10.09.2019
Zuchering / JUG Ingolstadt

Vorschau auf Webanwendungen mit MVC 1.0 und Eclipse Krazo

INNOQ

Tobias Erdle

Consultant
bei INNOQ Deutschland GmbH

Tobias arbeitet als Consultant bei INNOQ und entwickelt seit mehreren Jahren JVM-basierte Desktop- und Webanwendungen. Dabei liegt sein Schwerpunkt im Backend, ab und an wagt er sich aber auch ins Frontend. Wenn er sich nicht gerade um die Probleme von Maschinen kümmert, beschäftigt er sich mit Alternativen zu aktuellen agilen Vorgehensmodellen.

Frontends von JEE / JakartaEE (bisher)

INNOQ
www.innoq.com

Java Server Pages (JSP)
- Seit 1999 Teil von JEE
- Kompilieren direkt in Java Code (Servlets)
- Template-Engine nicht wählbar
- MVC Pattern nur mit Frameworks (Struts) umsetzbar

Java Server Faces (JSF)
- Seit 2003 Teil von JEE
- Basiert auf zentralem `FacesServlet`
- Template-Engine nicht wählbar
- Komponentenbasiertes MVC
- Komplexer Request-Cycle

Irgendetwas fehlt...

... und zwar ein actionbasiertes Frontend!

Frontends von JEE / JakartaEE (bald)

www.innoq.com

Java Server Pages (JSP)

Java Server Faces (JSF)

MVC API

Ein wenig Geschichte...

- **ab 2014:** Entwicklung der MVC API 1.0 (JSR 371) für Java EE durch Oracle
- **Januar 2017:** Oracle gibt Specification Lead an Community ab
- **September 2018:** Eintritt in die „Final Release“ Phase des Java Community Process
- **Anfang 2019:** Referenzimplementierung Eclipse Krazo erfüllt TCK zu 100%

Eigenschaften der API

- Möglichkeit actionbasierte, serverseitig gerenderte UIs zu entwickeln
- Bietet SPI zur Anbindung anderer View-Engines
  - JSP und Facelets müssen grundsätzlich unterstützt werden
  - diverse Krazo-Extensions existieren schon (eingeschränkte Features)
- Basiert auf JAX-RS 2.1
- CDI und Bean Validation nutzbar
- Nutzung der HTTP-Semantik möglich (Verben, etc.)

Und wie funktioniert das nun?
Probieren wir es aus!

Benötigte Abhängigkeiten

```
plugins {
    id 'java'
    id 'war'
}

repositories {
    jcenter()
    mavenCentral()
    maven {
        url "https://oss.sonatype.org/content/repositories/snapshots"
    }
}

dependencies {
    //…

    implementation 'javax.mvc:javax.mvc-api:1.0-pfd‘

    implementation 'org.eclipse.krazo:krazo-resteasy:1.0.0-SNAPSHOT‘

    // implementation 'org.eclipse.krazo.ext:krazo-thymeleaf:1.0.0-SNAPSHOT'


    providedCompile 'javax:javaee-api:8.0.1‘

    //…
}
```

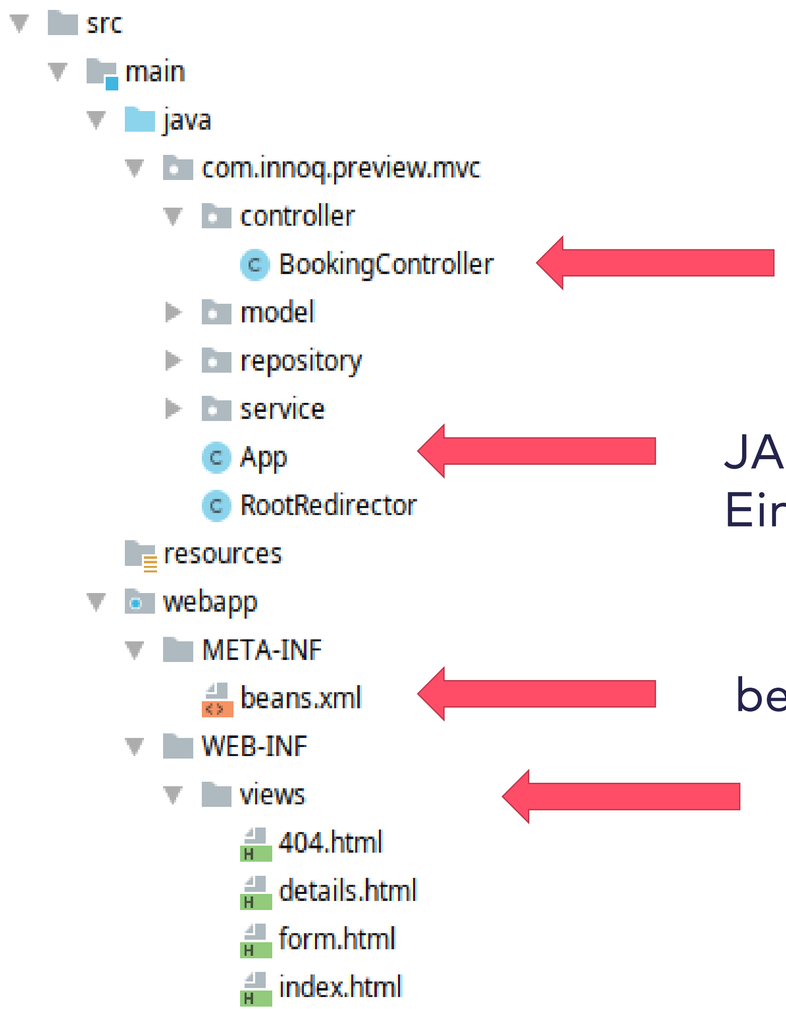
Snapshot-Repo für MVC und Eclipse Krazo Dependencies

MVC API

RESTEasy Support von Eclipse Krazo (für Wildfly)

Thymeleaf-Extension als optionale View-Engine

Projektstruktur



Controller mit View-Logik

JAX-RS Applikation als Einstiegspunkt

beans.xml für CDI-Konfiguration

View-Templates im Standardverzeichnis

JAX-RS Applikation

```java
import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@ApplicationPath("/mvc")
public class App extends Application {

}
```

Context Root sollte nicht verwendet werden.

"The reason for this is that MVC implementations often forward requests to the Servlet container, and the use of the aforementioned values may result in the unwanted processing of the forwarded request by the JAX-RS servlet once again"

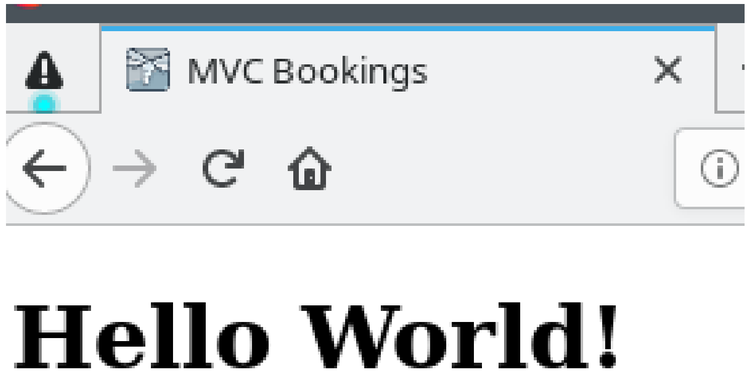
- MVC Specification

Hello World!

GET http://<host>/mvc/hello

```java
import javax.ws.rs.GET;
import javax.ws.rs.Path;

@Path("hello")

public class
HelloWorldController {

    @GET
    public String sayHello() {

    }

}
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello World</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```



Die Booking-Applikation

Buchungsübersicht

GET http://<host>/mvc/bookings

Buchungsübersicht

GET http://<host>/mvc/bookings

```
@Path("/bookings")
@Controller
public class BookingController {

    // ...

    @Inject
    private Models models;

    @GET
    public String index() {
        final var bookings = bookingService.getAllBookings();

        models.put("bookings", bookings);

        return "index.jsp";
    }

    // ...
```

Model für View

„bookings“ für View bereitstellen

Verwendung des Models

GET http://<host>/mvc/bookings

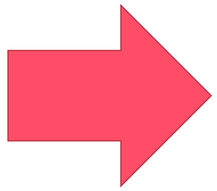
```java
@Path("/bookings")
@Controller
public class BookingController {

    // ...

    @Inject
    private Models models;

    @GET
    public String showIndex() {
        // ...
        models.put("bookings", bookings);

        return "index.jsp";
    }

    // ...
}
```

```jsp
<c:forEach var="booking" items="${bookings}">
    <tr>
        <td>${booking.bookingNumber}</td>
        <td>${booking.name}</td>
    </tr>
</c:forEach>
```

Buchungsübersicht

GET http://<host>/mvc/bookings

```
@Path("/bookings")
@Controller
public class BookingController {

    // ...

    @Inject
    private Models models;

    @GET
    public String showIndex() {
        final var bookings =
          bookingService.getAllBookings();
        models.put("bookings", bookings);

        return "index.jsp";
    }

    // ...
}
```



Booking number	Name	Status
116f1dad-577f-4ca0-abe0-432bd0730717	Augsburg -> München	Active
4b0e1c6a-b7b6-45ee-b5b0-f5b7423086a3	Berlin -> München	Cancelled
cae9e31f-d4d0-42ca-ad48-cc36a53e9611	München -> Augsburg	Active
0f4faa58-f5e1-4593-8014-53e5115ce12d	München -> Berlin	Cancelled

Nun kennen wir alle MVC Bestandteile!

Model

```
@Inject
private Models models;
```

View

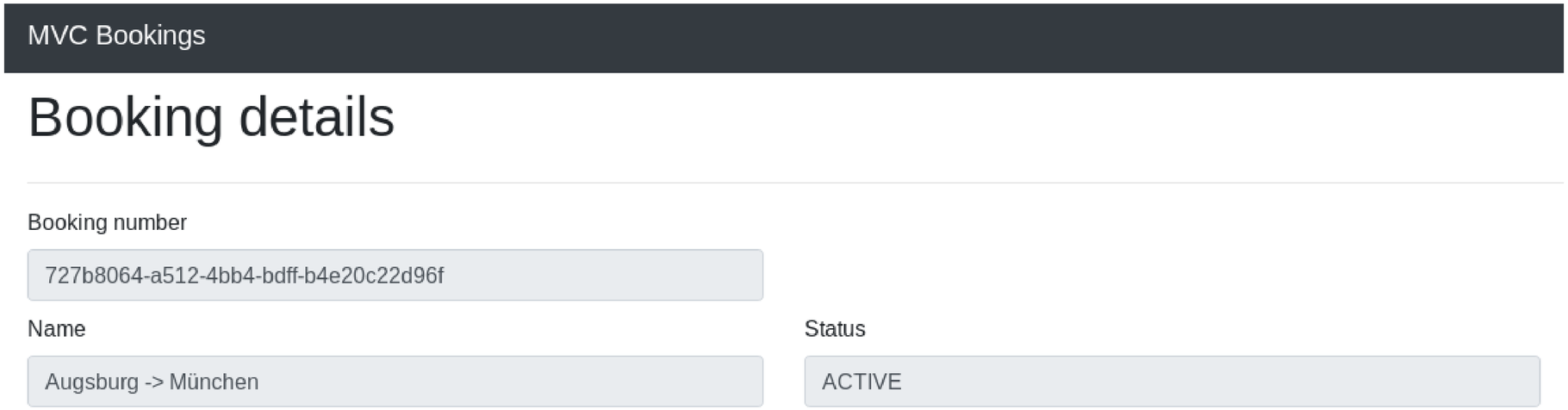
```
<c:forEach var="booking" items="${bookings}">
    <tr>
        <td>${booking.bookingNumber}</td>
        <td>${booking.name}</td>
          …
</c:forEach>
```

Controller

```
@Path("/bookings")
@Controller
public class BookingController {

    // ...

    @Inject
    private Models models;

    @GET
    public String showIndex() {
        // ...
        return "index.jsp";
    }

    // ...
}
```

Buchungsdetails

GET http://<host>/mvc/bookings/{bookingNumber}



Buchungsdetails

GET http://<host>/mvc/bookings/{bookingNumber}

```
@Path("/bookings")
@Controller
public class BookingController {

    // ...

    @GET
    @Path("/{bookingNumber}")
    public Response showDetailsOfBooking(@PathParam("bookingNumber") final UUID bookingNumber) {
        final var booking = bookingService.findBookingForBookingNumber(bookingNumber);

        if (booking.isPresent()) {
            models.put("booking", booking.get());
            return Response.ok("details.jsp").build();
        } else {
            models.put("bookingNumber", bookingNumber);
            return Response.status(Response.Status.NOT_FOUND).entity("404.jsp").build();
        }
    }

    // ...
```

Required URL-Parameter mit Booking Number

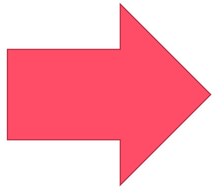
HTTP Status 200 & Booking werden zurückgegeben

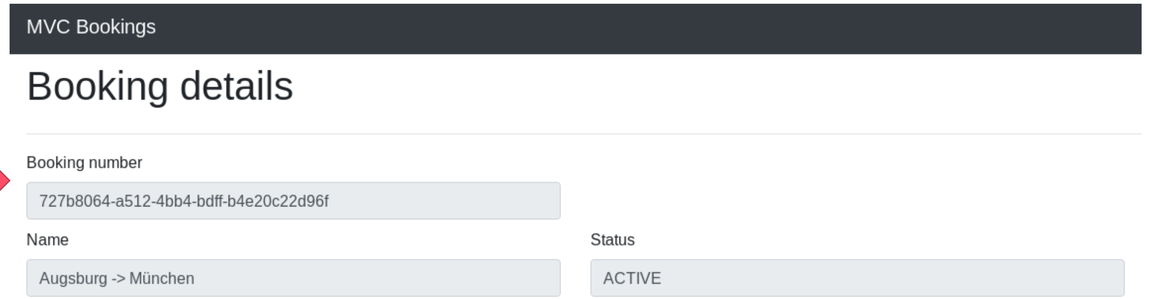
HTTP Status 404 & ungültige Booking Number

Buchungsdetails– Buchung gefunden

GET http://<host>/mvc/bookings/{bookingNumber}

```
@GET
@Path("/{bookingNumber}")
public Response showDetailsOfBooking(
@PathParam("bookingNumber") final UUID bookingNumber) {
    // ...
    if (booking.isPresent()) {
        models.put("booking", booking.get());
        return Response.ok("details.jsp").build();
    } else {
        // ...
    }
}
```

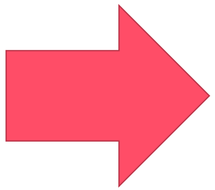


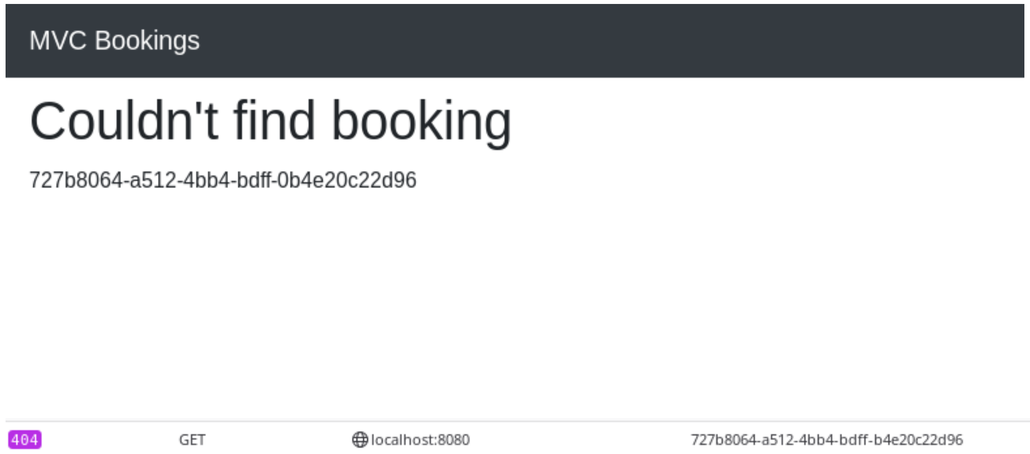


Buchungsdetails– Buchung nicht gefunden

GET http://<host>/mvc/bookings/{bookingNumber}

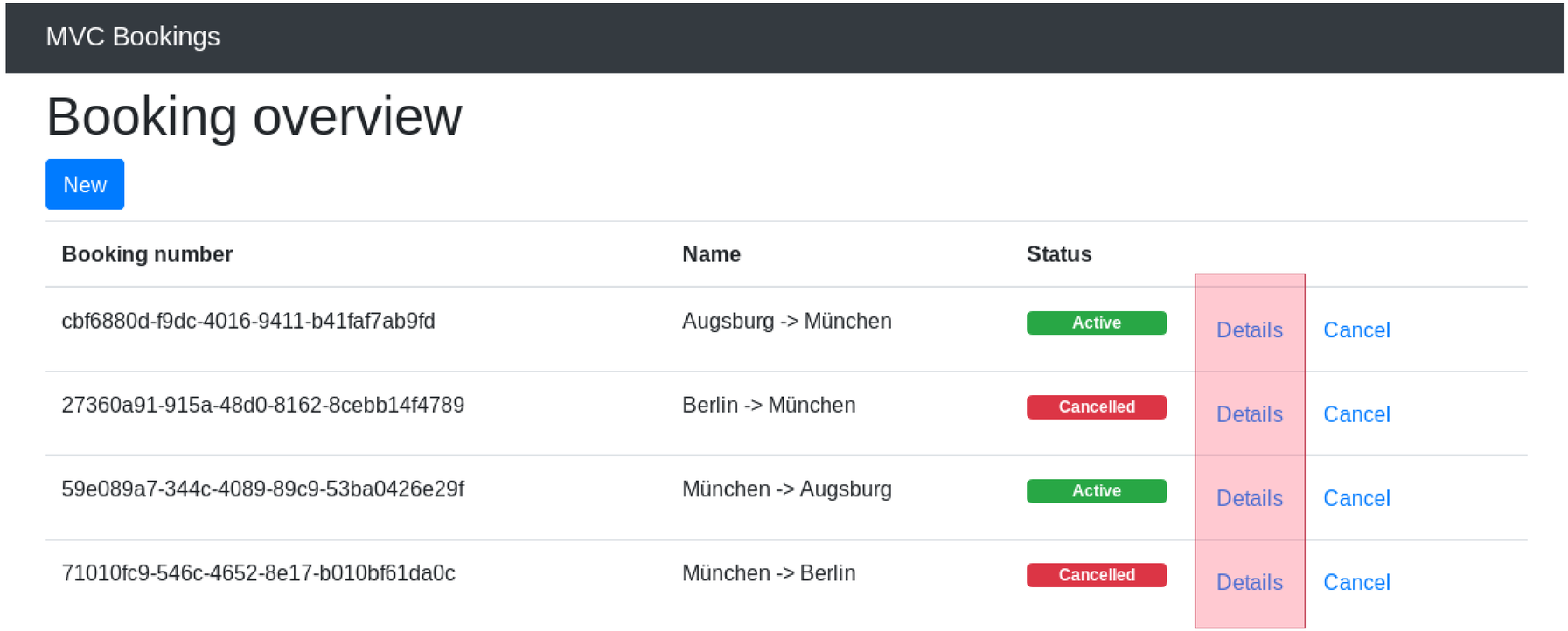
```
@GET
@Path("/{bookingNumber}")
public Response showDetailsOfBooking(
@PathParam("bookingNumber") final UUID bookingNumber) {
    // ...
    if (booking.isPresent()) {
        // ...
    } else {
        models.put("bookingNumber", bookingNumber);
        return Response
            .status(Response.Status.NOT_FOUND)
            .entity("404.jsp")
            .build();
    }
}
```





Links zu Buchungsdetails

GET http://<host>/mvc/bookings



Links zu Buchungsdetails

GET http://<host>/mvc/bookings

```
<a class="btn btn-link"
href="/bookings/${booking.bookingNumber}">
    Details
</a>
```

Hartcodierte, konkatenierte URL

Links zu Buchungsdetails

GET http://<host>/mvc/bookings

```
<a clas          k"
href="    ings/      ng.bookin      er}">
    De
</a>
```

Hartcodierte, konkatenierte URL

Links zu Buchungsdetails

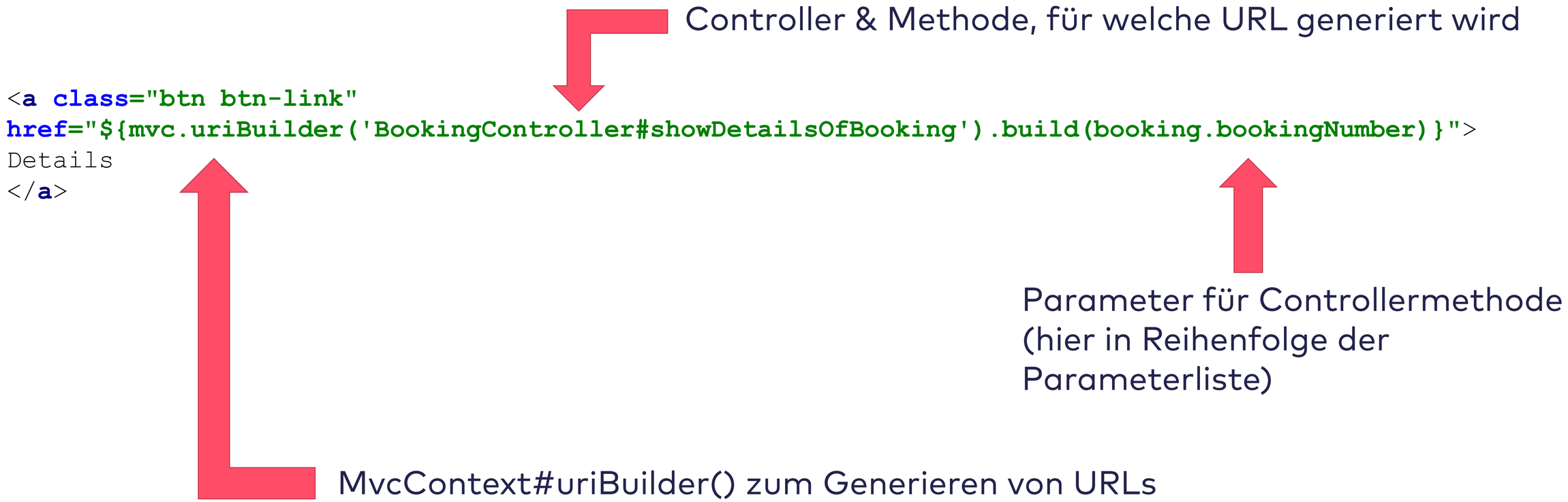
GET http://<host>/mvc/bookings

```
<a class="btn btn-link"
href="/bookings/${booking.bookingNumber}">
    Details
</a>
```

- Schwieriges Refactoring
- Ressourcen können nicht ohne weiteres umbenannt werden
- Fehler werden erst spät erkannt und führen zu unklaren Stacktraces

Links zu Buchungsdetails– Besser!

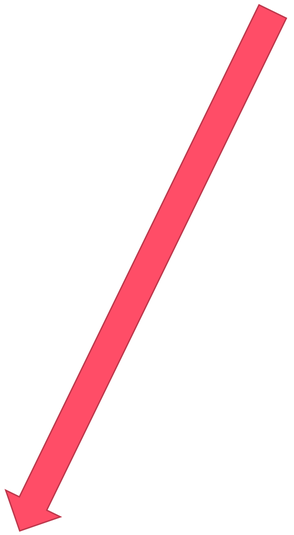
GET http://<host>/mvc/bookings



Links zu Buchungsdetails – Besser!

GET http://<host>/mvc/bookings

```
<a class="btn btn-link"
href="${mvc.uriBuilder('BookingController#showDetailsOfBooking').build(booking.bookingNumber)}">
Details
</a>
```



```
@Path("/bookings")
@Controller
public class BookingController {

    // ...

    @GET
    @Path("/{bookingNumber}")
    public String showDetailsOfBooking(@PathParam("bookingNumber") final UUID bookingNumber) {
        // ...
    }

    // ...
```

Links zu Buchungsdetails – Besser!

GET http://<host>/mvc/bookings

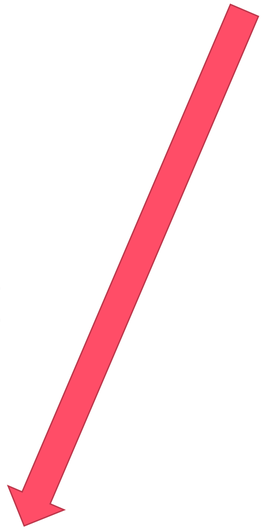
```
<a class="btn btn-link" th:href="${mvc
.uriBuilder('BookingController#showDetailsOfBooking')
.build(booking.bookingNumber)}">
Details
</a>
```

- Refactoring einfach
- Ressourcen können einfach umbenannt werden
- Nicht existierende Controller oder Methoden werden im Stacktrace genannt → einfaches Debugging

Links zu Buchungsdetails – Noch besser!

GET http://<host>/mvc/bookings

```
<a class="btn btn-link"
href="${mvc.uriBuilder('showBookingDetails').build(booking.bookingNumber)}">Details
</a>
```



```
@Path("/bookings")
@Controller
public class BookingController {

    // ...

    @GET
    @Path("/{bookingNumber}")
    @UriRef("showBookingDetails")
    public String showDetailsOfBooking(@PathParam("bookingNumber") final UUID bookingNumber) {
        // ...
    }

    // ...
```

Links zu Buchungsdetails – Noch besser!

GET http://<host>/mvc/bookings

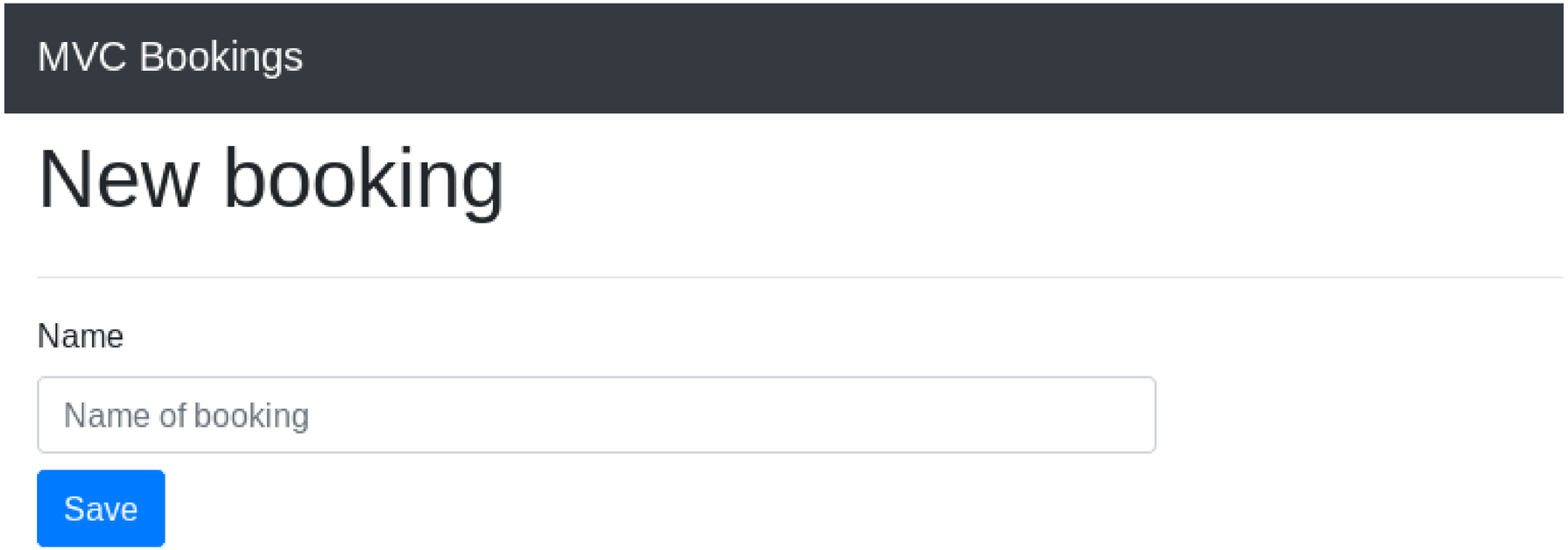
```
<a class="btn btn-link" href="${mvc
.uriBuilder('showBookingDetails')
.build(booking.bookingNumber)}">
Details
</a>
```

- Refactoring einfach
- Ressourcen können einfach umbenannt werden
- Nicht existierende Controller oder Methoden werden im Stacktrace genannt → einfaches Debugging

- **Keine Abhängigkeit zu konkreter Implementierung**

Buchung anlegen

GET http://<host>/mvc/bookings/new



Buchung anlegen

GET http://<host>/mvc/bookings/new

```
@Path("/bookings")
@Controller
public class BookingController {

    // ...

    @GET
    @Path("/new")
    public String showNewBookingForm() {
        return "form.jsp";
    }

    // ...
}
```

Buchung anlegen

GET http://<host>/mvc/bookings/new

```
<form method="post" action="${mvc.uri('BookingController#createNewBooking')}">
    <div class="form-row">
        <div class="col-md-6">
            <label for="name">Name</label>
            <input id="name" name="name" class="form-control" placeholder="Name of booking" required>
        </div>
    </div>
    <div class="form-row mt-2">
        <div class="col-md-3">
            <button type="submit" class="btn btn-primary">Save</button>
        </div>
    </div>
</form>
```

Booking anlegen mit POST

POST http://<host>/mvc/bookings

```
// ...

@Inject private BindingResult bindingResult;

@POST
public String createNewBooking(@MvcBinding @FormParam("name")
          @NotBlank @Size(min = 5, max = 255) final String name) {

   if (bindingResult.isFailed()) {
        models.put("errors", new ArrayList<>(bindingResult.getAllErrors()));
        return "form.jsp";
    }

    final var booking = bookingService.createNewBooking(name);

    return "redirect:/bookings/" + booking.getBookingNumber();
}

// ...
```

```
<input id="name" name="name" class="form-control"
placeholder="Name of booking" required>
```

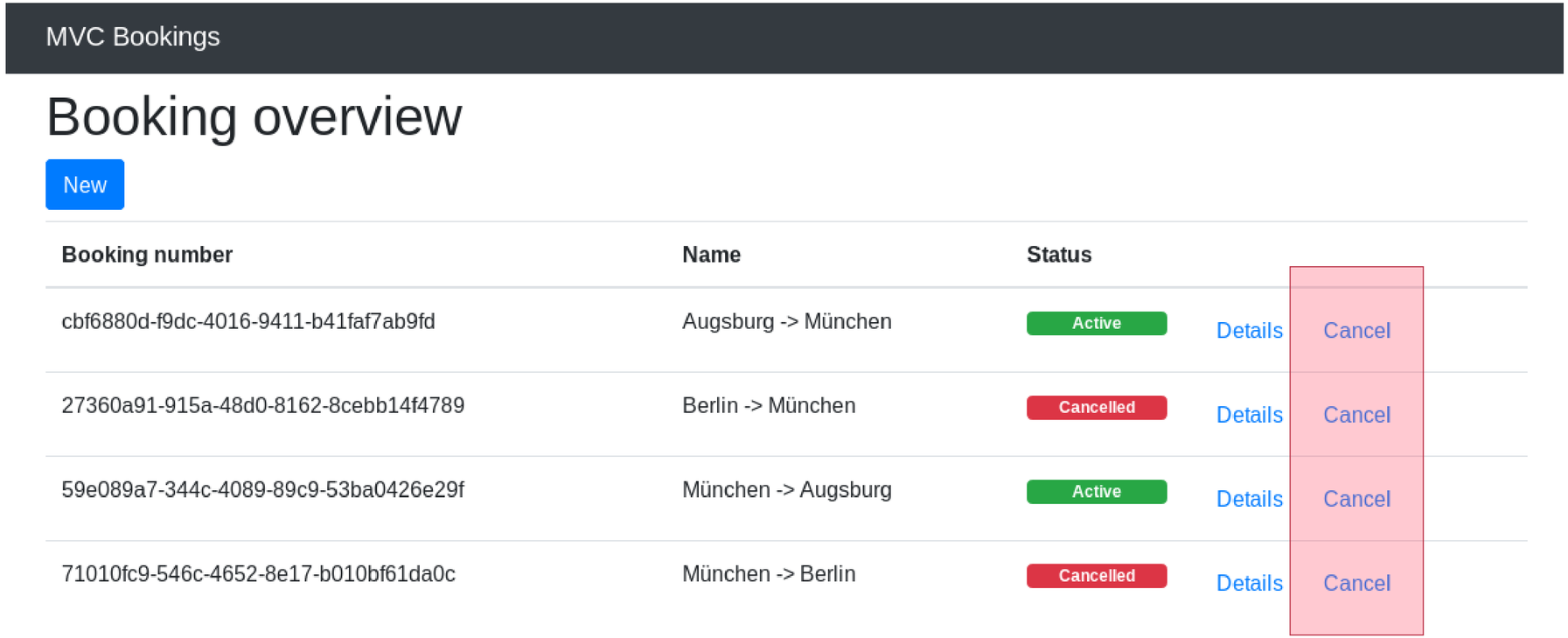
Fehlerbehandlung bei POST

POST http://<host>/mvc/bookings

```
<form method="post" action="${mvc.uri('BookingController#createNewBooking')}" accept-charset="UTF-8">
    <div class="form-row">
        <div class="col-md-6">
            <label for="name">${msg.get("form.label.name")}</label>
            <input id="name" name="name" class="form-control" placeholder="${msg.get("form.label.name.placeholder")}" required>
        </div>
    </div>
    <c:choose>
        <c:when test="${not errors.isEmpty()}">
            <div class="row mt-2">
                <c:forEach var="error" items="${errors}">
                    <div class="col-md-12">
                        <div class="alert alert-danger">${error.getParamName()}: ${error.getMessage()}</div>
                    </div>
                </c:forEach>
            </div>
        </c:when>
    </c:choose>
    <div class="form-row mt-2">
        <div class="col-md-3">
            <button type="submit" class="btn btn-primary">${msg.get("form.btn.save")}</button>
        </div>
    </div>
</form>
```

Buchung stornieren

GET http://<host>/mvc/bookings



Buchung stornieren

DELETE http://<host>/mvc/bookings/{bookingNumber}

```
// ...

@DELETE
@Path("/{bookingNumber}")
public String cancelBooking(@PathParam("bookingNumber") final UUID bookingNumber) {
    final var isCancelled = bookingService.cancelBooking(bookingNumber);

    return isCancelled ? "redirect:/bookings" : "404.jsp";
}

// ...
```

Andere HTTP Methoden in Forms (Krazo only)

Buchung stornieren

GET http://<host>/mvc/bookings

```
<form class="d-inline" method="post"
      action="${mvc.uriBuilder('BookingController#cancelBooking').build(booking.bookingNumber)}"
      onsubmit="return confirm('Do you really want to cancel the booking?');">

    <input type="hidden" name="_method" value="DELETE">

    <button type="submit" class="btn btn-link">Cancel</button>
</form>
```

Achtung:
PATCH Methode wird erst ab JAX-RS 2.1 unterstützt

Internationalisierung - LocaleResolver

```java
/**
 * <p>Locale resolvers are used to determine the locale of the current request and are
 * discovered using CDI.
 *</p>
 *
 * …
 */
public interface LocaleResolver {

    /**
     * <p>Resolve the locale of the current request given a {@link LocaleResolverContext}.</p>
     *
     * …
     *
     * @param context the context needed for processing.
     * @return The resolved locale or <code>null</code>.
     */
    Locale resolveLocale(LocaleResolverContext context);
}
```

Internationalisierung - LocaleResolver

```java
/**
 * <p>Locale resolvers are used to determine the locale of the current request and are
 * discovered using CDI.
 *</p>
 *
 * …
 */
public interface LocaleResolver {

    /**
     * <p>Resolve the locale of the current request given a {@link LocaleResolverContext}.</p>
     *
     * …
     *
     * @param context the context needed for processing.
     * @return The resolved locale or <code>null</code>.
     */
    Locale resolveLocale(LocaleResolverContext context);
}
```

- Default LocaleResolver muss vorhanden sein
- Auslesen des Accept-Language Headers als Quelle der Locale
- Ergebnis wird z.B. von `org.eclipse.krazo.locale.LocaleResolverChain` in `MvcContext` gesetzt

Internationalisierung - Übersetzungen

Beispiel: ResourceBundle `messages` für eine Anwendung

```
//messages.properties
…

#Index Page
page.title=My awesome application

#Hello World
hello.world=Hello World!
```

```
//messages_fr.properties

#Hello World
hello.world=Bonjour le monde!
```

```
//messages_de.properties

#Hello World
hello.world=Hallo Welt!
```

Internationalisierung – Übersetzung finden

Beispielhafte Implementierung

```
@RequestScoped
@Named("msg")
public class Messages {

  @Inject
  private MvcContext mvcContext;

  public final String get(final String key) {
    LOGGER.info(String.format("Getting message '%s' for locale '%s'", key,
                                                        mvcContext.getLocale()));

    final var bundle = ResourceBundle.getBundle("messages", mvcContext.getLocale());

    return bundle.containsKey(key) ? bundle.getString(key) : formatUnknownKey(key);
  }

  private static String formatUnknownKey(final String key) {
    return String.format("???%s???", key);
  }
}
```

Internationalisierung – Übersetzung anwenden

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>${msg.get("page.title")}</title>
</head>
<body>
    <h1>${msg.get("hello.world")}</h1>
</body>
</html>
```

Internationalisierung - Übersetzungen

Beispiel: ResourceBundle *messages* für Booking Anwendung

```
//messages.properties

<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
      My awesome application
    </title>
</head>
<body>
    <h1>
      Hello World!
    </h1>
</body>
</html>
```

```
//messages_fr.properties

<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
       My awesome application
    </title>
</head>
<body>
    <h1>
      Bonjour le monde!
    </h1>
</body>
</html>
```

```
//messages_de.properties

<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
       My awesome application
</title>
</head>
<body>
    <h1>
      Hallo Welt!
    </h1>
</body>
</html>
```

Mehr Details: https://www.mvc-spec.org/learn/cookbook/multilang_de.html

Und wann geht die API live?

- Hoffentlich bald ☺
- Aber es gibt aktuell noch zwei Blocker:
    - Dokumentation für TCK erstellen
    - Lizenzrechtliche Prüfungen durch Oracle bei Krazo

Danke! Fragen?

Tobias Erdle
tobias.erdle@innoq.com
@erdlet

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
+49 2173 3366-0

Ohlauer Str. 43
10999 Berlin
Germany
+49 2173 3366-0

Ludwigstr. 180E
63067 Offenbach
Germany
+49 2173 3366-0

Kreuzstr. 16
80331 München
Germany
+49 2173 3366-0

Hermannstrasse 13
20095 Hamburg
Germany
+49 2173 3366-0

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
+41 41 743 0116

Quellen & weiterführende Infos

- MVC Specification Homepage (https://www.mvc-spec.org/)
- Eclipse Krazo Github Projekt (https://github.com/eclipse-ee4j/krazo)
- MVC Users Mailing List (https://groups.google.com/forum/#!forum/jsr371-users)
- JSR 371 (https://jcp.org/en/jsr/detail?id=371)
- Booking-Applikation (https://github.com/erdlet/mvc-krazo-talk-example)
- MVC Multilanguage Examples (https://github.com/erdlet/mvc-international-examples)
    - Cookbook: https://www.mvc-spec.org/learn/cookbook/
- MVC Toolbox (https://github.com/chkal/mvc-toolbox)