**Business requirements**

We’re building a system for managing online events (e.g. conferences).

We already have a 3rd party video streaming service purchased, but the rest of the software and infrastructure should be built from scratch.

The video streaming service provides us with the following API

* create and dispose virtual rooms - the environment for holding the events
* issue time-limited tokens for accessing a virtual room (as a streamer or a spectator)
* creating a video stream by token - returns a stream id
* connecting to a stream as a spectator by token and stream id
* starting/stopping a video stream by token and stream id
* retrieving a video record by stream id

Thus, we have users who can

* register using their email
* enroll as a spectator for an event
* request an event as a speaker - this leads to allocating a virtual room for a certain time slot

An event request includes

* the speaker id
* expected duration
* content plan - free-form text
* topic tags

Event requests are handled by moderators

* they monitor pending event requests
* may ask to correct the content plan/topic tags
* may refuse event requests
* speakers should be notified via email about what’s going on with their event requests

Users may subscribe to a newsletter about upcoming events by specific tags.

Once an event is complete, its recording may be published on YouTube based upon the speakers’ decision.

We would like to build a vendor-agnostic solution in the cloud. We plan to launch in specific countries only. We plan to start with an audience of about 10000 watchers, 2000 speakers, and 500 events a week.

**Online Event Management System Documentation**

**Overview**

* The system is designed to manage online events, enabling users to register, enroll as spectators, and request to host events as speakers. The system integrates with a third-party video streaming service for managing virtual rooms and video streams.
* Initial launch in selected countries with around 10,000 spectators, 2,000 speakers, and 500 events per week.
* Vendor-agnostic, cloud-based infrastructure to support scaling

**Functional Requirements**

**User Registration**

* **Action:** Users register using their email addresses.
* **Outcome:** Registered users can either enroll as spectators or request events as speakers.

**Event Management**

* **Speakers:**  
  Speakers can request events by providing the following:
  + Speaker ID
  + Expected duration
  + Content plan (free-form text)
  + Topic tags
* **Moderators:**  
  Moderators are responsible for managing event requests:
  + Monitor pending requests.
  + Request corrections to content plans or tags.
  + Approve or deny event requests.
  + Send email notifications to speakers regarding the status of their requests.

**Video Streaming Service Integration**

* The third-party video service offers APIs for:
  + Creating and disposing of virtual rooms.
  + Issuing time-limited tokens for streamers and spectators.
  + Starting/stopping video streams by token and stream ID.
  + Retrieving video records after the stream.

**Newsletters & Subscriptions**

* **Action:** Users can subscribe to newsletters based on specific tags (e.g., topics of interest).
* **Outcome:** Users will be notified about upcoming events related to their subscribed tags.

**Non-Functional Requirements**

**Performance**

* Must support an audience of 10,000 watchers, 2,000 speakers, and 500 events per week.

**Scalability**

* The system should be scalable to accommodate growth and peak traffic.

**Reliability**

* Ensure high availability of event streams and related infrastructure.

**Security**

* Tokens for accessing streams should be time-limited and securely managed.
* Personal data (emails, subscriptions) should be encrypted and secured.

**Technical Architecture**

**Cloud Infrastructure**

* Vendor-agnostic cloud setup to ensure flexibility and cost-efficiency.

**Third-Party Video Streaming Service**

* **API Integration:**
  + Creating virtual rooms.
  + Managing stream tokens.
  + Retrieving video streams and records.

**Database**

* **Users:** Store user information, subscriptions, and event participation history.
* **Events:** Store event details

**Event Request Lifecycle**

* **Creation:** Speaker submits a request → Moderator reviews → Event approved or rejected.
* **Notifications:** Email notifications for status updates to speakers.

**User Roles**

**Spectators**

* Register via email.
* Enroll for events.

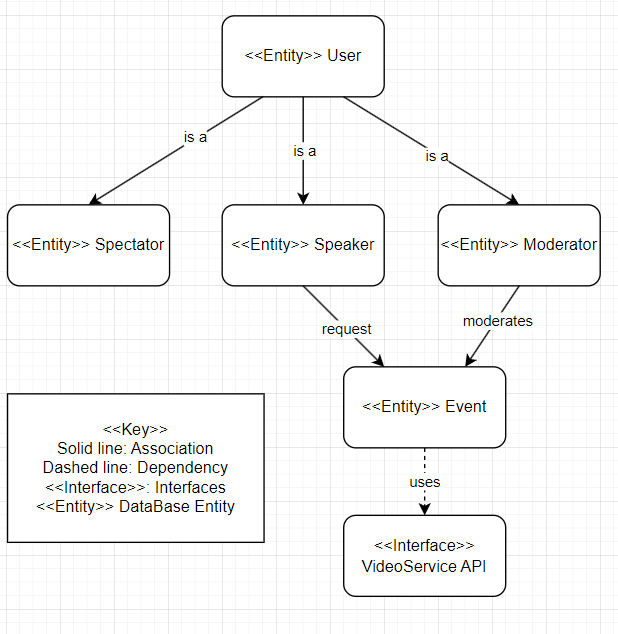
**Speakers**

* Request to host events.

**Moderators**

* Monitor, approve, or reject event requests.
* Request corrections to event details if needed.
* Notify speakers of the event status.

**UML Online Event Management System**



**Key**: Explaining the notation used (solid lines for associations, dashed lines for dependencies).

**Entities**: User, Spectator, Speaker, Moderator, Event, and VideoServiceAPI.

**Relationships**: Shows how users interact with events and the VideoServiceAPI

The purpose of the diagram is to visually represent the key entities and relationships in the online event management system, illustrating how users (spectators, speakers, moderators) interact with events and the video streaming service. It provides a simplified overview of the system's core components and their dependencies.

**Use-Case Document: Event Registration**

**Actors:**

* **Speaker**: Initiates the event request.
* **System**: Manages event validation and scheduling.
* **Moderator**: Reviews and approves or denies event requests.

**Preconditions:**

* The speaker must be registered in the system.
* The speaker has a valid account and is authenticated.

**Postconditions:**

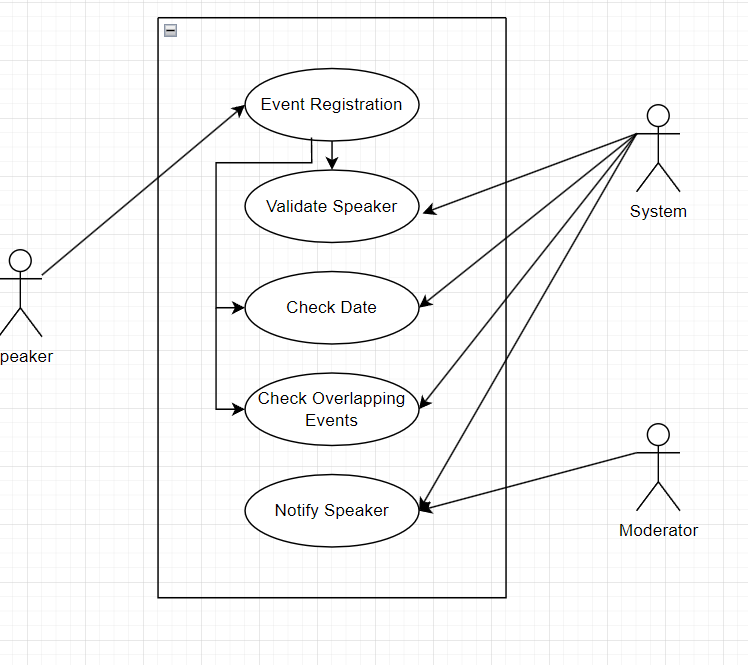
* The event is successfully registered if all validation checks pass.
* The speaker is notified of the request status (pending, approved, or rejected).

**Main Flow:**

1. **Speaker submits event request**:
   * Speaker provides the content plan, topic tags, and expected duration.
   * The speaker selects the desired date and time slot for the event.
2. **System validates the request**:
   * **Validation 1**: Check if the speaker exists in the system.
   * **Validation 2**: Verify that the event date and time are not in the past.
   * **Validation 3**: Ensure no other events are scheduled at the same time for the same speaker.
3. **System forwards the request to a moderator** for review and approval.
4. **Moderator reviews the request**:
   * **If approved**, the system allocates a virtual room for the event and notifies the speaker of approval.
   * **If rejected**, the system sends feedback to the speaker to correct the request or informs them of rejection.

**Exceptions:**

* **System error**: If there’s an issue creating the virtual room, the system notifies the speaker and moderator of a technical error.
* **Speaker unavailable**: If the speaker's account is inactive or suspended, the request is automatically rejected.



The UML use case diagram for the event registration process visually represents the interactions and operations involved when a speaker wants to register an event.

**Speaker**:

* + This is the primary actor who initiates the event registration process. The speaker provides necessary details for the event, such as date, time, content plan, and topic tags.

**System**:

* + The system is responsible for performing various validation checks and operations. It processes the event registration details submitted by the speaker.

**Moderator**:

* + The moderator acts as a secondary actor who reviews the submitted event details for approval or requests modifications.

**Event Registration**:

* + This use case is central to the diagram and represents the process initiated by the speaker to register an event.

**Validate Speaker**:

* + A process included in the event registration to verify if the speaker is registered and recognized by the system.

**Check Date**:

* + This operation ensures that the event date provided by the speaker is not in the past and is a feasible date for scheduling the event.

**Check Overlapping Events**:

* + This validation check ensures that there are no other events scheduled for the same speaker at the same time, preventing double bookings.

**Notify Speaker**:

* + Once the moderator reviews the event details, this operation involves sending notifications to the speaker about the status of their event registration, whether approved, requiring modifications, or rejected.