| %>  
Code ergänzen (entspricht "Fachgebiet erfassen")  
<% end %>
```

Testen der Applikation

Testen Sie die verschiedenen Seiten und Funktionen der Fachgebiete und Bücher. Wenn Sie mit dem Ergebnis zufrieden sind, zeigen Sie es der Lehrperson.

Applikation mit Autoren ergänzen

Diese Aufgabe ist als Zusatzaufgabe für schnelle Lernende zu verstehen und gehört nicht zum obligatorischen Stoff des Moduls.

Model Author erstellen und Migration AuthorsBooks erstellen

1. Erstellen Sie in der Konsole das neue Model `Author` (englische Schreibweise für Autor).
2. Tragen Sie die n:n-Beziehung ein.

`app/models/book.rb`:

```
class Book < ApplicationRecord
  belongs_to :subject
  has_and_belongs_to_many :authors
  Allfällige Validationen...
end
```

`app/models/author.rb`:

```
class Author < ApplicationRecord
  has_and_belongs_to_many :books
end
```

3. Löschen Sie die Datei `db/migrate/NNNNNNNNNNNNNNNN_create_author.rb`
4. Erstellen Sie in der Konsole die neue Migration `AuthorsBooks`.
5. Passen Sie die soeben erstellte Datei `db/migrate/NNNNNNNNNNNNNNNN_authors_books.rb` wie folgt an:

```
class AuthorsBooks < ActiveRecord::Migration[5.2]
  def change
    create_table :authors do |t|
      t.string :name
      t.timestamps
    end
    create_table :authors_books, id: false do |t|
      t.belongs_to :author, index: true
      t.belongs_to :book, index: true
    end
  end
end
```

Führen Sie anschliessend die Migration durch.

Es werden die Tabelle `authors` und die für die n:n-Beziehung notwendige Zwischentabelle `authors_books` erstellt.

Routen für die Autoren erstellen

Ergänzen Sie die Datei `config/routes.rb` mit den Autoren:

```
Rails.application.routes.draw do
  resources :subjects
  resources :books
  resources :authors
end
```

Controller Authors erstellen

Erstellen Sie in der Konsole den Controller `Authors`.

Es sollte für Sie kein Problem sein, die Methoden für die Autoren zu erstellen. Der Code ist praktisch identisch mit den Fachgebieten.

View authors#index erstellen

Autoren			
Autor	Bücher		
Herbert Liechti	Python for Dummies, Linux Administration,	Ändern	Löschen
John Doe	Linux Server, Python for Dummies, JavaScript,	Ändern	Löschen
Michael Abplanalp	Python for Dummies, SQL Database, HTML 5 Handbuch,	Ändern	Löschen
Ralph Maurer	PHP & MySQL, Ruby on Rails, Python for Dummies, Linux Administration, JavaScript,	Ändern	Löschen
Neuer Autor			

In dieser View sind alle Autoren mit ihren Büchern aufgelistet (0, 1 oder mehrere Bücher pro Autor). Der Controller muss nicht angepasst werden, denn mit den Autorenobjekten werden automatisch auch die Bücher mitgeliefert.

Erstellen Sie `app/views/authors/index.html.erb` wie folgt:

```
<table>
  usw...
  <% @authors.each do |a| %>
    Titel des Buches anzeigen
    <% a.books.collect.each do |b| %>
      <%= b.title + ", " %>
    <% end %>
  usw...
  <% end %>
</table>
```

`a.books.collect` "sammelt" alle Bücher des Autoren ein, die `each`-Schleife kennen Sie ja bereits. Innerhalb der Schleife kann auf alle Attribute der Bücher zugegriffen werden.

View authors#new erstellen

Autor erfassen	
Name:	<input type="text"/>
Autor speichern	Abbrechen

View authors#edit erstellen

Autor ändern

Name:

View subjects#index anpassen

Die View `subjects#index` soll wie folgt aussehen:

Fachgebiete			
Fachgebiet	Bücher		
Linux	Linux Administration, Linux Server,	Ändern	Löschen
Python	Python for Dummies,	Ändern	Löschen
Rails	Ruby on Rails,	Ändern	Löschen
SQL	SQL Databaseee,	Ändern	Löschen
Web Programming	HTML 5 Handbuch, PHP & MySQL, JavaScript,	Ändern	Löschen
Neues Fachgebiet			

Es sollen neu die Bücher (0, 1 oder mehrere) pro Fachgebiet angezeigt werden. Der Controller muss nicht angepasst werden.

Analog zu der Autorenliste können Sie in der View `app/views/subjects/index.html.erb` mit `s.books.collect.each do |s|` pro Fachgebiet eine Schleife über alle Bücher machen und den Buchtitel ausgeben.

View books#index anpassen

Die View `books#index` soll wie folgt aussehen:

Bücherliste				
Titel	Autoren	Fachgebiet		
HTML 5 Handbuch	Michael Abplanalp,	Web Programming	Ändern	Löschen
JavaScript	John Doe, Ralph Maurer,	Web Programming	Ändern	Löschen
Linux Administration	Ralph Maurer, Herbert Liechti,	Linux	Ändern	Löschen
Linux Server	John Doe,	Linux	Ändern	Löschen
PHP & MySQL	Ralph Maurer,	Web Programming	Ändern	Löschen
Python for Dummies	John Doe, Ralph Maurer, Herbert Liechti, Michael Abplanalp,	Python	Ändern	Löschen
Ruby on Rails	Ralph Maurer,	Rails	Ändern	Löschen
SQL Databaseee	Michael Abplanalp,	SQL	Ändern	Löschen
Neues Buch				

Es sollen neu die Autoren (0, 1 oder mehrere) und das Fachgebiete (genau 1) pro Buch angezeigt werden. Der Controller muss nicht angepasst werden.

Analog zu der Autoren- und Fachgebietsliste können Sie in der View `app/views/bookss/index.html.erb` mit `b.authors.collect.each do |a|` pro Buch eine Schleife über alle Autoren machen und deren Namen ausgeben.

Mit `b.subject` kann auf das Fachgebiet des jeweiligen Buches zugegriffen werden.



Nun kommen wir zur grössten Herausforderung: Das Erfassen der n:n-Beziehung. Wir streben eine Lösung an, die möglichst einfach für den Benutzer ist: Die Autoren werden direkt beim Erfassen bzw. Ändern eines Buches erfasst, mittels Liste, die eine Mehrfachauswahl erlaubt.

Die Programmierung ist nicht ganz einfach, mit der folgenden Hilfe sollte sie jedoch gelingen.

Books-Controller anpassen (new und edit)

Als Erstes müssen wir den Books-Controller anpassen, damit beim Anzeigen des Erfassen- und Ändern-Formulars die Autoren angezeigt werden: Speichern Sie in beiden Methoden neben den Fachgebieten auch alle Autoren in eine Instanzvariable (`Author.all`).

View books#new anpassen

Buch erfassen

Name:

Preis:

Fachgebiet:

Autoren:

Beschreibung:

Linux

Herbert Liechti

John Doe

Michael Abplanalp

Ralph Maurer

Buch speichern

Abbrechen

Wir erstellen eine Select-Liste, in der eine Mehrfachauswahl möglich ist:

```
<%= f.collection_select :author, @authors, :id, :name, {}, {size: "6", multiple: true} %>
```

f	Bezieht sich auf das Formular
:author	Der Name der Liste
@authors	Alle Autoren (Objekte)
:id	Attribut von @authors, der Wert wird beim Listeneintrag hinterlegt
:name	Attribut von @authors, der Wert wird in der Liste angezeigt
size	So viele Einträge werden in der Liste angezeigt
multiple	Bewirkt, dass mehrere Einträge ausgewählt werden können (mit <Ctrl>)

Folgender HTML-Code wird erzeugt:

```
<select size="6" multiple="multiple" name="book[author][]" id="book_author">  
  <option value="4">Herbert Liechti</option>  
  ..USW.  
</select>
```

View books#edit anpassen

Buch ändern

Name:

Preis:

Fachgebiet:

Autoren:

Beschreibung:

Linux

John Doe

Ralph Maurer

Herbert Liechti

Michael Abplanalp

Buch speichern

Abbrechen

Das Formular ist wie das Erfassen-Formular aufgebaut. Zusätzlich müssen die Autoren des Buches angezeigt werden. Der Rails-Helper sieht wie folgt aus:

```
<%= f.collection_select :author, @authors, :id, :name,  
  {selected: @book.authors.map(&:id)}, {size: "6", multiple: true} %>
```

Wir geben in den Optionen mit: `{selected: @book.authors.map(&:id)}`.
Das bewirkt, dass die Autoren dieses Buches in der Liste ausgewählt sind.

Controller Books anpassen (create und update)

Nun müssen wir noch den Controller anpassen, damit die ausgewählten Autoren auch in die Datenbank gespeichert werden.

Dazu verwenden wir folgende Methode, die sowohl in `create` als auch in `update` aufgerufen wird:

```
def insert_authors(book)
  params[:book][:author].each do |a|
    if a.to_i > 0
      book.authors << Author.find(a)
    end
  end
end
```

Wir geben das aktuelle Buch als Parameter `book` der Methode mit.

In der Methode werden alle Einträge der Autorenliste durchlaufen: `params[:book][:author]` ist die Liste, die vom Formular übergeben worden ist. Im `if` wird geprüft, ob der Listeneintrag ausgewählt worden ist (`if a.to_i > 0`). 0 bedeutet "nicht ausgewählt", eine positive Zahl bedeutet "ausgewählt".

Falls der Autor ausgewählt worden ist: Der Autor wird aus der Datenbank geholt `Author.find(a)` und dem Buch als Autor hinzugefügt `book.authors <<`. Dabei wird ein Eintrag in der Zwischentabelle `authors_books` generiert.

Beim Editieren eines Buches kann es sein, dass die Autoren geändert werden. Die einfachste Möglichkeit zum Ändern ist, zuerst die alten Autoren zu löschen und dann die neuen einzufügen.

Die Methode zum Löschen wird in `update` aufgerufen und sieht wie folgt aus:

```
def delete_authors(book)
  bookAuthors = Book.find(params[:id]).authors
  bookAuthors.each do |ba|
    book.authors.delete(ba)
  end
end
```

Wir geben das aktuelle Buch als Parameter `book` der Methode mit.

Zuerst werden alle Autoren des Buches aus der Datenbank geholt `Book.find(params[:id]).authors`. Danach werden in einer `each`-Schleife alle Autoren durchlaufen gelöscht `book.authors.delete(ba)`.