**Domain Driven Design of Large Software Systems**

Master Digital Science / Software Architecture, WS 2025 / 26

Prof. Dr.-Ing. Stefan Bente

# DDD Case Study Project: Current Status

16.01.2026

Technology Arts Sciences
TH Köln

# Agenda

- Status
  - Current size
  - Integration
- Survey
  - Course Organization
  - Project Organization
  - Technical Issues (Spring Modulith / jMolecules / …)
- Development Recommendations
- Some Sample Comments to your Code

# Status

# Current Size (LoC), only Java files counted

- Original main: 260 LoC

- origin/client-api-cooking-pickup-voice: 2135 LoC
- origin/development-FoodItem-Ordering: 1077 LoC
- origin/feature-ordering-implementation: 1395 LoC
- origin/tablemanagement-reservation: 1051 LoC

- => There is evidently work done on the project => great
- However, most of it is not visible in `main`

# Integration

- Mainly three branches currently, one per subteam
- One merge into main (yesterday)
- 2 problems:
  1. No merge of `main` into feature branch before merging back
  2. Tests are red on `main` => this is a no-go

- All in all, this is not how we discussed this on 21.11.
  - see slides on branching strategies

- You need to integrate more often!
  - at least 1x per week

- Do you really need subteam dev branches?
  - Require a double merge to get results back to main
  - Consider using pure feature branches only (see also later slides)

# Survey

# Course Organization

- Too much time for the first part („specification workshops“) of the course (3x)
  - Coding should start earlier
- More resources on Spring basics (2x)
  - Let‘s discuss if I can still point out resources – what exactly?
    - from next year onwards, this will be „Advanced DDD“
    - „DDD fundamentals“ will then (by and large) be ST2 for those students who didn‘t deal with this in their BA studies
  - Please use the Discord channels more to get help
  - The pretask should exactly help you get familiar with Spring Modulith, did that fail?

# Course Organization (2)

- Communication via Discord not ideal (2x)
  - „it bothers me a bit that messages in discord are often late at night (often right before the lecture the next day) since I mostly sleep at the time (after 21pm) and they come to my phone”
  - “I would like important messages from you to be communicated via email (and discord). Discord is not a bad communication medium, but there is still the possibility, that things may be overlooked.“
  - => what would be a good policy?

# Course Organization (3)

- Example repo for Spring Modulith wanted (1x)
  - I know exemplary code is provided from other repos linked on the archi lab website, but the code you want as a lecturer might be different than the code some other person in the world would do it.
  - => ??
  - https://www.archi-lab.io/infopages/spring/spring-modulith-getting-started.html#get-the-inofficial-spring-modulith-example-applications
    - none of the two has been written by me
    - one is even by the inventor of Spring Modulith
    - beyond these two, I indeed know of no other public examples (last time I looked, anyway)

# Course Organization (4)

- Time Conflicts (1x)
  - Parallel modules
  - Xmas break / block week after New Years Eve led to less working time than one would think
  - => **what is a realistic submission date?**

- Overall rating of the course is positive / interesting (2x)

- Distribution among the teams a bit unbalanced (1x)
  - not fully clear – workload? experience?
  - “It’s not dramatic, but I personally might have chosen a different approach here — for example, parallel programming of individual tasks or a more fine-grained distribution of tasks.”

# Project Organization

- Communication between subteams needs improvement (4x)
  - Regular status meeting?
    - Should we do this?
    - Weekly / biweekly?
  - “Communication with other teams is a weakpoint - probably for every team. We coded what we could do alone for now. Then we waited until everybody in our team approved the code. Not everyone from our team had time to do so quickly. Now we are ready to discuss what we need from the other teams. So we would have asked the other teams anyway now. Or we will discuss what we need tomorrow.“

- => see recommendations for development (last section of the slides)

# Project Organization (2)

- Collaboration within subteam good / relaxed / nice (3x)

- More „external“ governance wanted (2x)
  - „ It might be helpful to set some intermediate goals in order to achieve part ’X’ of the implementation and speak more about the implementation status between the teams.”
  - The other point is that we are obliged to split up the workload for ourselves - when we do what and how much. There are no deadlines or something similar, which may lead to people working less on DDD - which in itself is probably a good learning when it comes to project work in teams - you have to motivate yourself and you are responsible for fulfilling your tasks.
  - => What would help you to getter a clearer task description?

# Technical Issues (Spring Modulith / jMolecules / …)

- How to handle IDs in Events (Identifiers ? UUIDs?)
  - IDs should be exported between the aggregates
  - => then you can use IDs in events
  - otherwise UUIDs can easily be confused (not typesafe)
    - e.g. parameters sequence in a method confused
    - => table ID interpreted as order ID, and vice versa
    - => exception

- When the Service is annotated with jMolecules `@service` tag, it is not possible to use it in the RestController
  - This is not true
  - jMolecules is **by design technology-agnostic**
  - You need to run the ByteBuddy plugin beforehand, this adds the required JPA Tag
    - see example e.g. in tablemanagement branch

# Development Recommendations

# General Guidelines

- Start top-down, from the API inwards
  - Synchronous API => REST => ✓
  - Asynchronous API => Events
  - Make sure that all the events you identified during the specification process are created and visible in your module
- What is visible, what is hidden?
  - **Visible** (top-level in module, unless configured otherwise by `package-info.java`):
    - Events
    - Identifier
    - Aggregate Root
    - (Domain / Application) Service
  - **Not visible** (in some subfolder):
    - Listener
    - Repository
    - RESTController

# General Guidelines (2)

- First things to completely create and make public:
  1. All your events
  2. All your Aggregate Root Identifiers
  3. All your Aggregate Roots (at least in base version)

- How to make stuff public, if it resides in a subfolder?
  - … to avoid clutter on the module top level
  - Only really works for interfaces
  - Let‘s assume you have a package „order“ and a sub-package „api“

```
@ApplicationModule
package com.example.order;

import org.springframework.modulith.ApplicationModule;
```

*package-info.java (order)*

```
@NamedInterface("api")
package com.example.order.api;

import org.springframework.modulith.NamedInterface;
```

*package-info.java (order.api)*

# Development Guidelines for a Feature

1. Select a feature to implement from your specification
2. Create a feature branch for it
3. How can you test it? Classify:
   a. Trigger event => *computation* => result event
   b. Trigger event => *computation* => state has changed
   c. User action (REST call) => *computation* => result event
   d. User action (REST call) => *computation* => state has changed
      (unlikely case, because this usually should include (c) as well)
4. Write your acceptance test for it (=> see examples next slides)
5. Write the "*computation*" part, make sure to publish result event
6. Debug until test is green
7. Merge main into your feature branch
8. Merge your feature branch into main

# (a) Trigger event => computation => result event

- You can use the Pretask tests as examples

```
CarReserved carReserved = /* instantiate your trigger event */
scenario.publish( carReserved )
        .andWaitForEventOfType( CarOrderedForCarFleet.class )
        .toArriveAndVerify( carOrderedForFleet -> {
            assertTrue( carOrderedForFleet.getBecauseOfEmployeeId().isPresent() );
            assertEquals( … );
            assertEquals( … );
        } );

// then
assertEquals( 0, /* some service method call result */ );
assertEquals( 1, /* some service method call result */ );
//…
```

- (you can combine this with checks on state change, see last asserts)

# (b) Trigger event => computation => state has changed

```
CarReserved carReserved1 = /* instantiate your trigger event */

// when
scenario.publish( carReserved1 )
        .andWaitForStateChange(
                () -> billingAPI.getTotalCostFor( idAlfred ),
                cost -> cost != 0 )
        .andVerify( cost -> assertEquals( COST_PER_DAY.getFee(), cost ) );
```

# (c) User action (REST call) => *computation* => result event

```
@Test
public void testPlanRegularJourney( Scenario scenario ) {
   scenario.stimulate( () ->  employeeAPI.planJourney( idAlfred, d1, d4, true ) )
         .andWaitForEventOfType( CarReserved.class )
         .toArriveAndVerify( carReserved -> {
            assertEquals( idAlfred, carReserved.getEmployeeId() );
         } );
}
```

# (d) User action (REST call) => *computation* => state has changed

- This example is from another repo

  see https://git.archi-lab.io/public-repos/internship/internship-rest-solution/-/blob/master/src/test/java/thkoeln/archilab/internship/internship/InternshipRESTAttendanceCheckTests.java?ref_type=heads

```
@Test
public void testAddAttendanceCheckViaREST() throws Exception {
   // given
   ObjectMapper objectMapper = new ObjectMapper();
   objectMapper.registerModule( new JavaTimeModule() );
   String json1 = objectMapper.writeValueAsString( sampleData.ATT_1_0 );

   // when
   mockMvc.perform( post("/internships/" + INTERSHIP_AMELIE_ID + "/attendanceChecks" )
                .contentType(APPLICATION_JSON).content( json1 ))
         .andExpect(status().isCreated())
         .andReturn();

  // … and now some status checks by calling service methods …
}
```

# Some Sample Comments to your Code

# Findings on `main` Branch

- Public Classes (top level on module) – what are they?
  - Currently just POJOs
  - Are they Aggregate Roots?

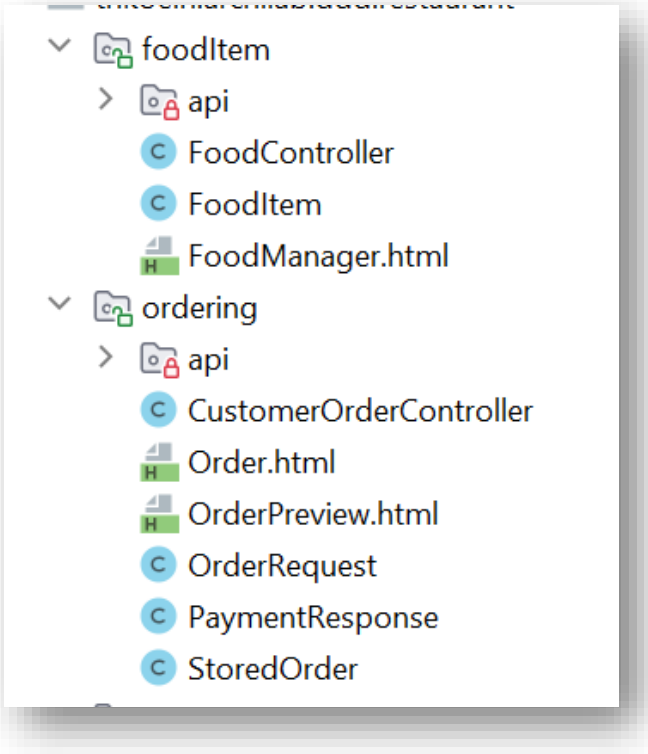
```
@Getter
@Setter
public class FoodItem {
   private Long id;
   private String name;
   private String recipe;
   private boolean active;
   private Double price;

   public FoodItem() {}

   public FoodItem(Long id, String name, String recipe, boolean active, Double price) {
      this.id = id;
      this.name = name;
      this.recipe = recipe;
      this.active = active;
      this.price = price;
   }
}
```

# Findings on `main` Branch (2)

- You should check the visibility rules
  - Controllers don‘t need to be public
  - HTML files neither (I guess, unless I overlook something)



# Findings on `main` Branch (3)

- The API interfaces don‘t have implementing services yet

```
@ApplicationModuleTest
public class FoodItemUnitTests {
   private final FoodItemAdminAPI foodItemAdminAPI;

   private Identifier idPizza, idPasta, unknownId;

   public FoodItemUnitTests(FoodItemAdminAPI foodItemAdminAPI) {
      this.foodItemAdminAPI = foodItemAdminAPI;
   }
   //…
}
```