SPRING BOOT IN DER PRAXIS

Praxisbeispiele aus einer neuen monolithischen Spring Boot Smart Farming Anwendung

Bern | jug.ch | 22.03.2022

1. Kurzportrait
2. DRY mit Vererbung in Controllern & Tests im Java-Code
3. DRY mit eigener Thymeleaf Erweiterung
4. Reaktionsschnelles Frontend mit serverseitigem Rendering, ohne JS-Framework
5. Multi-Tenancy auf Basis von Row-Permissions in der PostgreSQL-Datenbank
6. Auto Setup/Teardown der Integrationsumgebung mit GitLab-CI und ansible
7. Fragen

Agenda

KURZPORTRAIT

Referent

Kurzportrait

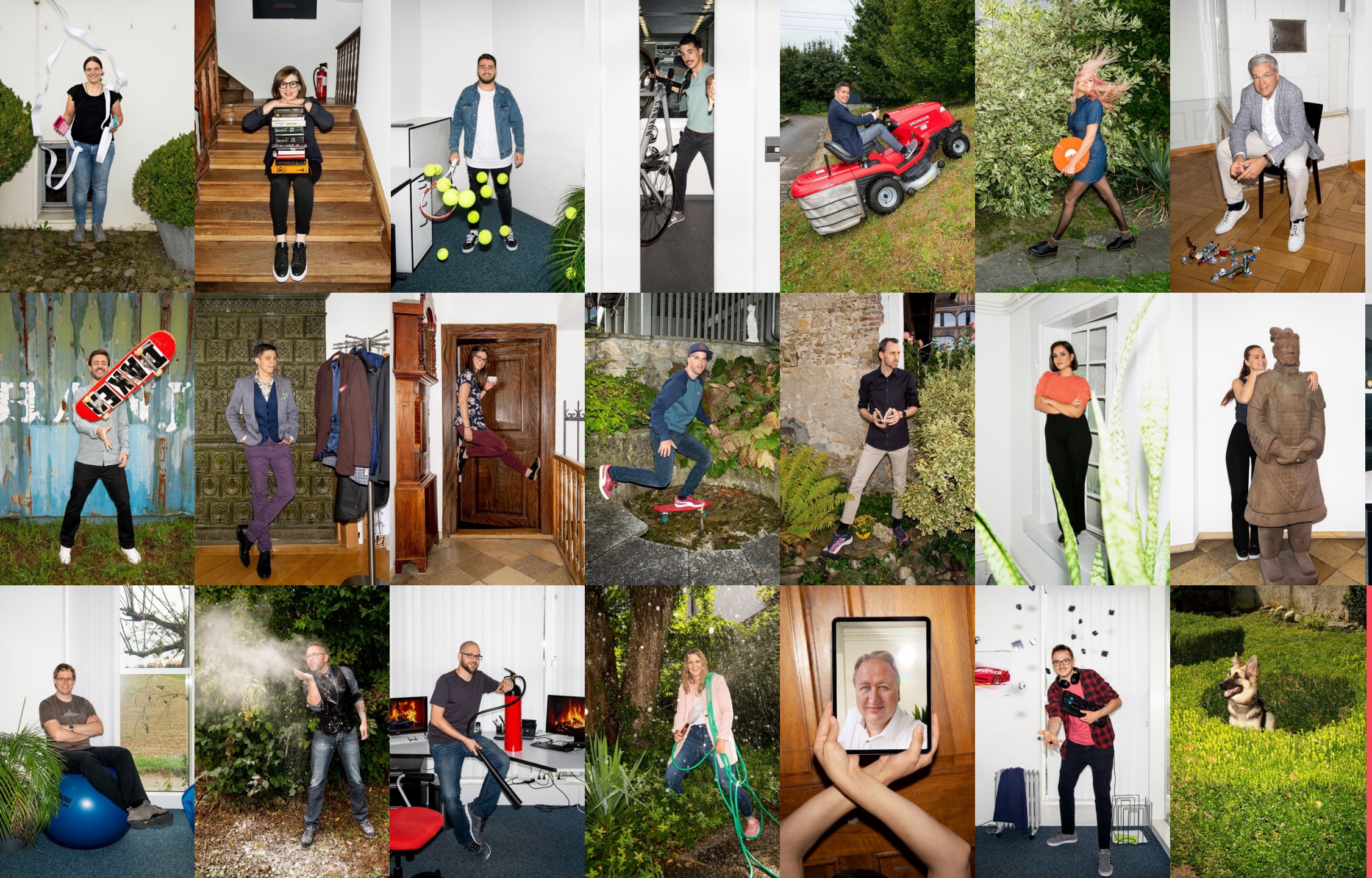
MORITZ KOBEL

CTO

@tandemblog

Qube

creatives




ABWASSERVERBAND REGION LENZBURG
AESCHBACH QUARTIER AARAU
AK MOBIL CC
Aargauische Kantonalbank
amag
ANA Altersitz Neuhaus Aaretal
anox
aph HASLE-RÜEGSAU
auris relocation switzerland
BACO
bankLeerau
BARMELWEID
Bracher & Partner
BBA Bus Aarau
coop rechtsschutz einfach anders.
EDILTECNICA
entero
EPIC SUISSE
EVU Partners
FC Aarau
forumvera
G+P INGENIEURE
BUCHS
MÖRIKEN WILDEGG
GEMEINDE RIGGISBERG
Grundmann Die Baumeister
huchuli KIES · BETON · ELEMENTE
city märt Das Einkaufszentrum in Aarau.
IM LENZ URBAN LEBEN. ZENTRAL ARBEITEN.
imkp
IMS AG
[IMS] Treuhand Revision
INTER CHEESE AG DIE KÄSEKOMPETENZ
KANTON AARGAU
KANTONS POLIZEI AARGAU
Kita läpurzel
SEIT 1908 KOBEL'S HOF
Kooperation Arbeitsmarkt
Läderach chocolatier suisse
LÖSINGER MARAZZI
MAPEI
MOBIMO Leidenschaft für Immobilien
novoglas
OOT OPEN OPERATING THEATRE
PFADI kanton bern
PRESYN Baustoffe mit Mehrwert.
pro pede Zentrum für Orthopädie-Schuhtechnik und Podologie
realit
SAMSUNG
Schöftland
Scholl
schwarz
Schweizerische Eidgenossenschaft Confédération suisse Confederazione Svizzera Confederaziun svizra
spineclinic Wirbelsäulenmedizin
spineinstitute Förderung der Wirbelsäulenmedizin
STADT AARAU
sva AARGAU Sozialversicherung
viacar
WEHRLIS
WOLFER
XO FRESH INFUSED WATER
YLEX RECHT EINFACH.
zamba

Kunden

- **Makoni ist eine Smart Farming Software für die Direktvermarktung.**
- **Makoni ist selbstfinanziert, unabhängig und wird in der Schweiz entwickelt und gehostet.**
- **Makoni ist eine gemeinsame Marke der Qube AG und Kobels Hof.**

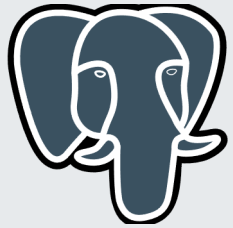
Makoni

Kurzportrait

Architektur

Kurzportrait

Spring Boot 2.X
Applikation



PostgreSQL
als Datenbank

Thymeleaf
als Template Engine

CI/CD
Mit GitLab

DRY MIT VERERBUNG IN CONTROLLERN & TESTS IM JAVA-CODE

Problem

DRY mit Vererbung in Controllern & Tests im Java-Code

- ähnlicher Code für CRUD
- ähnlicher Code für Kommentare
- ähnlicher Code für Tests

Lösungsansatz

DRY mit Vererbung in Controllern & Tests im Java-Code

- Controller-Interfaces bringen mit default-Methoden Funktionalität
- Überschreibbare Methoden für individuelle Funktionalität
- Gleiches Prinzip für Tests
- Knackpunkt: Security-Annotations

```java
@BackendController
@RequestMapping("/pasture")
public class PastureController implements CommentableCrudController<Pasture, PastureComment>,
ServerSideTableHandling<Pasture> {

    @Override
    public Map<String, Object> addCustomShowAttributes(Pasture entity) {
        Map<String, Object> attributes = CommentableCrudController.super
                                .addCustomShowAttributes(entity);
        attributes.put("cattleList", cattleRepository
                                .findAllOnFarmOnPastureOrderByDateOfBirthAscWithCommentsLastCalf(entity));
        return attributes;
    }

    @Override
    public PastureRepository getRepository() {
        return pastureRepository;
    }
```

PastureController

DRY mit Vererbung in Controllern & Tests im Java-Code

```
public interface CommentableCrudController<E extends BaseEntity & Commentable<C> &
DecapitalizedClassNameAware,
        C extends Comment<E>> extends CrudController<E>, CommentableController<E, C> {

    @Override
    default Map<String, Object> addCustomShowAttributes(E entity) {
        Map<String, Object> attributes = new HashMap<>();
        attributes.put("comments", getCommentRepository()
                        .findAllByEntityAndEnabledIsTrueOrderByDateCreatedDesc(entity));
        return attributes;
    }
}
```

CommentableCrudController

DRY mit Vererbung in Controllern & Tests im Java-Code

```
public interface CrudController<E extends BaseEntity> extends EntityRepositoryAware<E>, MessageSourceAware,
LoggerAware, ApplicationEventPublisherAware {

    @RequestMapping("")
    default String index(Model model) {

        List<E> entityList = getRepository().findAll();

        model.addAttribute("entityList", entityList);
        model.addAllAttributes(addCustomListAttributes(entityList));

        return newEntityInstance().getDecapitalizedClassName() + "/index";
    }

    @GetMapping("/create")
    default String create(@ModelAttribute("entity") E entity, Model model) {
        model.addAllAttributes(addCustomEditAttributes(entity));

        return newEntityInstance().getDecapitalizedClassName() + "/edit";
    }
```

CrudController

DRY mit Vererbung in Controllern & Tests im Java-Code

```
@PostMapping("/create")
default String createSave(@Validated @ModelAttribute("entity") E entity, BindingResult bindingResult, Model model, RedirectAttributes
redirectAttributes, Locale locale) {

    if (bindingResult.hasErrors()) {
        model.addAllAttributes(addCustomEditAttributes(entity));
        return newEntityInstance().getDecapitalizedClassName() + "/edit";
    }

    E c = newEntityInstance();
    BeanUtils.copyProperties(entity, c, "id", "version", "dateCreated", "lastUpdated");

    try {
        c = getRepository().save(c);
    } catch (Exception e) {
        model.addAllAttributes(addCustomEditAttributes(entity));
        return newEntityInstance().getDecapitalizedClassName() + "/edit";
    }

    redirectAttributes.addFlashAttribute(Flash.SUCCESS, getMessageSource().getMessage(newEntityInstance().getDecapitalizedClassName() +
".create.success", null, locale));

    return "redirect:/" + newEntityInstance().getDecapitalizedClassName() + "/show/" + c.getId();
```

CrudController

DRY mit Vererbung in Controllern & Tests im Java-Code

```
public interface CommentableController<E extends Commentable<C> & DecapitalizedClassNameAware,
        C extends Comment<E>>
        extends EntityRepositoryAware<E>, CommentRepositoryAware<E, C>, UserRepositoryAware, MessageSourceAware,
LoggerAware {

    @GetMapping("/comment/{entityId}/create")
    default String createComment(@PathVariable long entityId, C comment, Model model) {

        BaseEntityRepository<E> repo = getRepository();
        E entity = repo.findById(entityId).orElseThrow(() -> new ObjectNotFoundException(entityId, "commentable"));
        model.addAttribute("entity", entity);
        model.addAttribute("comment", comment);

        return "comment/edit";
    }

    @PostMapping("/comment/{entityId}/create")
    default String createCommentSave(@PathVariable long entityId, @Validated @ModelAttribute("comment") C comment,
BindingResult bindingResult, Model model, @CurrentUser UserPrincipal user, Locale locale, RedirectAttributes redirectAttributes) {

        BaseEntityRepository<E> repo = getRepository();
        E entity = repo.findById(entityId).orElseThrow(() -> new ObjectNotFoundException(entityId, "commentable"));
```

CommentableController

```java
@WebMvcTest(controllers = PastureController.class)
@WithMockUser(roles = {"CATTLE"})
public class PastureControllerTest extends BaseControllerTest implements CommentableCrudControllerTest {

    @BeforeEach
    public void initMocks() {
        Pasture pasture = new Pasture();
        pasture.setId(1L);
        pasture.setName("test");
        Mockito.when(pastureRepository.findById(any())).thenReturn(Optional.of(pasture));

        PastureComment pastureComment = new PastureComment();
        pastureComment.setId(1L);
        pastureComment.setEntity(pasture);
        pastureComment.setMessage("message");
        Mockito.when(pastureCommentRepository.findById(any())).thenReturn(Optional.of(pastureComment));

        Mockito.when(pastureRepository.findAll(any(Specification.class), any(Pageable.class))).thenAnswer((x)
                -> new PageImpl<>(List.of(pasture), x.getArgument(1), 1));
    }


    @Override
    public String getBaseUrl() {
        return "/pasture";
    }
```

PastureControllerTest

DRY mit Vererbung in Controllern & Tests im Java-Code

```java
public interface CommentableCrudControllerTest extends CrudControllerTest {

    @InheritableTest
    default void createCommentForm() throws Exception {
        long id = 1L;
        getMockMvc().perform(get(getBaseUrl() + "/comment/" + id + "/create"))
                .andExpect(status().isOk());
    }

    @InheritableTest
    @WithAnonymousUser
    default void createCommentFormUnauthenticated() throws Exception {
        long id = 1L;
        getMockMvc().perform(get(getBaseUrl() + "/comment/" + id + "/create"))
                .andExpect(status().isFound());
    }

    @InheritableTest
    default void editCommentForm() throws Exception {
        long id = 1L;
        getMockMvc().perform(get(getBaseUrl() + "/comment/" + id + "/edit/1"))
                .andExpect(status().isOk());
    }
```

CommentableCrudControllerTest

DRY mit Vererbung in Controllern & Tests im Java-Code

DRY MIT EIGENER THYMELEAF ERWEITERUNG

Problem

DRY mit eigener Thymeleaf Erweiterung

- ähnlicher Code in den HTML Templates
- alles generiert ist aufwändig & unflexibel

Lösungsansatz

DRY mit eigener Thymeleaf Erweiterung

- Meshing für Header, Menu, Footer,... mit Thymeleaf Layout Dialect
- Eigene Thymeleaf Tags für generische Inputs & Outputs (analog Grails fields)
- input & innerInput
- Themes & Variants

```
<html xmlns:fields="http://www.itds.ch/thymeleaf/fields">

<form th:object="${customer}">
   <!— no fields:bean -> uses object from th:object —>
   <fields:input fields:property="company"/>
   <fields:input fields:property="lastName" fields:variant="editForm"/>
   <!— with fields:bean -> uses object from fields:bean —>
   <fields:input fields:bean="${customer}" fields:property="firstName"/>
   <fields:input fields:bean="${customer}" fields:property="comment" fields:variant="large"/>
</form>

</html>
```

Thymeleaf Fields Dialect im Detail: Anwendung

DRY mit eigener Thymeleaf Erweiterung

```
<th:block xmlns:th="http://www.thymeleaf.org" xmlns:fields="http://www.itds.ch/thymeleaf/fields">
  <!—/*@thymesVar id="bean" type="java.lang.Object"*/—>
  <!—/*@thymesVar id="beanName" type="java.lang.String"*/—>
  <!—/*@thymesVar id="property" type="java.lang.String"*/—>
  <!—/*@thymesVar id="variant" type="java.lang.String"*/—>
  <!—/*@thymesVar id="value" type="java.lang.Object"*/—>
  <!—/*@thymesVar id="constraints" type="ch.itds.thymeleaf.fields.FieldConstraints"*/—>
  <div class="form-group row">
    <label class="col-sm-2 col-form-label" th:for="${property}"
        th:text="#{__${beanName}__.__${property}__.label}"></label>
    <div class="col-sm-10">
      <fields:inlineInput fields:bean="${bean}" fields:property="${property}"
fields:variant="${variant}"/>
      <div class="invalid-feedback">
        <p th:each="error: ${#fields.errors(property)}" th:text="${error}">Invalid data</p>
      </div>
    </div>
  </div>
</th:block>
```

Thymeleaf Fields Dialect im Detail: input

DRY mit eigener Thymeleaf Erweiterung

```
<th:block xmlns:th="http://www.thymeleaf.org">
  <!—/*@thymesVar id="bean" type="java.lang.Object"*/—>
  <!—/*@thymesVar id="beanName" type="java.lang.String"*/—>
  <!—/*@thymesVar id="property" type="java.lang.String"*/—>
  <!—/*@thymesVar id="value" type="java.lang.Object"*/—>
  <!—/*@thymesVar id="constraints" type="ch.itds.thymeleaf.fields.FieldConstraints"*/—>
  <input class="form-control" th:classappend="${not #lists.isEmpty(#fields.errors(property))} ? 'is-invalid'"
      th:field="*{__${property}__}" th:id="${property}"
      type="text">
</th:block>
```

Thymeleaf Fields Dialect im Detail: inlineInput

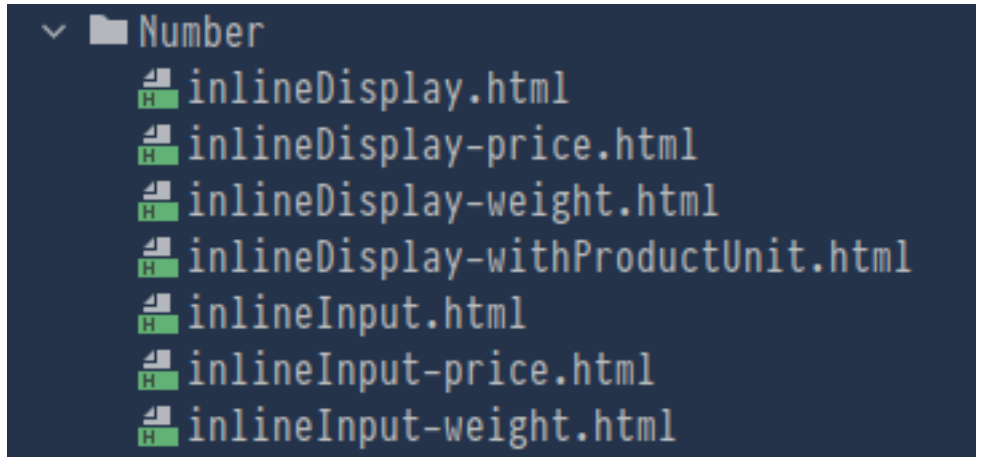
DRY mit eigener Thymeleaf Erweiterung

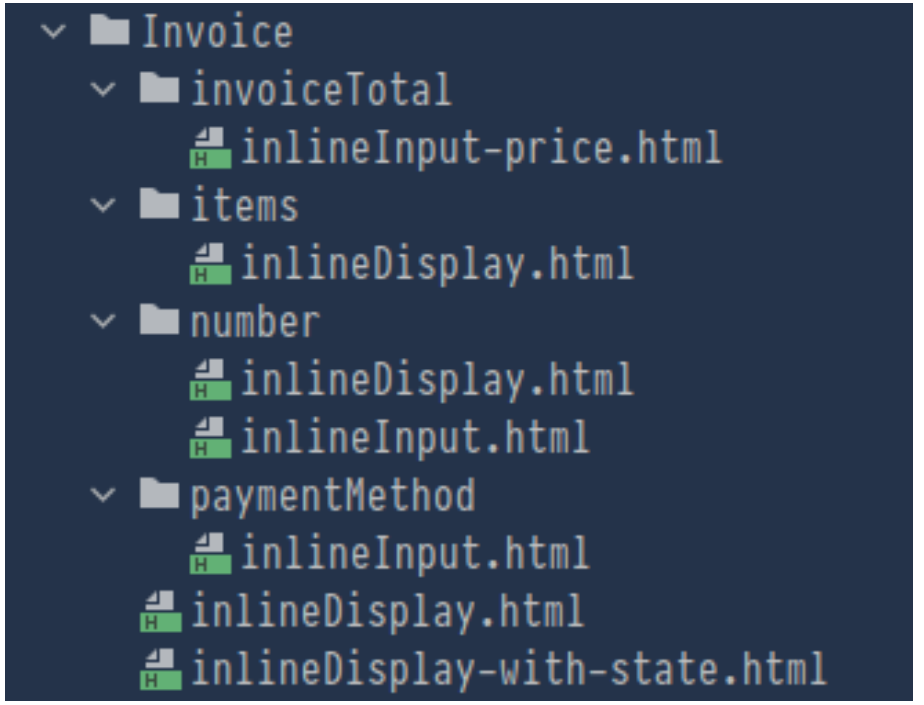
```
fields/theme/className/attributName/tagName-variant
fields/theme/className/attributName/tagName
fields/theme/className/attributeClassName/tagName-variant
fields/theme/className/attributeClassName/tagName
fields/theme/className/attributeSuperClassName/tagName-variant
fields/theme/className/attributeSuperClassName/tagName
fields/theme/superClassName/attributeClassName/tagName-variant
fields/theme/superClassName/attributeClassName/tagName
fields/theme/superClassName/attributeSuperClassName/tagName-variant
fields/theme/superClassName/attributeSuperClassName/tagName
fields/theme/attributeClassName/tagName-variant
fields/theme/attributeClassName/tagName
fields/theme/attributeSuperClassName/tagName-variant
fields/theme/attributeSuperClassName/tagName
fields/theme/tagName-variant
fields/theme/tagName
```

```
fields/className/attributName/tagName-variant
fields/className/attributName/tagName
fields/className/attributeClassName/tagName-variant
fields/className/attributeClassName/tagName
fields/className/attributeSuperClassName/tagName-variant
fields/className/attributeSuperClassName/tagName
fields/superClassName/attributeClassName/tagName-variant
fields/superClassName/attributeClassName/tagName
fields/superClassName/attributeSuperClassName/tagName-variant
fields/superClassName/attributeSuperClassName/tagName
fields/attributeClassName/tagName-variant
fields/attributeClassName/tagName
fields/attributeSuperClassName/tagName-variant
fields/attributeSuperClassName/tagName
fields/tagName-variant
fields/tagName
tagName
```

Thymeleaf Fields Dialect im Detail: Lookup

DRY mit eigener Thymeleaf Erweiterung





Ordnerstruktur

DRY mit eigener Thymeleaf Erweiterung

```
<div layout:fragment="content">

    <ul class="nav justify-content-end content-nav-with-bottom-margin float-right mt-1">
        <li class="nav-item">
            <a class="btn btn-outline-primary" th:href="@{/product}" th:text="#{general.list}">Liste</a>
        </li>
        <li class="nav-item">
            <a class="btn btn-primary" th:href="@{/product/edit/{id}(id=${entity.id})}" th:text="#{general.edit}">Bearbeiten</a>
        </li>
    </ul>

    <h1 th:text="|#{product.show.title} ${entity.name}|">Produkt</h1>

    <div class="row">
        <div class="makoni-main">
            <table class="table table-borderless table-sm table-no-full-width" th:object="${entity}">
                <fields:display fields:property="name"/>
                <fields:display fields:property="code"/>
                <fields:display fields:property="category"/>
                <fields:display fields:property="unit"/>
                <fields:display fields:property="activePrices"/>
            </table>
        </div>
    </div>

</div>
```

product/show.html

DRY mit eigener Thymeleaf Erweiterung

```
<th:block th:fragment="field">
  <!—/*—>
  <!—@thymesVar id="bean" type="java.lang.Object"—>
  <!—@thymesVar id="beanName" type="java.lang.String"—>
  <!—@thymesVar id="property" type="java.lang.String"—>
  <!—*/—>
  <a th:href="@{/invoice/show/{id}(id=${bean.__${property}__.id})}"
    th:text="${bean.__${property}__.number}">
  </a>
  <a th:href="@{/invoice/export/{id}.pdf(id=${bean.__${property}__.id})}">
    <i class="far fa-file-pdf"></i>
  </a>
  <th:block th:text="${#i18n.i18nEnum(bean.__${property}__.state)}"></th:block>
</th:block>
```

fields/Invoice/inlineDisplay-with-state.html

REAKTIONSSCHNELLES FRONTEND MIT SERVERSEITIGEM RENDERING, OHNE JS-FRAMEWORK

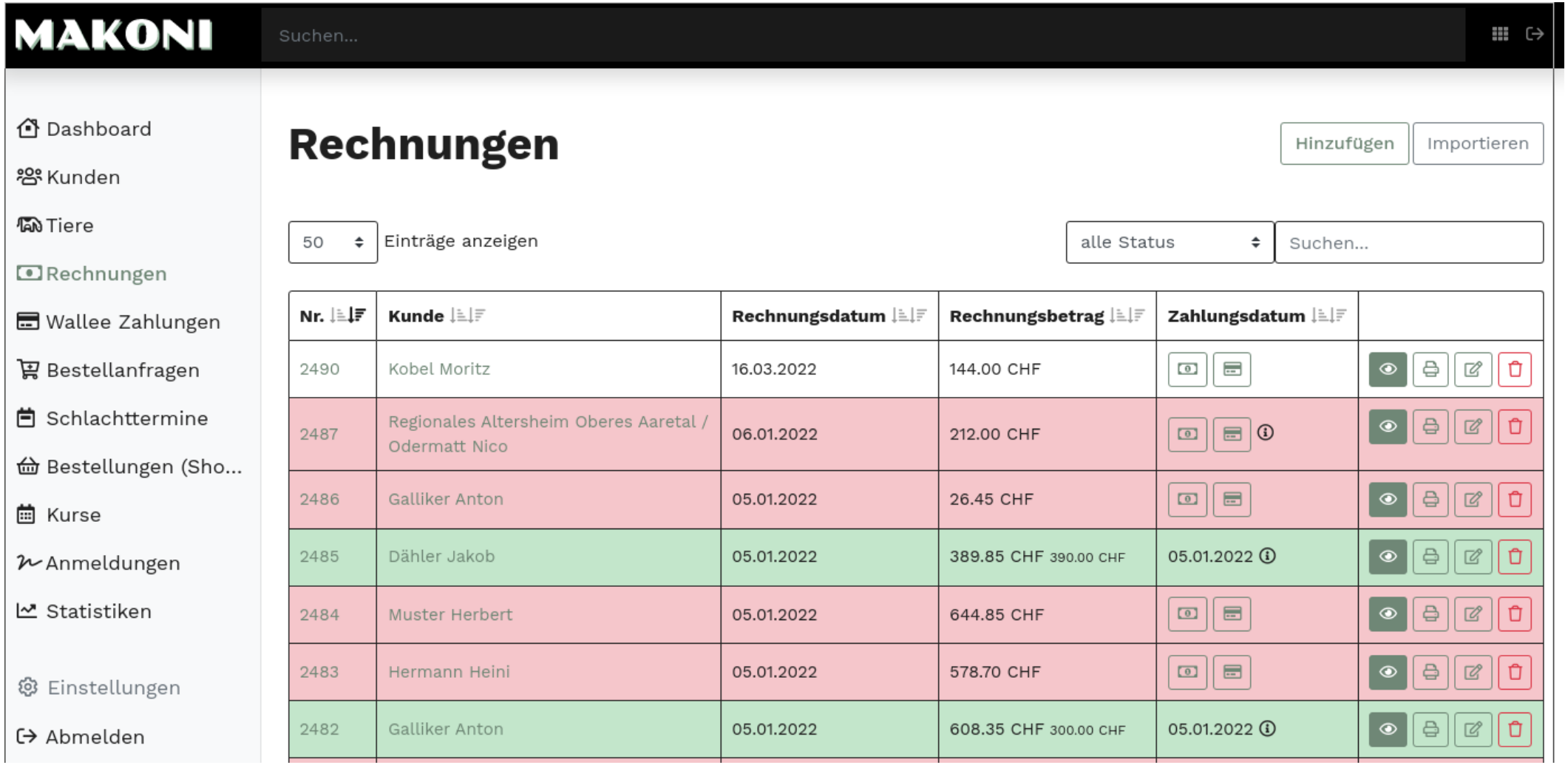


Tabelle der Rechnungen

Reaktionsschnelles Frontend mit serverseitigem Rendering, ohne JS-Framework

Problem

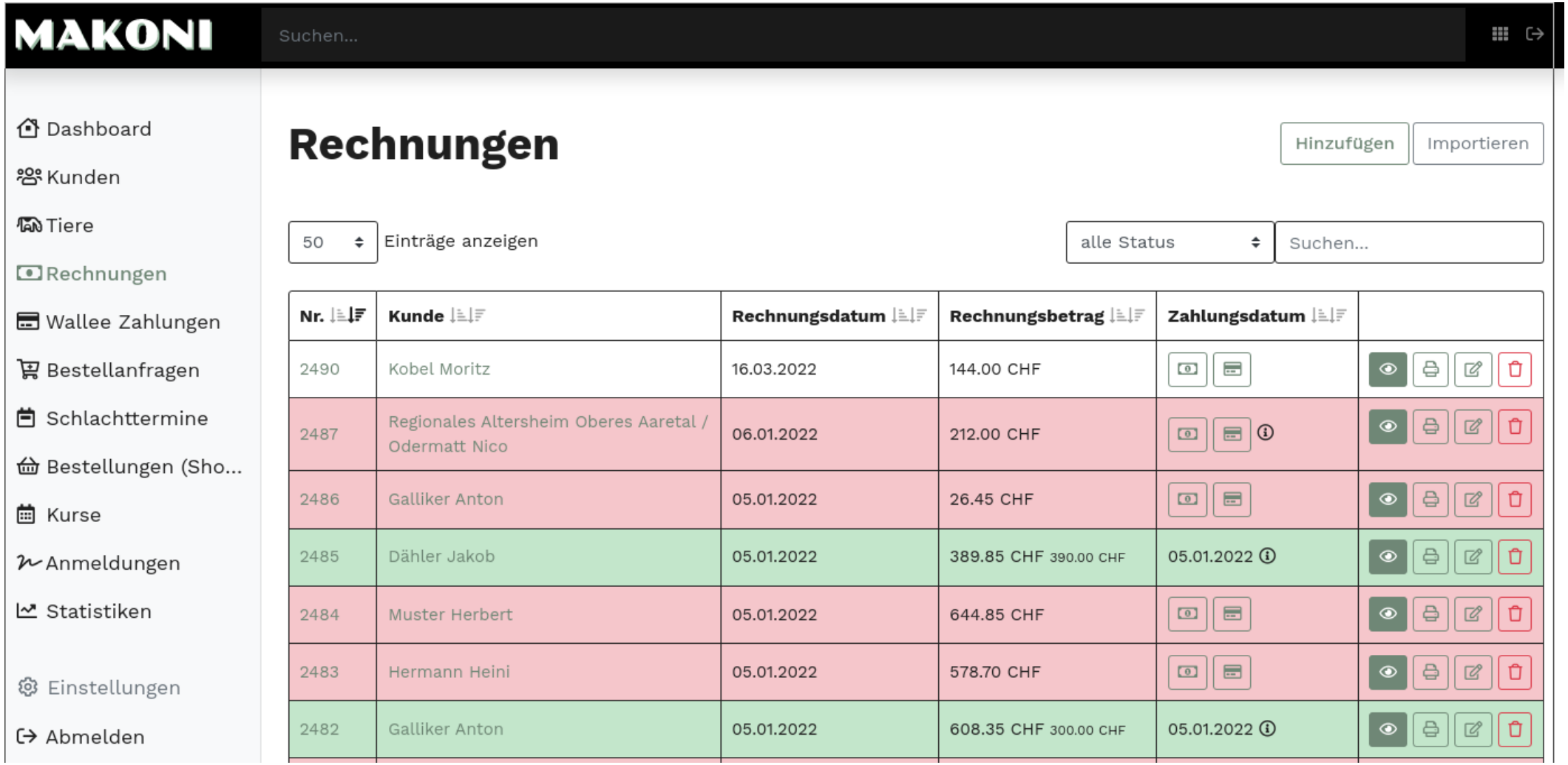
Reaktionsschnelles Frontend mit serverseitigem Rendering, ohne JS-Framework

- Nicht bei allen Aktionen wollen wir die ganze Seite neu laden
- Durchsuchbare Tabellen wie Datatables sind schön, aber nicht so elegant zum Implementieren
- Ich will nicht alles fürs Frontend duplizieren

Lösungsansatz

Reaktionsschnelles Frontend mit serverseitigem Rendering, ohne JS-Framework

- So viel wie möglich serverseitig, von Thymeleaf profitieren
- Mit wenig JS Code Inhaltsbereiche ersetzen, URL nachführen
- ServerSideTableHandling-Interface für Controller



Beispiel: Tabelle der Rechnungen

Reaktionsschnelles Frontend mit serverseitigem Rendering, ohne JS-Framework

```java
@BackendController
@RequestMapping("/invoice")
public class InvoiceController implements CommentableCrudController<Invoice, InvoiceComment>,
ServerSideTableHandling<Invoice> {

    @Override
    public GeneralTableQuery getDefaultGeneralTableQuery() {
        HttpServletRequest curRequest =
                ((ServletRequestAttributes) RequestContextHolder.currentRequestAttributes())
                        .getRequest();
        String stateString = curRequest.getParameter("state");
        InvoiceState state = null;
        if (StringUtils.hasText(stateString) && !"null".equalsIgnoreCase(stateString)) {
            state = InvoiceState.valueOf(stateString);
        }
        return InvoiceTableQuery.of(0, 50, "number", Sort.Direction.DESC.name(), state);
    }

    @Override
    public Specification<Invoice> getServerSideTableHandlingSpecification(GeneralTableQuery query) {
        return ServerSideTableHandlingSpecifications.createInvoiceQuery(query);
    }
```

InvoiceController

Reaktionsschnelles Frontend mit serverseitigem Rendering, ohne JS-Framework

```
public static Specification<Invoice> createInvoiceQuery(GeneralTableQuery tableQuery) {
  return (root, query, builder) -> {
    List<Predicate> combinedPredicates = new ArrayList<>();
    if (StringUtils.hasText(tableQuery.getName())) {
      Predicate customerNameLike = builder.like(builder.upper(unaccent(root.get("customer").get("displayName"), builder)),
                  unaccent(tableQuery.getNameQuery(), builder));
      Predicate numberLike = builder.like(root.get("number").as(String.class), tableQuery.getNameQuery());
      Predicate invoiceTotalLike = builder.like(root.get("invoiceTotal").as(String.class), tableQuery.getNameQuery());
      Predicate paymentTotalLike = builder.like(root.get("paymentTotal").as(String.class), tableQuery.getNameQuery());
      combinedPredicates.add(builder.or(customerNameLike, numberLike, invoiceTotalLike, paymentTotalLike));
    }
    if (tableQuery instanceof InvoiceTableQuery) {
      InvoiceState state = ((InvoiceTableQuery) tableQuery).getState();
      if (state != null) {
        switch (state) {
          case PAID:
            combinedPredicates.add(builder.isNotNull(root.get("paymentDate")));
            break;
          case OVER_DUE:
            combinedPredicates.add(builder.isNull(root.get("paymentDate")));
            combinedPredicates.add(builder.lessThan(builder.sum(root.get("invoiceDate"),
                                    builder.sum(root.get("terms").get("paymentDueDays"), 5)).as(LocalDate.class), LocalDate.now()));
            break;
          case DUE:
            combinedPredicates.add(builder.isNull(root.get("paymentDate")));
            combinedPredicates.add(builder.greaterThanOrEqualTo(builder.sum(root.get("invoiceDate"),
                                    builder.sum(root.get("terms").get("paymentDueDays"), 5)).as(LocalDate.class), LocalDate.now()));
            break;
        }
      }
    }
```

ServerSideTableHandlingSpecification.createInvoiceQuery

```
public interface ServerSideTableHandling<E extends BaseEntity>
        extends ServerSideTableHandlingEntityRepositoryAware<E>, CrudController<E> {

    GeneralTableQuery getDefaultGeneralTableQuery();

    Specification<E> getServerSideTableHandlingSpecification(GeneralTableQuery query);

    @Override
    @RequestMapping("")
    default String index(Model model) {

        GeneralTableQuery query = getDefaultGeneralTableQuery();

        Page<E> entityPage = getPage(query);

        model.addAttribute("entityPage", new PageWrapper<>(entityPage));
        model.addAttribute("query", query);
        model.addAllAttributes(addCustomListAttributes(entityPage.getContent()));

        return newEntityInstance().getDecapitalizedClassName() + "/index";
    }
```

ServerSideTableHandling

Reaktionsschnelles Frontend mit serverseitigem Rendering, ohne JS-Framework

```
<th:block th:fragment="filter">
  <!—/* @thymesVar id="query" type="ch.itds.makoni.domain.invoice.InvoiceTableQuery" */—>

  <select class="form-control custom-select" id="custom-field-state"
          name="state" onchange="generalTableSearchWithUpdatedCustomField('state')">
    <option value="null" th:text="#{InvoiceState.filter.ALL}">alle Rechnungen</option>
    <option th:each="dropdownValue:
                     ${T(ch.itds.makoni.domain.invoice.InvoiceState).values()}"
            th:text="|${#i18n.i18nEnum(dropdownValue)}|"
            th:value="${dropdownValue.name()}" th:selected="${query.state==dropdownValue}">
    </option>
  </select>

</th:block>
```

invoice/inline-search-table.html

Reaktionsschnelles Frontend mit serverseitigem Rendering, ohne JS-Framework

```
<div id="searchTable"
    data-search-custom-fields="state"
    th:attr="data-search-url=@{/invoice/inline-search-table(state=query.state)},data-search-name=${query.name},data-search-sort-by=${query.sortBy},data-search-
sort-direction=${query.sortDirection},data-search-page-number=${query.page},data-search-page-size=${query.size},data-search-custom-field-state=${(query.state)}">
   <table class="table table-bordered row-clickable mt-4">
      <thead><tr>
         <th class="minimal-width no-wrap">
            <th:block th:text="#{invoice.number.short.label}"/>                <ui:sort-handles ui:sort-by="number" ui:query="${query}"/>
         </th>
         <th class="no-wrap">
            <th:block th:text="#{invoice.customer.label}"/>          <ui:sort-handles ui:sort-by="customer.displayName" ui:query="${query}"/>
         </th>
         <th class="no-wrap">
            <th:block th:text="#{invoice.invoiceDate.label}"/> <ui:sort-handles ui:sort-by="invoiceDate" ui:query="${query}"/>
         </th>
         <th class="no-wrap">
            <th:block th:text="#{invoice.invoiceTotal.label}"/><ui:sort-handles ui:sort-by="invoiceTotal" ui:query="${query}"/>
         </th>
         <th class="no-wrap">
            <th:block th:text="#{invoice.paymentDate.label}"/>                <ui:sort-handles ui:sort-by="paymentDate" ui:query="${query}"/>
         </th>
         <th class="minimal-width"></th>
      </tr></thead>
      <tbody>
      <tr th:if="${entityPage.getContent().isEmpty()}">
         <td colspan="6" class="table-warning" th:text="#{invoice.list.empty}"></td>
      </tr>
      <tr th:classappend="|invoice-state-${item.state.name()}|" th:each="item : ${entityPage.getContent()}"
```

invoice/inline-search-table.html

Reaktionsschnelles Frontend mit serverseitigem Rendering, ohne JS-Framework

MULTI-TENANCY AUF BASIS VON ROW-PERMISSIONS IN DER POSTGRESQL-DATENBANK

Problem

- Mehrere Anwendungsinstanzen lohnen sich nicht
- Mehrere Datenbanken sind bei Schemaänderungen schwierig zum Pflegen
- Eine tenant-Spalte pro Zeile ist in allen Queries aufwändig

Lösungsansatz

Multi-Tenancy auf Basis von Row-Permissions in der PostgreSQL-Datenbank

- Eine tenant-Spalte pro Zeile geht sehr gut!
- Row Level Security
- Tenant wird pro Requests via Filter anhand der URL gesetzt

https://callistaenterprise.se/blogg/teknik/2020/10/24/multi-tenancy-with-spring-boot-part6/

```java
@MappedSuperclass
@Getter
@Setter
@NoArgsConstructor
@EntityListeners(TenantListener.class)
public abstract class TenantAwareBaseEntity extends BaseEntity implements TenantAware {

    @Column(name = “tenant_id”)
    private Long tenantId;

}
```

TenantAwareBaseEntity

```java
public class TenantListener {

    @PreUpdate
    @PreRemove
    @PrePersist
    public void setTenant(TenantAware entity) {
        final Long tenantId = TenantContext.getTenantId();
        entity.setTenantId(tenantId);
    }
}
```

TenantListener

```
public class TenantAwareDataSource extends DelegatingDataSource {

    @Override
    public Connection getConnection() throws SQLException {
        final Connection connection = Objects.requireNonNull(getTargetDataSource()).getConnection();
        setTenantId(connection);
        return getTenantAwareConnectionProxy(connection);
    }

    private void setTenantId(Connection connection) throws SQLException {
        try (Statement sql = connection.createStatement()) {
            Long tenantId = TenantContext.getTenantId();
            log.debug("set tenant = " + tenantId);
            sql.execute("SET app.tenant_id TO '" + tenantId + "'");
        }
    }
    private void clearTenantId(Connection connection) throws SQLException {
        try (Statement sql = connection.createStatement()) {
            sql.execute("RESET app.tenant_id");
        }
    }
    // Connection Proxy that intercepts close() to reset the tenant_id
    protected Connection getTenantAwareConnectionProxy(Connection connection) {
        return (Connection) Proxy.newProxyInstance(
                ConnectionProxy.class.getClassLoader(),
                new Class[]{ConnectionProxy.class},
                new TenantAwareDataSource.TenantAwareInvocationHandler(connection));
    }
```

TenantAwareDataSource

```
ALTER TABLE beeforder
        ENABLE ROW LEVEL SECURITY;

DROP POLICY IF EXISTS beeforder_tenant_isolation_policy ON beeforder;

CREATE POLICY beeforder_tenant_isolation_policy ON beeforder
        USING (tenant_id = current_setting('app.tenant_id')::BIGINT);
```

CREATE POLICY

- **Liquibase Konfiguration etwas aufwändiger**
  - **separater Benutzer mit eigenen Rechten**
  - **Applikationsbenutzer über Variablen in Liquibase (für Docker)**
- **Async mit TenantAwareTaskDecorator**

Schwierigkeiten

AUTOMATISCHES SETUP/TEARDOWN DER INTEGRATIONSUMGEBUNG MIT GITLAB-CI UND ANSIBLE

Problem

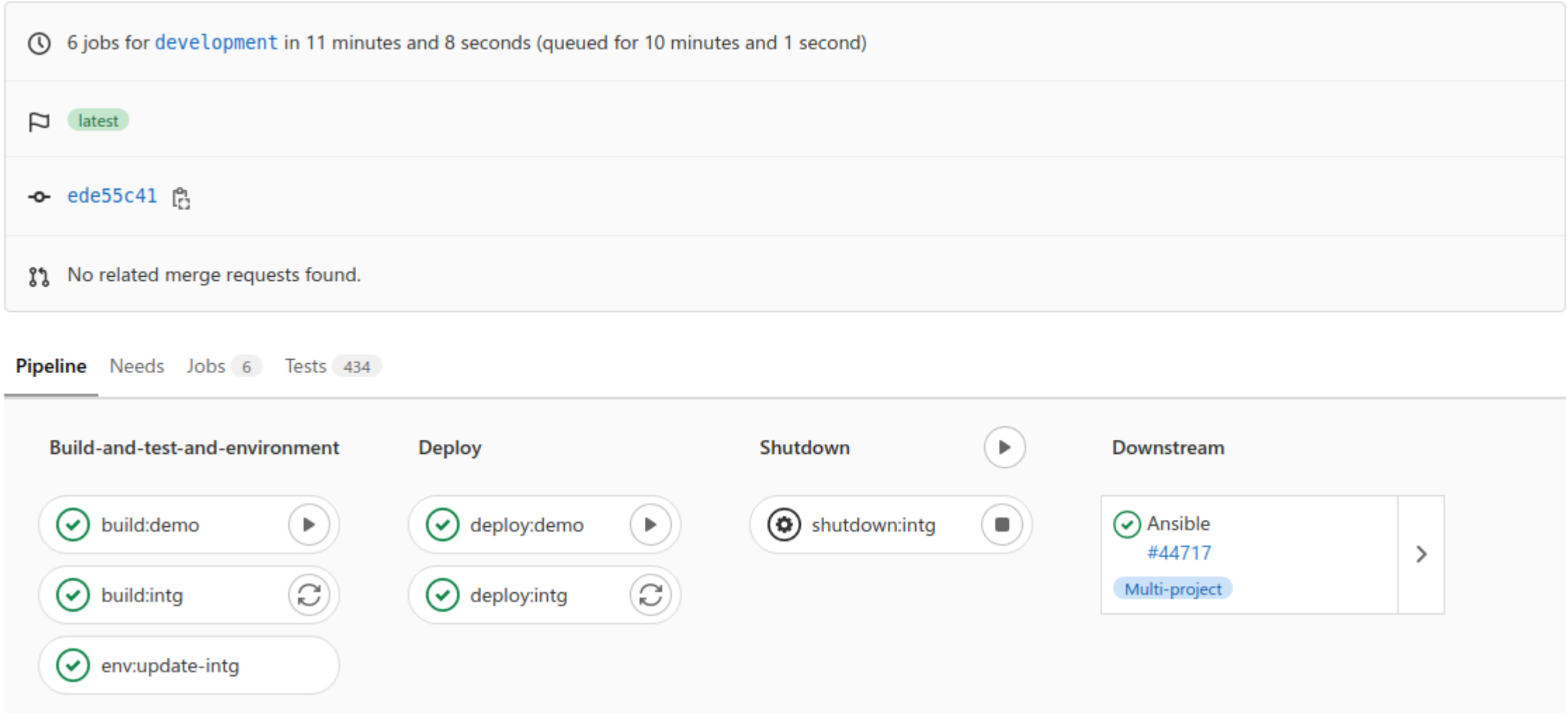
Auto Setup/Teardown der Integrationsumgebung mit GitLab-CI und ansible

- Integrationsinstanzen kosten Geld
- Manuelles ein/ausschalten ist mühsam

Lösungsansatz

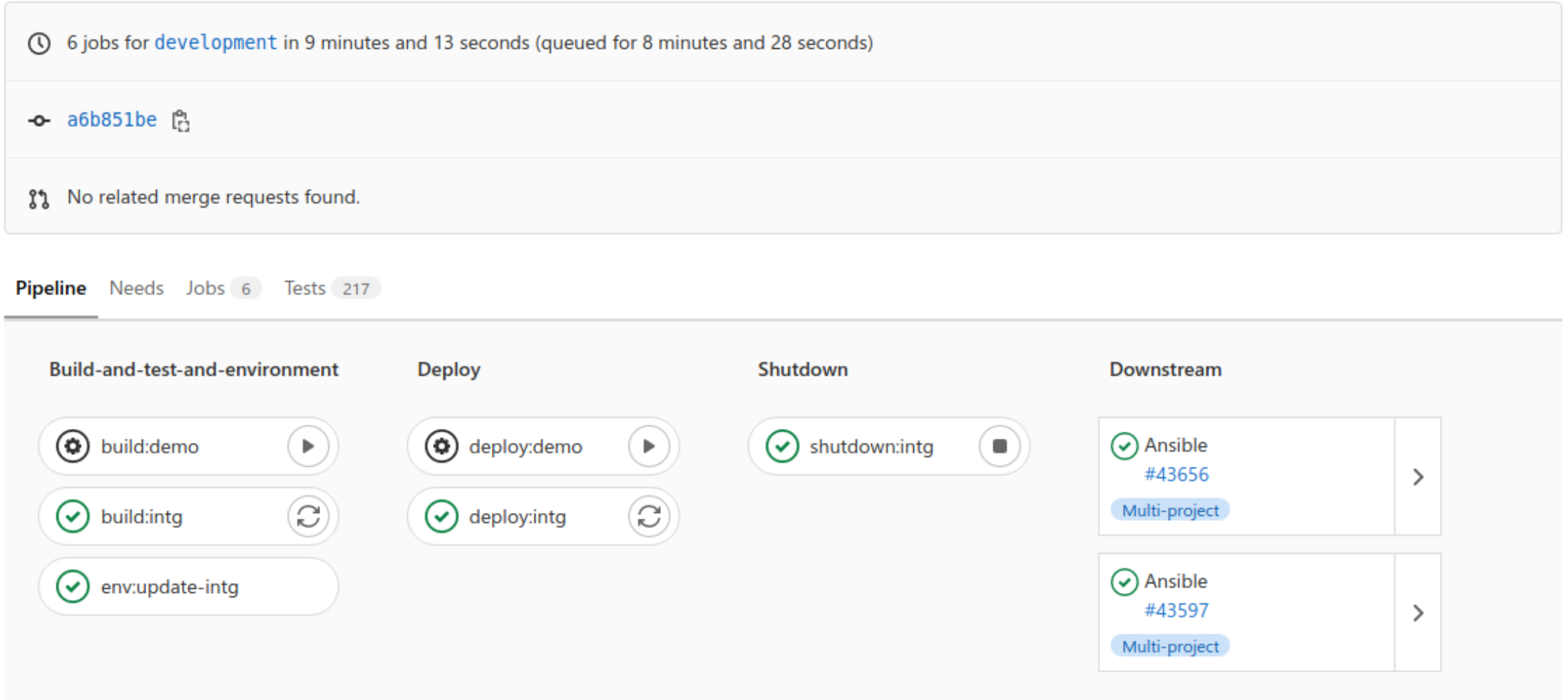
Auto Setup/Teardown der Integrationsumgebung mit GitLab-CI und ansible

- GitLab-CI weis, wann die Infrastruktur benötigt wird
- Setup & Teardown mit ansible



Pipeline

Auto Setup/Teardown der Integrationsumgebung mit GitLab-CI und ansible



Pipeline

Auto Setup/Teardown der Integrationsumgebung mit GitLab-CI und ansible

```
env:update-intg:
  stage: build-and-test-and-environment
  trigger:
    project: inf/ansible
    branch: master
    strategy: depend
  variables:
    SERVER: example.itds-net.ch
    DOMAIN: example.itds-test.ch
    APPLICATION: makoni-integration
    STATE: present
  only:
    - development
  except:
    - schedules
    - tags
```

start trigger in .gitlab-ci.yml

```
deploy:configure-spring-boot-and-webproxy:
  stage: deploy
  only:
    variables:
      - $SERVER =~ /^[a-z]+\.itds-net\.ch$/
      - $DOMAIN =~ /^[a-z0-9\-\.]+$/
      - $APPLICATION =~ /^[a-z0-9]+\-[a-z0-9]+$/
  script:
    - if [ “$STATE” != “” ] ; then
              ./ci-set-instance-state.sh $SERVER $APPLICATION $DOMAIN $STATE ; fi
    - ansible-playbook -i inventory library/springboot.yml -l $SERVER
              -t user,service,config,monitoring -e limit_applications=$APPLICATION
    - ansible-playbook -i inventory library/configure-webserver.yml -l $SERVER
              -t instance-config-only -e limit_webproxy=$DOMAIN
  resource_group: infrastructure
```

ansible in .gitlab-ci.yml

```
deploy:intg:
  extends: .deploy:template
  only:
    - development
  variables:
    <<: *default-deploy-variables
    TH: example.itds-net.ch
  environment:
    name: integration
    url: https://${INT_PROJECT_NAME}.itds-test.ch/
    on_stop: shutdown:intg
    auto_stop_in: 1 week
  resource_group: integration-deployment
```

deploy mit auto_stop in .gitlab-ci.yml

```
shutdown:intg:
  image: debian-sshclient:latest
  stage: shutdown
  when: manual
  environment:
    name: integration
    action: stop
  script:
    - curl —request POST —form “token=$CI_JOB_TOKEN” —form ref=master
                —form variables[SERVER]=example.itds-net.ch
                —form variables[DOMAIN]=example.itds-test.ch
                —form variables[APPLICATION]=makoni-integration
                —form variables[STATE]=absent
                “https://gitlab/api/v4/projects/123456/trigger/pipeline”
  only:
    - development
  except:
    - schedules
    - tags
```

shutdown trigger in .gitlab-ci.yml

qube.ag

Fragen?

Schauen Sie vorbei qube.ag

Impressum

Qube creatives

Aarau
Qube AG
Laurenzenvorstadt 21
CH-5000 Aarau

Bern
Qube AG
Schulhausgasse 22
CH-3113 Rubigen