Concordia Conference System

Yuelin Yao ID#40194926
Gina Cody School of Engineering and Computer Science
Concordia University
Montreal, Canada
yuelin.yao@outlook.com

Xueying Zou ID#40102538
Gina Cody School of Engineering and Computer Science
Concordia University
Montreal, Canada
zouxueying126@gmail.com

***Abstract*—The content of Assignment 1 is to design a conference management system. This report is used to demonstrate all the issues required within Assignment 1, including the five sections of requirements engineering, system context analysis, model-view-controller design, UI design, and implementation. The development process and code of the implementation section will also be shown in this report accordingly.**

***Index Terms*—conference system, requirements engineering, system context analysis, model-view-controller design, UI design, and implementation**

I. Requirements Engineering

The requirements engineering section consists of two small parts, the first one documents the user story and the second one shows the use case diagram.

A. User Story

There are three roles in the case of Assignment 1. They are the speaker, the attendee, and the conference organizer. The user stories for each role are as follows:

1) Speaker:

- As a speaker, I want to manage the talks, so that I can edit the title, abstract and files.

2) Attendee:

- As an attendee, I want to register for a conference, so that I can access to the schedule.
- As an attendee, I want to vote for the talks, so that I can give my feedback.
- As an attendee, I want to obtain the recording, so that I can review a talk.

3) Conference Organizer:

- As a conference organizer, I want to organize an event, so that I can assign the room and time.
- As a conference organizer, I want to have access to a messaging service, so that I can notify the attendees when there are some changes.

B. Use Case Diagram

Use case diagrams can show the different ways in which users interact with the system. The use case diagram drawn for this assignment is shown in Figure 1.

As can be seen in Figure 1, the use case diagram covers the main functions of the system, corresponding to the user stories shown in the previous part.

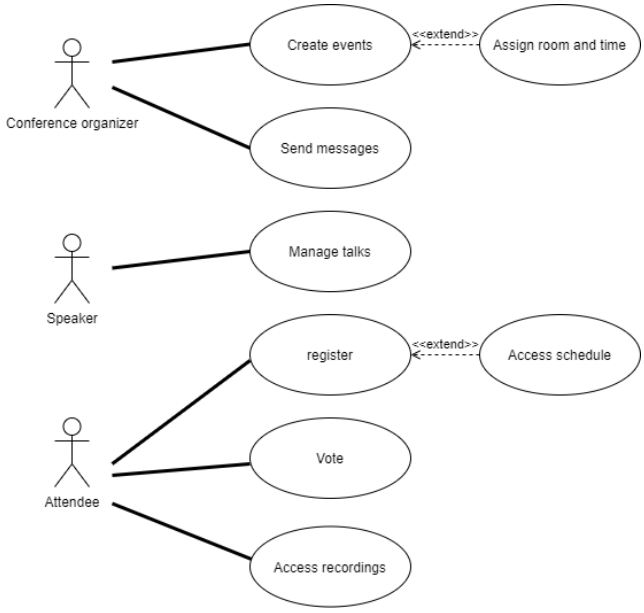

Fig. 1. Use case diagram.

II. System Context Analysis

Figure 2 shows the system context diagram of the conference management system, which depicts the roles, external systems and information flows in the context of the system.

There are three roles in the conference management system, and the information interaction between each role and the conference management system is consistent with that introduced in the first section.

For the external and internal systems of the conference management system, there are roughly five systems: the browser, the custom mobile application, the on-site kiosks available, the messaging service, and the recording system. The attendees can choose a way from the browser, the custom mobile application, or the on-site kiosks to vote on the conferences or events they attend. The messaging service can allow the conference organizer to notify all attendees about the changes in the schedule. With the recording system, every conference or event can be recorded automatically and the recordings can be obtained by attendees.

In the conference management system, there are also information flows between the system and the date or room. Each

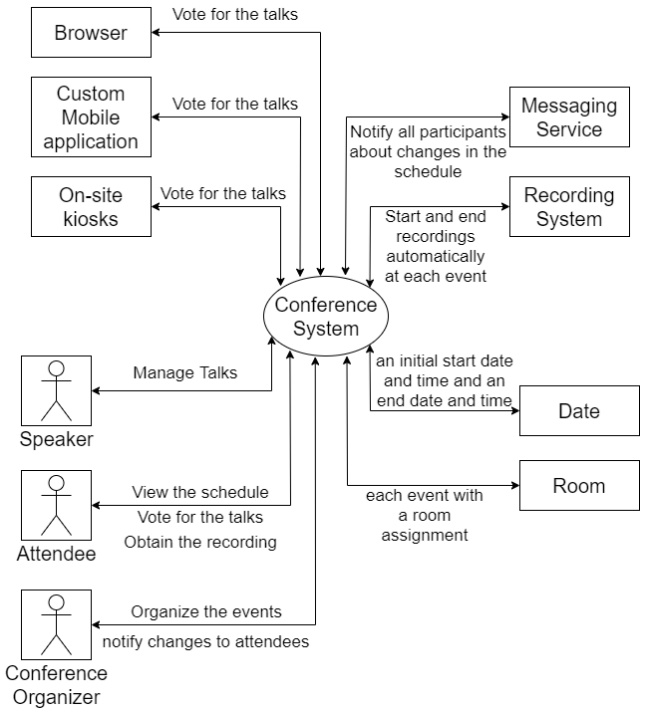

Fig. 2. System context diagram.

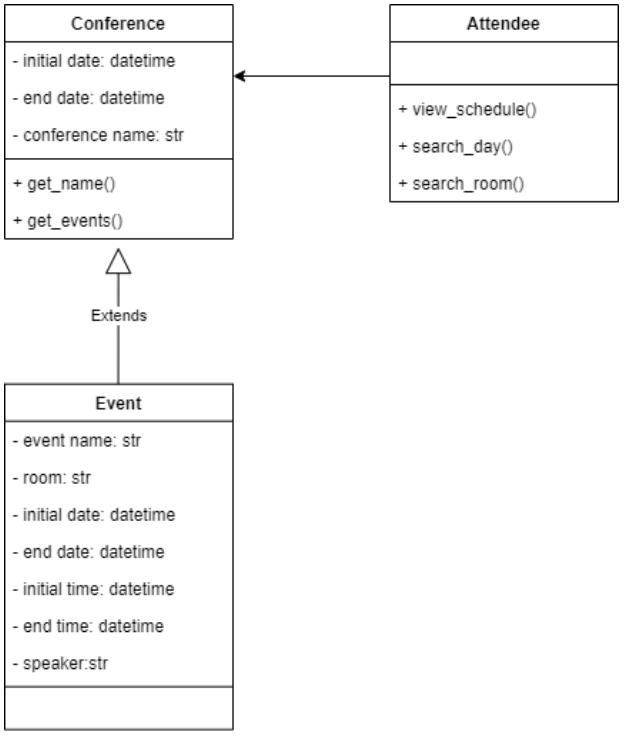

Fig. 3. Class diagram.

conference has an initial date and an end date. Each event has an initial date and time, end date and time, and the room assignment.

III. Model-View-Controller Design

The functionality for this section is *"Attendees that register for a conference can access the schedule for the conference (all events happening, including their room assignment) and view it by day or room"*. This functionality can be implemented in two main sub-functions. These two sub-functions are: to show all the events in a conference and to filter the events by date or room, respectively. The class diagram for this functionality is shown in Figure 3.

The Conference class consists of three main attributes: initial date, end date, and conference name. And it involves roughly two methods, getting the name of the conference and the name of the event. There are relatively more attributes in the Event class, mainly event time, event room, initial date, end date, initial time, end time, and the speaker. To achieve the above functionality, there is another role that may be involved, and that is the attendee. The attendees can have access to a list of all conferences, and a list of all events. They can also search for the corresponding event information based on date or room.

The Model in model-view-controller can manage the data items that are available for users to view and may change as a result of user interactions. The View in model-view-controller can describe the presentation of the data and control elements. The Controller in model-view-controller can handle the user input and prepare the necessary data sets for the view part of the work. Therefore, any calculations and data transformation in the system are performed in the controller. Due to the use of the Python programming language, the MTV laws that Django's organization conforms to are very similar in idea to MVC, with some discrepancies in naming. In the MVC system, each capital letter represents model, view, and controller respectively, while in the MTV system, each capital letter represents model, template, and view respectively. i.e., the view in the MVC system is expressed in the MTV system by template, and the controller in the MTV system is represented by the view. According to the MVC system, the class diagram can be derived from Figure 4.

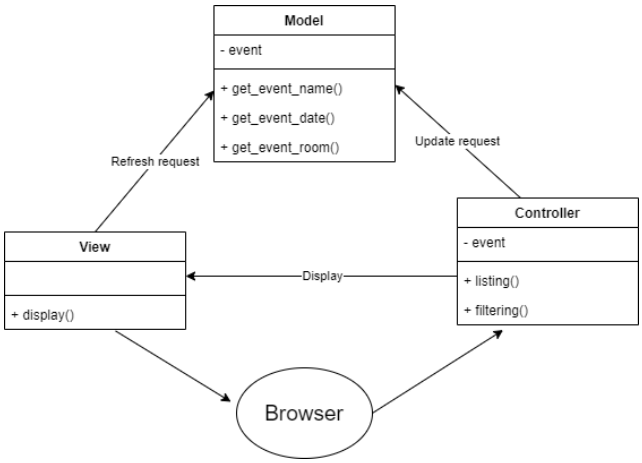

Fig. 4. MVC class diagram.

IV. UI Design

A wireframe is a schematic or blueprint that can be used to show the initial shape of the structure of the software or website being built. The wireframes of the user interface for the functionality described in the previous section can be shown as Figure 5 and Figure 6. Figure 5 shows the interface for the schedule list of all conferences. By clicking on the event details in the last column, the attendees can see all the information about the events under that conference. Figure 6 shows the detailed information of all events, which can be searched by date and room number to get information of all events that meet the criteria. The menu bar below the logo enables jumping between the two main function pages.

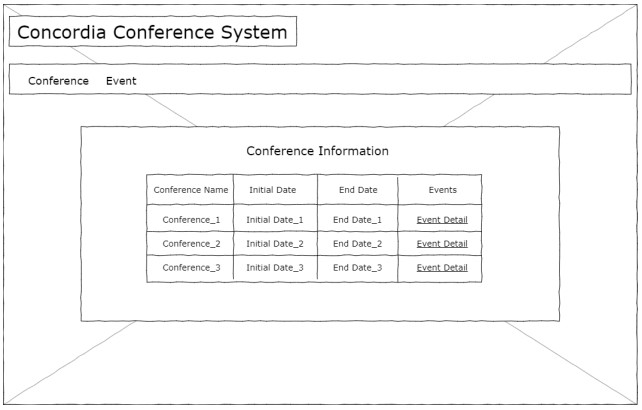

Fig. 5. User interface for listing the conference and event.

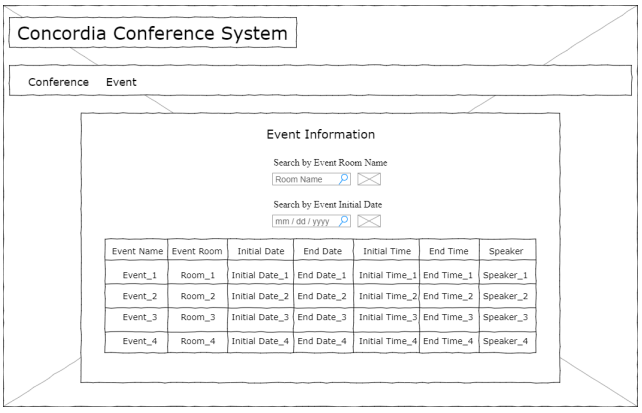

Fig. 6. User interface for filtering the events.

V. Implementation

The Django framework and MySQL database are used to implement the functionality required in the third section. Django framework based on Python web framework, following the model-template-view architectural pattern, is a high-level Python web framework which allows people to do rapid development. Also, its design is simple and pragmatic.

The first step of development is to build a big framework of Assignment 1 with Django, in which only part of the settings content and urls path assignment needs to be modified. The big framework of the Assignment 1 corresponds to the assignment folder in the source code folder.

A. Back-end

The second step is to create a folder parallel to the big framework for writing functional codes. It corresponds to the system folder in the source code folder. Three files are mainly used in this folder: models.py, urls.py and views.py. The specific source code and the corresponding explanations will be presented in the subsequent content.

1) models.py:

Among the three main files in the system folder, models.py is mainly used to operate with the database, this file contains Django's preconfigured functions to add, delete, edit, and check the database. And the attributes defined in the model will generate the corresponding fields in the corresponding database tables. There are two classes in the models.py file, namely, conference and event, and the code is shown in Figure 7 and Figure 8, respectively.

- Class Conference

```
4  class Conference(models.Model):
5      # Attributes for conference table
6      conference_name = models.CharField(max_length=12)
7      conference_initial_date = models.DateField(auto_now_add=False,auto_now=False,blank=True,null=True)
8      conference_end_date = models.DateField(auto_now_add=False,auto_now=False,blank=True,null=True)
9
10     # Define the output format
11     def __str__(self):
12         return self.conference_name
13
14     # Custom table name
15     class Meta:
16         db_table = "conference"
```

Fig. 7. Source code for class conference.

As shown in Figure 7, a table with the custom name conference will be created in the database, and the content of the table will be the conference name, conference initial date and conference end date. In this table, the conference name is a string field, and the conference initial and end date are date fields.

- Class Event

```
19 class Event(models.Model):
20     # Attributes for conference table
21     event_name = models.CharField(max_length=8)
22     event_room = models.CharField(max_length=6)
23     event_initial_date = models.DateField(auto_now_add=False,auto_now=False,blank=True,null=True)
24     event_end_date = models.DateField(auto_now_add=False,auto_now=False,blank=True,null=True)
25     event_initial_time = models.TimeField(auto_now_add=False,auto_now=False,blank=True,null=True)
26     event_end_time = models.TimeField(auto_now_add=False,auto_now=False,blank=True,null=True)
27     event_speaker = models.CharField(max_length=9)
28     conference = models.ForeignKey("Conference", to_field='id', on_delete=models.CASCADE)
29
30     # Define the output format
31     def __str__(self):
32         return self.event_name
33
34     # Custom table name
35     class Meta:
36         db_table = "event"
```

Fig. 8. Source code for class event.

Similarly, the code shown in Figure 8 creates and returns a table in the database with a custom name of event, with the event name, event room, event initial date and time, and event end date and time. In this table, the event initial date and end date are date fields, the event initial time and end time are time fields, and the rest are string fields.

It is worth noting that there is a foreign key in this class which is used to establish and strengthen the linking relationship between the data of the two tables mentioned above. This foreign key uses a one-to-many relationship, i.e., one conference can correspond to multiple events for subsequent search operations.

2) views.py:

The role of views.py file is similar to the ”C” in MVC pattern, which plays the role of project control. The views.py file can be called by urls.py to perform different operations on the database, so views.py plays the role of a bridge between front and back-end interaction. In the view.py file, three functions are defined, which are to display the list of all conferences, to display the list of all events, and to search the corresponding event information by date or room in the list of all events. The corresponding source code and explanations for these three functions are shown in Figure 9, Figure 10, and Figure 11.

- Display Conference

```python
# Display conference information
def display_conference(request):
    try:
        conference_list = Conference.objects.all()
        conference_context = {"conferencelist": conference_list}
        # Load Template
        return render(request, "system/conference/conlist.html", conference_context)
    except:
        return HttpResponse("Error: no conference information.")
```

Fig. 9. Source code for display conference list.

As shown in Figure 9, this function will fetch all the information within the conference table and then combine it with a given template to return a rendered HttpResponse object.

- Display Event

```python
# Display event information
def display_event(request):
    try:
        event_list = Event.objects.all()
        # Use filter
        event_filter = EventFilter(request.GET, queryset=event_list)
        event_list = event_filter.qs
        event_context = {"eventlist": event_list, "eventfilter": event_filter}
        # Load Template
        return render(request, "system/event/evelist.html", event_context)
    except:
        return HttpResponse("Error: no event information.")
```

Fig. 10. Source code for display event list.

The function shown in Figure 10, like the previous one, displays the full event information. On top of that, it adds the role of filtering, which can retrieve the corresponding event information from a conference and present it according to the user requests issued by the front-end.

- Search Event

The function shown in Figure 11 adds a search function to the display of the full list of event information, allowing the user to filter the results of queries the user wants to display based on the keywords entered.

```python
def search_event(request):
    key_word = request.GET.get('q')
    if isinstance(key_word, datetime.date()):
        to_filter = f'event_initial_date={key_word}'
    else:
        to_filter = f"event_room={key_word}"
    try:
        event_list = Event.objects.filter(to_filter)
        # Use filter
        event_filter = EventFilter(request.GET, queryset=event_list)
        event_list = event_filter.qs
        event_context = {"eventlist": event_list, "eventfilter": event_filter}
        # Load Template
        return render(request, "system/event/evelist.html", event_context)
    except:
        return HttpResponse("Error: no event information.")
```

Fig. 11. Source code for search event by day or room.

3) urls.py:

A URL is the entry point to a web service, and any request sent by a user through a browser will be sent to a specified URL address and then responded to. URL routing is represented in the Django project by the urls.py file.

```python
urlpatterns = [
    # Configure conference information
    path('conference', views.display_conference, name='display_conference'),
    # Configure event information
    path('event', views.display_event, name='display_event'),
]
```

Fig. 12. Source code for urls.

The URL routing is configured within this file. As shown in Figure 12, the first element within the configuration path is the name used when accessing the URL, and the second element is the name of the defined function that needs to be looked up once the path returns the views.py file. By searching for the value in the third element, people can find its corresponding URL address, which is what reflection does.

B. Front-end

The above is the back-end part. After the back-end is completed, the third step is to build the front-end part. As with the previously mentioned system folder, the templates folder for the front-end is created in parallel. Then a folder with the same name as the folder including the back-end function code is created under the templates folder, and then different folders and .html files can be created according to different needs to realize the front-end development.

TABLE I
MAIN ATTRIBUTES FOR IMPLEMENTATION

Conference Name	Date	Event Name	Date	Room
conference_1	2022/07/25	event1_1	2022/07/25	room_1
conference_2	2022/07/25	event2_1	2022/07/25	room_2
	2022/07/26	event2_2	2022/07/26	room_1
conference_3	2022/07/25	event3_1	2022/07/25	room_3
	2022/07/26	event3_2	2022/07/26	room_3

To implement the functionality mentioned in the third section, two front-end pages are needed, there are buttons on both pages to enable interoperability of the pages. The corresponding source code and explanations will be given in the

subsequent content. In order to demonstrate the functionality of the conference management system more clearly, a sample was designed and the values of the main attributes involved in the implementation are shown in Table 1.

1) conlist.html:

As shown in Figure 5, the conlist.html page displays a list of all conferences, and by clicking on the event detail in the last column, people can also view information about all events under that conference.

```
<body>
    <h1 style="font-size:45px" class="threed">Concordia Conference System</h1>

    <div id="href">
        <ul class="item">
            <li><a href="{% url 'display_conference' %}" target="_blank">Conference</a></li>
            <li><a href="{% url 'display_event' %}" target="_blank">Event</a></li>
        </ul>
    </div>

    <div id="conference_information">
        <h2 style="font-size:35px">Conference Information</h2>
        <table width = "900" border = "1" style="background-color:rgba(255, 255, 255, 0.5)">
            <tr>
                <th style="font-size:20px">Conference name</th>
                <th style="font-size:20px">Initial date</th>
                <th style="font-size:20px">End date</th>
                <th style="font-size:20px">Events</th>
            </tr>
            {% for conference in conferencelist %}
                <tr>
                    <td style="font-size:20px">{{conference.conference_name}}</td>
                    <td style="font-size:20px">{{conference.conference_initial_date}}</td>
                    <td style="font-size:20px">{{conference.conference_end_date}}</td>
                    <td>
                        <a href="event?conference={{ conference.id }}" style="font-size:18px">Event Detail</a>
                    </td>
                </tr>
            {% endfor %}
        </table>
    </div>
</body>
```

Fig. 13. Source code for conlist.

The main source code of this page is shown in Figure 13. The main method is to use a for loop to display all the relevant information in the database in order in the table. When clicking the event detail button, it will display all the event information within a particular conference accordingly, based on the one-to-many relationship set by the foreign key. The source code for beautifying the page using CSS can be found in the source code folder.

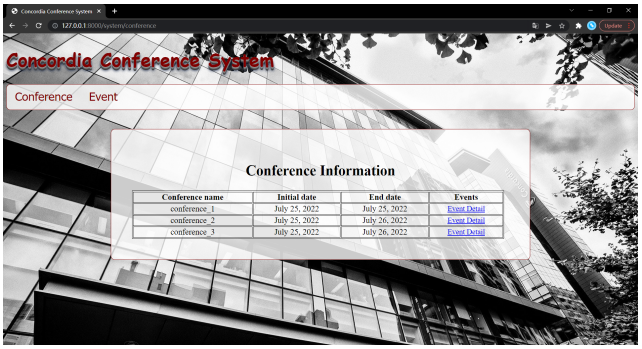

Fig. 14. Effect of listing all the conference information.

According to the sample information set in Table 1, the effect of listing all the conference information can be obtained as shown in Figure 14. By clicking on the event detail button of conference_1, all the event information in that conference can be obtained as shown in Figure 15.

2) evelist.html:

The evelist.html page as shown in Figure 6 can display all the event information. There are two search buttons above the

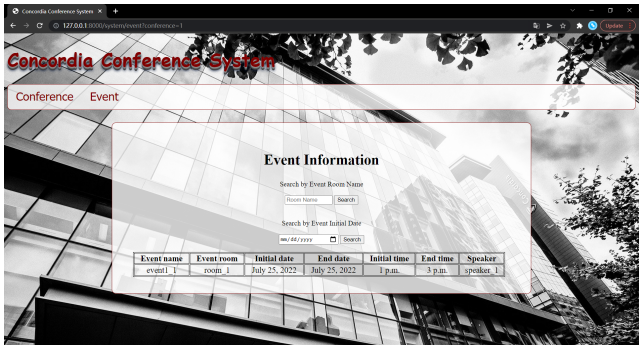

Fig. 15. Effect of listing event details of a specific conference.

information table, and people can retrieve the corresponding event by using the date and room number respectively according to the information prompted in the search box.

```
<body>
    <h1 style="font-size:45px" class="threed">Concordia Conference System</h1>

    <div id="href">
        <ul class="item">
            <li><a href="{% url 'display_conference' %}" target="_blank">Conference</a></li>
            <li><a href="{% url 'display_event' %}" target="_blank">Event</a></li>
        </ul>
    </div>

    <div id="conference_information">
        <h2 style="font-size:35px">Event Information</h2>
        <p style="font-size:17px">Search by Event Room Name</p>
        <div class="row">
            <div class="col">
                <div class="event search based on room name">
                    <form method="get" action="/system/event">
                    <input type="search" name="event_room" placeholder="Room Name" required style="width: 115px;">
                    <button type="submit">Search</button>
                    </form>
                </div>
            </div>
        </div>
        <br>

        <p style="font-size:17px">Search by Event Initial Date</p>
        <div class="row">
            <div class="col">
                <div class="event search based on event date">
                    <form method="get" action="/system/event">
                    <input type="date" name="event_initial_date" placeholder="Event Date e.g. 2022-07-25" required>
                    <button type="submit">Search</button>
                    </form>
                </div>
            </div>
        </div>
        <br>
        <table width = "900" border = "1">
            <tr>
                <th style="font-size:20px">Event name</th>
                <th style="font-size:20px">Event room</th>
                <th style="font-size:20px">Initial date</th>
                <th style="font-size:20px">End date</th>
                <th style="font-size:20px">Initial time</th>
                <th style="font-size:20px">End time</th>
                <th style="font-size:20px">Speaker</th>
            </tr>
            {% for event in eventlist %}
                <tr>
                    <td style="font-size:20px">{{event.event_name}}</td>
                    <td style="font-size:20px">{{event.event_room}}</td>
                    <td style="font-size:20px">{{event.event_initial_date}}</td>
                    <td style="font-size:20px">{{event.event_end_date}}</td>
                    <td style="font-size:20px">{{event.event_initial_time}}</td>
                    <td style="font-size:20px">{{event.event_end_time}}</td>
                    <td style="font-size:20px">{{event.event_speaker}}</td>
                </tr>
            {% endfor %}
        </table>
    </div>
</body>
```

Fig. 16. Source code for evelist.

The main source code of this page is shown in Figure 16. Like the conference information list page, the event information list uses a for loop to get all the event information. The effect of listing all the event information can be obtained as shown in Figure 17.

The form is then used to collect the user input and provide a submit button to transmit the data to the server. We use two forms that allow retrieving the corresponding event information by date and room number. The effect of filtering the corresponding event information by date and room number is shown in Figure 18 and Figure 19, respectively.

Fig. 17. Effect of listing all the event information.

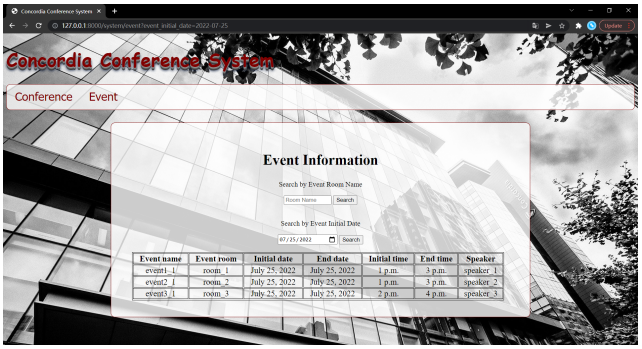

Fig. 18. Effect of filtering the event based on date.

As can be seen from the display effect in Figure 18 and Figure 19, the conference management system can filter and display the corresponding event information by date or room number without any problem.

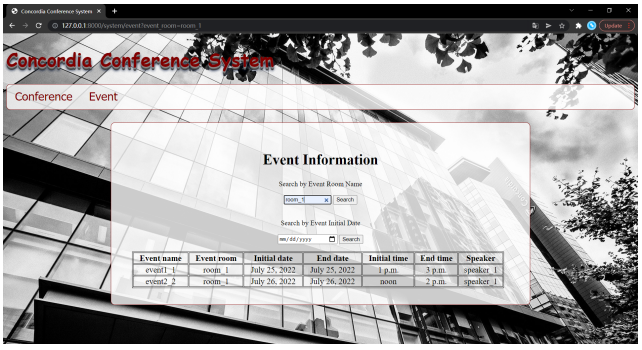

Fig. 19. Effect of filtering the event based on room.

These are all the diagrams of all the problems covered in Assignment 1 and their corresponding code and explanations. The complete source code is in the assignment folder, which contains a readme file describing how to run the program and the database-related operations.