052400-1 VU Information Management and Systems Engineering (2025W)

Milestone 2

Group 5

Student 1: Aziz, Iftekher, 12338137

Student 2: Baur, Lennard, 12018378

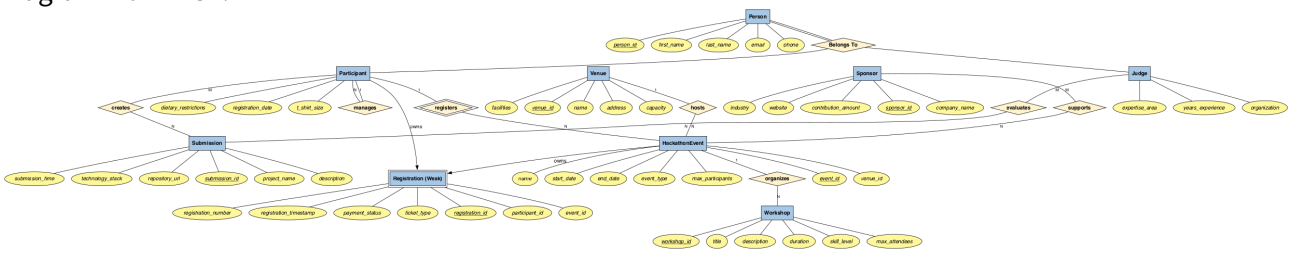
[*Date*], Vienna

Contents

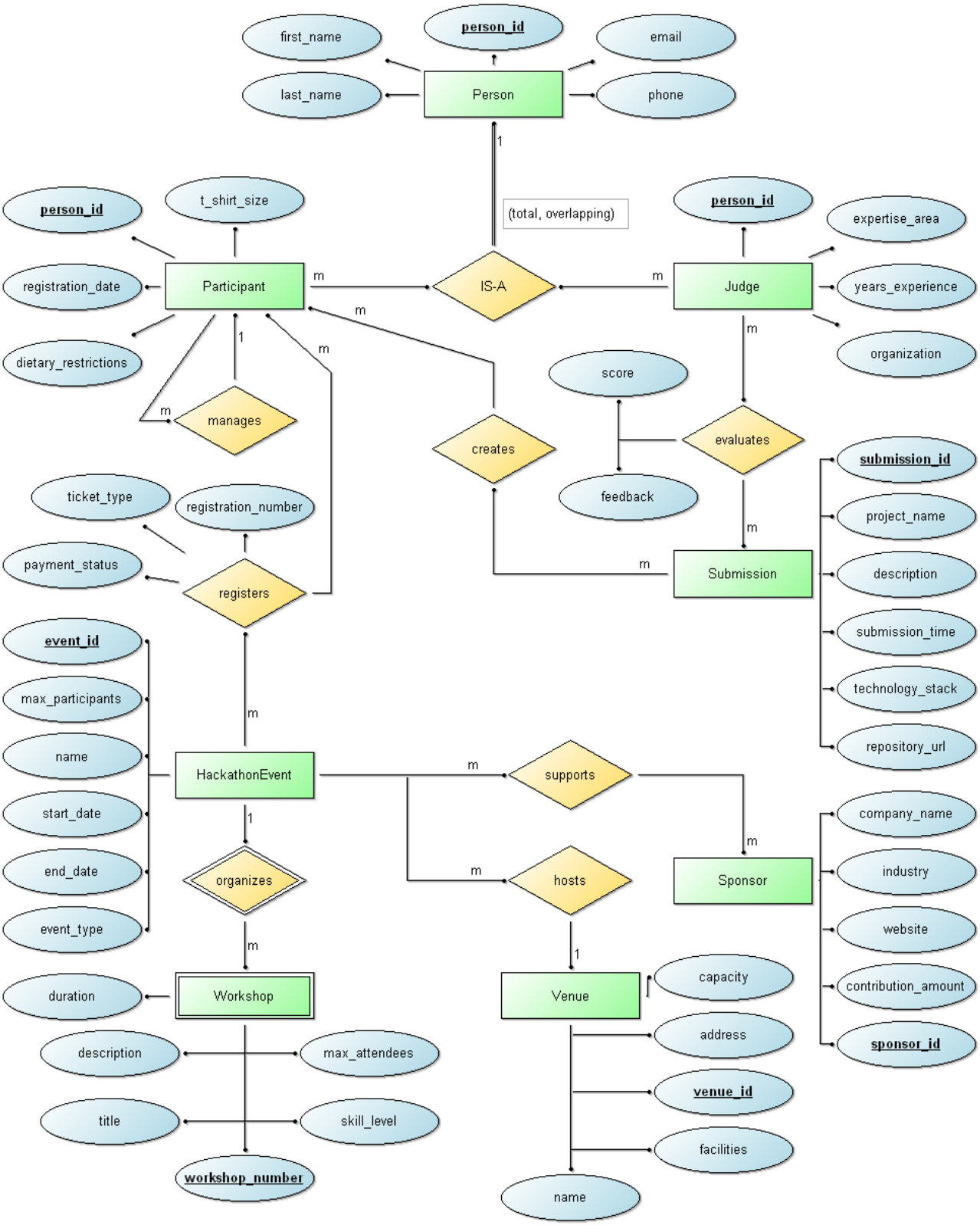
1. Milestone 1 - Recap and Revisions

1.1. Recap: ER-Diagram

ER-Diagram from MS1:



New ER-Diagram (for modeling the program BeeUp has been used, therefore the ChanNoation is a bit different as it only offers M:M instead of M:N)



1.2. Revisions and Changes

Based on the feedback received for MS1, the following changes were made to the ER diagram (as a Team):

Weak Entity (Feedback: “Weak entity sounds more like a bridging table”)

- **Before:** Registration was modeled as a weak entity with a composite key (participant_id, event_id).
- **After:** Registration is now a **relationship with attributes** between Participant and HackathonEvent. Also Workshop has become the new Weak Entity with a chnaged attribute from "workshop_id" to "workshop_number".

IS-A Relationship (Feedback: “IS-A relationship not properly modelled”)

- The IS-A relationship now uses proper **triangle notation** with **(total, overlapping)**.

Cardinalities (Feedback: “Cardinalities are either wrong or not properly assignable”)

- **Before:** “manages” was M:M and “creates” was M:1.
- **After:** “manages” corrected to **1:N** and “creates” corrected to **M:N**. All cardinalities are now clearly placed next to their respective entities.

Visual Presentation (Feedback: “Model is difficult to read”)

- **Before:** Cluttered layout with overlapping lines and unclear attribute placement.
- **After:** Reorganized layout with clear spacing, consistent color coding (green for entities, yellow for relationships, blue for attributes), and improved readability.

1.2.1. Changes Student 1 Individual Part

Based on the feedback I received for MS1, I made the following changes to the individual sections:

Textual Description (Feedback: “Relevant action missing in Main Flow description”)

Use Case Name: Submit Hackathon Project

Goal/Purpose: Allow a registered participant to submit a hackathon project for evaluation by judges, either as an individual or as part of a team.

Trigger: The participant selects the option “Submit Project” from their dashboard within the Hackathon Management System.

Precondition(s):

1. These conditions are assumed to be true before the use case starts.
2. The participant is logged in (a valid *Person* and *Participant* record exists).
3. The participant is registered for an active *HackathonEvent*.
4. The event’s project submission period is open.
5. For team submissions: All potential team members must be registered for the same HackathonEvent.
6. Once these conditions hold, the use case begins at “Click *Submit Project.*”.

Main Flow: Submit a hackathon project

1. Participant navigates to the registered hackathon event dashboard and selects “Submit Project.”
2. **System prompts: “Is this an individual or team submission?”**
    - **If Individual submission:** Proceed directly to step 3

- **If Team submission:**
  (a) Participant selects “Add Team Members”
  (b) System displays a list of all registered participants for this hackathon event
  (c) Participant selects one or more team members from the list
  (d) Participant confirms team selection, then proceed to step 3

3. Participant fills in the project details: project name, description, technology stack, and repository URL.
4. System validates the input data (checking for required fields, URL format, character limits).
5. System creates a new record in the Submission table with the provided project details.
6. **System creates relationship record(s) in the Creates table:**
   - **For individual submissions:** One record linking the submitting participant (IS-A Person) to the new Submission
   - **For team submissions:** Multiple records linking each team member (participant_id) to the same Submission (submission_id), establishing the m:n relationship between Participant and Submission
7. System confirms successful submission and displays a success message to the participant.

Postcondition(s):

1. A new project record is inserted into the Submission table with participant-provided details.
2. **One or more link records are created in the Creates table:**
   - **For individual submissions:** One record linking the Participant to the Submission
   - **For team submissions:** Multiple records linking each team member (Participant) to the Submission (representing the m:n relationship)
3. All team members (if applicable) are notified of the successful submission.
4. The participant’s submission becomes available for viewing or updating until the event deadline.

Entities Involved: ***Person*** serves as the superclass in the **IS-A hierarchy**, from which ***Participant*** is derived as a **specialized entity** performing the project submission. The use case also involves ***Submission***, an **independent entity** that stores project details, and the **associative relationship *Creates***, which links each *participant* to their corresponding *Submission*.

Graphical Representation (Feedback: “Relevant flow missing, Controlflow wrong..”)

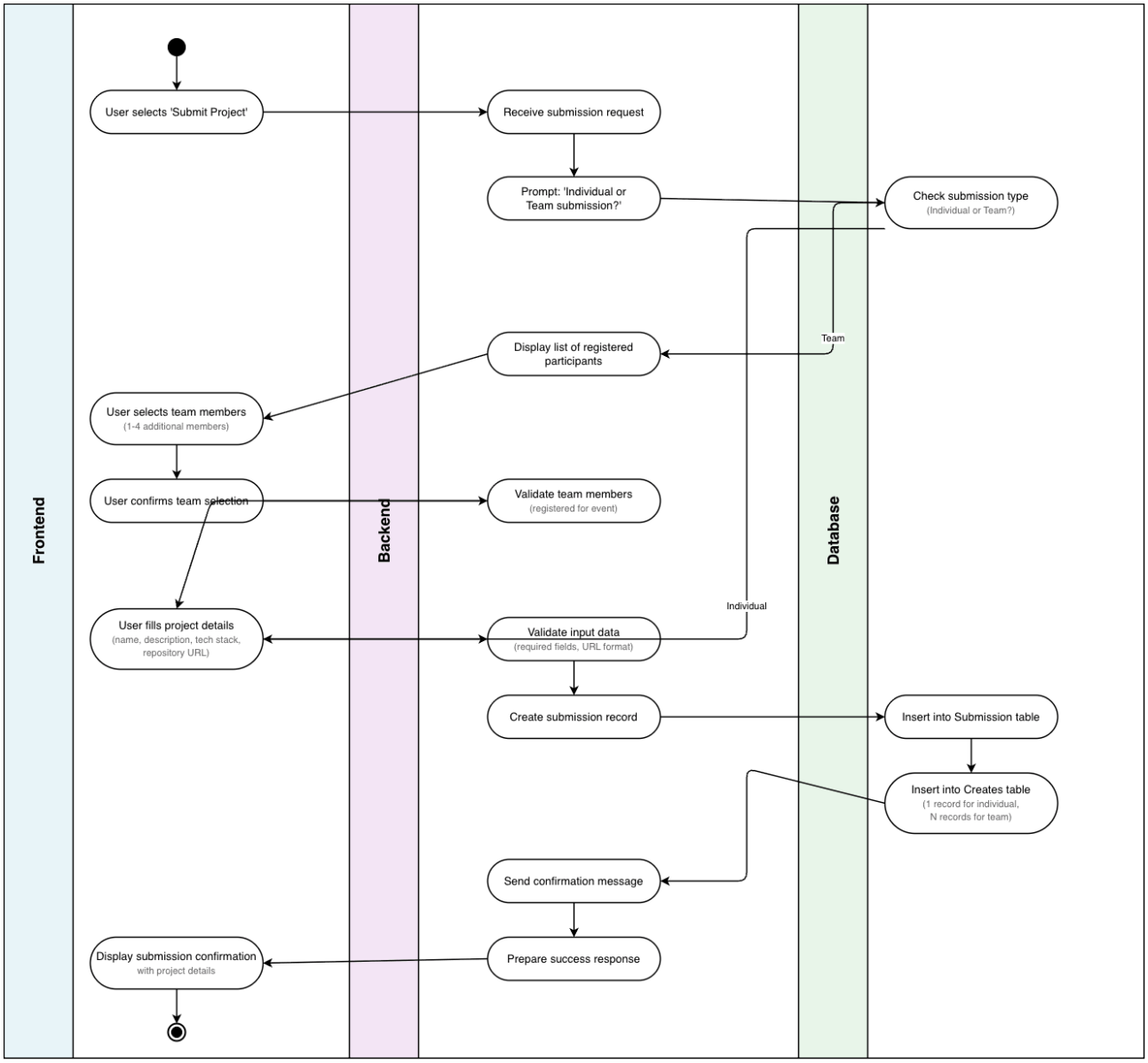


Figure 1: Activity Diagram: Submit Project for Evaluation

1.2.2. Changes Student 2 Individual Part

Based on the feedback I received for MS1, I made the following changes to the individual sections:

Analytics Report Concept (Feedback: "SQL statement missing")

The SQL query statement was missing from the Analytics Report Concept section. The following query has been added to provide the complete analytics report:

```
SELECT
    e.event_id,
    e.name AS event_name,
    e.event_type,
    e.start_date,
    e.end_date,
    e.max_participants,
    v.name AS venue_name,
    v.address AS venue_address,
    v.capacity AS venue_capacity,
    COUNT(r.person_id) AS total_registrations,
    ROUND((COUNT(r.person_id) * 100.0 / e.max_participants), 2)
        AS capacity_percentage,
    SUM(CASE WHEN r.payment_status = 'completed' THEN 1 ELSE 0 END)
        AS paid_registrations,
    SUM(CASE WHEN r.payment_status = 'pending' THEN 1 ELSE 0 END)
        AS pending_payments,
    GROUP_CONCAT(
        CONCAT(p.first_name, ' ', p.last_name, ' (', r.ticket_type, ')')
        SEPARATOR ', '
    ) AS registered_participants
FROM HackathonEvent e
JOIN Venue v ON e.venue_id = v.venue_id
LEFT JOIN Registration r ON e.event_id = r.event_id
LEFT JOIN Person p ON r.person_id = p.person_id
WHERE e.event_type = 'Hackathon'
GROUP BY e.event_id, e.name, e.event_type, e.start_date, e.end_date,
         e.max_participants, v.name, v.address, v.capacity
ORDER BY total_registrations DESC;
```

This query involves five entities (Person, Participant, HackathonEvent, Venue, Registration), uses `event_type` as the filter field, and displays registration statistics including capacity utilization and payment status.

NoSQL Design (Feedback: "Keeps several ids, although only one is necessary")

The original NoSQL design included redundant identifiers in the document structure. Both MongoDB's automatic `_id` field and custom ID fields (e.g., `participant_id`, `event_id`) were present, which is unnecessary.

Before (Redundant IDs):

```
// Participants Collection - OLD
{
    "_id": ObjectId("507f1f77bcf86cd799439012"),
    "participant_id": 5,
    "first_name": "Lisa",
```

```
    "last_name": "Wagner",
    ...
}

// Events Collection - OLD
{
    "_id": ObjectId("507f1f77bcf86cd799439011"),
    "event_id": 1,
    "name": "AI Innovation Hackathon 2025",
    ...
}
```

After (Single ID - using _id as the identifier):

```
// Participants Collection - NEW
{
    "_id": 5,
    "first_name": "Lisa",
    "last_name": "Wagner",
    "email": "lisa.wagner@email.com",
    "phone": "+43 690 5678901",
    "registration_date": ISODate("2024-05-12T00:00:00Z"),
    "t_shirt_size": "M",
    "dietary_restrictions": null,
    "manager_id": 4,
    "manager_name": "David Fischer",
    "event_history": [...]
}

// Events Collection - NEW
{
    "_id": 1,
    "name": "AI Innovation Hackathon 2025",
    "start_date": ISODate("2025-03-15T00:00:00Z"),
    "end_date": ISODate("2025-03-17T00:00:00Z"),
    "event_type": "Hackathon",
    "max_participants": 150,
    "venue": {...},
    "registrations": [...],
    "registration_count": 2
}
```

By using MongoDB's `_id` field directly as the business identifier, we eliminate redundancy while maintaining the same query functionality. References within embedded documents now use `_id` instead of separate ID fields.

2. Milestone 2

2.1. Infrastructure

Describe your infrastructure and provide instructions to run your code.

2.2. RDBMS Implementation

2.2.1. DB Setup / Data Import / Base Function of Web System

Briefly describe the DB setup and how you import the data. Note any changes to the relational model since Milestone 1.

Student 1

2.2.2. Implementation of the Use Case and the Analytics Report

Briefly describe the goal of your use case and report, including how the report's outcome changes after the use case is triggered in the web system. Note any changes to the use case or analytics report since Milestone 1.

Student 2

2.2.2. Implementation of the Use Case and the Analytics Report

Briefly describe the goal of your use case and report, including how the report's outcome changes after the use case is triggered in the web system. Note any changes to the use case or analytics report since Milestone 1.

2.3. NoSQL Implementation

2.3.1. NoSQL Re-Design

See guidelines. Include a copy or screenshot of your collections / schema in the report.

2.3.2. NoSQL DB Setup and Data Migration

Briefly describe the DB setup and data migration process, outlining the steps taken to transfer data from one system to another.

Student 1

2.3.3. NoSQL Analytics Report Statement

Briefly summarize the NoSQL implementation of the use case and report, and compare their results/outcome with those from 2.2.

2.3.4. NoSQL Analytics Report Statement

See guidelines

2.3.5. NoSQL Indexing

See guidelines

Student 2

2.3.3. NoSQL Analytics Report Statement

Briefly summarize the NoSQL implementation of the use case and report, and compare their results/outcome with those from 2.2.

2.3.4. NoSQL Analytics Report Statement

See guidelines

2.3.5. NoSQL Indexing

See guidelines