Workshop 2

Design Artifacts and System Modeling

Students:

Juan David Castaneda Cardenas
Laura Vanesa Alvarez Lafont
Nicolas Andres Bolanos Fernandez
Santiago Andres Fetecua Pulgarin
Sergio Nicolas Correa Escobar

Professor:

Carlos Andres Sierra Virguez

Software Engineering II
School of Engineering
Universidad Nacional de Colombia

1. CRC Cards

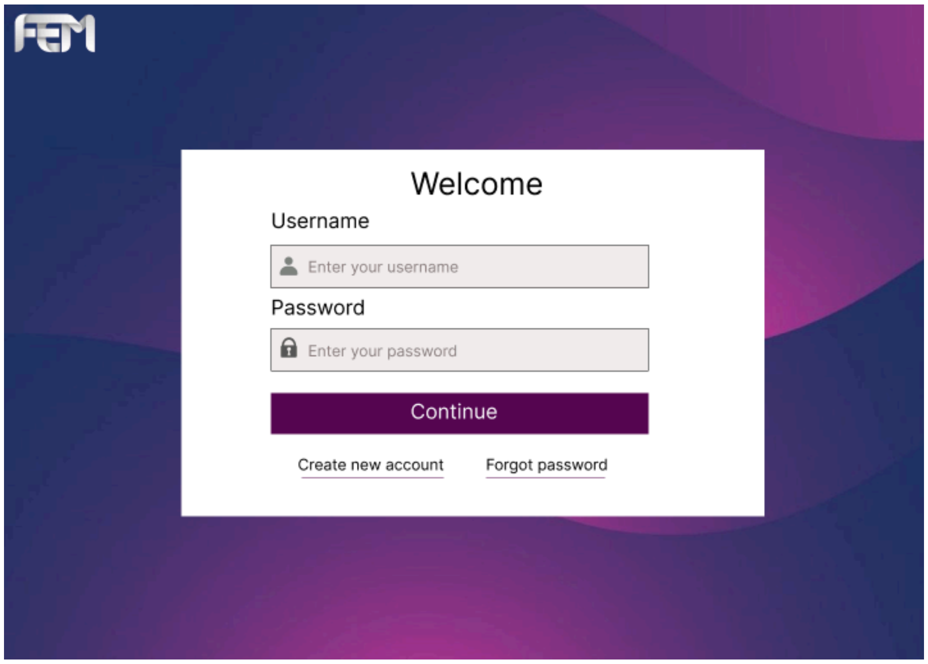
USER	
Responsibilities	Collaboration
Create EVENTS	EVENTS
Share EVENTS	
Delete EVENTS	
Knows password	
Can change password	
Knows name	
Knows user name	
Knows email	

EVENTS	
Responsibilities	Collaboration
Knows name	ATTENDANCE
Knows Date	
Knows Time	
Knows Location	
Knows the attendance capacity	
Provides a shareable link	
Knows the number of confirmed ATTENDANCE	

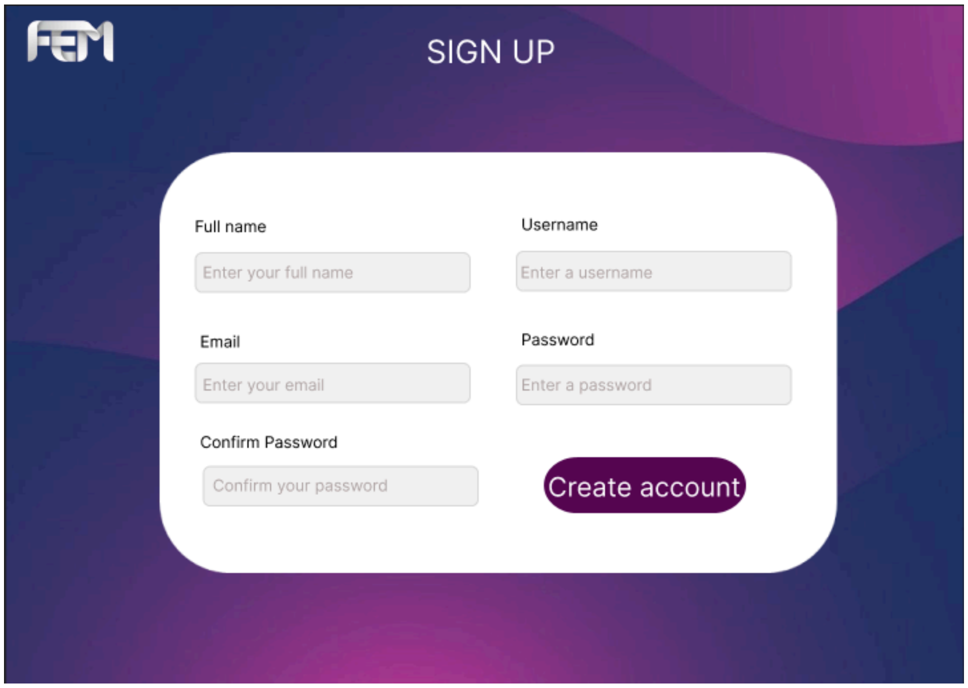
ATTENDANCE	
Responsibilities	Collaboration
Knows name	EVENTS
Knows email	
Knows contac number	
Knows Document ID	
Can confirm their attendance at an event	

These cards show the principal classes of the project, their responsibilities and the way in which they collaborate.

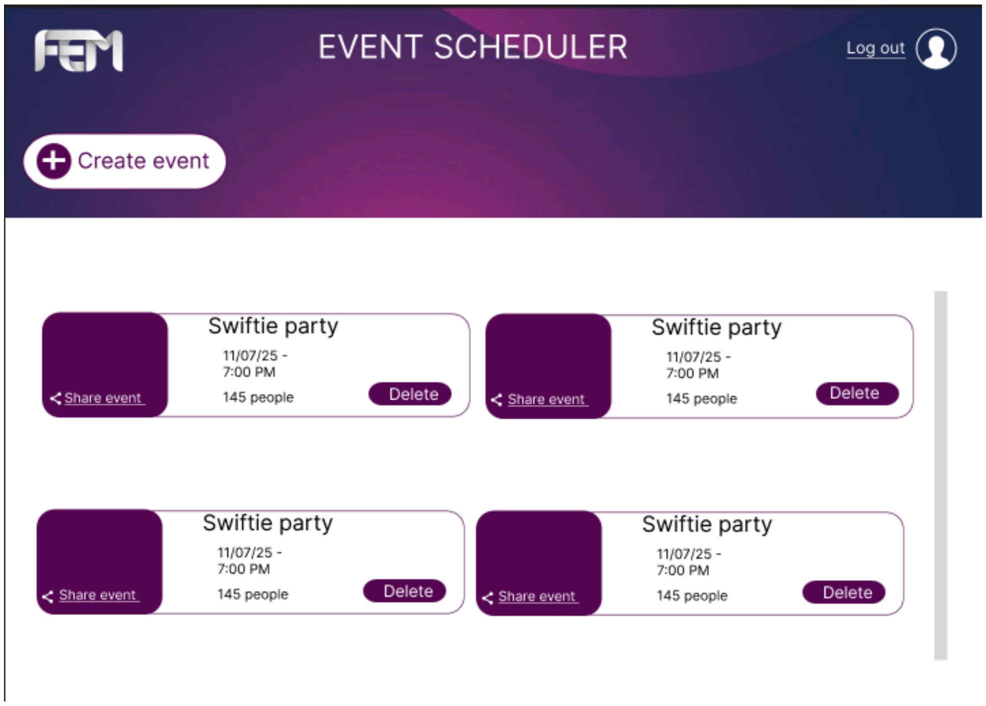
2. Mock Up



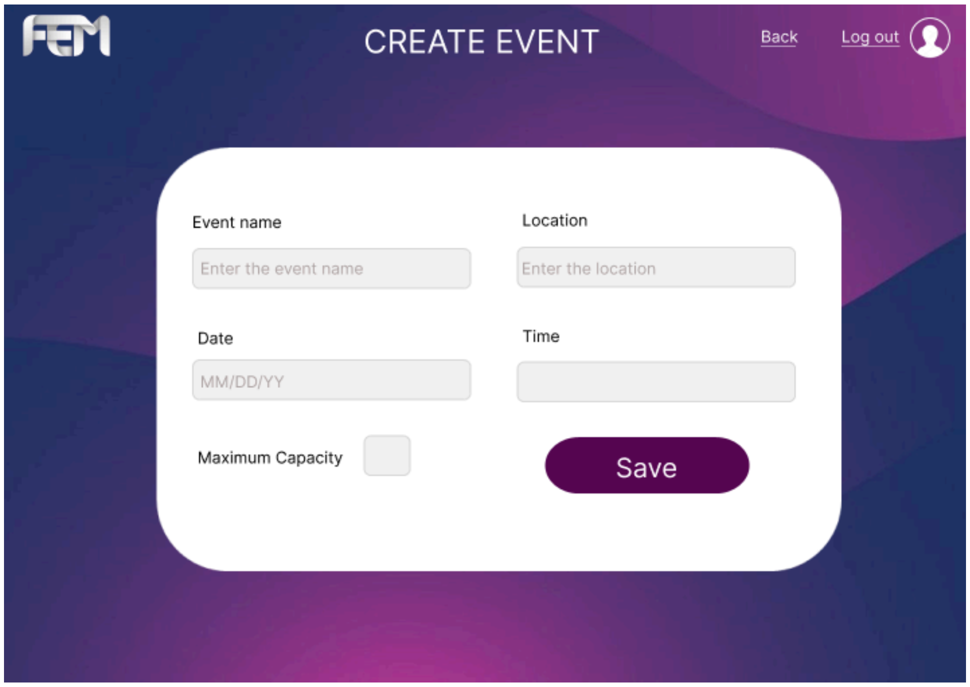
The Login Screen is the entry point of the application and is used by the organizer user to sign into their account. When the user enters their information and clicks Continue, the system verifies the credentials.



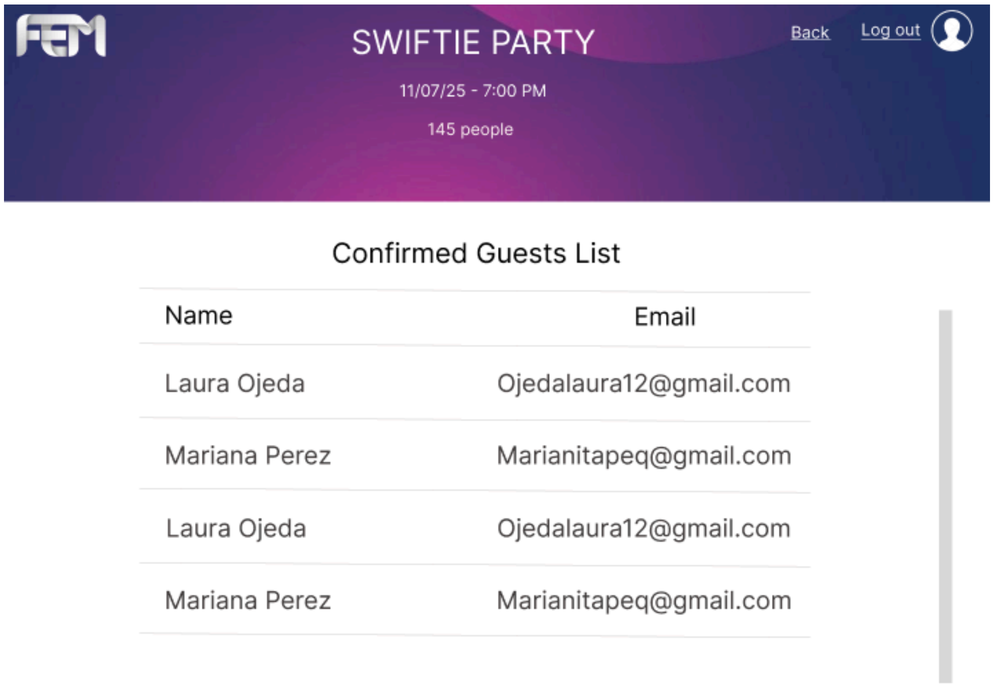
The Sign Up Screen is used by new organizer users to create an account in the system before accessing the event management features. This screen contains a form with fields to enter full name, username, email, password, and password confirmation, along with a “Create account” button that submits the information to the system. If everything is correct, a new account is registered and the user can then proceed to log in.



The Event Scheduler Screen is the main dashboard displayed to the organizer user after successfully logging into the system. Its purpose is to allow the organizer to view, manage, and interact with all the events they have created. The screen contains a “Create event” button at the top for adding new events, a list of event cards that show the event title, date and time, and the number of confirmed attendees, along with two actions for each event: “Share event” to copy or distribute the attendance link, and “Delete” to remove the event from the system. It also includes a “Log out” option at the top right to exit the session.



The Create Event Screen is used by the organizer user to register a new event in the system. This screen displays a form where the organizer can enter the event name, location, date, time, and maximum capacity, along with a “Save” button to confirm the creation of the event. When the user fills in all the required fields and clicks the Save button, the system validates the information and stores the new event in the database so that it appears later on the Event Scheduler dashboard. The screen also includes navigation options such as “Back” to return to the previous page and “Log out” to end the current session. This screen is only shown to organizer users who want to create new events.



SWIFTIE PARTY

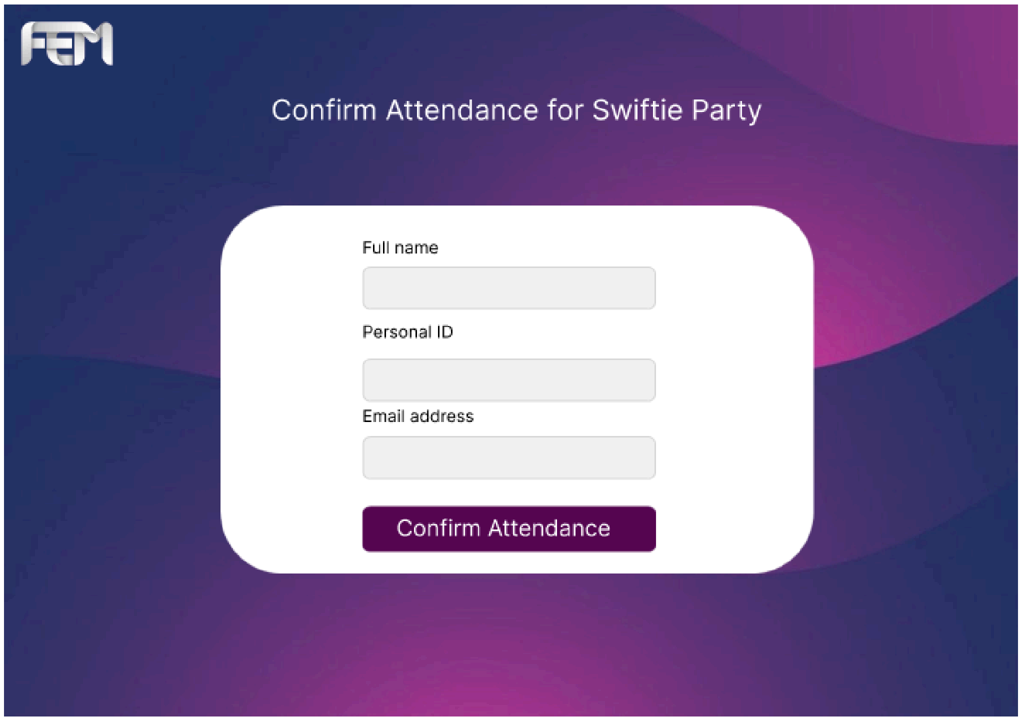
11/07/25 - 7:00 PM

145 people

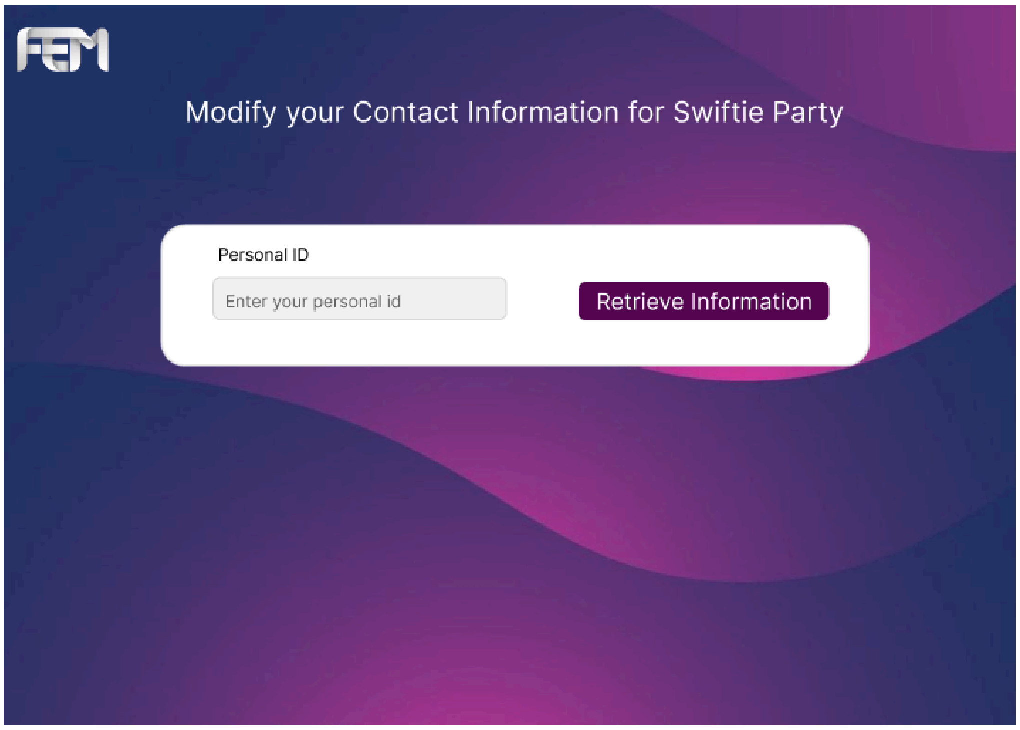
Confirmed Guests List

Name	Email
Laura Ojeda	Ojedalaura12@gmail.com
Mariana Perez	Marianitapeq@gmail.com
Laura Ojeda	Ojedalaura12@gmail.com
Mariana Perez	Marianitapeq@gmail.com

The Event Details Screen is displayed to the organizer user when they select one of their created events from the main dashboard. Its purpose is to show the detailed information of the chosen event, including the event name, date, time, and the total number of confirmed attendees. Below this information, the screen presents a table titled “Confirmed Guests List,” which displays the names and contact emails of all guests who have confirmed their attendance. This allows the organizer to easily review, reference, or copy guest information if necessary. The screen also includes navigation elements such as a “Back” link to return to the event dashboard and a “Log out” option to close the session. This screen is used solely by the organizer to monitor and manage guest information for a specific event.

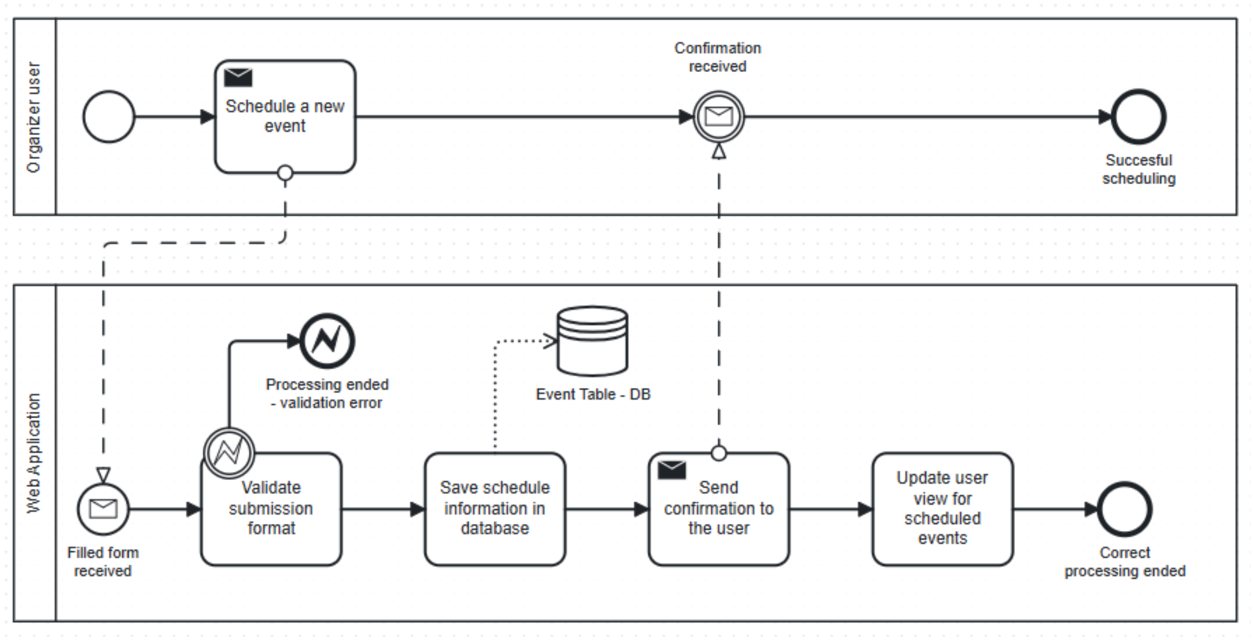


Is designed for guest users who have received a link to confirm their participation in an event, in this case, the Swiftie Party. The interface presents a simple attendance confirmation form, which includes three input fields: Full name, Personal ID, and Email address. After filling in the information, users click the "Confirm Attendance" button to submit their details.



Allows a guest to modify their contact information for a specific event. The interface contains a single input field labeled Personal ID, where the user must enter their personal identification number. By clicking the “Retrieve Information” button, the system retrieves the guest’s previously stored data from the database, allowing them to update their personal details if needed. This functionality is intended for event guests who have already confirmed attendance, ensuring that the organizer has accurate and up-to-date contact information.

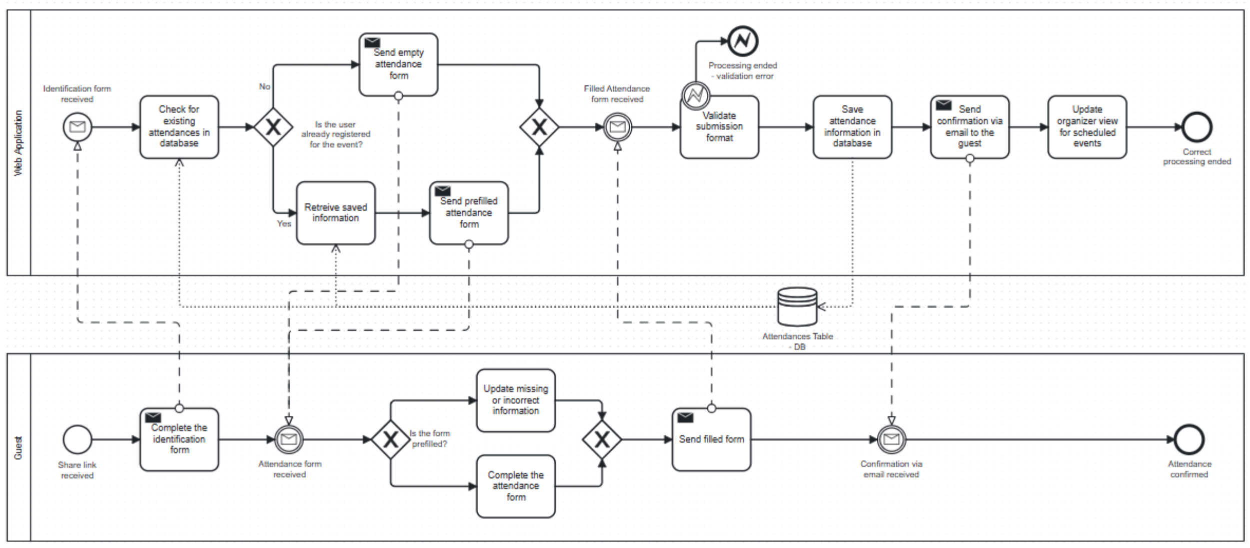
3. Business Model Processes


Organizer user
Schedule a new event
Confirmation received
Succesful scheduling
Web Application
Processing ended - validation error
Event Table - DB
Filled form received
Validate submission format
Save schedule information in database
Send confirmation to the user
Update user view for scheduled events
Correct processing ended

Process 1. Scheduling a new event (by an organizer)

Description: This process represents one of the core functionality of the application: an organizer user schedules a new event through the platform. It begins when the user decides to create a new event and completes the scheduling form. Then, the application receives this form and validates the format of the answers – if there is a format inconsistency, the system throws an error and the process ends–. After that, it saves the information into the system database, sends the user a confirmation message and updates the scheduled events of the user in the homepage.

Role in the application: This is the basis of functionality in the system,as it enables registered users to create and manage one of the main entities that the application is built around: the events. It ensures that new events are properly registered and persistent over time, in order to be visible in the user’s account. Other key activities (such as confirming attendance) depend on the correct execution of this process.

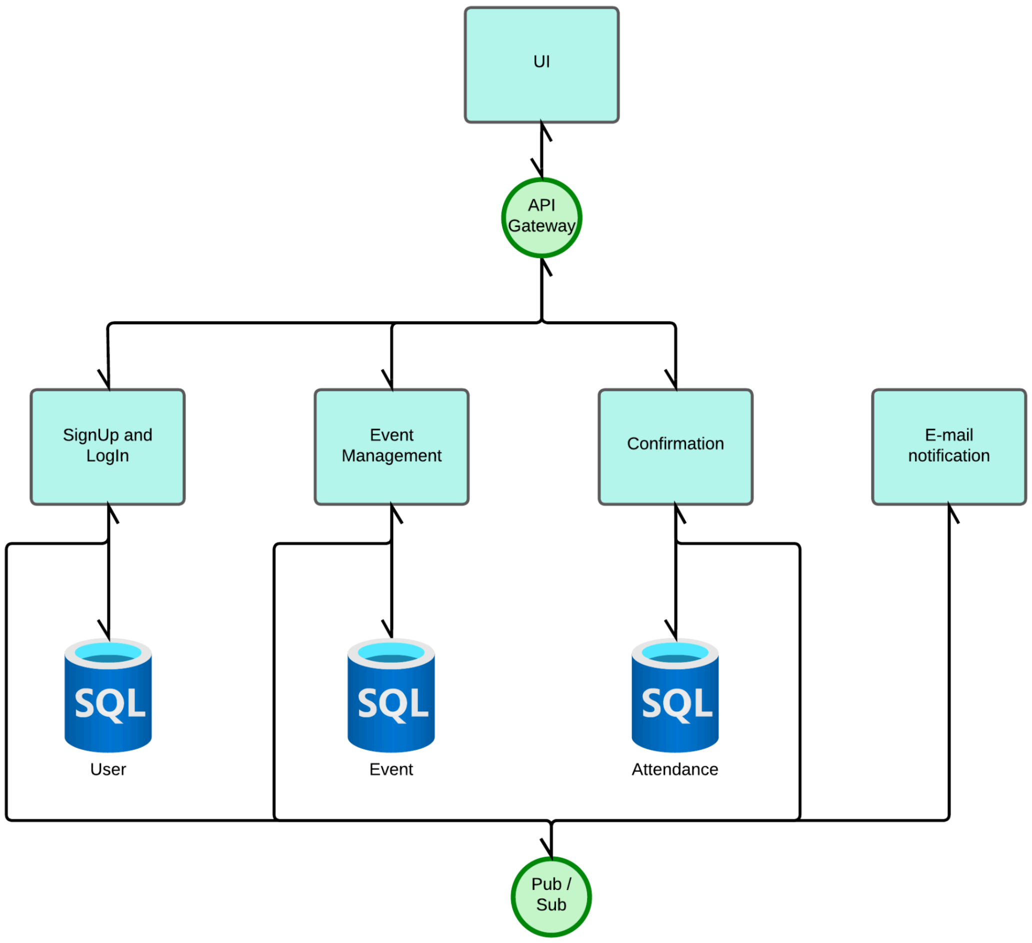

Web Application
Identification form received
Check for existing attendances in database
No
Yes
Is the user already registered for the event?
Send empty attendance form
Retreive saved information
Send prefilled attendance form
Filled Attendance form received
Processing ended - validation error
Validate submission format
Save attendance information in database
Send confirmation via email to the guest
Update organizer view for scheduled events
Correct processing ended
Attendances Table - DB
Guest
Share link received
Complete the identification form
Attendance form received
Is the form prefilled?
Update missing or incorrect information
Complete the attendance form
Send filled form
Confirmation via email received
Attendance confirmed

Process 2. Confirming attendance (by a guest)

Description: This process depicts how a guest confirms their attendance to an event they have been invited to. It begins when the guest receives a link to a specific event, allowing them to fill an identification form (just asking for personal id) to check if there are existing attendances for that guest in that event.
If no record is found, the system loads an empty form requesting essential information for the attendance; else, the system loads a prefilled form with previously saved information. After that, the guest can then fill in or update the information and submit the form.
The application validates its format and, if there are no format errors, it saves the information in the database. Finally, the system sends a confirmation email to the guest, and updates the organizer's view to reflect the number of confirmed attendees for the event.

Role in the application: This process is one of the most valuable core functionalities of the system. It allows a guest (an external user, without a registered account) to confirm their attendance to an event. This process completes the management basis of event resources, since the information of each guest is sent to the organizer.

4. Architecture Diagram



As shown in the picture above, the project will be developed using a REST API based microservices architecture. It is composed by four main modules:

- **The SignUp and LogIn Microservice:** Its responsibilities are the storage of all the users personal information when a new account is created, the verification of the credentials for the login process and the update of the password when it is required.

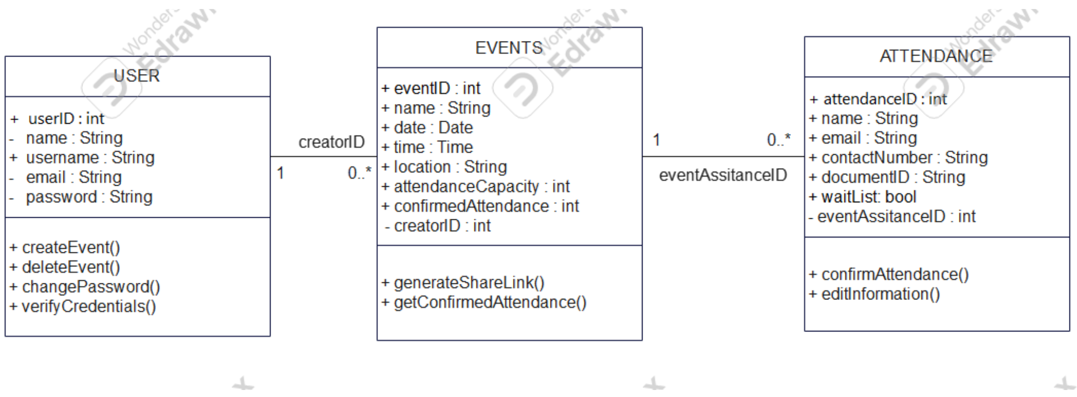
- **The Event Management Microservice:** Its responsibilities are the creation, storage, deletion and update of all the event information associated with a user, and the link generation process to share the event with the guests.

- **The Confirmation Microservice:** Its responsibilities are the processing and storage of the confirmed guests information, and its update when it is required. Additionally, when an event capacity is at its limit, this module manages the consequential waiting list.

- **The E-mail Notification Microservice:** Its responsibilities are the acknowledgment via e-mail of the following successful events:
    - SignUp.
    - Password Change.
    - Attendance confirmation.

Furthermore, the project includes other three relevant pieces:
- **The User Interface:** The main endpoint to connect the user with the application.
- **The API Gateway:** The router for the synchronous REST API communications between the application components.
- **The Pub/Sub:** The event buffer for the asynchronous requests between components, specifically those addressed to the E-Mail Notification Microservice.

5. Class Diagram



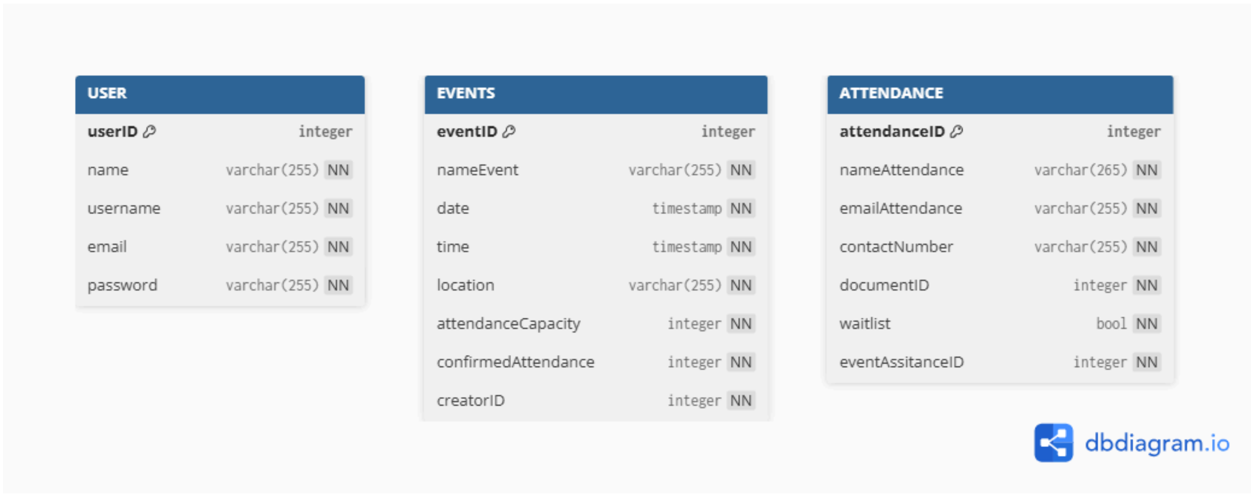
In this web app, **one user can create multiple events**, but each event is created by **only one user**.
Similarly, **one event can have multiple attendances**, and **each attendance record belongs to a single event.**

The relationships between the classes are represented as follows:

- The **USER** class is linked to the **EVENTS** class through the attribute **creatorID**, which stores the ID of the user who created the event.

- The **EVENTS** class is linked to the **ATTENDANCE** class through the attribute **eventAssistanceID**, which represents the ID of the event each attendance belongs to.

6. Relational Database Diagram



USER		
userID	integer	
name	varchar(255)	NN
username	varchar(255)	NN
email	varchar(255)	NN
password	varchar(255)	NN

EVENTS		
eventID	integer	
nameEvent	varchar(255)	NN
date	timestamp	NN
time	timestamp	NN
location	varchar(255)	NN
attendanceCapacity	integer	NN
confirmedAttendance	integer	NN
creatorID	integer	NN

ATTENDANCE		
attendanceID	integer	
nameAttendance	varchar(265)	NN
emailAttendance	varchar(255)	NN
contactNumber	varchar(255)	NN
documentID	integer	NN
waitlist	bool	NN
eventAssitanceID	integer	NN

The relational database diagram for the app is composed of three main tables: **USER**, **EVENTS**, and **ATTENDANCE**, which correspond to the entities defined in the class diagram. Each table has a primary key (PK) and the relationships between tables are established through foreign keys (FK).

Primary Keys (PK)

- **USER.userID** is the primary key used to uniquely identify each user.
- **EVENTS.eventID** is the primary key used to uniquely identify each event.
- **ATTENDANCE.attendanceID** is the primary key used to uniquely identify each attendance record.

Foreign Keys (FK) and Relationships

Even if the relationships and FKs are not shown in the diagram, they will still be implemented through the REST API communication. The project does this abstraction, because each table is actually a detached database of a different microservice; hence, the tables only know each other through their FK simulated attributes:

- The **EVENTS** table contains the “foreign key” **creatorID**, which references **USER.userID**, establishing a **one-to-many relationship** in which one user can create multiple events.
- The **ATTENDANCE** table contains the “foreign key” **eventAssistanceID**, which references **EVENTS.eventID**, establishing a **one-to-many relationship** in which one event can have multiple attendance records.