
Play! Online-Kochbuch

Modul MI-WebT-B



Sebastian Boosz, M.Sc.

Lehrstuhl für Medieninformatik

Fakultät für Wirtschaftsinformatik und Angewandte Informatik

Otto-Friedrich-Universität Bamberg, 96047 Bamberg

Telefon: 0951 863-2853

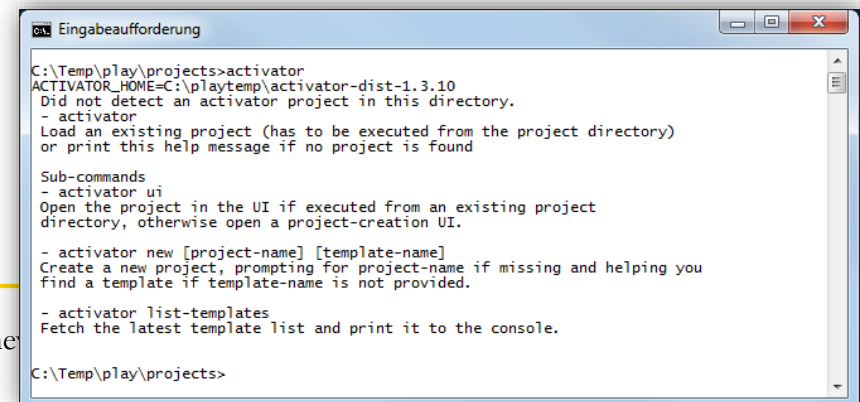
Email: sebastian.boosz@uni-bamberg.de

WWW: <http://www.uni-bamberg.de/minf/>

Getting Started: Installation

<http://www.playframework.com/documentation/2.5.x/Installing>

- Voraussetzung ist Java Development Kit (**JDK**) 8 oder höher
 - Sicherstellen, dass Java in der **PATH**-Systemvariable eingetragen ist
 - (bei Windows: Start → Computer → Einstellungen → Erweiterte Systemeinstellungen → Umgebungsvariablen ...)
- Aktuelle Version von **Play** herunterladen
 - <http://www.playframework.com/>
 - Download: browse all versions → **Offline Distribution**
 - **Extrahieren** des Archivs an einen beliebigen Ort
 - Pfad zum „bin“-Verzeichnis des entpackten Ordners der **PATH**-Systemvariable hinzufügen
- **Überprüfen**, ob der Befehl ‚activator‘ der Windows Eingabeaufforderung bekannt ist



```

C:\Temp\play\projects>activator
ACTIVATOR_HOME=C:\playtemp\activator-dist-1.3.10
Did not detect an activator project in this directory.
- activator
  Load an existing project (has to be executed from the project directory)
  or print this help message if no project is found

Sub-commands
- activator ui
  Open the project in the UI if executed from an existing project
  directory, otherwise open a project-creation UI.
- activator new [project-name] [template-name]
  Create a new project, prompting for project-name if missing and helping you
  find a template if template-name is not provided.
- activator list-templates
  Fetch the latest template list and print it to the console.

C:\Temp\play\projects>
  
```

An den Rechnern im Pool

- Download & extrahieren beispielsweise nach C:\activator-1.3.10
- Command Prompt öffnen Systempfad erweitern
 - > Set PATH=%PATH%;C:\activator-1.3.10\bin
 - > Set PATH=%PATH%;C:\Program Files\Java\jdk\bin
 - Tab-Taste hilft teils beim Vervollständigen von Pfadangaben
- In ein beliebiges Verzeichnis wechseln und einen Ordner *projects* für zukünftige Projekte anlegen
 - > mkdir projects
- Nach play wechseln (> cd projects)

Getting Started: neues Projekt

- Ein neues Projekt erstellen (in neues Verzeichnis projects)
 - > `activator new` („>“ steht für die Eingabeaufforderung)
 - Erstellt ein Verzeichnis `recipes/` mit den wichtigsten Dateien und Verzeichnissen

```

C:\Windows\system32\cmd.exe

C:\Temp\play\projects>activator new
ACTIVATOR_HOME=C:\playtemp\activator-dist-1.3.10

Fetching the latest list of templates...

Browse the list of templates: http://lightbend.com/activator/templates
Choose from these featured templates or enter a template name:
  1) minimal-akka-java-seed
  2) minimal-akka-scala-seed
  3) minimal-java
  4) minimal-scala
  5) play-java
  6) play-scala
(hit tab to see a list of all templates)
> 5
Enter a name for your application (just press enter for 'play-java')
> recipes
OK, application "recipes" is being created using the "play-java" template.

To run "recipes" from the command line, "cd recipes" then:
C:\Temp\play\projects\recipes>activator run

To run the test for "recipes" from the command line, "cd recipes" then:
C:\Temp\play\projects\recipes>activator test

To run the Activator UI for "recipes" from the command line, "cd recipes" then:
C:\Temp\play\projects\recipes>activator ui

C:\Temp\play\projects>
  
```

Die wichtigsten Verzeichnisse

- Die wichtigsten Dateien und Verzeichnisse im Überblick
 - app/
 - Hauptteil der Anwendung, beherbergt controllers/, models/ und views/
 - Beinhaltet die *.java und *.html Dateien
 - conf/
 - Hier liegen alle wichtigen Konfigurationsdateien wie
 - application.conf und routes
 - public/
 - Beinhaltet alle öffentlichen Ressourcen: Bilder, *.js und *.css
 - test/
 - Beinhaltet mögliche JUnit-Tests der Anwendung

Die Play Konsole

■ Die Play Konsole

- Innerhalb jedes Play Projekts vorhanden
- Wird über den Befehl > `activator` gestartet
- `$ help` liefert eine Übersicht an Befehlen mit Erläuterung
- `$ run` startet den mitgelieferten Webserver (<http://localhost:9000/>)
 - Beenden des laufenden Webserver mit Strg + d
 - `$ compile` kompiliert den Quelltext auch ohne dass der Server läuft
- Konsole starten via `activator -jvm-debug 9999`, Anwendung kann über Port 9999 mit Java Platform Debugger Architecture gedebuggt werden
 - <http://www.playframework.com/documentation/2.5.x/IDE>
- `$ exit` beendet die Konsole und kehrt zur Eingabeaufforderung zurück

```

C:\Windows\system32\cmd.exe - activator

C:\Temp\play\projects\recipes>activator
ACTIVATOR_HOME=C:\playtemp\activator-dist-1.3.10
[info] Loading project definition from C:\Temp\play\projects\recipes\project
[info] Set current project to recipes (in build file:/C:/Temp/play/projects/recipes/)
[recipes] $
  
```

Getting Started

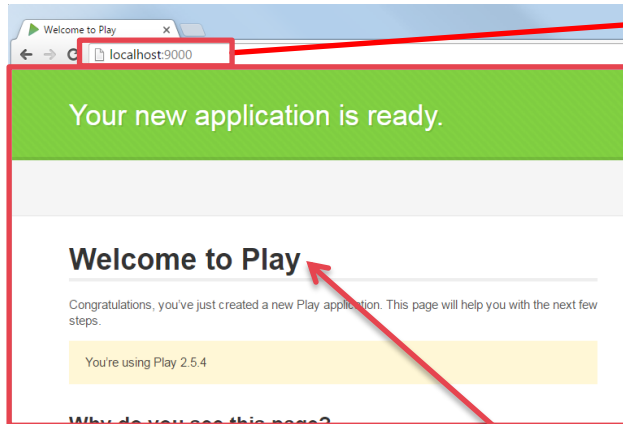
- Verwenden einer integrierten Entwicklungsumgebung (IDE)
 - Eine Play-Anwendung kann **ohne jegliche IDE** entwickelt werden, da das Framework selbst den Quelltext aktualisiert und kompiliert.
 - Play unterstützt dennoch einige IDE's, da diese Vorteile wie Autovervollständigung und Syntaxhervorhebung mit sich bringen
 - Zur Einrichtung der jeweiligen IDE siehe <http://www.playframework.com/documentation/2.5.x/IDE>



Routing einer Anfrage

Server

Client



- ❶ Anfrage des Wurzelverzeichnis /
- ❷ Nachschlagen der Route von / zu einem Controller und Aufruf des Controllers
- ❸ Auswahl einer View
- ❹ Rendern der View und Übertragung an den Browser

```

1 # Routes
2 # This file defines all application routes (Higher priority routes first)
3 # ~~~~
4
5 ❶ An example controller showing a sample home page
6 GET / controllers.HomeController.index
7 ❷ An example controller showing how to use dependency injection
8 GET /count controllers.CountController.count
9 # An example controller showing how to write asynchronous code
10 GET /message controllers.AsyncController.message

```

conf/routes

```

11 public class HomeController extends Controller {
12
13     /**
14      * An action that renders an HTML page
15      * The configuration in the <code>routes</code>
16      * this method will be called when the
17      * <code>GET</code> request with a path
18      */
19     public Result index() {
20         return ok.index.render("Your new application is ready.");
21     }
22 }

```

action-Methode: erzeugt ein Result (hier 200 OK Response mit Body gemäß Template)

app/controllers/Application.java

```

1 @ (message: String)
2
3 @main("Welcome to Play") {
4
5     @play20.welcome(message, style = "Java")
6
7 }

```

app/views/index.scala.html



Objekt-relationales Mapping mit Ebeans

- Verwendung von Ebeans als ORM-Mechanismus (implementiert Java Persistence API)
 - Erweiterung der `project/plugins.sbt`-Datei (Zeile existiert bereits, ist aber auskommentiert)

```
// Play Ebean support, to enable, uncomment this line, and  
// enable in your build.sbt using enablePlugins(PlayEbean).  
addSbtPlugin("com.typesafe.sbt" % "sbt-play-ebean" % "1.0.0")
```

Scala

- Aktivieren von PlayEbean in der `/build.sbt`-Datei

```
version := "1.0-SNAPSHOT"  
  
lazy val root =  
    (project in file(".")).enablePlugins(PlayJava, PlayEbean)  
  
scalaVersion := "2.11.7"
```

Scala



Bereitstellen einer Datenbank

- Verwendung der In-Memory-Datenbank **H2**
 - Erweiterung der `conf/application.conf`-Datei
(Zeilen existieren teilweise bereits, sind aber auskommentiert)

```
config = "db"                # Verwende Konfiguration db
default = "default"          # mit den default-Werten
. . .
default.driver=org.h2.Driver  # Treiber für DB
default.url="jdbc:h2:mem:play" # Verbindung für DB
. . .
ebean.default = ["models.*"] # EBean als objektrelationaler
                             # Mapper, für Objekte im models
                             # Package
```

z.B. am Ende der Datei einfügen

Scala

- In den Klassen des `models-Package` Verwendung von **Java-Annotations** als Metadaten, um Objekte zu persistieren
 - beschreiben die Tabelle einer Klasse
 - Z.B. `@Entity`, `@Id`, ...

Ein simples Beispiel

- Online-Kochbuch (recipes)
 - Zutaten (Ingredient)
 - C_{reate} R_{ead} U_{pdate} D_{ele}te
 - Initiale Daten in die Datenbank
 - Rezept (Recipe)
 - Relationen

Zutaten (Ingredient)

- Projekt recipes anlegen (und in Eclipse importieren)
- Ebean und H2 Datenbank konfigurieren
- Im Verzeichnis app ein neues Package models anlegen
- Das Model für Ingredient anlegen
- Den Controller für Ingredient anlegen
- Die Routen anpassen
- Die Views für Ingredient anlegen
- Den Controller erweitern

Zutaten (Ingredient) – Model

Ingredient.java

```
package models;

import java.util.List;
import javax.persistence.Entity;
import javax.persistence.Id;
import play.data.validation.Constraints.Required;
import com.avaje.ebean.Model;

@Entity
public class Ingredient extends Model {
    @Id
    public Long iid;
    @Required
    public String name;
    public String description;
    public static Finder<Long, Ingredient> find = new Finder<Long, Ingredient>(Ingredient.class);

    public static void create(Ingredient ingredient) {
        ingredient.save();
    }
    public static List<Ingredient> read() {
        return find.all();
    }
    public static void update(Ingredient updatedIngredient) {
        updatedIngredient.update();
    }
    public static void delete(Long iid) {
        find.ref(iid).delete();
    }
}
```

Zutaten (Ingredient) – Controller

IngredientsCtrl.java

```
package controllers;

import models.Ingredient;
import play.data.FormFactory;
import play.mvc.Controller;
import play.mvc.Result;

public class IngredientsCtrl extends Controller {

    @Inject
    private FormFactory formFactory;
    public Result createIngredient() {
        return TODO;
    }

    public Result readIngredients() {
        return TODO;
    }

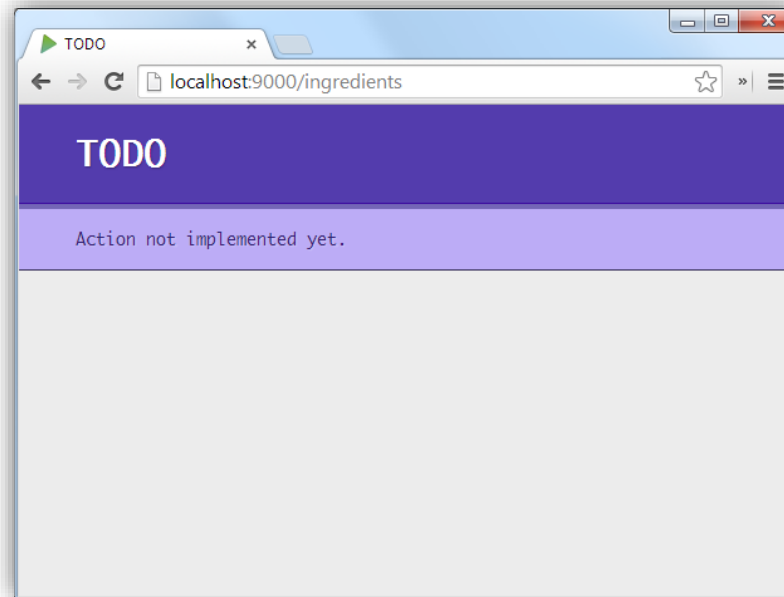
    public Result updateIngredient(Long iid) {
        return TODO;
    }

    public Result deleteIngredient(Long iid) {
        return TODO;
    }

    public Result storeIngredient() {
        return TODO;
    }
}
```

Zutaten (Ingredient) – Routen

# Ingredients			routes
GET	/ingredients	controllers.IngredientsCtrl.readIngredients()	
POST	/ingredients/store	controllers.IngredientsCtrl.storeIngredient()	
GET	/ingredients/create	controllers.IngredientsCtrl.createIngredient()	
POST	/ingredients/update/:iid	controllers.IngredientsCtrl.updateIngredient(iid: Long)	
POST	/ingredients/delete/:iid	controllers.IngredientsCtrl.deleteIngredient(iid: Long)	



Zutaten (Ingredient) – View

ingredients.scala.html

```
@(ingredients: List[Ingredient])

@main("Ingredients") {

  <h1>List of ingredients</h1>

  <a href="@routes.IngredientsCtrl.createIngredient()">Create Ingredient</a>

  <table>
  @for(ingredient <- ingredients) {
    <tr>
      <td>@ingredient.name</td>
      <td>@ingredient.description</td>
      <td>
        @helper.form(action = routes.IngredientsCtrl.updateIngredient(ingredient.iid)) {
          <input type="submit" value="Update">
        }
      </td>
      <td>
        @helper.form(action = routes.IngredientsCtrl.deleteIngredient(ingredient.iid)) {
          <input type="submit" value="Delete">
        }
      </td>
    </tr>
  }
  </table>
}
```


Zutaten (Ingredient) – Ctrl II

IngredientsCtrl.java

```
package controllers;

import models.Ingredient;
import play.data.FormFactory;
import play.mvc.Controller;
import play.mvc.Result;

public class IngredientsCtrl extends Controller {

    @Inject
    private FormFactory formFactory;

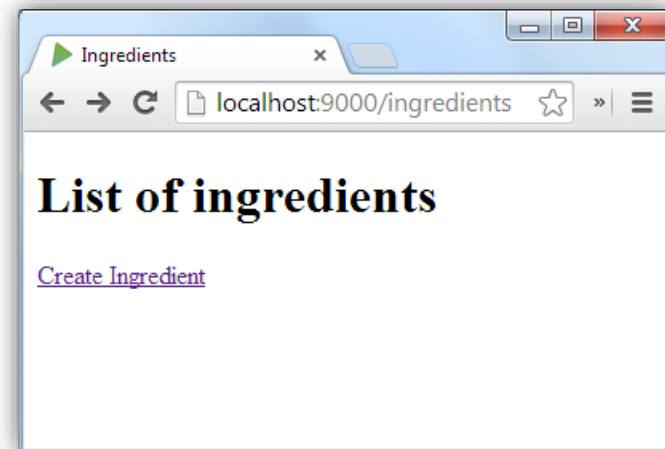
    public Result createIngredient() {
        ...
    }

    public Result readIngredients() {
        return ok(views.html.ingredients.render(Ingredient.read()));
    }

    public Result updateIngredient(Long iid) {
        ...
    }

    public Result deleteIngredient(Long iid) {
        ...
    }

    public Result storeIngredient() {
        ...
    }
}
```



Zutaten (Ingredient) – View II

```
@(formType: String, ingredientForm: Form[Ingredient])
```

ingredientsForm.scala.html

```
@main(formType + " Ingredient") {
```

```
  <h1>@formType Ingredient</h1>
```

```
  @helper.form(action = routes.IngredientsCtrl.storeIngredient()) {
```

```
    @helper.inputText(ingredientForm("name"))
```

```
    @helper.inputText(ingredientForm("description"))
```

```
    <input type="submit" value="Save">
```

```
  }
```

```
}
```

Zutaten (Ingredient) – Ctrl III

IngredientsCtrl.java

```
package controllers;

import models.Ingredient;
import play.data.FormFactory;
import play.data.Form;
import play.mvc.Controller;
import play.mvc.Result;

public class IngredientsCtrl extends Controller {

    ...

    public Result storeIngredient() {
        Form<Ingredient> ingredientForm = formFactory.form(Ingredient.class);
        Form<Ingredient> filledForm = ingredientForm.bindFromRequest();
        if (filledForm.hasErrors())
            return ok(views.html.ingredientsForm.render("Correct", filledForm));
        else {
            Ingredient ingredient = filledForm.get();
            Ingredient.create(ingredient);
            return redirect(routes.IngredientsCtrl.readIngredients());
        }
    }
}
```

Zutaten (Ingredient) – View III

```
@(formType: String, ingredientForm: Form[Ingredient])
```

ingredientsForm.scala.html

```
@main(formType + " Ingredient") {
```

```
  <h1>@formType Ingredient</h1>
```

```
  @helper.form(action = routes.IngredientsCtrl.storeIngredient()) {
```

```
    <input type="hidden" id="@ingredientForm("iid").id"
```

```
      name="@ingredientForm("iid").name"
```

```
      value='@ingredientForm("iid").value' />
```

```
    @helper.inputText(ingredientForm("name"))
```

```
    @helper.inputText(ingredientForm("description"))
```

```
    <input type="submit" value="Save">
```

```
  }
```

```
}
```

Zutaten (Ingredient) – Ctrl IV

IngredientsCtrl.java

```
package controllers;

import models.Ingredient;
import play.data.*;
import play.mvc.Controller;
import play.mvc.Result;

public class IngredientsCtrl extends Controller {

    ...

    public Result storeIngredient() {
        Form<Ingredient> ingredientForm = formFactory.form(Ingredient.class);
        Form<Ingredient> filledForm = ingredientForm.bindFromRequest();
        if (filledForm.hasErrors()) {
            return ok(views.html.ingredientsForm.render("Correct", filledForm));
        } else {
            Ingredient ingredient = filledForm.get();
            if (ingredient.iid == null){
                Ingredient.create(ingredient);
            } else {
                Ingredient.update(ingredient);
            }
            return redirect(routes.IngredientsCtrl.readIngredients());
        }
    }
}
```

Zutaten (Ingredient) – Ctrl V

IngredientsCtrl.java

```
package controllers;

import models.Ingredient;
import play.data.*;
import play.mvc.Controller;
import play.mvc.Result;

public class IngredientsCtrl extends Controller {

    ...

    public Result deleteIngredient(Long iid) {
        Ingredient.delete(iid);
        return redirect(routes.IngredientsCtrl.readIngredients());
    }

}
```

Initiale Daten in die Datenbank

- H2 In-Memory-Datenbank bei jedem Start leer
- Im Konstruktor der **ApplicationTimer**-Klasse können initiale Daten erstellt werden
 - Die Klasse wird bei Verwendung des *play-java* Templates automatisch erstellt
 - `app/services/ApplicationTimer.java`
 - Verhalten bei Beenden des Servers kann ebenfalls definiert werden

ApplicationTimer

ApplicationTimer.java

```
@Singleton
public class ApplicationTimer {

    private final Clock clock;
    private final ApplicationLifecycle applifecycle;
    private final Instant start;

    @Inject
    public ApplicationTimer(Clock clock, ApplicationLifecycle applifecycle) {
        this.clock = clock;
        this.applifecycle = applifecycle;
        // This code is called when the application starts.
        start = clock.instant();
        Logger.info("ApplicationTimer demo: Starting application at " + start);

        // Create ingredients here:
        Ingredient paprika = new Ingredient();
        paprika.name = "Paprika";
        paprika.description = "Rotes Gemüse";
        Ingredient.create(paprika);

        Ingredient kartoffel = new Ingredient();
        kartoffel.name = "Kartoffel";
        kartoffel.description = "Wohlschmeckendes Nachtschattengewächs";
        Ingredient.create(kartoffel);

        Ingredient nudeln = new Ingredient();
        nudeln.name = "Nudeln";
        nudeln.description = "Schön al dente";
        Ingredient.create(nudeln);

        ...
    }
}
```



Rezept (Recipe)

- Das Model für Recipe anlegen
- Das Model für Ingredient erweitern
- Den Controller für Recipe anlegen
- Die Views für Recipe anlegen
- Die Routen anpassen

Rezept (Recipe) – Model

Recipe.java

```
package models;

import java.util.List;
import javax.persistence.*;
import play.data.validation.Constraints.Required;
import com.avaje.ebean.Model;

@Entity
public class Recipe extends Model {
    @Id
    public Long rid;
    @Required
    public String name;
    public String description;

    @ManyToMany(cascade = CascadeType.REMOVE)
    public List<Ingredient> ingredients;

    public static Finder<Long, Recipe> find = new Finder<Long, Recipe>(Recipe.class);

    public static void create(Recipe ingredient) {
        ingredient.save();
    }

    public static List<Recipe> read() {
        return find.all();
    }

    public static void update(Recipe updatedIngredient) {
        updatedIngredient.update();
    }

    public static void delete(Long rid) {
        find.ref(rid).delete();
    }
}
```



Zutaten (Ingredient) – Model erweitert

Ingredient.java

```

package models;
import java.util.HashMap; java.util.List; java.util.Map; javax.persistence.Entity; javax.persistence.Id;
javax.persistence.ManyToMany; play.data.validation.Constraints.Required; com.avaje.ebean.Model;
@Entity
public class Ingredient extends Model {
    @Id
    public Long iid;
    @Required
    public String name;
    public String description;
    @ManyToMany(mappedBy = "ingredients")
    public List<Recipe> recipes;
    public static Finder<Long, Ingredient> find = new Finder<Long, Ingredient>(Ingredient.class);
    public static void create(Ingredient ingredient) {
        ingredient.save();
    }
    public static List<Ingredient> read() {
        return find.all();
    }
    public static void update(Ingredient updatedIngredient) {
        updatedIngredient.update();
    }
    public static void delete(Long iid) {
        find.ref(iid).delete();
    }
    public static Map<Long, String> getAllAsMap() {
        HashMap<Long, String> ingredientsMap = new HashMap<Long, String>();
        List<Ingredient> ingredients = Ingredient.read();
        for (Ingredient ingredient : ingredients) {
            ingredientsMap.put(ingredient.iid, ingredient.name);
        }
        return ingredientsMap;
    }
}

```

Rezept (Recipe) – View I

recipes.scala.html

```
@(recipes: List[Recipe])

@import helper._

@main("Recipes") {
  <h1>List of Recipes</h1>
  <a href="@routes.RecipesCtrl.createRecipe()">Create Recipe</a>
  <table>
    @for(recipe <- recipes) {
      <tr>
        <td>@recipe.name</td>
        <td>
          <ul>
            @for(ingredient <- recipe.ingredients) {
              <li>@ingredient.name</li>
            }
          </ul>
        </td>
        <td>@recipe.description</td>
        <td>
          @form(routes.RecipesCtrl.updateRecipe(recipe.rid)) {
            <input type="submit" value="Update">
          }
        </td>
        <td>
          @form(routes.RecipesCtrl.deleteRecipe(recipe.rid)) {
            <input type="submit" value="Delete">
          }
        </td>
      </tr>
    }
  </table>
}
```

Rezept (Recipe) – View II

recipesForm.scala.html

```

@ (formType: String, recipeForm: Form[Recipe], ingredients: List[models.Ingredient])

@import helper._

@main(formType + " Recipe") {
  <h1>@formType Recipe</h1>
  @form(routes.RecipesCtrl.storeRecipe()) {
    <input type="hidden" id="@recipeForm("rid").id"
      name="@recipeForm("rid").name"
      value='@recipeForm("rid").value' />
    @inputText(recipeForm("name"))
    <div>
      <div>
        <strong>Ingredients</strong>
      </div>
      <div>
        @for(ingredient <- ingredients) {
          <label class="checkbox">
            <input type="checkbox" name="ingredients.iid[]" value="@ingredient.iid"
              @if(recipeForm.value.isDefined &&
                recipeForm.value.get.ingredients.contains(ingredient)) {
                checked="checked"
              }
            />
            @ingredient.name
          </label>
        }
      </div>
    </div>
    @textarea(field = recipeForm("description"), args = 'rows -> 3, 'cols -> 50)
    <input type="submit" value="Save">
  }
}

```

Rezept (Recipe) – Controller I

RecipesCtrl.java

```
package controllers;
import java.util.*;
import models.Ingredient;
import models.Recipe;
import play.data.Form;
import play.mvc.*;

public class RecipesCtrl extends Controller {
    @Inject
    private FormFactory formFactory;

    public Result createRecipe() {
        return ok(views.html.recipesForm.render("Create", formFactory.form(Recipe.class),
            Ingredient.read()));
    }

    public Result readRecipes() {
        return ok(views.html.recipes.render(Recipe.read()));
    }

    public Result updateRecipe(Long iid) {
        Recipe recipe = Recipe.find.byId(iid);
        Form<Recipe> filledForm = formFactory.form(Recipe.class).fill(recipe);
        return ok(views.html.recipesForm.render("Update", filledForm, Ingredient.read()));
    }

    public Result deleteRecipe(Long rid) {
        Recipe.delete(rid);
        return redirect(routes.RecipesCtrl.readRecipes());
    }
    ...
}
```

Rezept (Recipe) – Controller II

RecipesCtrl.java

```
...
public Result storeRecipe() {
    Form<Recipe> recipeForm = formFactory.form(Recipe.class);
    Form<Recipe> filledForm = recipeForm.bindFromRequest();
    if (filledForm.hasErrors()) {
        return ok(views.html.recipesForm.render("Correct", filledForm,
            Ingredient.read()));
    } else {
        Recipe recipe = filledForm.get();
        List<Long> selectedIngredients = RecipesCtrl.getMultiSelectIDs(
            filledForm.data(), "ingredients.iid");
        for (Long ingredientID : selectedIngredients) {
            Ingredient tmpIngredient = Ingredient.find.byId(ingredientID);
            recipe.ingredients.add(tmpIngredient);
        }
        if (recipe.rid == null) {
            Recipe.create(recipe);
        } else {
            Recipe.update(recipe);
        }
        return ok(views.html.recipes.render(Recipe.read()));
    }
}

public List<Long> getMultiSelectIDs(Map<String, String> formMap, String multiName) {
    ArrayList<Long> selectedIDs = new ArrayList<Long>();
    Set<String> fieldNames = formMap.keySet();
    for (String fieldName : fieldNames) {
        if (fieldName.toLowerCase().contains(multiName.toLowerCase())) {
            Long tmpID = Long.parseLong(formMap.get(fieldName));
            selectedIDs.add(tmpID);
        }
    }
    return selectedIDs;
}
```

Rezept (Recipe) – Routen

```
# Recipes
```

```
GET    /recipes          controllers.RecipesCtrl.readRecipes()  
POST   /recipes/store     controllers.RecipesCtrl.storeRecipe()  
GET    /recipes/create    controllers.RecipesCtrl.createRecipe()  
POST   /recipes/update/:rid controllers.RecipesCtrl.updateRecipe(rid: Long)  
POST   /recipes/delete/:rid controllers.RecipesCtrl.deleteRecipe(rid: Long)
```

routes