CacheNCarry 🛍

Where "left behind" meets LEFT JOIN.

CacheNCarry is an end-of-semester storage management system designed to track and manage student belongings (cloakroom luggage + a mattress). The system ensures easy check-in/check-out processes, reduces human error, and provides traceability for checked-in bags and misplaced mattresses.



1. Student Registration:

At the end of each semester, a student registers their belonging(s):

- Belongings can be of two types: luggage or mattress.
- Each item is assigned a unique UUID.
- A QR code is generated for each item and must be printed and physically attached to the respective belonging by the student.

2. Item Check-In:

At the warehouse:

- A staff member scans a QR code on the student's phone to begin a session, associating the session with that student, and logging check-in time.
- The QR codes on each belonging are scanned to link them to the student and register them into the system.
- The staff ends the session once all items have been scanned.

3. Storage Period:

- Belongings remain in the warehouse during the student's absence.
- Mattresses are handled separately: at the start of the semester, they are moved directly to student rooms by university workers.

4. Item Retrieval (Check-Out):

Upon the student's return:

- The student visits the warehouse.
- Staff scan the student's QR code to open a check-out session.
- For each item:
 - **Luggage**: The system validates whether the scanned luggage belongs to the student in session. If ownership does not match, the item cannot be checked out.
 - **Mattress**: Since the mattress is already placed in the student's room, the staff scan both the student and the mattress QR code to validate ownership.

• The staff ends the session once all items have been scanned.

5. Misplacement Handling:

If a scanned mattress does not belong to the student:

- An incident is opened.
- Because all mattresses are uniquely tagged and linked to their rightful owners, it can be easily determined:
 - Who the mattress actually belongs to.
 - Who mistakenly received another student's mattress.
- This allows staff to resolve such incidents efficiently without manual investigation.

% Tech stack

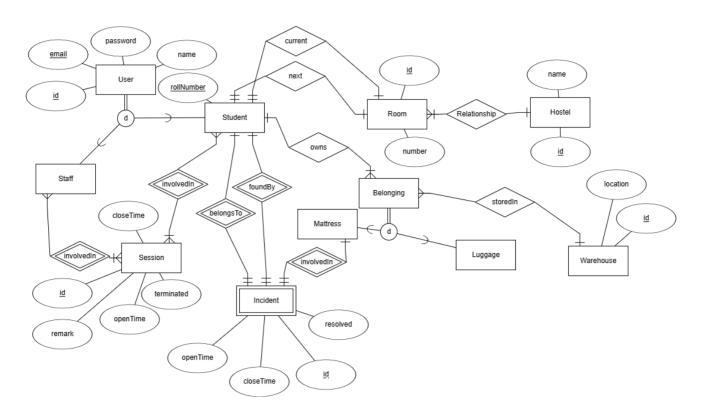
• Frontend & API: SvelteKit + TailwindCSS

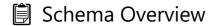
Database: MySQL

• **DB Connector**: Prisma

• Containerization: Docker

Entity Relationship Diagram





Entities

user

All students and staff register as users.

Field	Туре	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY	UUID identification for user
name	VARCHAR(191)		user's real name
email	VARCHAR(191)	UNIQUE	unique organization email id
password	VARCHAR(191)	UNIQUE	hashed password

staff

A staff member oversees sessions and incidents.

Field	Туре	Constraints	Description
	\/ADCHAD(101)	PRIMARY KEY, FOREIGN KEY	UUID identification for staff corressponding
id VARCHAR(191)	(user.id)	to user	

student

A student can register their items and check them in with QR codes attached to them.

Field	Туре	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY, FOREIGN KEY (user.id)	UUID identification for student corressponding to user
roll_number	VARCHAR(12)	UNIQUE	unique roll number
current_room_id	VARCHAR(191)	FOREIGN KEY (room.id)	UUID identification of current occupancy
next_room_id	VARCHAR(191)	FOREIGN KEY (room.id), NULLABLE	UUID identification of next occupancy

hostel

A hostel is a place of residence containing rooms.

Field	Туре	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY	UUID identification for hostel
name	VARCHAR(191)	PRIMARY KEY	name of the hostel

Represents a room within a hostel.

Field	Туре	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY	UUID identification for a room
number	VARCHAR(6)	<pre>UNIQUE per hostel (hostel_id, number)</pre>	hostel room number
hostel_id	VARCHAR(191)	FOREIGN KEY (hostel.id)	UUID identification for the corressponding hostel

warehouse

A warehouse is a physical storage unit for student belingings.

Field	Туре	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY	UUID identification for a warehouse
location	VARCHAR(191)		information about general location

belonging

A belonging is a general entity for storable items.

Field	Туре	Constraints	Description
id	INTEGER	PRIMARY KEY	UUID identification for each belonging
description	VARCHAR(191)	NULLABLE	general description of the item
is_checked_in	BOOLEAN	DEFAULT FALSE	indication of whether the item is checked in
checked_in_at	DATETIME	NULLABLE	timestamp for when the item was checked in
checked_out_at	DATETIME	NULLABLE	timestamp for when the item was checked out
student_id	VARCHAR(191)	FOREIGN KEY (student.id)	UUID identification for the owning student
warehouse_id	VARCHAR(191)	FOREIGN KEY (warehouse.id), NULLABLE	UUID identification for the storage warehouse

luggage

Student luggage that can be checked in and out in direct sessions.

Field	Type	Constraints	Description

Field	Туре	Constraints	Description
÷ d	VADCHAD(101)	PRIMARY KEY, FOREIGN KEY	LILITD identification for each bag
id	VARCHAR(191)	(belonging.id)	UUID identification for each bag

mattress

A mattress can be stored over the semester break, and gets transported to a student's room before they arrive. An ownership discrepancy leads to an incident.

Field	Туре	Constraints	Description
	\/ADCHAD(101)	PRIMARY KEY, FOREIGN KEY	UUID identification for a
id	VARCHAR(191)	(belonging.id)	mattress

session

Represents a warehouse interaction involving a student and staff member.

Field	Туре	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY	UUID identification for each session
remark	VARCHAR(191)	NULLABLE	Optional notes about the session
open_time	DATETIME	DEFAULT NOW()	timestamp for when the session started.
close_time	DATETIME	NULLABLE	timestamp for when the session ended.
terminated	BOOLEAN	DEFAULT FALSE	indication of whether the session was manually terminated.
staff_id	VARCHAR(191)	FOREIGN KEY (staff.id)	UUID identification for the staff member managing the session
student_id	VARCHAR(191)	FOREIGN KEY (student.id), UNIQUE per staff (staff_id, student_id)	UUID identification for the student who owns the belongings

incident

Represents misplacement or ownership conflict involving mattresses.

Field	Туре	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY	UUID identification for each incident.

Field	Туре	Constraints	Description	
mattress_id	VARCHAR(191)	FOREIGN KEY (mattress.id)	UUID identification for the mattress involved in the incident.	
found_by	VARCHAR(191)	FOREIGN KEY (student.id)	UUID identification for the student who discovered the misplaced mattress.	
belongs_to	VARCHAR(191)	FOREIGN KEY (student.id)	UUID identification for the student to whom the mattress actually belongs.	
resolved	BOOLEAN	DEFAULT FALSE	indication of whether the incident has been resolved.	
open_time	DATETIME	DEFAULT NOW()	timestamp for when the incident was reported.	
close_time	DATETIME	NULLABLE	timestamp for when the incident was resolved.	

Constraints

Table	Foreign Key	References	On Delete	On Update
staff	id	user(id)	CASCADE	CASCADE
student	id	user(id)	CASCADE	CASCADE
student	current_room_id	room(id)	RESTRICT	CASCADE
student	next_room_id	room(id)	SET NULL	CASCADE
room	hostel_id	hostel(id)	CASCADE	CASCADE
belonging	student_id	student(id)	RESTRICT	CASCADE
belonging	warehouse_id	warehouse(id)	SET NULL	CASCADE
luggage	id	belonging(id)	CASCADE	CASCADE
mattress	id	belonging(id)	CASCADE	CASCADE
session	staff_id	staff(id)	RESTRICT	CASCADE
session	student_id	student(id)	RESTRICT	CASCADE
incident	mattress_id	mattress(id)	RESTRICT	CASCADE
incident	found_by	student(id)	RESTRICT	CASCADE
incident	belongs_to	student(id)	RESTRICT	CASCADE

& Database Normalization

The following is a concise normalization analysis for each table:

user

```
CREATE TABLE `user` (
   `id` VARCHAR(191) NOT NULL,
   `name` VARCHAR(191) NOT NULL,
   `email` VARCHAR(191) NOT NULL,
   `password` VARCHAR(191) NOT NULL,

UNIQUE INDEX `user_email_key`(`email`),
   PRIMARY KEY (`id`)
)
```

- **1NF**: All attributes atomic
- 2NF: All non-key attributes depend on full PRIMARY KEY (id)
- **3NF**: No transitive dependencies

staff

```
CREATE TABLE `staff` (
   `id` VARCHAR(191) NOT NULL,

PRIMARY KEY (`id`)
)
```

- 1NF: Minimal table structure
- 2NF: Inherits PRIMARY KEY from user (no partial dependencies)
- **3NF**: No non-key dependencies

student

```
CREATE TABLE `student` (
    `id` VARCHAR(191) NOT NULL,
    `rollNumber` VARCHAR(12) NOT NULL,
    `current_room_id` VARCHAR(191) NOT NULL,
    `next_room_id` VARCHAR(191) NULL,

UNIQUE INDEX `student_rollNumber_key`(`rollNumber`),
    UNIQUE INDEX `student_current_room_id_key`(`current_room_id`),
    UNIQUE INDEX `student_next_room_id_key`(`next_room_id`),
    PRIMARY KEY (`id`)
)
```

- 1NF: Atomic values
- 2NF: roll number depends on full PRIMARY KEY
- **3NF**: noom references are ID-only (no transitive dependencies)

hostel

```
CREATE TABLE `hostel` (
   `id` VARCHAR(191) NOT NULL,
   `name` VARCHAR(191) NOT NULL,

PRIMARY KEY (`id`)
)
```

- **1NF**: Simple structure
- 2NF: Single attribute depends on PRIMARY KEY
- **3NF**: No non-key dependencies

room

```
CREATE TABLE `room` (
    `id` VARCHAR(191) NOT NULL,
    `number` VARCHAR(6) NOT NULL,
    `hostel_id` VARCHAR(191) NOT NULL,

UNIQUE INDEX `room_hostel_id_number_key`(`hostel_id`, `number`),
    PRIMARY KEY (`id`)
)
```

- **1NF**: Proper identifiers
- 2NF: Composite unique key (hostel_id + number) depends on PRIMARY KEY
- **3NF**: No derived attributes

warehouse

```
CREATE TABLE `warehouse` (
   `id` VARCHAR(191) NOT NULL,
   `location` VARCHAR(191) NOT NULL,

PRIMARY KEY (`id`)
)
```

- 1NF: Basic table
- 2NF: Single non-key attribute
- **3NF**: No dependencies

belonging

```
CREATE TABLE `belonging` (
   `id` VARCHAR(191) NOT NULL,
   `description` VARCHAR(191) NULL,
   `is_checked_in` BOOLEAN NOT NULL DEFAULT false,
```

```
`student_id` VARCHAR(191) NOT NULL,
  `warehouse_id` VARCHAR(191) NULL,
  `checked_in_at` DATETIME(3) NULL,
  `checked_out_at` DATETIME(3) NULL,
  PRIMARY KEY (`id`)
)
```

- **1NF**: Atomic fields
- 2NF: All attributes depend on PRIMARY KEY
- **3NF**: warehouse reference is ID-only

luggage/mattress

```
CREATE TABLE `luggage` (
   `id` VARCHAR(191) NOT NULL,

PRIMARY KEY (`id`)
)

CREATE TABLE `mattress` (
   `id` VARCHAR(191) NOT NULL,

PRIMARY KEY (`id`)
)
```

- 1NF: Inherited PRIMARY KEY
- 2NF: No additional attributes
- 3NF: Inherited normalization from belonging

session

```
CREATE TABLE `session` (
   `id` VARCHAR(191) NOT NULL,
   `remark` VARCHAR(191) NULL,
   `open_time` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
   `close_time` DATETIME(3) NULL,
   `terminated` BOOLEAN NOT NULL DEFAULT false,
   `staff_id` VARCHAR(191) NOT NULL,
   `student_id` VARCHAR(191) NOT NULL,

UNIQUE INDEX `session_staff_id_student_id_key`(`staff_id`, `student_id`),
   PRIMARY KEY (`id`)
)
```

- **1NF**: Proper structure
- 2NF: All attributes depend on PRIMARY KEY

• 3NF: staff/student references are ID-only

incident

```
CREATE TABLE `incident` (
    `id` VARCHAR(191) NOT NULL,
    `found_by` VARCHAR(191) NOT NULL,
    `belongs_to` VARCHAR(191) NOT NULL,
    `mattress_id` VARCHAR(191) NOT NULL,
    `resolved` BOOLEAN NOT NULL DEFAULT false,
    `open_time` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    `close_time` DATETIME(3) NULL,

UNIQUE INDEX `incident_found_by_key`(`found_by`),
    UNIQUE INDEX `incident_belongs_to_key`(`belongs_to`),
    UNIQUE INDEX `incident_mattress_id_key`(`mattress_id`),
    PRIMARY KEY (`id`)
)
```

- 1NF: Atomic values
- 2NF: All fields relate to PRIMARY KEY
- 3NF: mattress/student references are ID-only

Trigger Deep Dive

belonging_checkin_log

Logic:

- WHEN: is_checked_in changes false → true
- WHAT: Sets checked_in_at to current timestamp
- WHY: Logging check-in request time

belonging_checkout_log

```
CREATE TRIGGER belonging_checkout_log

BEFORE UPDATE ON belonging
```

Logic:

- WHEN: is_checked_in changes true → false
- WHAT: Sets checked_out_at to current timestamp
- WHY: Logging check-out request time

session_log

```
CREATE TRIGGER session_log

BEFORE UPDATE ON session

FOR EACH ROW

BEGIN

IF OLD.terminated = FALSE AND NEW.terminated = TRUE THEN

SET NEW.close_time = NOW();

END IF;

END;
```

Logic:

- WHEN: terminated changes false → true
- WHAT: Sets close_time to current timestamp
- WHY: Logging session close time

incident_log

```
CREATE TRIGGER incident_log

BEFORE UPDATE ON incident

FOR EACH ROW

BEGIN

IF OLD.resolved = FALSE AND NEW.resolved = TRUE THEN

SET NEW.close_time = NOW();

END IF;

END;
```

Logic:

- WHEN: resolved changes false → true
- WHAT: Sets close_time to current timestamp
- WHY: Logging incident close time

? Query Examples

Finds the staff ID associated with the provided staff email address.

```
SELECT s.id
FROM user u
JOIN staff s ON u.id = s.id
WHERE LOWER(TRIM(u.email)) = ${trimmedStaffEmail}
LIMIT 1
```

Finds the warehouse ID based on the provided warehouse name.

```
SELECT id
FROM warehouse
WHERE location = ${trimmedWarehouseName}
LIMIT 1
```

Retrieves the student ID, check-in status, and current warehouse ID for a specific belonging ID.

```
SELECT id, student_id, is_checked_in, warehouse_id
FROM belonging
WHERE id = ${belongingId}
LIMIT 1
```

Checks if an active, non-terminated session exists between the specified staff and student IDs.

```
SELECT id
FROM session
WHERE staff_id = ${staffId}
  AND student_id = ${studentId}
  AND close_time IS NULL
  AND `terminated` = false
LIMIT 1
```

Updates a specific belonging to mark it as checked-in and assign it to the specified warehouse.

```
UPDATE belonging
SET is_checked_in = true,
    warehouse_id = ${warehouseId}
WHERE id = ${belongingId}
```

Selects the complete, updated record details for the specified belonging ID after the update.

```
SELECT
   id,
   description,
   is_checked_in,
   student_id,
   warehouse_id,
   checked_in_at,
   checked_out_at
FROM belonging
WHERE id = ${belongingId}
LIMIT 1
```

Retrieves belonging details, owner ID, check-in status, and checks if it's a mattress by joining with the mattress table.

```
SELECT
    b.id,
    b.student_id,
    b.is_checked_in,
    b.checked_out_at,
    m.id as mattress_id
FROM belonging b
LEFT JOIN mattress m ON b.id = m.id
WHERE b.id = ${belongingId}
LIMIT 1
```

Finds an active, non-terminated session associated with the specific staff member and retrieves the session ID and the student ID involved.

```
SELECT id, student_id
FROM session
WHERE staff_id = ${staffId}
AND close_time IS NULL
AND `terminated` = false
LIMIT 1
```

Checks for any existing unresolved incidents associated with the specific mattress ID.

```
SELECT id FROM incident
WHERE mattress_id = ${mattressId} AND resolved = false
LIMIT 1
```

Creates a new incident record if a mattress is checked out by someone other than its owner.

```
INSERT INTO incident (id, found_by, belongs_to, mattress_id)
VALUES (${newIncidentId}, ${sessionStudentId}, ${actualOwnerStudentId},
${mattressId})
```

Updates the specified belonging to mark it as checked out, set the checkout time, and clear the warehouse ID.

```
UPDATE belonging
SET is_checked_in = false,
    checked_out_at = NOW(),
    warehouse_id = NULL
WHERE id = ${belongingId}
```

Selects the updated state (ID, check-in status, checkout time, owner ID) of the belonging after the update.

```
SELECT id, is_checked_in, checked_out_at, student_id
FROM belonging
WHERE id = ${belongingId}
LIMIT 1
```

Closes the active session by setting the close time and adding a remark indicating the belonging checkout.

```
UPDATE session
SET close_time = NOW(),
    remark = ${remark}
WHERE id = ${closedSessionId}
```

Finds the Student ID based on the provided roll number.

```
SELECT id FROM student WHERE roll_number = ${trimmedRollNumber} LIMIT 1
```

Inserts a new belonging record with its ID, description, and associated student ID.

```
INSERT INTO belonging (id, description, student_id)
VALUES (${newBelongingId}, ${trimmedDescription}, ${studentId})
```

Inserts a corresponding record into the luggage table using the belonging ID.

```
INSERT INTO luggage (id) VALUES (${newBelongingId})
```

Inserts a corresponding record into the mattress table using the belonging ID.

```
INSERT INTO mattress (id) VALUES (${newBelongingId})
```

Selects the complete details of the newly created belonging, including related luggage/mattress/warehouse info.

```
SELECT
    b.id, b.description, b.is_checked_in, b.student_id, b.warehouse_id,
    b.checked_in_at, b.checked_out_at,
    l.id as luggage_id,
    m.id as mattress_id,
    w.location as warehouse_location
FROM belonging b
LEFT JOIN luggage 1 ON b.id = l.id
LEFT JOIN mattress m ON b.id = m.id
LEFT JOIN warehouse w ON b.warehouse_id = w.id
WHERE b.id = ${newBelongingId}}
LIMIT 1
```

Fetches all belongings for a given student ID, joining related luggage, mattress, and warehouse data.

```
SELECT
    b.id, b.description, b.is_checked_in, b.student_id, b.warehouse_id,
    b.checked_in_at, b.checked_out_at,
    l.id as luggage_id,
    m.id as mattress_id,
    w.location as warehouse_location
FROM belonging b
LEFT JOIN luggage 1 ON b.id = l.id
LEFT JOIN mattress m ON b.id = m.id
LEFT JOIN warehouse w ON b.warehouse_id = w.id
WHERE b.student_id = ${studentId}
ORDER BY b.checked_in_at ASC
```

Finds the User ID associated with the provided staff email address.

```
SELECT id FROM user WHERE email = ${trimmedStaffEmail} LIMIT 1
```

Verifies that the User ID found corresponds to an actual staff member.

```
SELECT id FROM staff WHERE id = ${userId} LIMIT 1
```

Inserts a new session record linking the provided staff and student IDs.

```
INSERT INTO session (id, staff_id, student_id)
VALUES (${newSessionId}, ${staffId}, ${studentId})
```

Selects the complete details of the newly created session record.

```
SELECT id, remark, open_time, close_time, `terminated`, staff_id, student_id
FROM session
WHERE id = ${newSessionId}
LIMIT 1
```

Deletes any active, non-terminated sessions matching the specific student (by roll number) and staff (by email).

```
DELETE FROM session
WHERE
    close_time IS NULL
    AND `terminated` = false
    AND student_id = (
        SELECT id
        FROM student
        WHERE roll_number = ${trimmedRollNumber}
        LIMIT 1
    AND staffid = (
        SELECT s.id
        FROM user u
        JOIN staff s ON u.id = s.id
        WHERE u.email = ${trimmedStaffEmail}
        LIMIT 1
    )
```