UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Faculty of Engineering.

Subject: Advanced Programming

Teacher: Carlos Andres Sierra

Project: Virtual Experience (chati)

Sergio Nicolas Mendivelso	**20231020227**
Daniel Santiago Pérez	**20231020203**

Bogotá, D.C.

10 April 2024.

Table of contents:

Project introduction

Virtual Xperience is a virtual event platform that helps the organization, management and the participation of these events. The users can register themselves so they can access the events they want. Also, they can create events with their own activities and assignments.

The application has a calendar, so the user can watch their schedule and programmed events and activities.

Business model

The business model of the virtual event platforms it's based on creating new events where the organizer can add activities and assignments online. The users also can design these events.

Some functions of the platforms are the following points:

- **Create events:** The organizer can design virtual events and activities, with date, hour, description, and type.
- **Inscription management:** The platform facilitates the inscription process for the participants, some product sales and email confirmations.
- **Real time interaction:** The platform provides tools to interact with other participants in real times, like chats, forums and question and ask sessions.
- **Access to the content:** The participants can access the event material, like files, videos, and relationed themes.

To design the business model, we have to create an architecture that can manage a lot of events at time. Another important thing is the security of this project.

Business rules

- The users (Organizers or participants) must be registered with an unique name, email and a password, so they can log in and access their information.
- The Organizers can create events and manage them, adding activities, and participants to the event.
- The participants can interact with each other, with the chats.
- The participants can interact with the material uploaded by the organizer of the event
- The users can watch their next activities in the calendar.

StakeHolders

- Event organizers: like business and people that want to do events.
- Participants: People that want to participate in those events.
- Sponsors: Companies that can finance some great events.
- Developers: The people that can program this platform.

Tools

Programming languages : Html, CSS,JS for frontend and Python for the backend
FrameWorks: Django, canvas.

User stories

- ☐ As user (Organizer or participant), I want to register myself with a unique name and password, so I can save information and protect my privacy.
- ☐ As user (Organizer or participant), I want to login with my account, so I can access my information and can know in which events I am organizer or participant.
- ☐ As organizer, I want to create virtual events, So I can plan the events in a successful way.
- ☐ As organizer I want to manage all my virtual events, also add the activities and assignments, so the participants can interact with these activities.
- ☐ As organizer I want to share documents and videos as material to my participants.
- ☐ As Organizer I want to restrict the access to some users, so I can decide who can assist in my events protecting my privacy.
- ☐ As user I want to see my Calendar, so I can watch my future activities.
- ☐ As Participant I want to Sign up to an event.
- ☐ As participant I want to see the date of the activities, an how long I have to upload these activities
- ☐ As Organizer I want to watch if the participants sent the activities at time.
- ☐ As participant I want to give my opinions about the virtual event, so give my feedback and suggestions to improve the event.
- ☐ As organizer I want to read the feedback and suggestion of the participants of my events, so I will decide what things I can change.

Entities

- User
- Account
- Event
- Organizer
- Participant
- Activity
- Assignment
- Type activity
- Activity description
- Documents
- Videos
- Access
- DashBoard
- Chat
- Group
- FeedBack.

Crc cards

The CRC Cards (Class, responsibility, collaborators) are cards to specify the responsibility of each one and the collaborators that make this possible. Basically specifies the interactions between classes.

The following diagrams are the specification of the responsibilities and the collaborator classes of each main class in the program.

User	
Responsability	Collaborators
Register himself Login in platform join events add himself to database	Event UsersDB EventsDB

Event	
Responsability	Collaborators
Create events Write event comments add himself to database	User Activity UsersDB ActivitiesDB EventsDB

Activity	
Responsability	Collaborators
create_activity add delivery add users id to the "at time list" Return itself info add himself to database	Event EventsDB ActivitiesDB UsersDB

UsersDB	
Responsability	Collaborators
create a table of users in the database return the data of the users	Activity User Event PostgresConnection

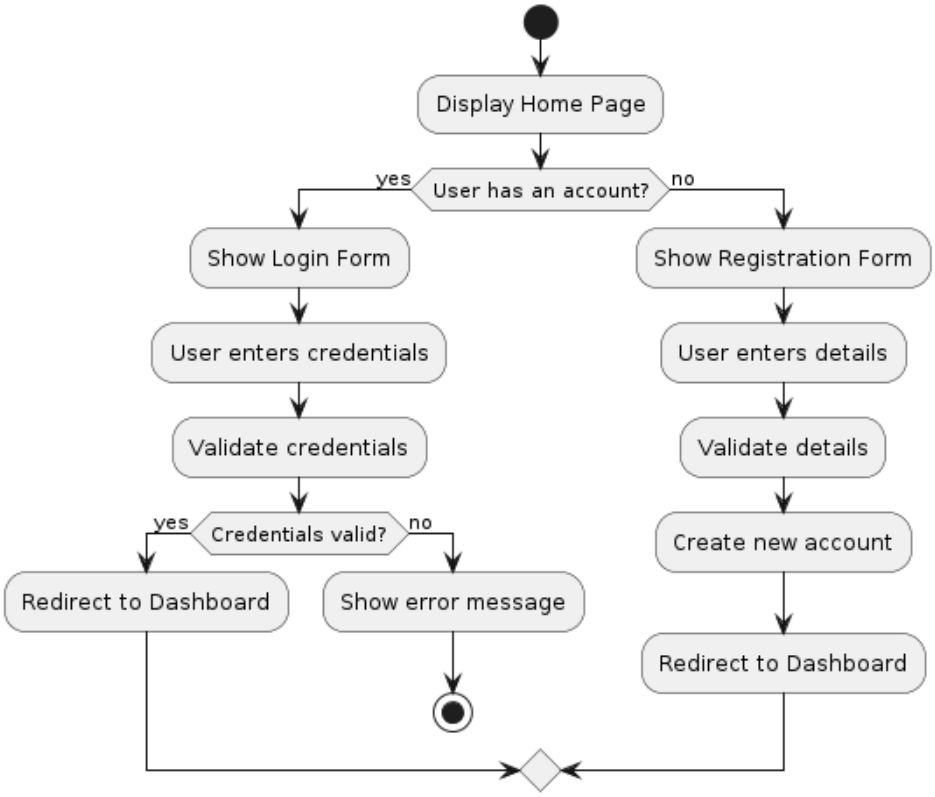
EventsDB	
Responsability	Collaborators
create a table of events in the database return the data of the events	Activity User Event PostgresConnection

ActivitiesDB	
Responsability	Collaborators
create a table of activities in the database return the data of the activities	Activity User Event PostgresConnection

PostgresConnection	
Responsability	Collaborators
Create an connection to the data base Create a sesion of the database	UsersDB EventsDB ActivitiesDB

Activity diagrams

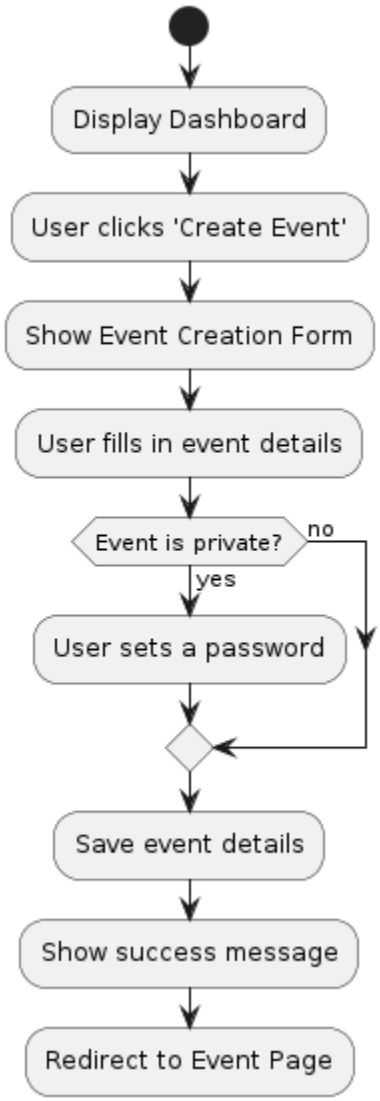
Registration and Login



Explanation:

- Users start on the home page and choose to either log in or register.
- If they have an account, they enter their credentials and log in.
- If not, they fill out a registration form, create an account, and then are redirected to the dashboard.

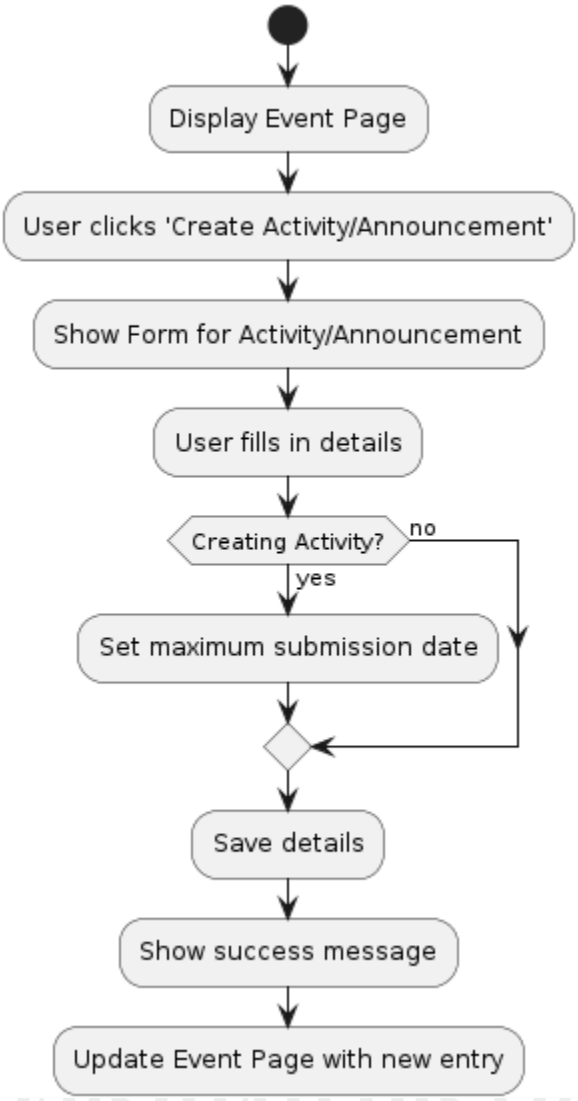
Event Creation



Explanation:

- From the dashboard, users can create a new event by filling out a form.
- They specify if the event is private or public.
- After saving the event details, they are redirected to the event page.

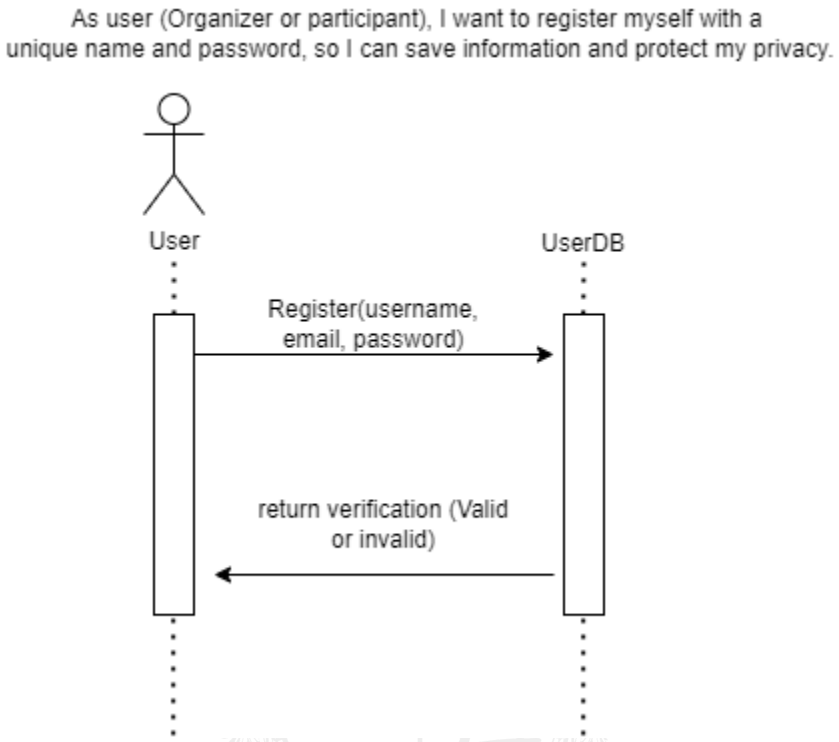
Publishing Activities and Announcements



Explanation:

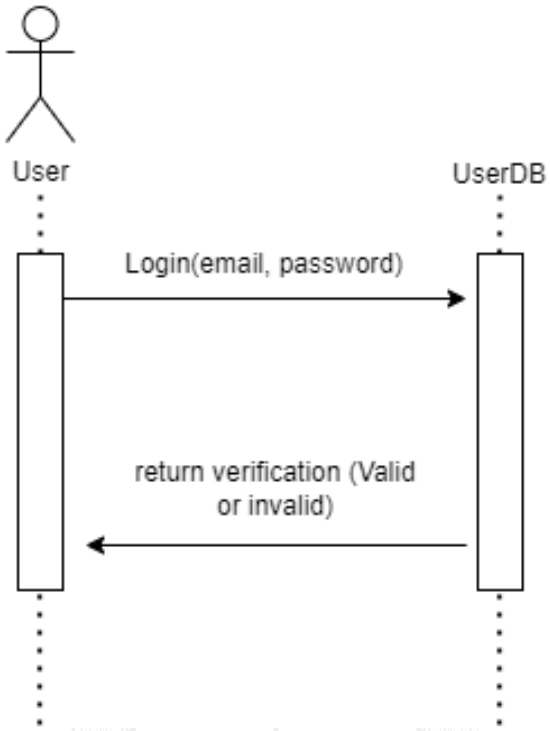
- On the event page, organizers can create activities or announcements.
- Activities have a maximum submission date, while announcements do not.
- After saving, the new entry appears on the event page

Sequence diagrams



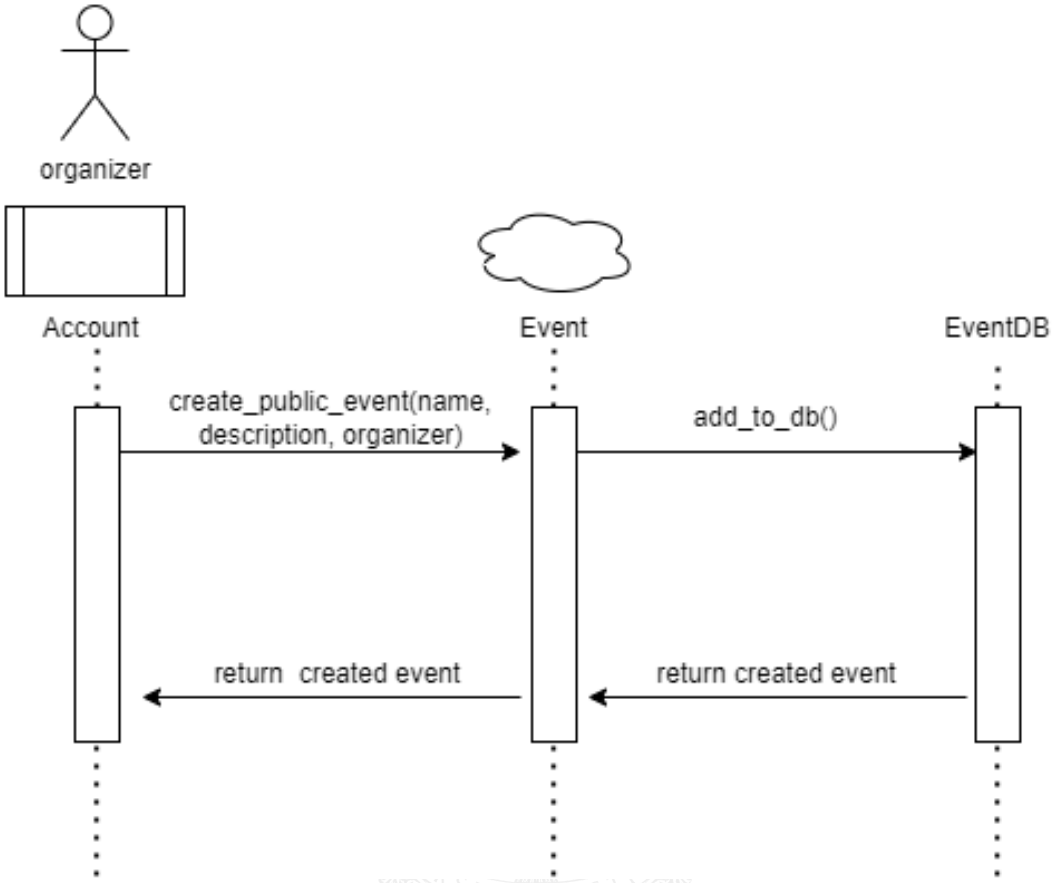
(This diagram shows the user Register with a method in order to get a verification of the Database. It will return the verification. It will true if the user and e-mail are already registered, and it will false if it not, so the program can guarantee the unique existence of an user.)

As user (Organizer or participant), I want to login with my account, so I can access my information and can know in which events I am organizer or participant.

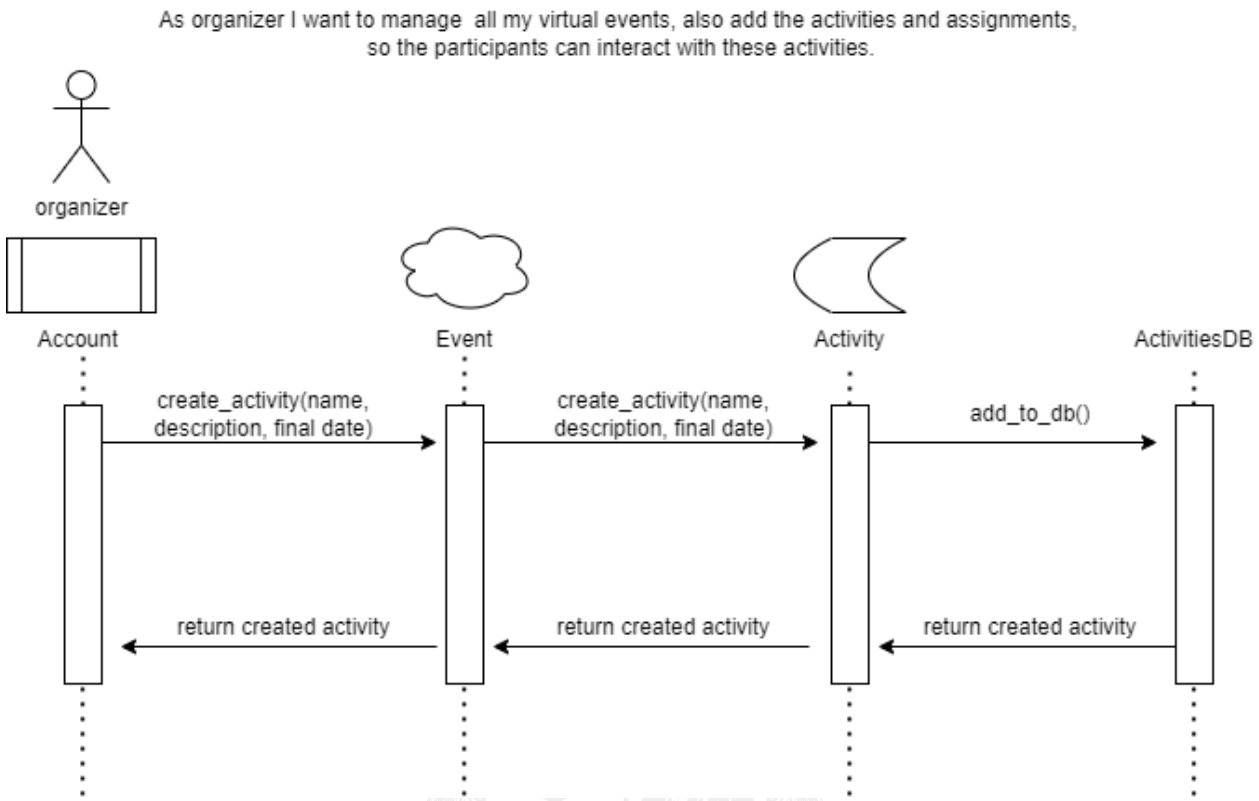


(This diagram shows the user Login with a method in order to get a verification of the Database. it will return the verification. It will true if the username and password are correct, and it will false if it not, so the program can guarantee the privacy and security of the users.)

As organizer, I want to create virtual events, So I can plan the events in a successful way.



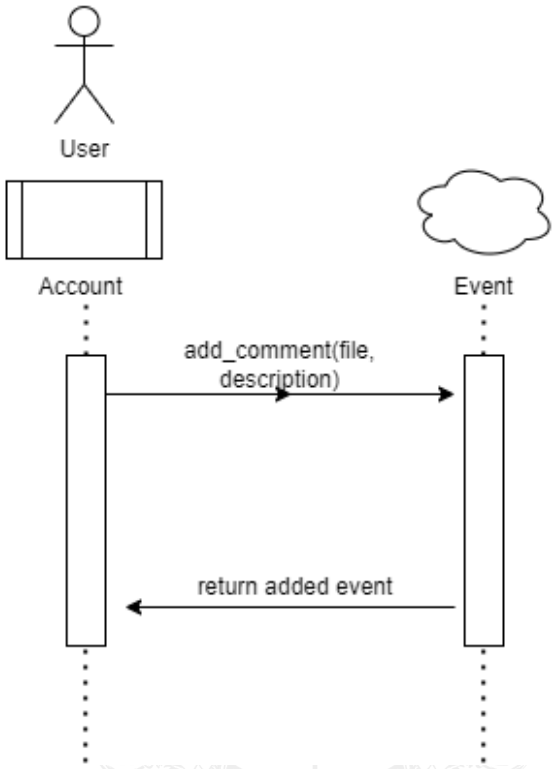
(This diagram shows how an organizer can create virtual events with the method __init__() of the Event class, and it will be added to the Database)



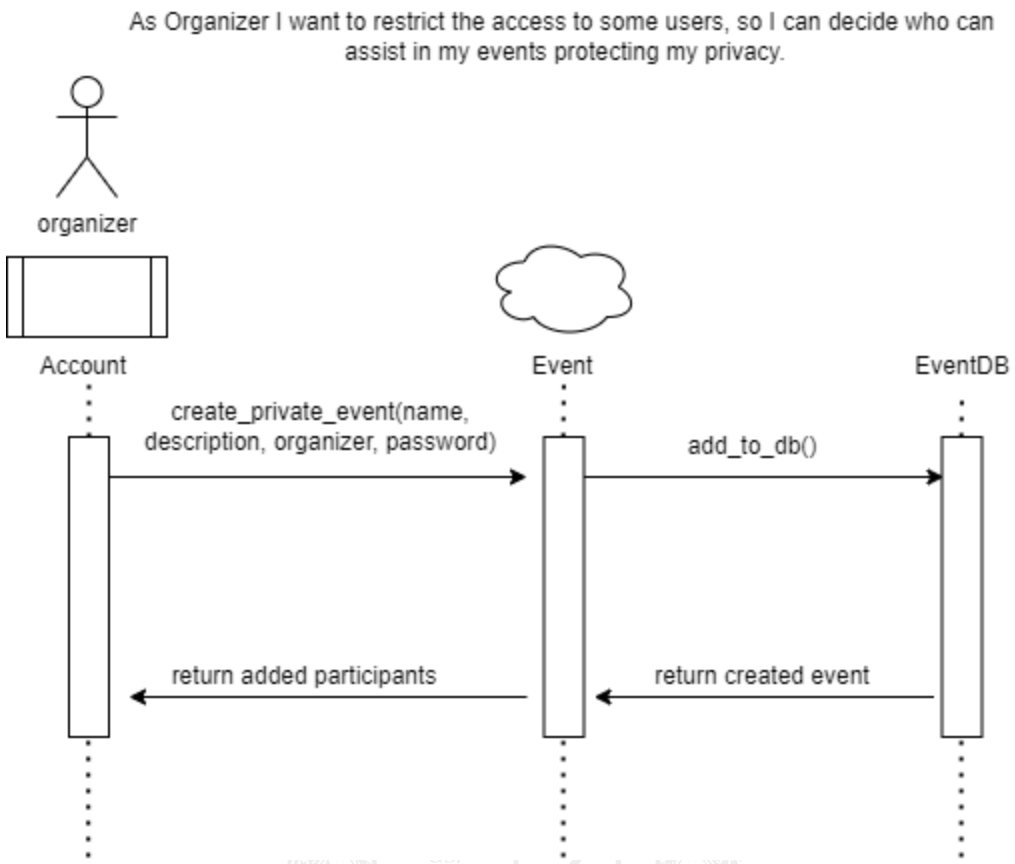
(This diagram shows how an organizer can create activities of the virtual events with the method __init__() of the Activity class, when is created it will be added to the activities_list of an event)

As organizer I want to share documents and videos as material to my participants.

As participant I want to give my opinions about the virtual event, so give my feedback and suggestions to improve the event.

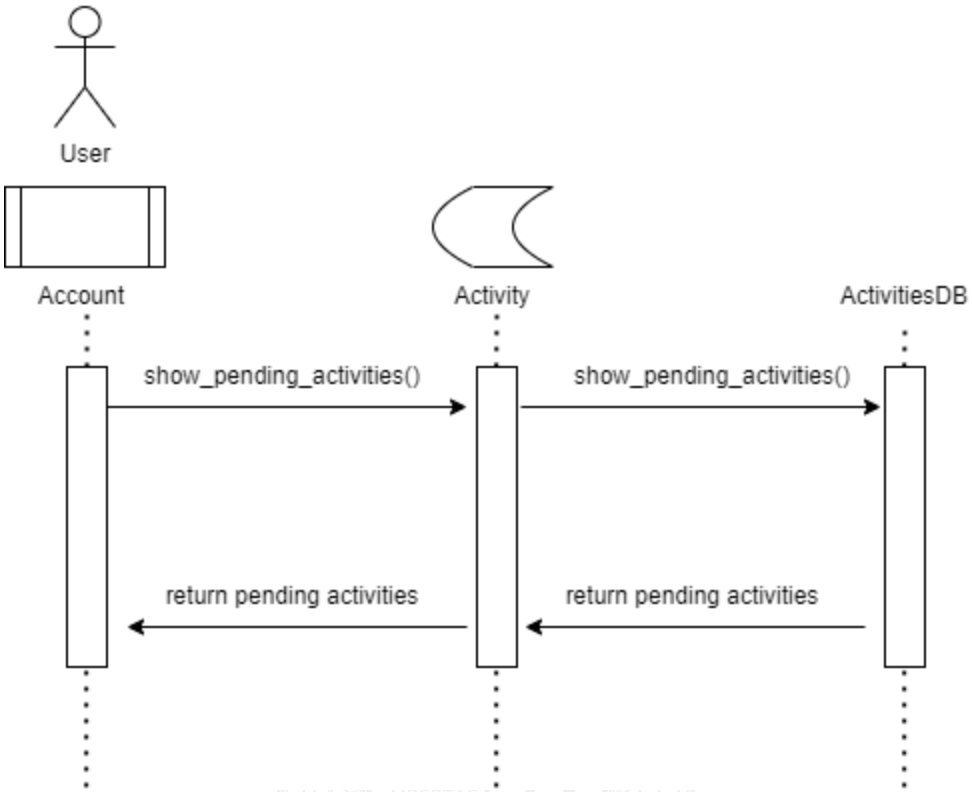


(This diagram shows how an organizer can add support material with the method add_comment() so that the participants of an event can access it. This diagram achieve with two user stories)

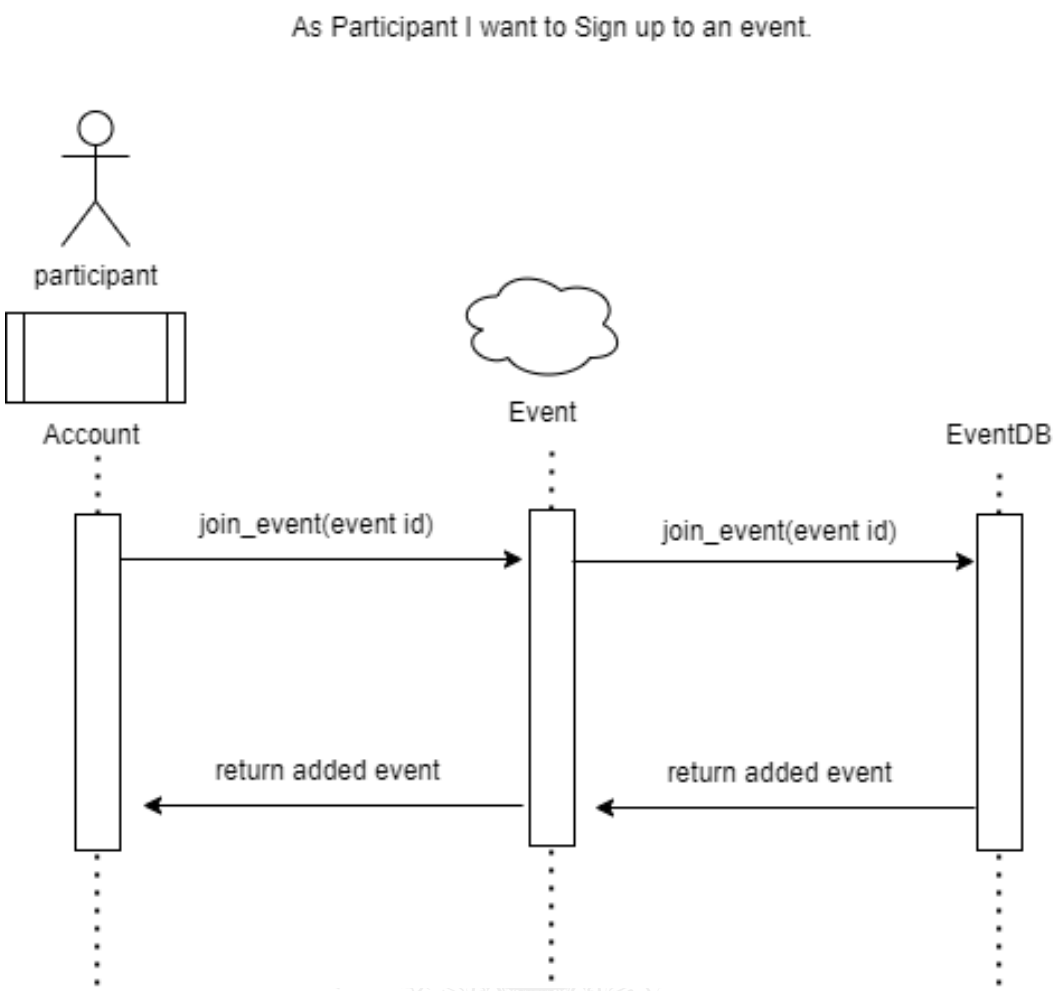


(This diagram shows how the organizer can create an private event with password, restricting the access to not invited users)

As user I want to see my Calendar, so I can watch my future activities.

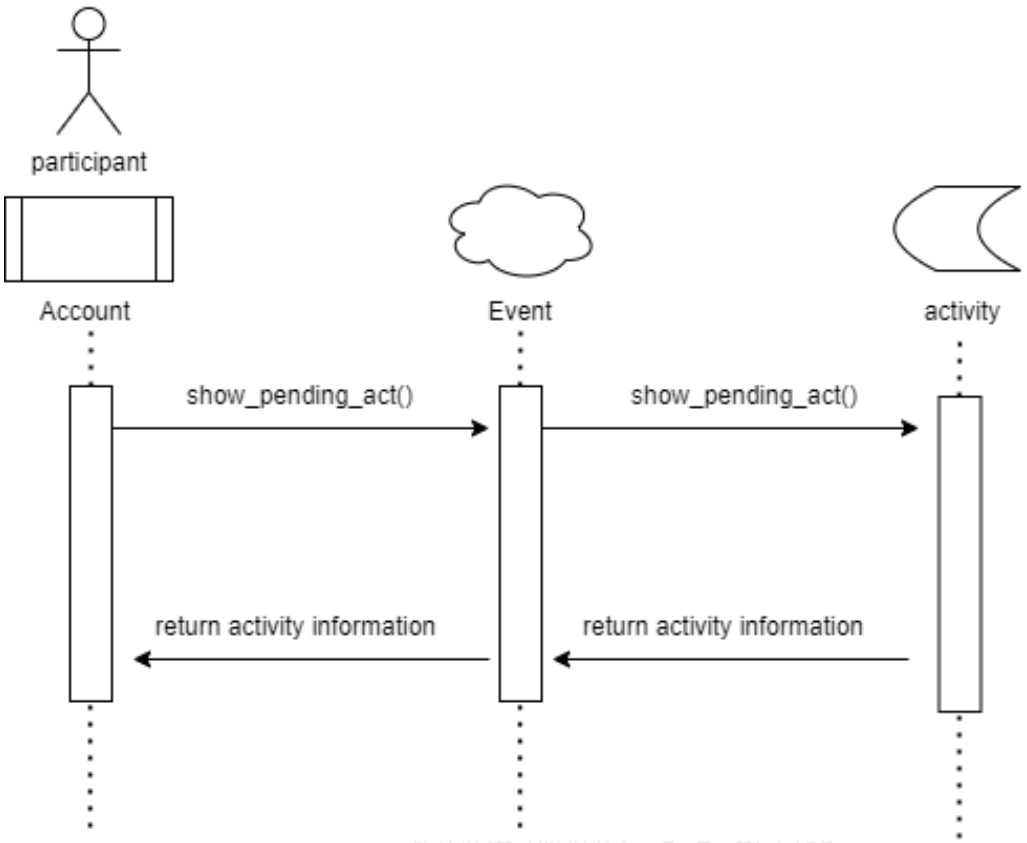


(This diagram shows how the User can access the DashBoard to see his activities.)

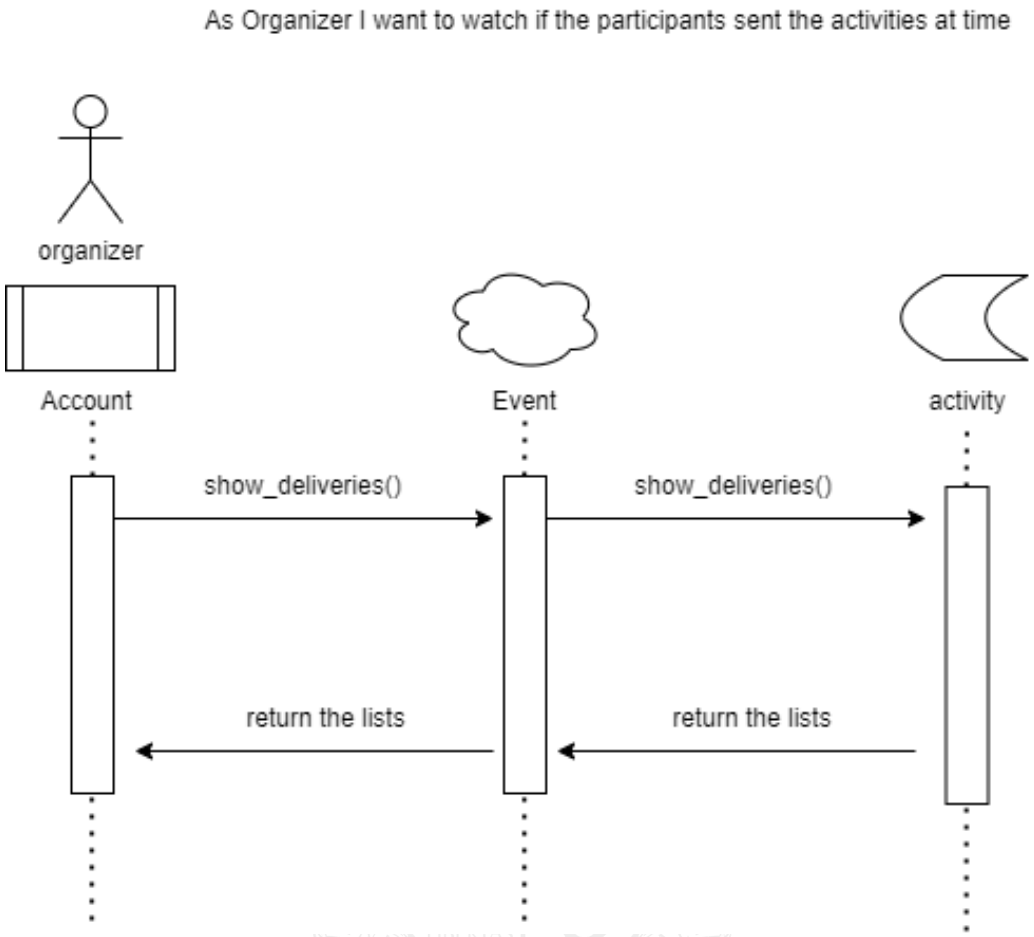


(This diagram shows how a participant can sign up to a event with the join_eventt() method, he can watch it in the DashBoard, so it can be all the public events in the Database).

As participant I want to see the date and hour of the activities, an how long I have to upload these activities



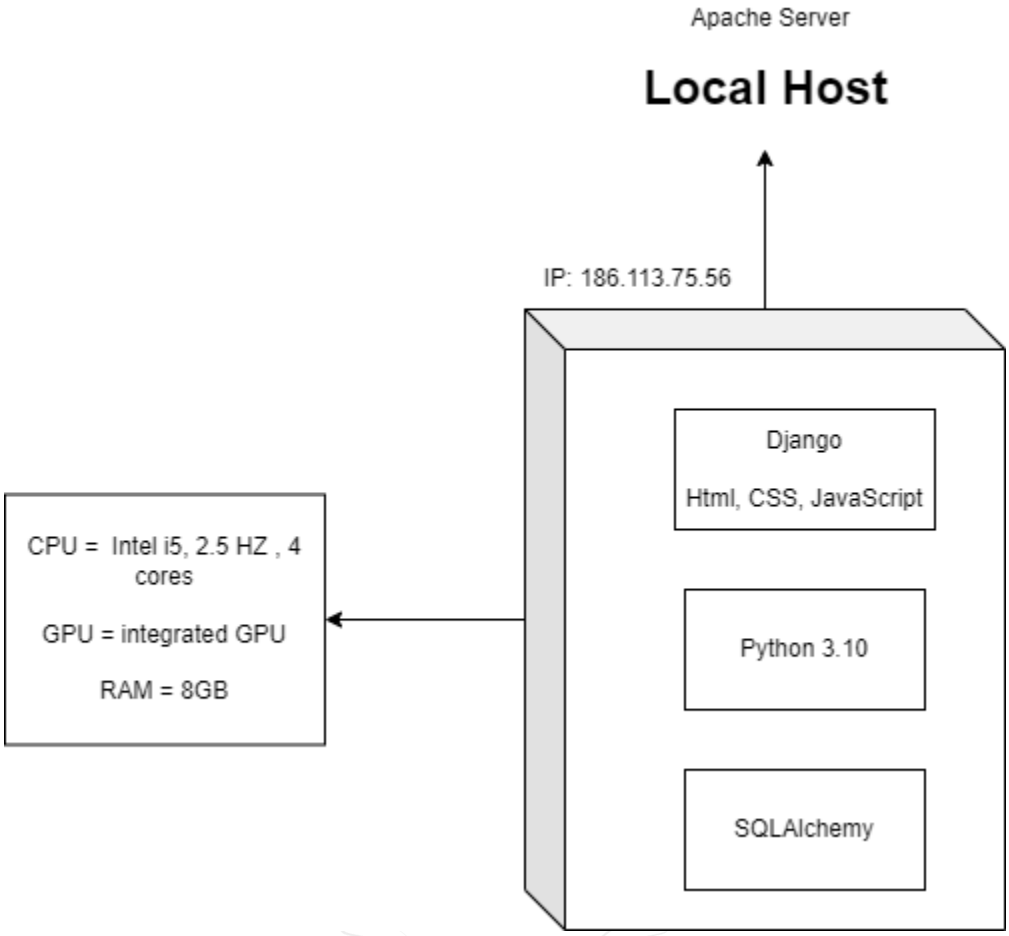
(This diagram shows how the participant can watch the attributes of an activity of an event).



(This diagram shows how an organizer can watch the deliveries of each participant that send their activities in a limit of time established by the organizer of an activity of an event).

State diagrams

Deployment diagram



(This diagram shows the physical requirements of the components of the server)

Explanation:

The before data was calculated having this information for the program:

1. Database Size Estimation

1.1. User Table

Assuming 5000 users:

- username: 20 characters (20 bytes)
- id: 20 characters (20 bytes)
- password: 64 characters (64 bytes)
- email: 30 characters (30 bytes)
- registered_events: Avg. 10 events/user, ID 20 bytes: 10 * 20 = 200 bytes
- verified: 1 byte
- uploaded_activities_id: Avg. 20 activities/user, ID 20 bytes: 20 * 20 = 400 bytes
- participant_events_id: Avg. 10 events/user, ID 20 bytes: 10 * 20 = 200 bytes
- organized_events_id: Avg. 5 events/user, ID 20 bytes: 5 * 20 = 100 bytes
- Total per user: 20 + 20 + 64 + 30 + 200 + 1 + 400 + 200 + 100 = 1035 bytes.

Total for 5000 users: 5000 * 1035 = 5,175,000 bytes ≈ 5.18 MB.

1.2. Events Table

Assuming 2000 events:

- name: 30 characters (30 bytes)
- id: 20 bytes
- description: 100 characters (100 bytes)
- organizer_id: 20 bytes
- privated: 1 byte
- password: 30 bytes (average case)
- participants_id: Avg. 50 participants/event, ID 20 bytes: 50 * 20 = 1000 bytes
- activities_id: Avg. 20 activities/event, ID 20 bytes: 20 * 20 = 400 bytes
- comments: Avg. 10 comments/event, comment ~100 characters: 10 * 100 = 1000 bytes
- Total per event: 30 + 20 + 100 + 20 + 1 + 30 + 1000 + 400 + 1000 = 2555 bytes.

Total for 2000 events: 2000 * 2555 = 5,110,000 bytes ≈ 5.11 MB.

1.3. Activities Table

Assuming 10,000 activities:

- name: 30 characters (30 bytes)
- id: 20 bytes
- description: 100 characters (100 bytes)
- event_id: 20 bytes
- start_date: 10 characters (10 bytes)

- final_date: 10 characters (10 bytes)
- deliveries: Avg. 20 deliveries/activity, ID 20 bytes: 20 * 20 = 400 bytes
- at_time_list: Avg. 20 entries/activity, ID 20 bytes: 20 * 20 = 400 bytes

Total per activity: 30 + 20 + 100 + 20 + 10 + 10 + 400 + 400 = 990 bytes.

Total for 10,000 activities: 10,000 * 990 = 9,900,000 bytes ≈ 9.9 MB.

2. Total Estimated Database Size

User table: 5.18 MB
Event table: 5.11 MB
Activity table: 9.9 MB
Total: 5.18 MB + 5.11 MB + 9.9 MB ≈ 20.19 MB.

3. CPU, GPU, and RAM Estimation

3.1 CPU and GPU

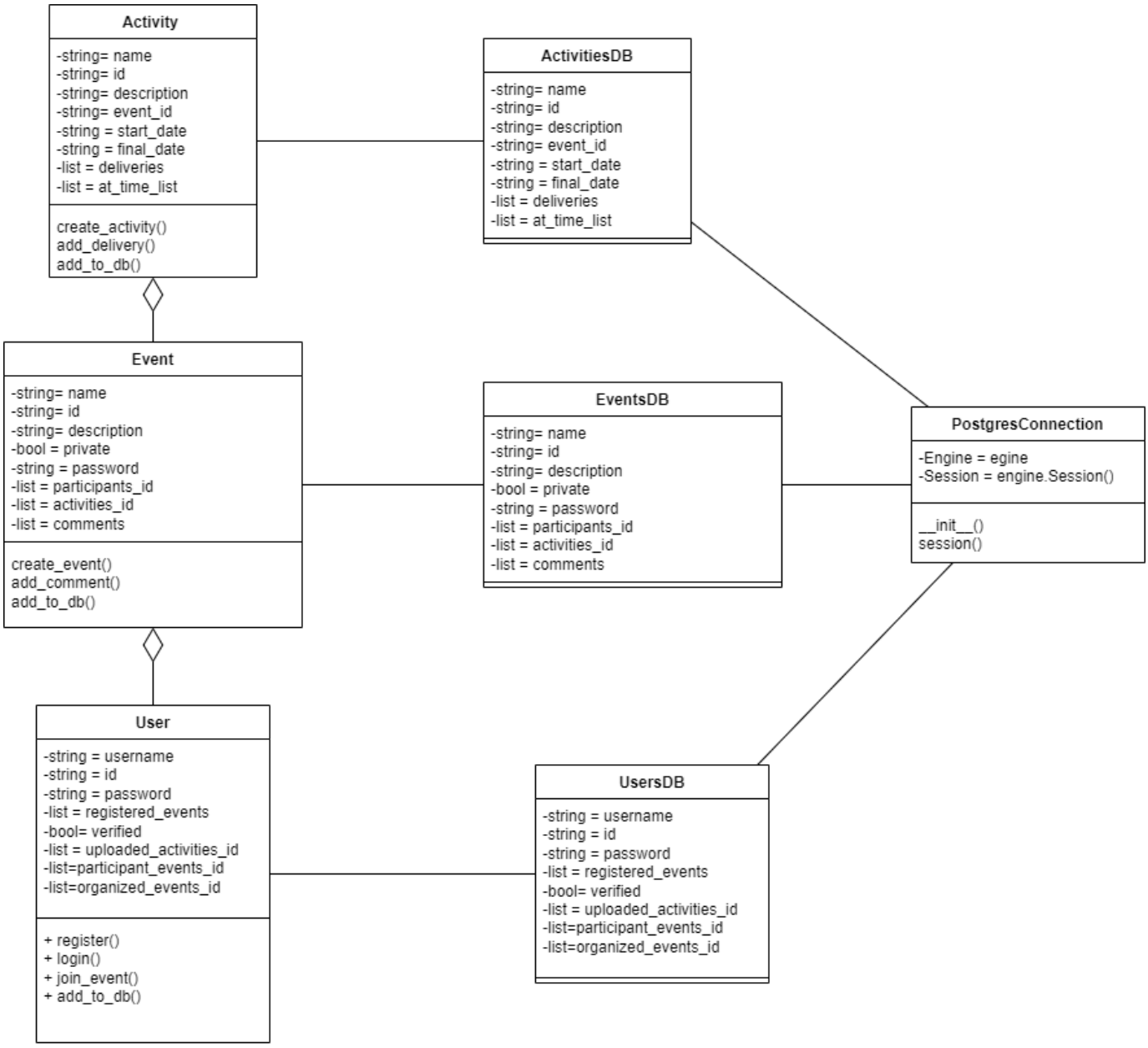
A database size of approximately 20.19 MB, we don't need significant CPU or GPU resources. A modern, entry-level CPU is okay.

3.2 RAM

Given the small database size, 4 GB - 8 GB of RAM should be more than enough to handle in-memory operations and run the database server efficiently.

Recommendations
CPU: Any modern entry-level CPU (e.g., Intel i3/i5 or equivalent AMD), with 4 cores and 2.5 Ghz
RAM: 4 GB - 8 GB.
Storage: An SSD with at least 50 GB for fast read/write operations and future growth.
GPU: Not necessary for this application; an integrated GPU is sufficient.

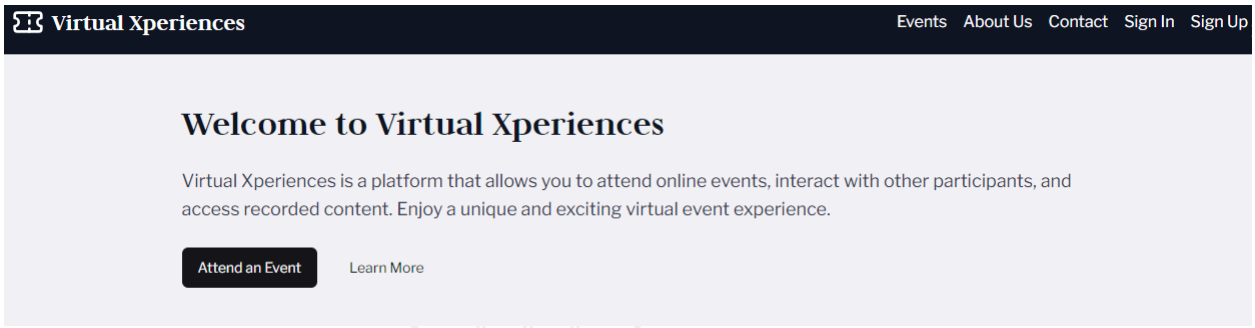
Class diagram



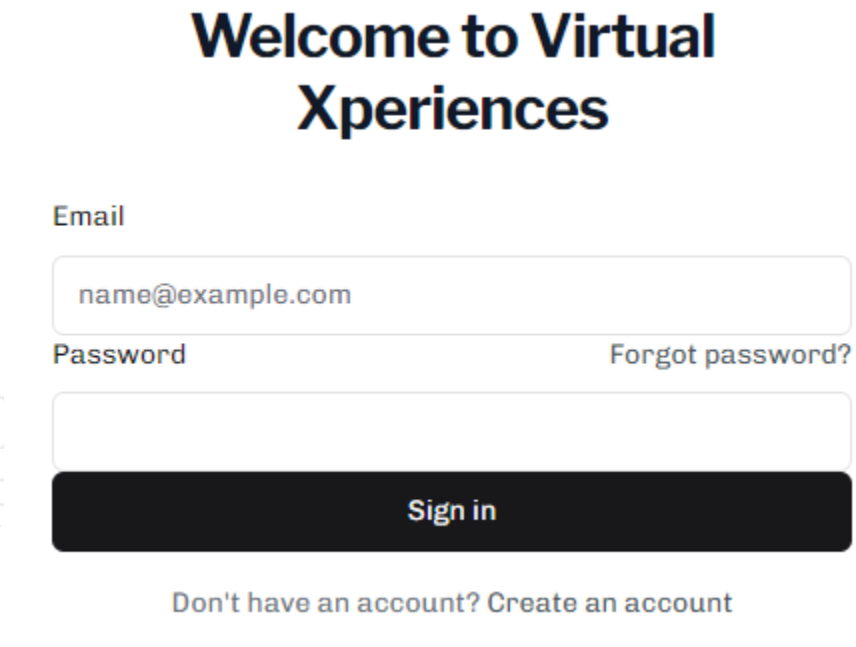
(This diagram shows how the classes interact one each other)

Mockups

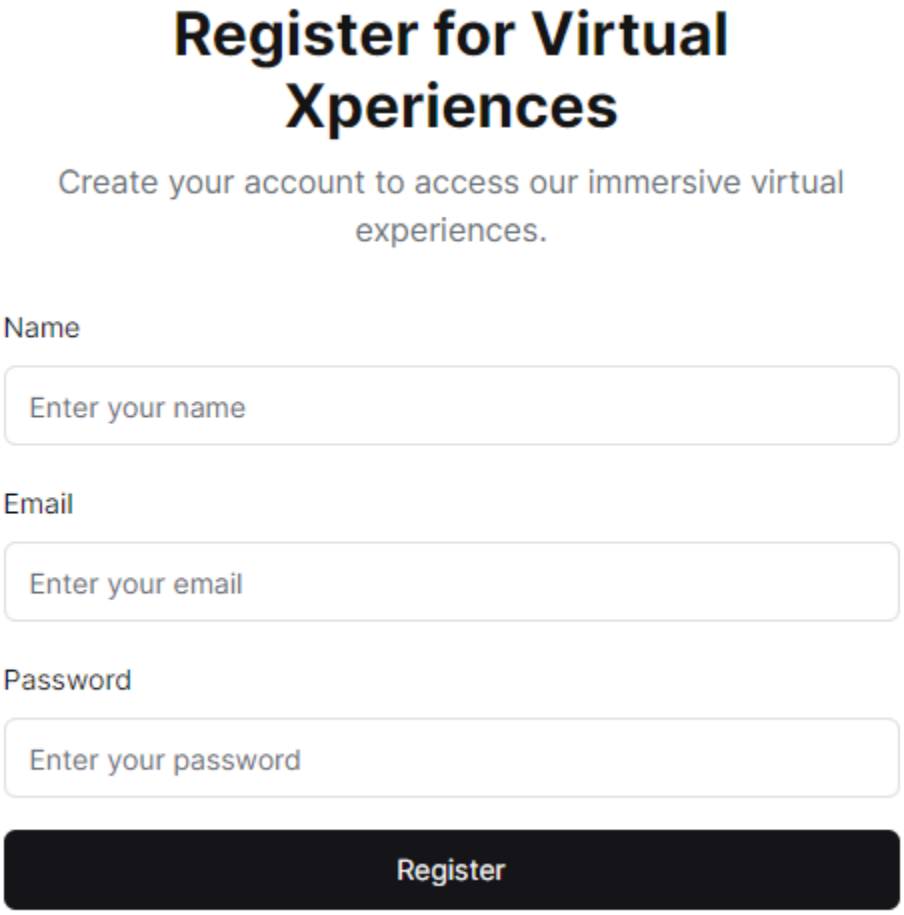
1. Home Page



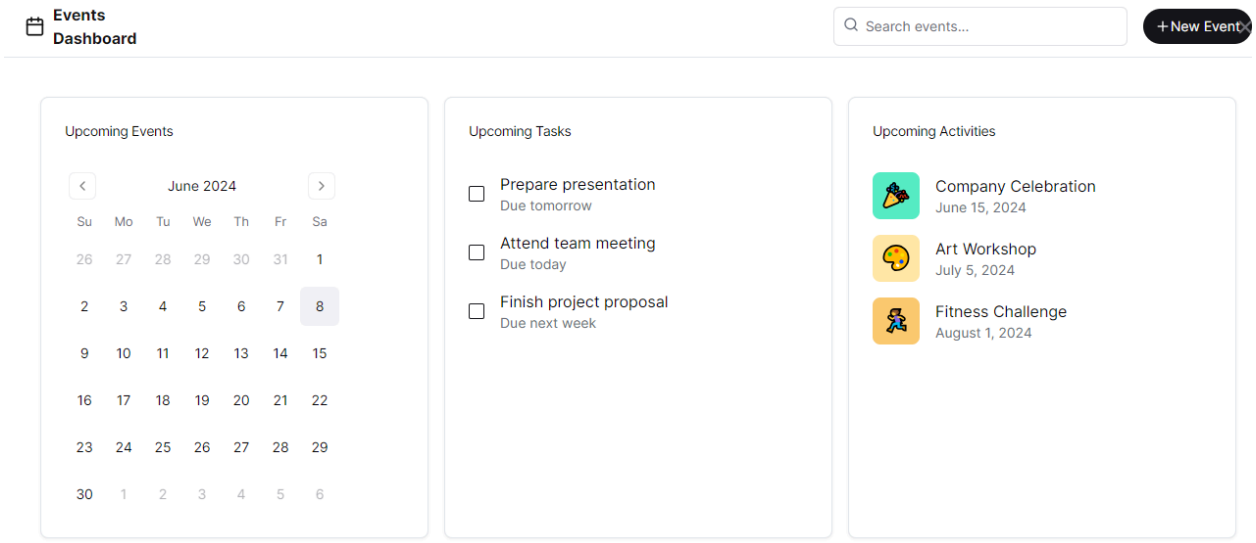
2. Login



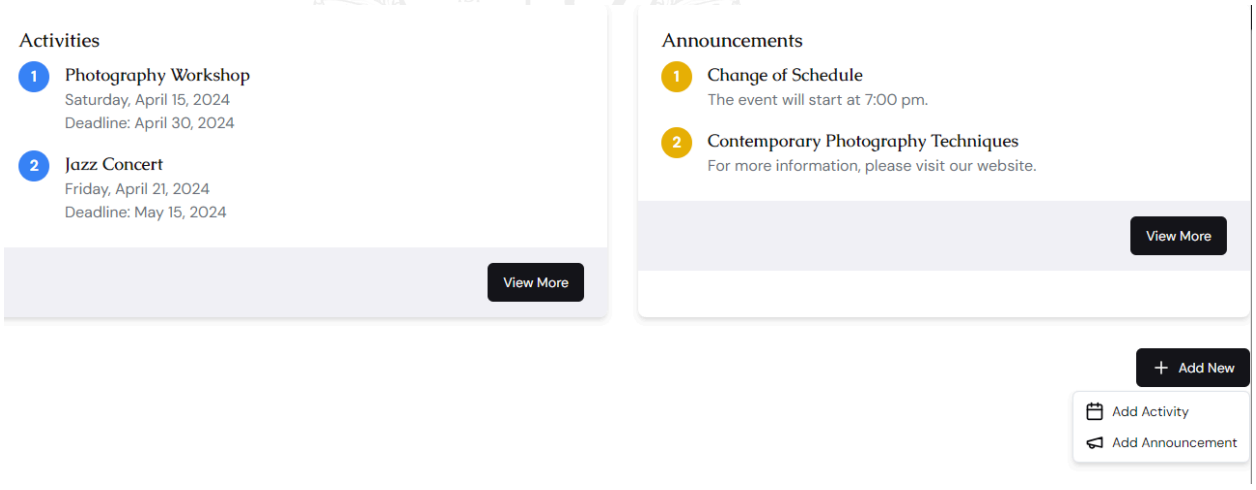
3. **Register**



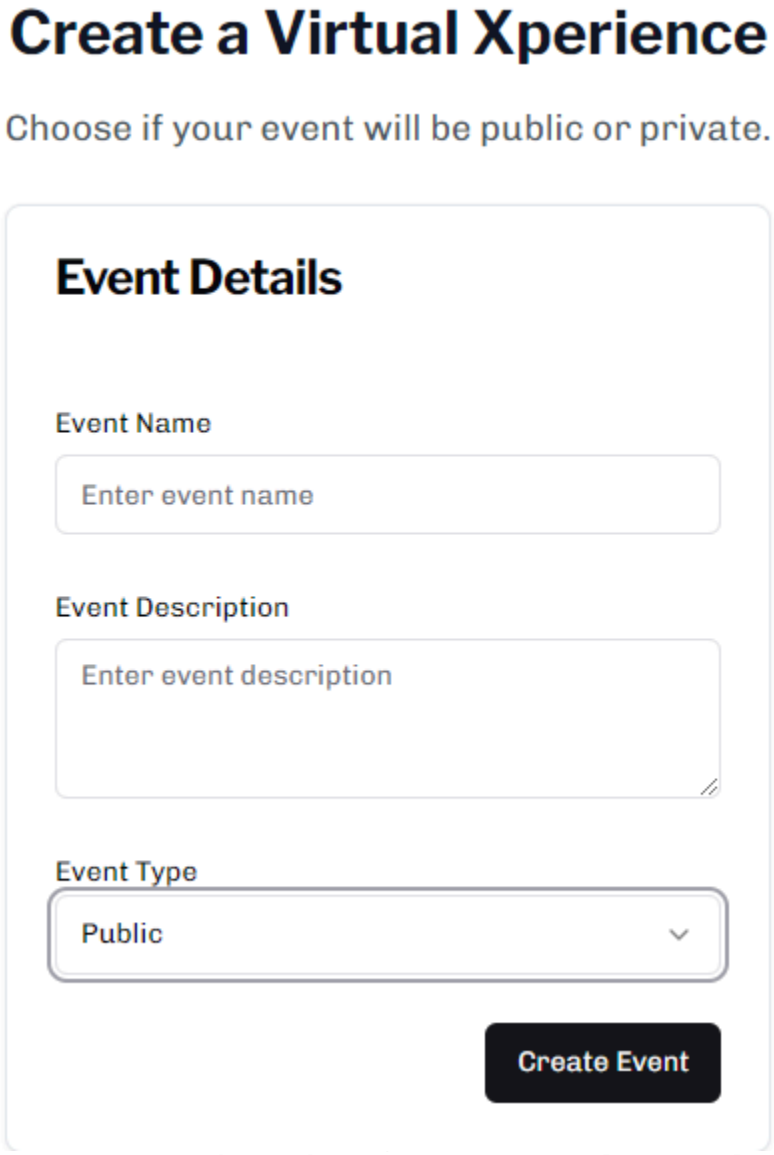
4. Dashboard



5. Event page



6. **Created event**



7. Activity page

User Interface Design Activity

This activity involves designing a user interface for a web-based task management application. You will need to create an attractive and functional design that meets the provided requirements.

Activity Details

Submission Deadline: June 30, 2023

Requirements:

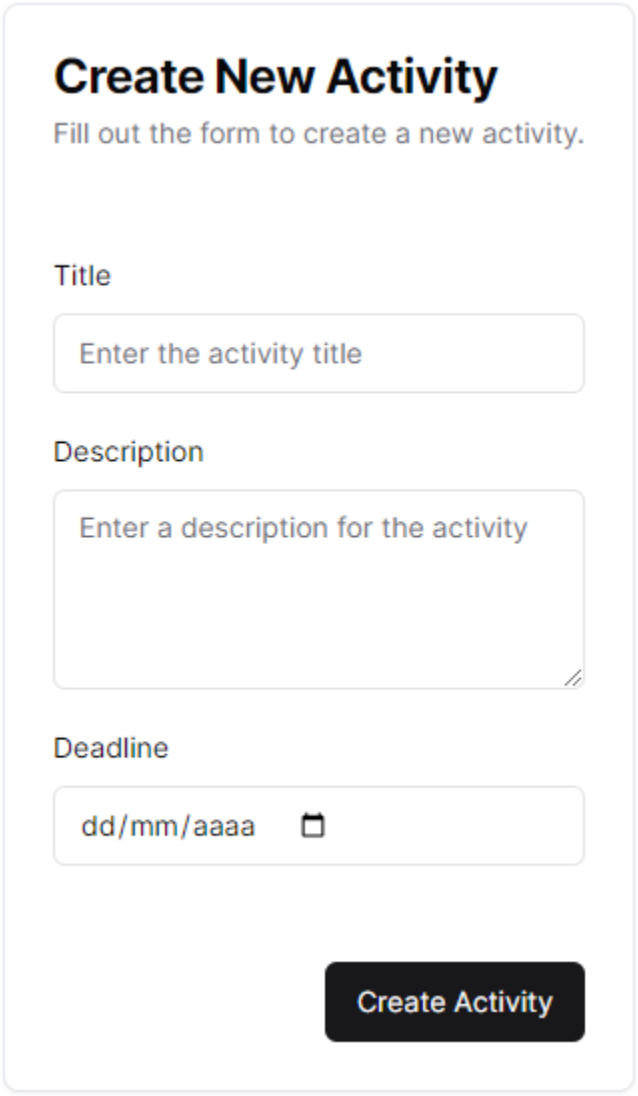
- Design for the homepage
- Design for the tasks page
- Design for the user profile page
- Consistent use of colors and typography
- Responsive design for mobile devices

Deliverables:

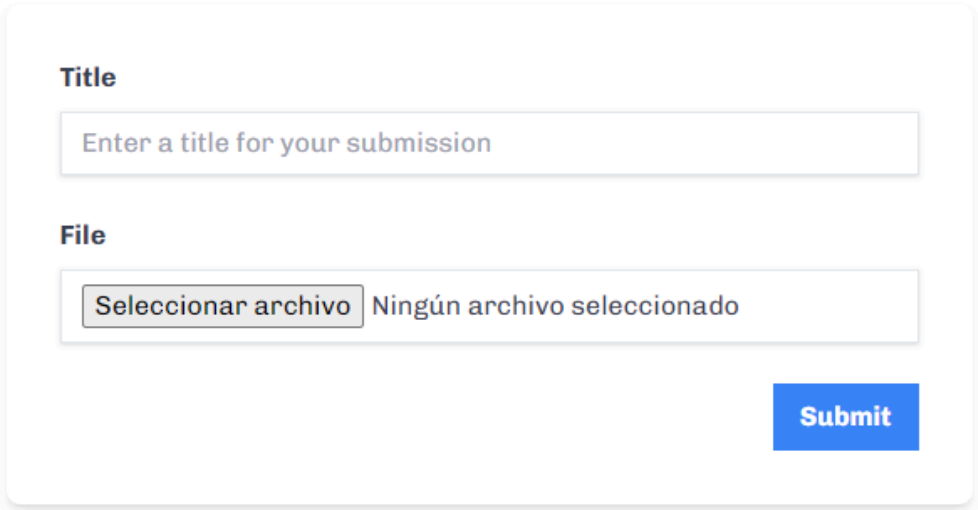
- Wireframes for the main pages
- High-fidelity designs in Figma
- Style guide with color palette and typography

Comments

8. **Created activity**



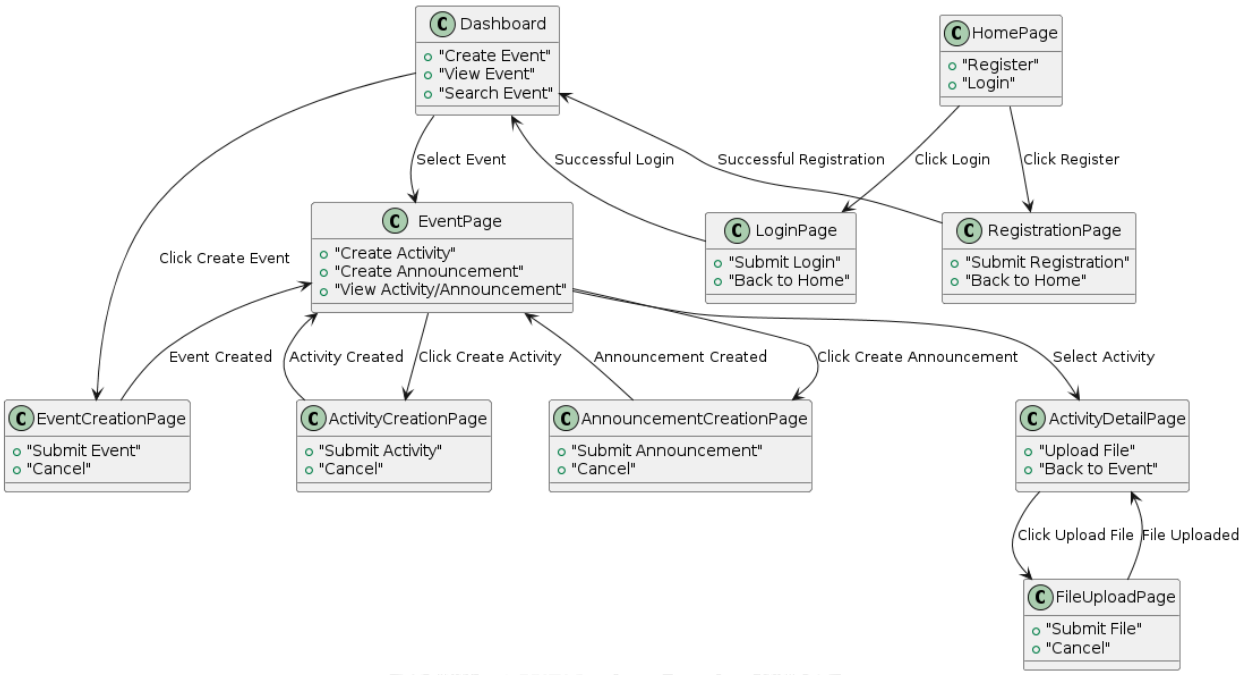
9. Activity submission



Instructions

Please make sure to select the correct file and add a descriptive title before submitting your work. Remember that the deadline for the submission is next Friday at 11:59 PM.

Navigation Map



URL_BASE: www.VirtualXperience.com
Home.py -> xperience/
Login.py -> xperience/login
Register.py -> xperience/register
Dashboard.py -> xperience/dashboard
CreatedEvent-> xperience/Createdevent
Search Event-> xperience/Searchevent
Event Page -> xperience/event
new Activity -> xperience/createdactivity
new announcement -> xperience/createdanou
announcement-> xperience/announcement
activity -> xperience/ activity
submit activity -> xperience/ submit

Design and Implementation Decisions

1. **Framework Choice**:
    - FastAPI was chosen for implementing the API due to its efficiency and simplicity in building fast and modern web applications. FastAPI offers static typing, improving code quality and error detection.

2. **Database Connection**:
    - SQLAlchemy was selected for interacting with the PostgreSQL database. SQLAlchemy provides a robust Object-Relational Mapping (ORM) that facilitates database manipulation using Python objects, improving code readability and maintenance.

    - A `PostgresConnection` class was created to manage the database connection, ensuring that connection credentials and parameters are centralized.

3. **Security:**
    - CORS (Cross-Origin Resource Sharing): CORS middleware was added to allow requests between different domains, which is crucial for web applications that may have frontends and backends hosted on different servers.

    - Password Hashing: Password hashing was implemented using `hash_password` and `verify_password` functions to ensure that user passwords are not stored in plaintext, enhancing application security.

4. **Database Structure**:
    - Data models for `User`, `Event`, and `Activity` were defined, each with their respective tables in the database (`UsersDB`, `EventsDB`, `ActivitiesDB`). This modular structure facilitates system extension and maintenance.

5. **User Management**:
    - Endpoints were implemented for registering and authenticating users (`/home/register` and `/home/login`), ensuring that data is properly validated before performing any database operations.

    - An online user (`user_online`) was managed to maintain the state of the authenticated user in the current session, facilitating the creation and management of personalized events.

6. **Event Creation and Participation**:
    - Public and Private Events: Public and private events were distinguished, implementing event creation functions with privacy parameters and password handling.

    - Participant Management: Checks were included to ensure that users cannot join events multiple times and that credentials provided for private events are correct.

7. **Exception Handling:**
    - `HTTPException` was used to handle errors and provide clear and specific feedback to the user when problems occur (e.g., invalid email, mismatched passwords, etc.). This improves the user experience when interacting with the API.

8. **Automation and Scalability:**
    - Database tables are automatically created when initializing the connection (`Base.metadata.create_all(bind=connection.engine)`), facilitating the initial setup and deployment of the application.

Examples of Decisions in Services

User Registration:

```
if '@' not in email:
    raise HTTPException(status_code=400, detail="The email address is not valid")
```

- Early validation of email format to ensure data integrity from the start.

User Login:

```
elif not verify_password(password, user_exists.password):
    raise HTTPException(status_code=400, detail="Invalid email or password")
```

- Verification of the provided password against the stored hashed password to securely authenticate users.

Event Creation:

```
event.organizer_id = user_online.id
event.privated = False
event.add_to_db()
```

- Assignment of the organizer and setting of event privacy during creation to ensure events are properly managed.

These decisions demonstrate a careful approach to creating a secure, efficient, and maintainable web application, using best programming practices and modern tools.

View Decisions

When deciding to use primarily black and white colors for the frontend design, several considerations come into play. Firstly, the choice of color scheme significantly influences the visual perception and overall user experience. Here's why opting for a black and white color palette with a simplified layout can be a strategic decision:

1. **Clarity and Simplicity:** Black and white colors offer a high level of contrast, which enhances readability and clarity of content. By minimizing distractions caused by a multitude of colors, the user's focus is directed towards the essential elements of the interface, resulting in a cleaner and more straightforward user experience.

2. **Timelessness:** Black and white color schemes have a timeless appeal that transcends trends and fads. Unlike trendy color palettes that may become outdated over time, black and white designs maintain their elegance and sophistication, ensuring the frontend remains visually appealing and relevant for an extended period.

3. **Accessibility:** High contrast color combinations, such as black text on a white background, are inherently accessible to users with visual impairments. This design choice aligns with accessibility standards, ensuring inclusivity and usability for all individuals, regardless of their abilities.

4. **Versatility:** Black and white colors serve as a neutral foundation that complements a wide range of design elements and content types. Whether integrating vibrant imagery, colorful graphics, or minimalist typography, the simplicity of the black and white palette allows for seamless integration and

versatility in design composition.

5. **Branding Emphasis:** In some cases, a black and white color scheme can enhance brand identity by conveying a sense of sophistication, professionalism, and elegance. By stripping away extraneous colors, the focus shifts to the brand's core values and messaging, establishing a strong visual identity that resonates with the target audience.

In conclusion, the decision to adopt a black and white color scheme with a simplified layout for the frontend design is based on considerations of clarity, timelessness, accessibility, versatility, and branding emphasis. By prioritizing these factors, the resulting interface offers a visually appealing, user-friendly, and cohesive experience for users while effectively communicating the brand's identity and message.