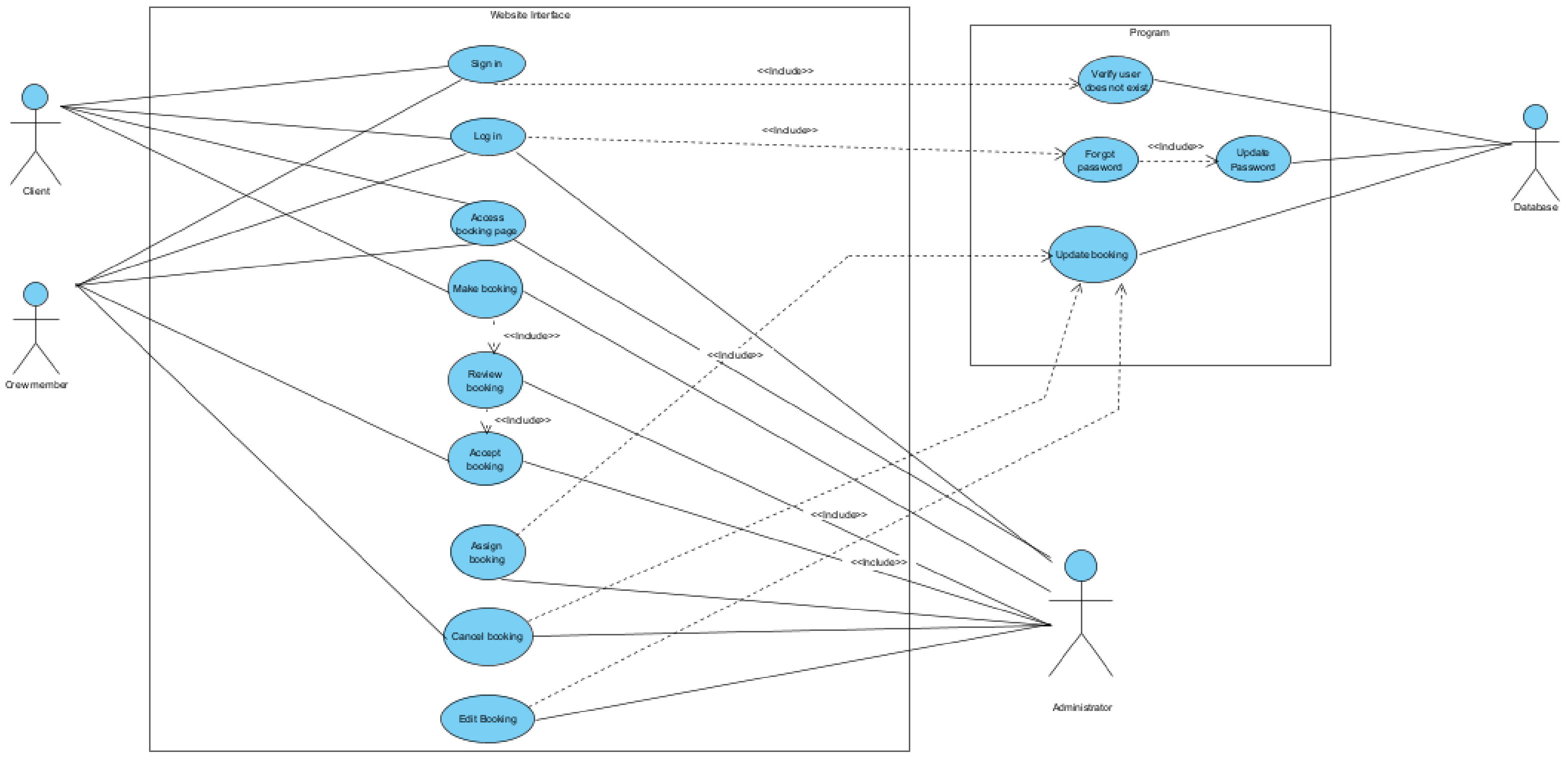
05.06.2022

Report on UML Diagrams and SQL Schema

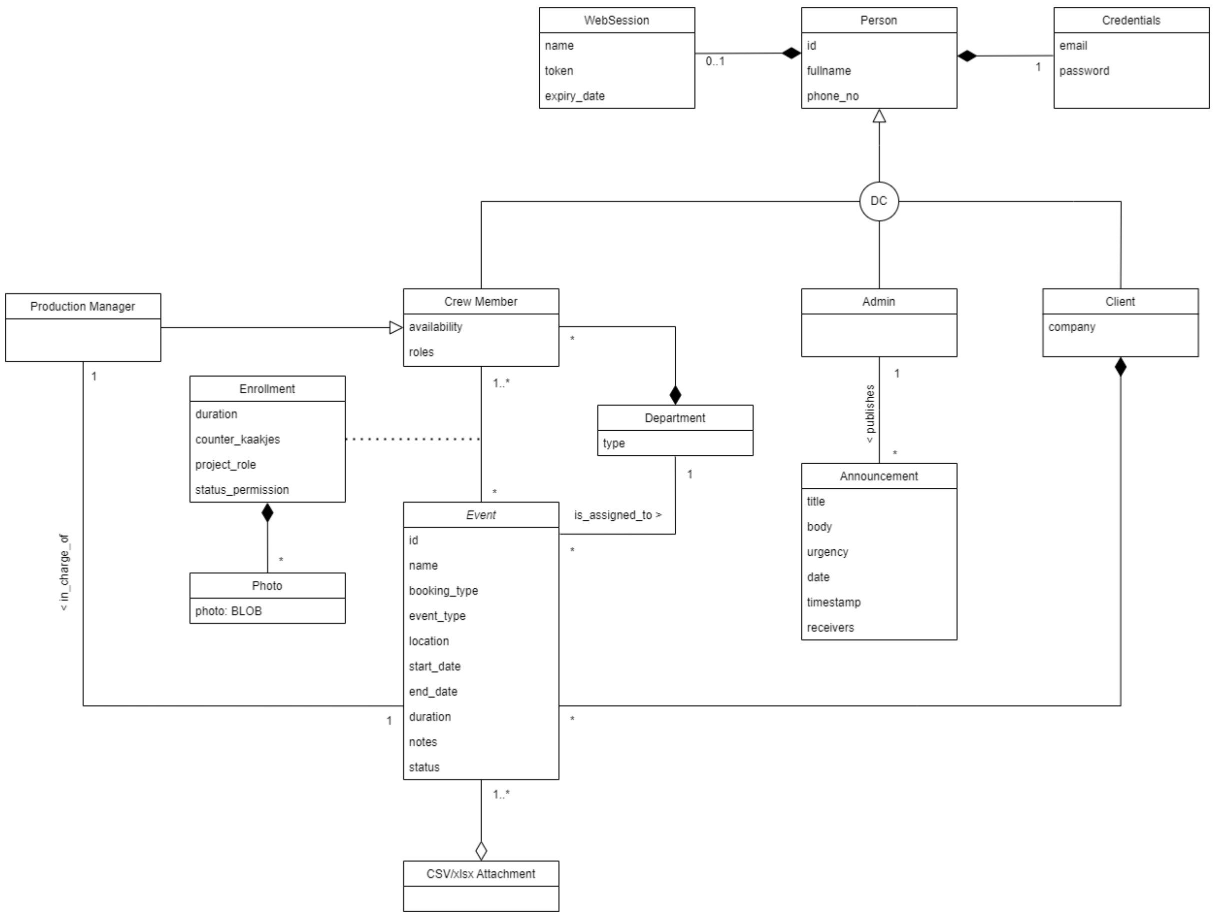
By Erick Divin, Serhii Lysin, Alexey Kovzel, Andreea Goga,
Marijke van Iperen, Tymur Astashov

UML Use Case Diagram



Our use case diagram focuses on the interactions between our users (crew, client, administrator) and the system which is represented by the database. It outlines the relationships between them and the ways they communicate with each other. Since the main purpose of the use case diagram is to provide a high level overview of the system, our diagram aims to do just that as it is quite simple and easy to understand.

UML Class Diagram



Our class diagram aims to visualize the system that we are making. Therefore, the structure of our project can be observed there. Attributes in classes correspond to those in the database. We decided to avoid making the diagram too crowded as it would hinder the understanding and would not provide any additional value. That is why only the most crucial information was included.

SQL Schema

```
STATUS(
    status_type CHAR(100) NOT NULL UNIQUE,
    PK (status_type)
);

STATUS_PERMISSION(
    status_type INTEGER NOT NULL UNIQUE,
    PK (status_type)
);

BOOKING(
    booking_type CHAR(100) NOT NULL UNIQUE,
    PK (booking_type)
);


EVENT_TYPE(
      event_type CHAR(100) NOT NULL UNIQUE,
      PK (event_type)
);


DEPARTMENT(
      department_type CHAR(100) NOT NULL UNIQUE,
      PK (department_type)
);

ROLE(
      role_type CHAR(100) NOT NULL UNIQUE,
      PK(role_type)
);


AVAILABILITY(
      availability_type CHAR(100) NOT NULL UNIQUE,
      PK (availability_type)
);
```

These first classes will function like an enum, they will have fixed data, that will probably never change. They have the fixed data such that the system doesn’t depend on if you type well or not. Those classes will support the other classes.

```
PERSON(
      id INTEGER NOT NULL UNIQUE,
      email CHAR(100),
      full_name CHAR(100) NOT NULL,
      phone_no CHAR(100),
      password CHAR(100),
      PK (id)
);
```
The person class will represent the user in the database, this a general user also one without an account.

```
WEB_SESSION(
      name CHAR(100) NOT NULL UNIQUE,
      token CHAR(100) NOT NULL,
      expiry_date TIMESTAMP NOT NULL,
      id INTEGER NOT NULL,
      PK (name),
      FK (id) REF PERSON(id),
      CHECK(id IN (SELECT id FROM PERSON))
);
```

The WEB_SESSION class makes the cookie management happen with expire_date, such that a user doesn't stay too long in one session.

```
CLIENT(
 company CHAR(100) NOT NULL,
 id INTEGER NOT NULL,
 department_type CHAR(100) NOT NULL,
 PK (id),
 FK (id) REF PERSON(id),
 CHECK(id IN (SELECT id FROM PERSON)),
 CHECK(id NOT IN (SELECT id FROM CREW_MEMBER))
);


ADMINS(
 id INTEGER NOT NULL UNIQUE,
 PK (id),
 FK (id) REF PERSON(id),
 CHECK(id IN (SELECT id FROM PERSON)),
 CHECK(id IN (SELECT id FROM CREW_MEMBER)),
 CHECK(id NOT IN (SELECT id FROM CLIENT))
);


CREW_MEMBER(
 id INTEGER NOT NULL UNIQUE,
 avaibilability CHAR(100),
 role CHAR(100),
 department_type CHAR(100),
 PK (id),
 FK (id) REF PERSON(id),
 FK (role) REF ROLE(role_type),
 FK (avaibilability) REF AVAILABILITY(availability_type),
 FK (department_type) REF DEPARTMENT(department_type),
 CHECK(id IN (SELECT id FROM PERSON)),
 CHECK(id NOT IN (SELECT id FROM CLIENT))
);

PRODUCTION_MANAGER(
 id INTEGER NOT NULL UNIQUE,
 PK (id),
 FK (id) REF PERSON(id),
 CHECK(id IN (SELECT id FROM CREW_MEMBER))
);
```
Classes client, admin, crew_member and production_manager represent the users with their different permissions.

```
EVENT(
    id INTEGER NOT NULL UNIQUE,
    name CHAR(100) NOT NULL,
      location CHAR(100),
    start_date DATE,
    end_date DATE,
    duration INTEGER,
    notes CHAR(400),
    number_of_crews INTEGER,
    product_manager INTEGER,
    client INTEGER NOT NULL,
    booking CHAR(100),
    status CHAR(100),
      event_type CHAR(100),
      department_type CHAR(100),
    PK(id),
    FK(product_manager) REF PRODUCTION_MANAGER(id),
/*Here to add a constraints that the productManager has the role crew*/
    FK(client) REF CLIENT(id),
    FK(booking) REF BOOKING(booking_type),
    FK(status) REF STATUS(status_type),
      FK(event_type) REF EVENT_TYPE(event_type),
      FK (department_type) REF DEPARTMENT(department_type),
      CHECK(start_date<end_date),
      CHECK(duration<= end_date - start_date),
      CHECK(client IN (SELECT id FROM CLIENT)),
      CHECK(product_manager IN (SELECT id FROM PRODUCTION_MANAGER))
);
```

This class brings everything together and represents the event.

```
ENROLMENT(
duration INTEGER,
counter_kaakjes INTEGER,
role CHAR(100),
status_permission INTEGER NOT NULL,
crew INTEGER,
event INTEGER NOT NULL UNIQUE,
PK(crew, event),
FK(event) REF EVENT(id),
FK(crew) REF CREW_MEMBER(id),
FK (role) REF ROLE(role_type),
FK (status_permission) REF STATUS_PERMISSION(status_type)
);
```

This class brings the crew and the event together.

```
ANNOUNCEMENT(
      id INTEGER NOT NULL UNIQUE,
      title CHAR(100),
      message VARCHAR,
      urgency CHAR(100),
      moment TIMESTAMP,
      admins INTEGER NOT NULL,
      receive INTEGER,
      PK (id),
      FK (admins) REF ADMINS(id),
      FK (receive) REF CREW_MEMBER(id)
);
ANNOUNCEMENT_DEPARTMENT(
      id INTEGER NOT NULL UNIQUE,
      title CHAR(100),
      message VARCHAR,
      urgency CHAR(100),
      moment TIMESTAMP,
      admins INTEGER NOT NULL,
      receive CHAR(100) NOT NULL,
      PK (id),
      FK (admins) REF ADMINS(id),
      FK (receive) REF DEPARTMENT(department_type)
);
```
These two classes store the announcement and make sure the right people have permission.

```
PHOTO(
      id INTEGER NOT NULL UNIQUE,
      event INTEGER NOT NULL,
      crew INTEGER NOT NULL,
      photo bytea,
      moment TIMESTAMP,
      PK (id),
      FK(event,crew ) REF ENROLMENT(event,crew),
);
```
This class stores all images for all clients.