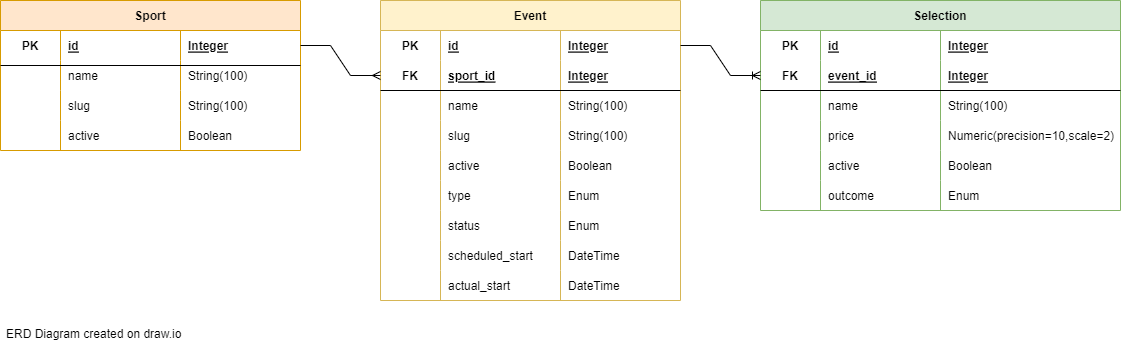
**Sport Management System**

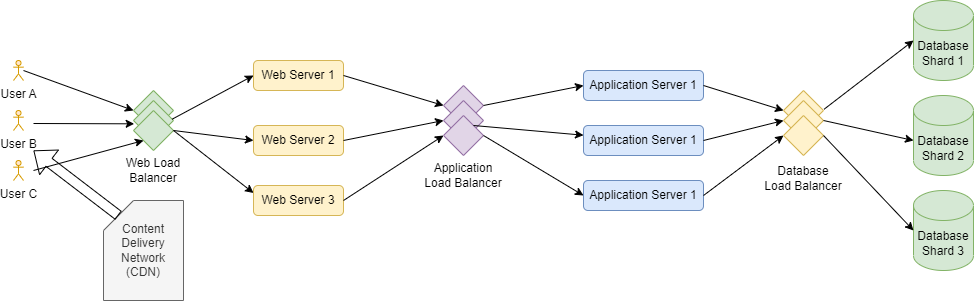
In this document, we delve into the intricacies of creating a seamless platform that allows you to effortlessly organize and oversee various sports categories, their corresponding events, and the intricate selection processes associated with them.

**Technologies used:** Python, Flask, Swagger, SQLite, Marshmallow for serialization/deserialization, Webargs for validation, Flask\_smorest for blueprint.

**ERD Diagram**



**System Design**



System design flows as follows:

* User first contacts the web load balancer which then passes their request to web server. There are many load balancing algorithms such as least connecting method, least response time, round robin or IP hash. We can opt any one of them. Web servers are responsible to render the HTML, CSS and other web page code.
* Then the request goes to application web server which then distributes the request to application server. This is where our API’s will be hosted.
* Finally the request passes to database load balancer which then distributes the request to the shard which contains our database. We can create shards based on IDs such as Sport ID’s from 1-1000 can be in first shard then next 1000 is second and so on or these can be filled in round robin.
* Also we can use CDN which act as cache to serve static content thus reducing bandwidth cost and improve website load time.

**Logging**

The API uses the Python logging module for logging events. The logging level is set to INFO. Errors and other important information are logged to provide insights into the application's behavior.

**Error Handling**

The API includes error handling for various scenarios, including database errors, validation errors, value errors, and general exceptions. Detailed error messages and logs are provided to help diagnose and resolve issues.

**Blueprint and Schemas**

The API utilizes the Flask-Smorest Blueprint to organize the endpoints and request/response schemas using Marshmallow.

**Dependencies**

The API relies on the Flask, Flask-Smorest, SQLAlchemy, and Marshmallow libraries for building the REST API, handling database operations, and validating data using schemas.

**SPORTS**

**Usage**

Developers can use this API to create, retrieve, update, and filter sports as needed for their applications.

## Create a New Sport

* **Method:** POST
* **Route:** **/sports**
* **Description:** Create a new sport.
* **Request Body:** JSON data containing the details of the new sport.
* **Response:** Created sport details or appropriate error response.

### Errors Handled:

* **SQLAlchemyError:** Handle database-related errors.
* **ValidationError:** Handle validation errors from Marshmallow schemas.
* **ValueError:** Handle general value errors.
* **Any other exceptions:** Handle unexpected errors.

## Update a Sport

* **Method:** PUT
* **Route:** **/sports/<int:sport\_id>**
* **Description:** Update the details of an existing sport.
* **Path Parameter:** **sport\_id** (ID of the sport to be updated)
* **Request Body:** JSON data containing the updated sport details.
* **Response:** Updated sport details or appropriate error response.

### Errors Handled:

* **SQLAlchemyError:** Handle database-related errors.
* **ValidationError:** Handle validation errors from Marshmallow schemas.
* **ValueError:** Handle general value errors.
* **Any other exceptions:** Handle unexpected errors.

## Search and Filter Sports

* **Method:** GET
* **Route:** **/sports/search**
* **Description:** Search and filter sports based on various criteria.
* **Query Parameters:** Parameters to filter sports based on name, active status, and slug.
* **Response:** List of sports matching the search criteria or appropriate error response.

### Errors Handled:

* **ValueError:** Handle value errors during filtering.
* **Any other exceptions:** Handle unexpected errors.

## Filter Sports by Regular Expression

* **Method:** GET
* **Route:** **/sports/regex**
* **Description:** Filter sports based on a regular expression pattern in the sport name.
* **Query Parameter:** **regex\_pattern** (regular expression pattern)
* **Response:** List of sports matching the regex pattern or appropriate error response.

### Errors Handled:

* **ValueError:** Handle value errors during filtering.
* **Any other exceptions:** Handle unexpected errors.

## Get Sports with Active Events Threshold

* **Method:** GET
* **Route:** **/sports/min\_active\_events/<int:min\_threshold>**
* **Description:** Get sports with a minimum threshold of active events.
* **Path Parameter:** **min\_threshold** (minimum count of active events)
* **Response:** List of sports meeting the active events threshold or appropriate error response.

### Errors Handled:

* **ValueError:** Handle value errors during threshold filtering.
* **Any other exceptions:** Handle unexpected errors.

**EVENTS  
  
Usage**

Developers can use this API to create, retrieve, update, and filter events as needed for their applications.

## Create a New Event

* **Method:** POST
* **Route:** **/events**
* **Description:** Create a new event.
* **Request Body:** JSON data containing the details of the new event.
* **Response:** Created event details or appropriate error response.

### Errors Handled:

* **SQLAlchemyError:** Handle database-related errors.
* **ValidationError:** Handle validation errors from Marshmallow schemas.
* **ValueError:** Handle general value errors.
* **Any other exceptions:** Handle unexpected errors.

## Update an Event

* **Method:** PUT
* **Route:** **/events/<int:event\_id>**
* **Description:** Update the details of an existing event.
* **Path Parameter:** **event\_id** (ID of the event to be updated)
* **Request Body:** JSON data containing the updated event details.
* **Response:** Updated event details or appropriate error response.

### Errors Handled:

* **SQLAlchemyError:** Handle database-related errors.
* **ValidationError:** Handle validation errors from Marshmallow schemas.
* **ValueError:** Handle general value errors.
* **Any other exceptions:** Handle unexpected errors.

## Search and Filter Events

* **Method:** GET
* **Route:** **/events/search**
* **Description:** Search and filter events based on various criteria.
* **Query Parameters:** Parameters to filter events based on name, status, type, slug, active, sport\_id, scheduled\_start range, and actual\_start range.
* **Response:** List of events matching the search criteria or appropriate error response.

### Errors Handled:

* **ValueError:** Handle value errors during filtering.
* **Any other exceptions:** Handle unexpected errors.

## Filter Events by Regular Expression

* **Method:** GET
* **Route:** **/events/regex**
* **Description:** Filter events based on a regular expression pattern in the event name.
* **Query Parameter:** **regex\_pattern** (regular expression pattern)
* **Response:** List of events matching the regex pattern or appropriate error response.

### Errors Handled:

* **ValueError:** Handle value errors during filtering.
* **Any other exceptions:** Handle unexpected errors.

## Get Events with Active Selections Threshold

* **Method:** GET
* **Route:** **/events/min\_active\_events/<int:min\_threshold>**
* **Description:** Get events with a minimum threshold of active selections.
* **Path Parameter:** **min\_threshold** (minimum count of active selections)
* **Response:** List of events meeting the active selections threshold or appropriate error response.

### Errors Handled:

* **ValueError:** Handle value errors during threshold filtering.
* **Any other exceptions:** Handle unexpected errors.

## Activate or Deactivate an Event

* **Method:** PUT
* **Route:** **/events/<int:event\_id>/activate-deactivate**
* **Description:** Activate or deactivate an event.
* **Path Parameter:** **event\_id** (ID of the event to be activated or deactivated)
* **Request Body:** JSON data containing the new active status for the event.
* **Response:** Updated event details or appropriate error response.

### Errors Handled:

* **SQLAlchemyError:** Handle database-related errors.
* **ValidationError:** Handle validation errors from Marshmallow schemas.
* **ValueError:** Handle general value errors.
* **Any other exceptions:** Handle unexpected errors.

**SELECTIONS**

**Usage**

Developers can use this API to create, retrieve, update, and filter selections as needed for their applications.

## Create a New Selection

* **Method:** POST
* **Route:** **/selections**
* **Description:** Create a new selection.
* **Request Body:** JSON data containing the details of the new selection.
* **Response:** Created selection details or appropriate error response.

### Errors Handled:

* **SQLAlchemyError:** Handle database-related errors.
* **ValidationError:** Handle validation errors from Marshmallow schemas.
* **ValueError:** Handle general value errors.
* **Any other exceptions:** Handle unexpected errors.

## Update a Selection

* **Method:** PUT
* **Route:** **/selections/<int:selection\_id>**
* **Description:** Update the details of an existing selection.
* **Path Parameter:** **selection\_id** (ID of the selection to be updated)
* **Request Body:** JSON data containing the updated selection details.
* **Response:** Updated selection details or appropriate error response.

### Errors Handled:

* **SQLAlchemyError:** Handle database-related errors.
* **ValidationError:** Handle validation errors from Marshmallow schemas.
* **ValueError:** Handle general value errors.
* **Any other exceptions:** Handle unexpected errors.

## Search and Filter Selections

* **Method:** GET
* **Route:** **/selections/search**
* **Description:** Search and filter selections based on various criteria.
* **Query Parameters:** Parameters to filter selections based on name, price, active status, outcome, and event\_id.
* **Response:** List of selections matching the search criteria or appropriate error response.

### Errors Handled:

* **ValueError:** Handle value errors during filtering.
* **Any other exceptions:** Handle unexpected errors.

## Filter Selections by Regular Expression

* **Method:** GET
* **Route:** **/selections/regex**
* **Description:** Filter selections based on a regular expression pattern in the selection name.
* **Query Parameter:** **regex\_pattern** (regular expression pattern)
* **Response:** List of selections matching the regex pattern or appropriate error response.

### Errors Handled:

* **ValueError:** Handle value errors during filtering.
* **Any other exceptions:** Handle unexpected errors.

**Room for improvement**

* Add more validation.
* Better error messaging.
* Can create YAML file for swagger leading better documentation.
* Optimization of code in terms of space/time complexity.