Designreport Shotmaniacs Team 1

A picture containing sketch, drawing, origami, design

Description automatically generatedThis report is about the class diagram, use case diagram and database schema of the current implementation that will be presented in the presentation on May 31, 2023, and some of the changes that are expected to be implemented in the next sprint(s).

Table of Contents

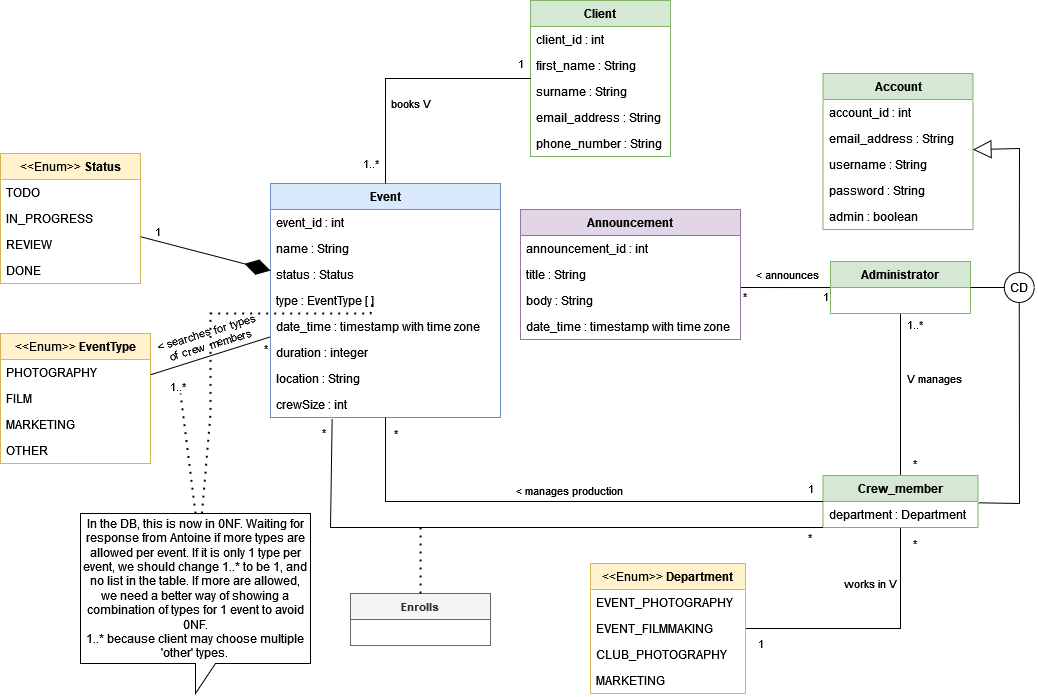
[Class Diagram 3](#_Toc136398614)

[Use Case Diagram 5](#_Toc136398615)

[Database Schema 6](#_Toc136398616)

# Class Diagram

The image underneath is the current implementation of the class diagram. Colors are used to group up certain objects; actors have a green tint and Enums have an orange tint.

The table Account used to be called User. When creating the DB, SQL did not allow ‘User’ as a valid table name, as it is a reserved word. As a result, to be certain that there is no confusion between the schema and class diagram, the User class is now renamed to Account to be consistent with the schema. This is an explanation as to why human actors are subclasses of an Account class, instead of using composition to show that the actors must have an account, it is just the naming of the class that has changed.

There are a few things that are subject to change about this class diagram. There are questions about this noted in Trello, which will be sent to Serkan (TA) to forward to Antoine, the representative of Shotmaniac. These changes can then be adapted in the third sprint. Other changes are already known, yet purposely not implemented.

Starting with the bottom left textbox. It is unclear if a booked event can have multiple types, or simply one type per event. Logically, a client would want more types per event, so that is how the current implementation is. However, this comes with its issue of normalizing the table, as there are infinite combinations of types possible. This may be solved with an association class, though there is no value in thinking of solutions, as we are waiting for an answer first. For this sprint, the minimum functionality is acceptable.

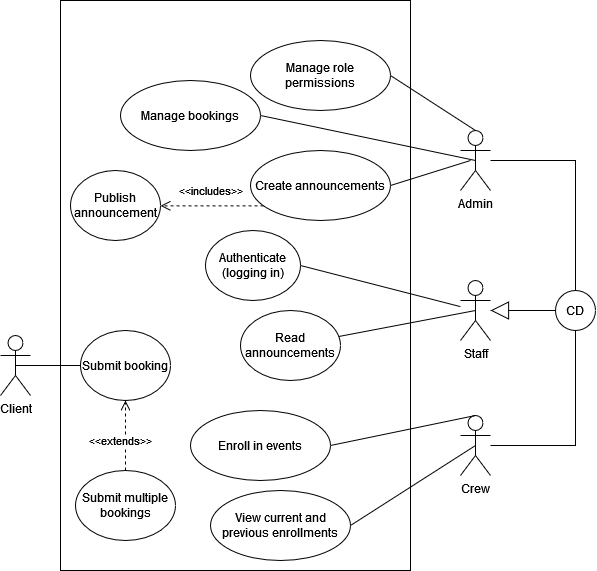
Something that comes close to that is the permission levels stated in Antoine’s previous answer to one of the questions in Discord. The overhanging question there is how the permission level is determined and if it is per department or differentiated within a single department, as well as how the event type is related to these. This could mean an addition of a class ‘Permission\_level’and where it would have to attach to: either Department or CrewMember.

The class Enrolls will also need a revision because this implementation does not allow crew members to enrol a second time after cancelling once. For now, this should be OK to show the main objective of the booking system. The necessary change would be to stop using an association class, but instead connect Enrolls to both Crew\_member and Event. The line for the production manager will however stay, as nothing is wrong with that.

Lastly, a change that is yet to be implemented is related to the account system in the class diagram and its variables. This diagram does not allow for clients to have an account, but instead only have an id. When a booking is made, the client’s authentication is the client ID, which will be shown to them. This is not secure, as some random person may cancel someone else’s events this way. Since it is not a main functionality of the booking system, this function is moved to the next sprints.

What will change in terms of the class diagram, is that there will be a generalization between Client and Account. The variable ‘admin : boolean’ now acts to differentiate administrators and crew members. With three subclasses, this creates a new problem. More on how this change affects the rest of the system’s back-end is mentioned below the database schema.

# Use Case Diagram



The use case diagram is very minimalistic, yet shows more than what will be possible in the prototype. Announcements are currently not the priority. In this sprint, it should be possible for staff to log in, clients to submit (at least) 1 booking for an event, and the crew to enrol in these. In the next sprints, the goal is to be able to connect the line from the client to ‘authenticate (logging in)’.  
On the right side, it is visible that the crew members and administrators are CD-generalized in staff. This is done to avoid too many added connecting lines in the box.

# Database Schema

This sprint’s database schema purely focuses on the main function of creating a booking of an event. That means that there are some challenges for the next sprints to improve the database further.

Event(

event\_id PK,

client\_id NOT NULL,

name,

type,

date\_time,

duration,

location,

production\_manager\_id NOT NULL,

crew\_size,

status,

FK(client\_id) REF Client(client\_id)

FK(production\_manager\_id) REF Crew\_member(user\_id)

);

Account(

account\_id PK,

username,

password,

email\_address,

admin

);

Crew\_member(

user\_id PK,

department,

FK(user\_id) REF Account(account\_id)

);

Administrator(

user\_id int PK,

FK(user\_id) REF Account(account\_id)

);

Announcement(

announcement\_id PK,

announcer NOT NULL,

title,

body,

date\_time,

FK(announcer) REF Administrator(user\_id)

);

Client(

client\_id PK,

first\_name,

last\_name,

email,

phone\_number

CHECK(client\_id IN (  
 SELECT client\_id

FROM Event

)

)

);

Enrolls(

event\_id,

crew\_id,

PK(event\_id, crew\_id),

FK(event\_id) REF Event(event\_id),

FK(crew\_id) REF Crew\_member(user\_id)

);

//For simplicity of the schema, Enums and sequences for id generations are left out, but they have been used in the creation of the database.

With a database using this structure, a client has no account and their information is not stored unless they create a booking. This becomes problematic when the same client books several events: a row is added to the Event table with a client\_id, and that id references a row in the Client table that will be added as well. This issue is that for every event a new client\_id is created and a row is stored that is, in reality, the same client. This is the main reason having an account for clients will be incredibly beneficial, as that singlehandedly solves this normalization issue **and**it has added authentication benefits.

Another issue with accounts is the fact that there is redundancy because of the covering and disjoint generalizations. The most efficient method is to delete the Account table altogether and add the necessary attributes (id, email, password, username) to each of the subclasses. The attribute ‘admin : boolean’ can be discarded and the column for the id is not repeated anymore in the sub -and superclasses.

Combining these two improvements should make it possible to have accounts for clients. The challenge that then arises is in the design of the login screen: How would the query know in which table to look for the login credentials? If a crew member logs in, it must be going to a different page than if a client or admin would log in. A possible solution is to have separate login screens, which come at a cost in the workload for the implementation, as that would mean at least two more pages to program.   
How this issue is tackled, is a problem for the next sprints.