Campus Event Management System

Comprehensive Report

Database Systems Course Project

Deshna Thunga - CS22B1020
Avanee Lakare - CS22B1080
Nimisha Thallapally - CS22B1082
Jyolsna J Isac - CS22B2034
Megha Hebbar- CS22B2039

Contents

1 Introduction

The Campus Event Management System (CEMS) project seeks to establish a centralized hub where students and staff of the Institute can access information regarding events like cultural festivities, academic seminars, sports competitions, and alumni gatherings taking place on campus.By consolidating these event details into a single platform, the aim was to provide easier access to event information.
This report provides an overview of the methodology, implementation, and technology stack used in developing such a system.

Our Github Repository Link - https://github.com/nimishathallapally/DBMS
In the src directory, run command python manage.py runserver to run the application locally in your system!

2 Workflow

In our project design and development processes, we implemented simple methodologies to ensure consistency and to streamline our workflow.
By establishing clear guidelines and procedures, we were able to maintain a consistent level of quality throughout the project lifecycle.
Additionally, regular reviews and feedback sessions were conducted to monitor progress and address any deviations from the decided procedures.

2.1 Database Design Methodology

Database design methodologies ensure that the database structure efficiently supports the requirements of the application.
We used the Entity-Relationship Model (ERM) to visualize and analyze the entities and relationships within a system. The ER diagram we came up with is given below in Figure 1.

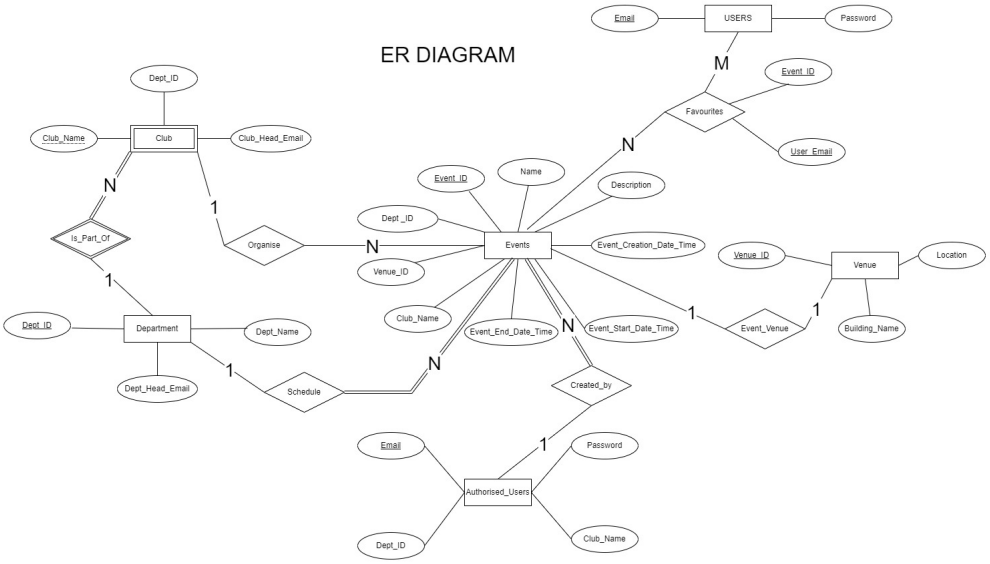


Figure 1: ER Diagram

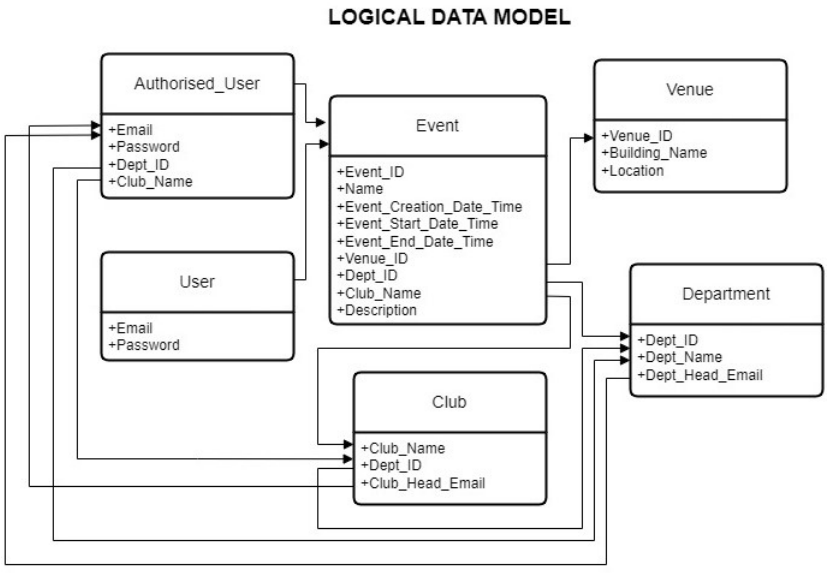


Figure 2: Logical Model

In designing our website and structuring our database, we utilized primary keys and foreign keys as outlined in our logical data model. This approach helped us connect different components required for the website, ensuring smooth functionality and efficient data management.

Then we used the Top-Down Methodology, to derive the logical and physical models from the conceptual model.We utilized the ER diagram as a reference to derive the tables in our database. This approach allowed us to accurately translate the conceptual model into a practical database structure. By analyzing the entities, relationships, and attributes represented in the ER diagram, we were able to identify the necessary tables and their corresponding fields. Each entity in the diagram was mapped to a table, while relationships between entities were translated into foreign key constraints or a separate table depending on the cardinality of the relationship. By following this process, we created a database schema that closely aligned with the requirements of our application, facilitating data management and retrieval.

2.2 Project Management Methodology

For managing our project, we adopted the waterfall methodology, which is characterized by a sequential development process. This approach involves distinct phases such as requirements gathering, design, implementation, testing, and maintenance, each executed in a linear fashion. This provided us with a clear roadmap for project progression, allowing us to prioritize tasks and manage dependencies effectively.

3 Technology Stack

3.1 Backend: Django

We chose Django,a high-level Python web framework, for the backend part of our project because of its ease of use and powerful features. Its built-in features such as an Object-Relational Mapping(ORM) system, authentication, and admin interface allowed us to rapidly develop our application. Its extensive documentation and helpful community further supported our decision, providing us with ample resources and support throughout the development process. Overall, Django proved to be the ideal choice for our project, enabling us to focus on building innovative features that brought our ideas into practice.

3.2 Frontend: HTML, CSS, JavaScript, Bootstrap

HTML, CSS, JavaScript, and Bootstrap formed a powerful toolkit for building the frontend of our application. They provided the necessary structure, styling, interactivity, and responsiveness that was required for a user-friendly and visually appealing interface.

- They are universally supported by all web browsers, ensuring compatibility across different devices and platforms.
- They offer a high degree of flexibility and customization for building web interfaces which we utilized to create a unique and visually appealing design for our CEMS.
- We used Bootstrap for a responsive web design, that ensures our CEMS website adapts seamlessly to different screen sizes.It also provides pre-built components like buttons, forms, navigation bars, and grids, that allowed for faster development.

3.3 Database: MySQL

We used MySQL, a popular open-source relational database management system (RDBMS) for our project.

- MySQL integrates seamlessly with various programming languages that are commonly used for web development, in our case Python.

- It can handle large datasets efficiently, making it suitable for managing data as the number of events and users in our CEMS grows.
- There's a vast community of developers and resources available online that provide support and information that we used in case of any challenges.

3.4 Version Control: Git, GitHub

With Git as the underlying version control system and GitHub as the hosting platform, we were able to streamline our development process and enhance collaboration among team members.

- Git's distributed nature allowed each of us to work independently on our local copies of the codebase, while GitHub served as a centralized repository for seamless integration and sharing of code.
- Additionally, the version history provided by Git and GitHub served as a safety net, allowing us to revert to previous states of the codebase if needed.

This setup enabled us to track changes, manage branches, and merge code with ease, ensuring a smooth and organized development workflow.

4 Methodology

4.1 System Architecture

Our CEMS follows a three-tier architecture:

- Presentation Tier: Web-based user interface for users to interact with the system.
- Application Tier: Business logic layer responsible for processing requests and managing data.
- Data Tier: Database for storing information related to events,clubs,departments and users.

4.2 Backend

The back-end of the CEMS was developed using Django,whose components were utilized to handle the server-side logic, data management, and interaction with the database.The folder structure of the app made in Django is given in Figure 3.

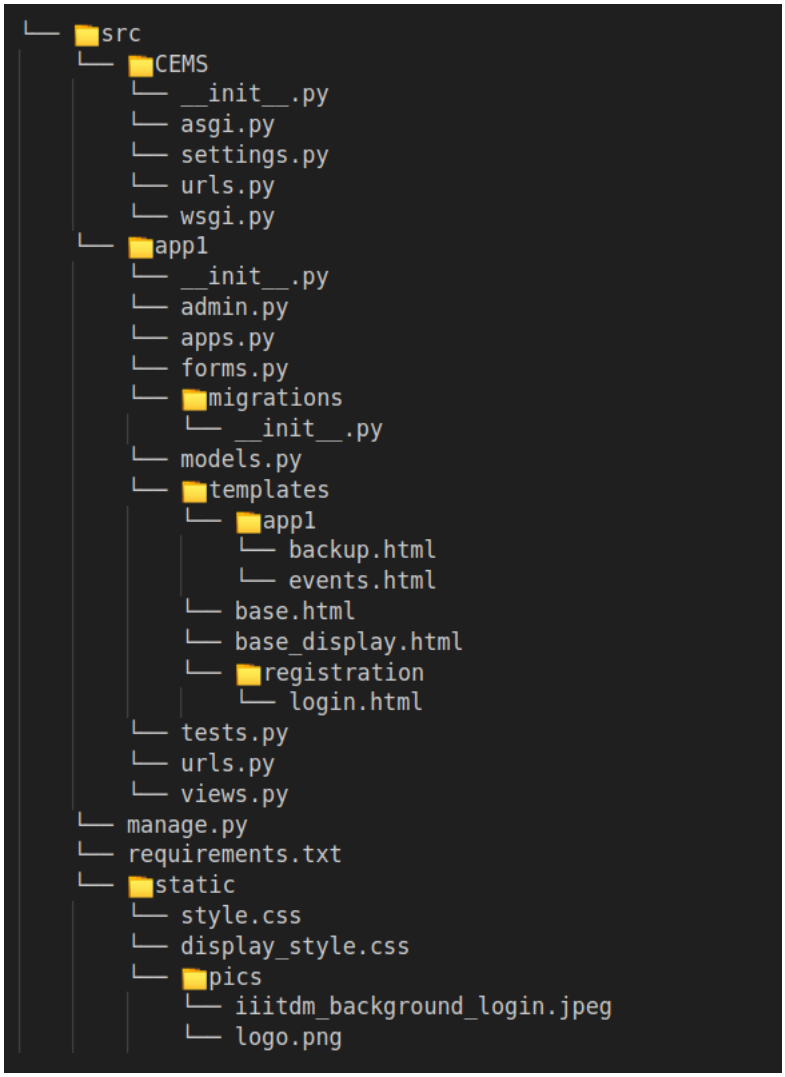


Figure 3: Folder structure of Application in Django

4.2.1 Models

Models in Django represent the data structure of the application. We used these model classes to create the tables in our database. Each model is defined in the models.py file of the Django app. The models we defined in our application are :

- **User :**
  By utilizing Django's built-in user model, we ensured that users could register, log in, and manage their accounts with ease. Additionally, Django's permissions system allowed us to define access levels and permissions for different user roles, ensuring that only authorized users could perform specific actions like adding, changing and deleting events.
- **Department :**

```python
class Department(models.Model):
    head = models.ForeignKey(User, models.DO_NOTHING,null=True)
    name = models.CharField(max_length=225)
```

Figure 4: Department Model

- **Club :**

```python
class Club(models.Model):
    name = models.CharField(unique=True, max_length=225)
    dept = models.ForeignKey('Department', models.DO_NOTHING)
    head = models.ForeignKey(User, models.DO_NOTHING,null=True)
```

Figure 5: Club Model

- **Venue :**

```python
class Venue(models.Model):
    building_name = models.CharField(max_length=225, blank=True, null=True)
    location = models.CharField(max_length=225, blank=True, null=True)
```

Figure 6: Venue Model

- **Event :**

```python
class Event(models.Model):
    # django has id for every model automatically so no need for specifying id
    name = models.CharField(max_length=225)
    description = models.TextField(blank=True, null=True)
    event_creation_date_time = models.DateTimeField(blank=True, null=True)
    event_start_date_time = models.DateTimeField()
    event_end_date_time = models.DateTimeField()
    club = models.ForeignKey(Club, models.DO_NOTHING, blank=True, null=True)
    venue = models.ForeignKey('Venue', models.DO_NOTHING, blank=True, null=True)
    dept = models.ForeignKey(Department, models.DO_NOTHING, blank=True, null=True)
    user = models.ForeignKey(User, models.DO_NOTHING,null=True)
```

Figure 7: Event Model

- **Favorites :**

```
class Favorites(models.Model):
    event = models.ForeignKey(Event, on_delete=models.CASCADE, related_name='fav', blank=True, null=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE, blank=True, null=True)
```

Figure 8: Favourites Model

4.2.2 Views

Views are Python functions that receive user requests made through a web browser and based on the request, it processes the information, performs the necessary logic, and generates an appropriate response. In our project the responses generated are HTML templates or JSON response.The view functions are defined in views.py

- **Logout view :** This function logs out the current user and redirects them to the home-page, by using Django's built-in logout() function.
- **Add event view :** This function allows authorized users (authorization done by django function login required()) , to add events. It creates a form instance (defined in forms.py) and populates the form based on the data received from the user.It then validates the form and saves the event if it is valid.
- **List club view :** This function retrieves valid Club objects to which an authorized User can add events to. This function returns a JSON response which is used to populate the club dropdown box in the add events page.
- **List venue view:** This returns a JSON response of available venues based on the provided start and end time of the events. This ensures that the venue chosen while adding an event does not clash with the venue of another event with overlapping time schedule.
- **Delete event view :** This function takes the eventID parameter from the URL and deletes the corresponding event from the database.
- **Events view :** This function displays events for a specific date and department. This view is used to filter the events based on date, department and favorite parameters as specified by the user.
- **Get agenda view :** This function returns events for a specific date to the Events View function.
- **Favorite event view :** This function adds an event favorited by an authorized user to the favorites table and returns a JSON response to indicate the successful execution.
- **Unfavorite event view :** This function removes an event favorited by an authorized user from the favorites table and returns a JSON response to indicate the successful execution.

4.2.3 Templates

Templates in Django are used to generate HTML dynamically based on data from views and models. Some templates we used in our project are:

- Home Page: The home page is the main page of the application.It displays all events such that the can be seen by anyone without logging in. It redirects you to the login page through the login button.
- Login Page: This page is used for users to enter their username and passwords which after successful authentication is redirected to the events page.
- Events Page: After authorization, Users are directed to this page. It provides a wide range of functionalities including favoriting and unfavoritng an event, filtering events based on department and favorites, displaying events on current day and upcoming days and allows authorized users to delete events. The add event button for redirecting to add events page is provided to authorized users.The logout button logs out the user and redirects to the Home-Page.
- Add Event Page: This page allows authenticated users to add new events. It has functionalities which include only displaying venues available at the time slot entered by the user and displaying clubs which are managed by the user. The cross button redirects you to the Events page.

4.2.4 URL

The URL configuration for the Website is given in the url.py file in the root directory.This urls.py file defines the URL patterns for routing requests within the application, including authentication URLs and custom app views related to events, venues, clubs, etc. The following are the url links used

- path(", views.events,name='Home-Page') - Maps the root URL to the events view with the name 'Home-Page'.
- path("accounts/", include("django.contrib.auth.urls")) - Includes authentication-related URLs provided by Django (login/, logout/, password_reset/, etc.).
- path('logout/', views.logout_view, name='logout') - Maps the logout/ URL to the logout_view view for user logout functionality.
- path('events/', views.events) - Maps to the events view after user login.
- re_path('events/(?P⟨datestr⟩[0-9a-zA-Z]+)/$', views.events) - Maps to the events view with a dynamic date parameter.
- re_path('events/(?P⟨datestr⟩[0-9a-zA-Z]+)/(?P⟨selector⟩[0-9a-zA-Z]+)/$', views.events) - Maps to the events view with dynamic date and selector parameters.
- re_path('events/fav/(?P⟨eventid⟩[0-9]+)/$',views.fav_event) - Maps to the fav_event view with a dynamic event ID.
- re_path('events/unfav/(?P⟨eventid⟩[0-9]+)/$',views.unfav_event) - Maps to the unfav_event view with a dynamic event ID.
- path('events/add/', views.add_event,name='add_event')- Maps to the add_event view for adding event functionality.
- path('events/listvenue/', views.list_venue, name='listvenue') - Maps to the list_venue view to list available venues.
- path('events/listclub/', views.list_club, name='listclub') - Maps to the list_club view to list available clubs.

- re_path('events/delete/(?P⟨eventid⟩[0-9]+)/$',views.delete_event) - Maps to the delete_event view with a dynamic event ID to delete event.
- path('admin/', admin.site.urls) - This pattern routes requests with the path admin/ to the Django admin site.

4.2.5 Form

The 'NewEvent' form is a well-structured Django 'ModelForm' that leverages the power of Django's forms framework to create, validate, and render form fields based on the 'Event' model. It uses custom widgets to enhance the appearance and usability of the form, ensuring a better experience for users when creating or updating events.This class is defined in the forms.py file in the root directory.

4.2.6 Admin

Creating Users and adding Clubs, Departments and Venues are done through the Django admin site.These tables can only be populated by a super user which in our case we have made through the command - python manage.py createsuperuser. 'admin@iiitdm.ac.in' is the superuser created for overseeing and managing the database.Other key features include

- We have a group called "authorised" which has permissions to create, edit, delete, and view events.All club and department email IDs belong to this group and inherit these permissions.
- The next set of users are users of IIITDM who only have view permissions and can favorite or unfavorite events.
- Users who are not logged in can only view the page and see upcoming events but cannot favorite or unfavorite any event.

The admin.py file in our project registers the Department, Club, Event, and Venue models with the Django admin site.

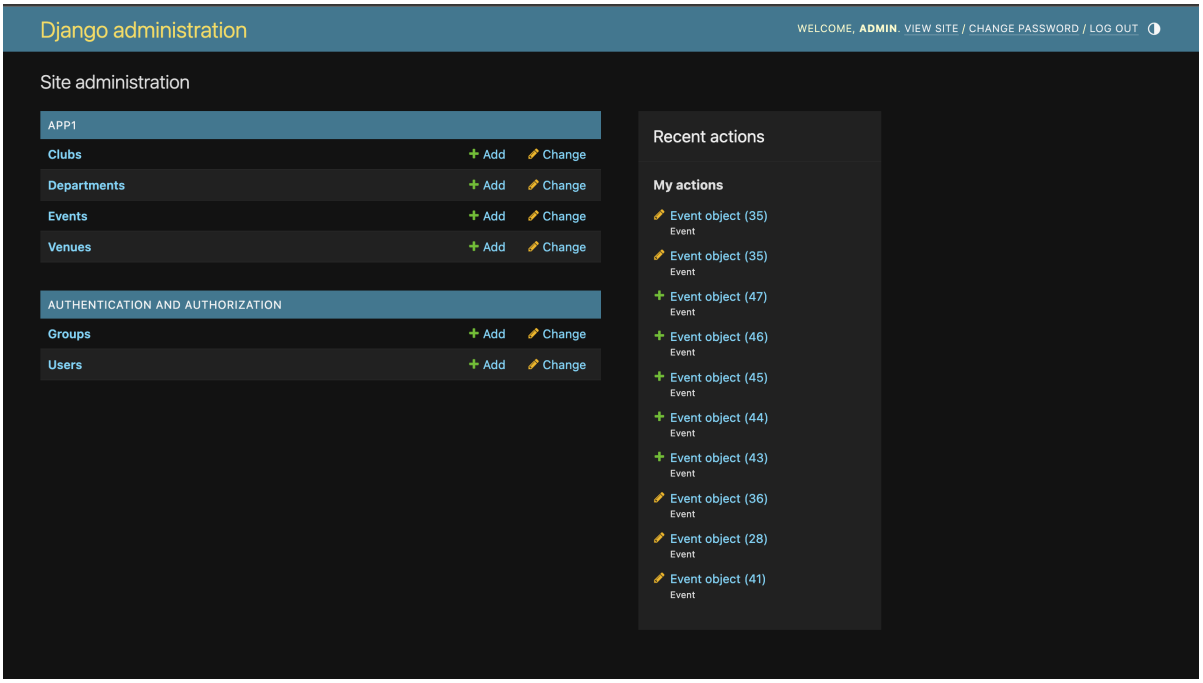


Figure 9: Tables in the Database

By following this structure, we ensure that users and groups are managed effectively, with appropriate permissions assigned to each group, and the models are accessible through the Django admin site for administrative tasks.

4.3 Database

We used a MySQL setup to create a CEMS database. From the ER diagram we derived the tables along with their respective columns that has to be included in the database. Each table corresponds to an entity and has columns representing its attributes.

4.3.1 Tables

Tables are defined in the Django framework as instances of Django class Model. These models are defined in the python file models.py. We run the command migrate as an argument while running manage.py to migrate or create the corresponding tables in MySQL. Figure shows all tables created by Django in the CEMS database.

Figure 10: Tables in the Database

4.3.2 Population

The database is populated through the Django Administration site by the superuser. Only evennts can be added through the website by authorized users.Our initial Populated Data includes the following :

- User

```
mysql> select id,username,is_superuser from auth_user;
+----+--------------------------------+--------------+
| id | username                       | is_superuser |
+----+--------------------------------+--------------+
|  5 | admin                          |            1 |
| 14 | cs22b1020@iiitdm.ac.in         |            0 |
| 15 | cs22b1080@iiitdm.ac.in         |            0 |
| 16 | cs22b1082@iiitdm.ac.in         |            0 |
| 17 | cs22b2034@iiitdm.ac.in         |            0 |
| 18 | cs22b2039@iiitdm.ac.in         |            0 |
| 19 | academic.affairs@iiitdm.ac.in  |            0 |
| 20 | alumni.affairs@iiitdm.ac.in    |            0 |
| 21 | cultural.affairs@iiitdm.ac.in  |            0 |
| 22 | general.affairs@iiitdm.ac.in   |            0 |
| 23 | hostel.affairs@iiitdm.ac.in    |            0 |
| 24 | placement@iiitdm.ac.in         |            0 |
| 25 | sports@iiitdm.ac.in            |            0 |
| 26 | technical.affairs@iiitdm.ac.in |            0 |
| 27 | mess.affairs@iiitdm.ac.in      |            0 |
| 28 | auv.society@iiitdm.ac.in       |            0 |
| 29 | csclub@iiitdm.ac.in            |            0 |
| 30 | epic@iiitdm.ac.in              |            0 |
| 31 | gdsc@iiitdm.ac.in              |            0 |
| 32 | mars@iiitdm.ac.in              |            0 |
| 33 | optica@iiitdm.ac.in            |            0 |
| 34 | saeclub@iiitdm.ac.in           |            0 |
| 35 | ipd@iiitdm.ac.in               |            0 |
| 36 | asme@iiitdm.ac.in              |            0 |
| 37 | idc@iiitdm.ac.in               |            0 |
| 38 | ieeesb@iiitdm.ac.in            |            0 |
+----+--------------------------------+--------------+
26 rows in set (0.00 sec)
```

Figure 11: User Data

- Authenticated Groups and corresponding users belonging to groups

```
mysql> select * from auth_group;
+----+------------+
| id | name       |
+----+------------+
|  1 | authorised |
+----+------------+
1 row in set (0.00 sec)

mysql> select * from auth_user_groups;
+----+---------+----------+
| id | user_id | group_id |
+----+---------+----------+
|  1 |       3 |        1 |
|  3 |      19 |        1 |
|  4 |      20 |        1 |
|  5 |      21 |        1 |
|  6 |      22 |        1 |
|  7 |      23 |        1 |
|  8 |      24 |        1 |
|  9 |      25 |        1 |
| 10 |      26 |        1 |
| 11 |      27 |        1 |
| 12 |      28 |        1 |
| 13 |      29 |        1 |
| 14 |      30 |        1 |
| 15 |      31 |        1 |
| 16 |      32 |        1 |
| 17 |      33 |        1 |
| 18 |      34 |        1 |
| 19 |      35 |        1 |
| 20 |      36 |        1 |
| 21 |      37 |        1 |
| 22 |      38 |        1 |
+----+---------+----------+
21 rows in set (0.00 sec)
```

Figure 12: Permissions Data

- Department

```
mysql> select * from app1_department;
+----+-----------------------------+---------+
| id | name                        | head_id |
+----+-----------------------------+---------+
|  1 | Academic Affairs            |      19 |
|  2 | Alumni Affairs              |      20 |
|  3 | Cultural Affairs            |      21 |
|  4 | General Affairs             |      22 |
|  5 | Hostel Affairs              |      23 |
|  6 | Training and Placement Cell |      24 |
|  7 | Sports Affairs              |      25 |
|  8 | Technical Affairs           |      26 |
|  9 | Mess Affairs                |      27 |
+----+-----------------------------+---------+
9 rows in set (0.00 sec)
```

Figure 13: Department Data

- Club

```
mysql> select * from app1_club;
+-----+----------------------+---------+---------+
| id  | name                 | dept_id | head_id |
+-----+----------------------+---------+---------+
| 301 | Aurora               |       3 |      21 |
| 302 | Cinematics           |       3 |      21 |
| 303 | Esports              |       3 |      21 |
| 304 | Footlights           |       3 |      21 |
| 305 | Imagix               |       3 |      21 |
| 306 | Literati Society     |       3 |      21 |
| 307 | Music                |       3 |      21 |
| 308 | Quriosity            |       3 |      21 |
| 309 | Tamizh Saalaram      |       3 |      21 |
| 310 | Theatrix             |       3 |      21 |
| 311 | Spicmacay            |       3 |      21 |
| 312 | IndicSense           |       3 |      21 |
| 801 | AUV                  |       8 |      28 |
| 802 | CS CLUB              |       8 |      29 |
| 803 | EPIC                 |       8 |      30 |
| 804 | GDSC                 |       8 |      31 |
| 805 | MARS                 |       8 |      32 |
| 806 | OPTICA               |       8 |      33 |
| 807 | ROBOTICS             |       8 |      26 |
| 808 | SAE                  |       8 |      34 |
| 809 | TAD                  |       8 |      37 |
| 810 | IPD                  |       8 |      35 |
| 811 | ASME STUDENT SECTION |       8 |      36 |
| 812 | IEEE STUDENT BRANCH  |       8 |      38 |
| 813 | SYSTEM CODING CLUB   |       8 |      26 |
| 814 | Red Ribbon Society   |       3 |      21 |
+-----+----------------------+---------+---------+
26 rows in set (0.01 sec)
```

Figure 14: Club Data

- Venue

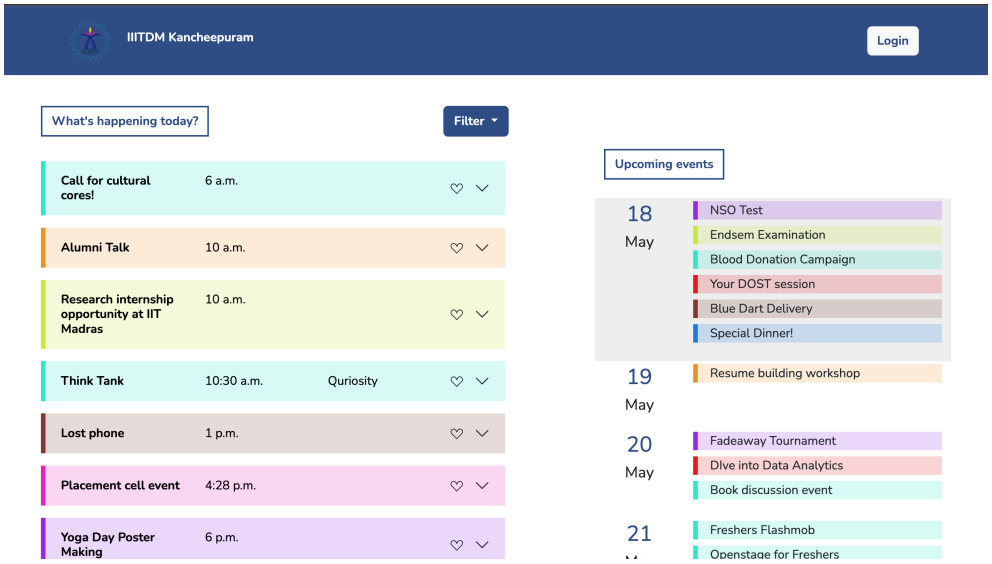
```
mysql> select * from app1_venue;
+----+----------------------+------------------+
| id | building_name        | location         |
+----+----------------------+------------------+
|  1 | Academic Block       | H05              |
|  2 | Academic Block       | H01              |
|  3 | Academic Block       | H02              |
|  4 | Academic Block       | H03              |
|  5 | Academic Block       | H04              |
|  6 | Academic Block       | H11              |
|  7 | Academic Block       | H12              |
|  8 | Academic Block       | H13              |
|  9 | Academic Block       | H14              |
| 10 | Academic Block       | H15              |
| 11 | Academic Block       | H21              |
| 12 | Academic Block       | H22              |
| 13 | Academic Block       | H23              |
| 14 | Academic Block       | H24              |
| 15 | Academic Block       | H25              |
| 16 | Academic Block       | H31              |
| 17 | Academic Block       | H32              |
| 18 | Academic Block       | H33              |
| 19 | Academic Block       | H34              |
| 20 | Academic Block       | H35              |
| 21 | Academic Block       | H41              |
| 22 | Academic Block       | H42              |
| 23 | Academic Block       | H43              |
| 24 | Academic Block       | H44              |
| 25 | Academic Block       | H45              |
| 26 | Arjuna Sports Complex | Basketball Court |
| 27 | Arjuna Sports Complex | Football Ground  |
| 28 | Arjuna Sports Complex | Cricket Ground   |
| 29 | Arjuna Sports Complex | Volleyball Court |
| 30 | Arjuna Sports Complex | Badminton Court  |
| 31 | Arjuna Sports Complex | Tennis Court     |
| 32 | Arjuna Sports Complex | Gym              |
| 33 | Admin                | Placement Office |
| 34 | Admin                | Seminar Hall     |
| 35 | Academic Block       | Old Library      |
| 36 | Akshaya              | Mess             |
+----+----------------------+------------------+
36 rows in set (0.00 sec)
```

Figure 15: Venue Data

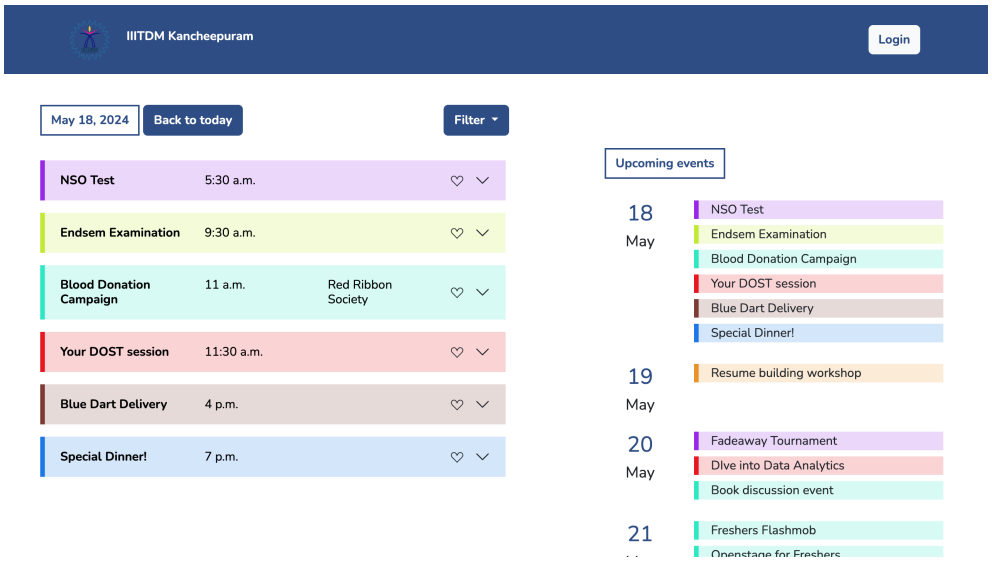
4.3.3 Functionalities

We have implemented various functionalities in our application which requires querying the database to get the required and relevant information. We have done querying using Django functions which translate into SQL queries and returns the data as Queryset variables. Django also helps us retrieve data with the help of foreign keys(indicated by '__')

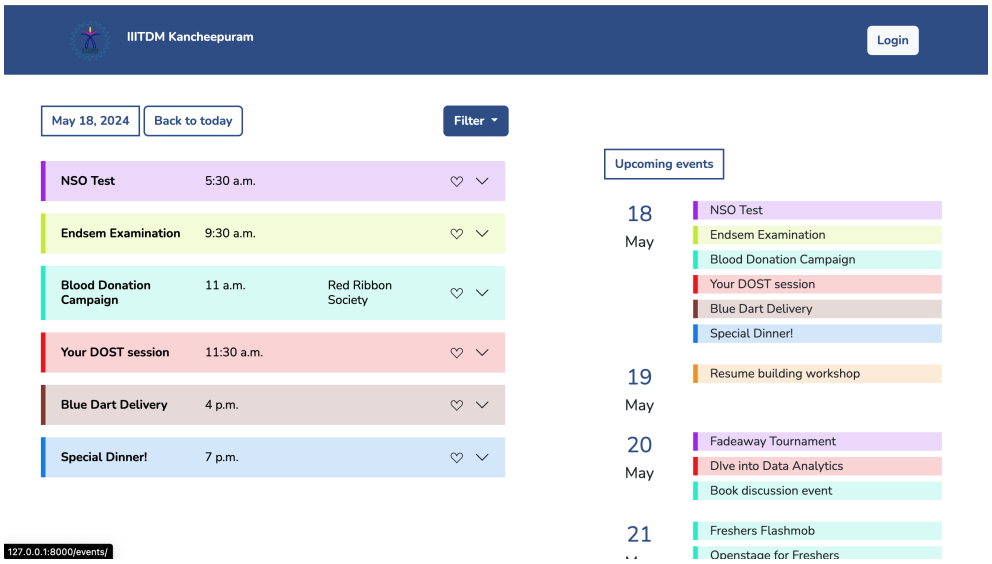
- Fetching Events occurring on given date - We are querying the Events table to retrieve events whose start date matches the given date Query - events_qs = Event.objects.filter(event_start_date_time__date=req_datetime)



(a) Hovering on 18th May



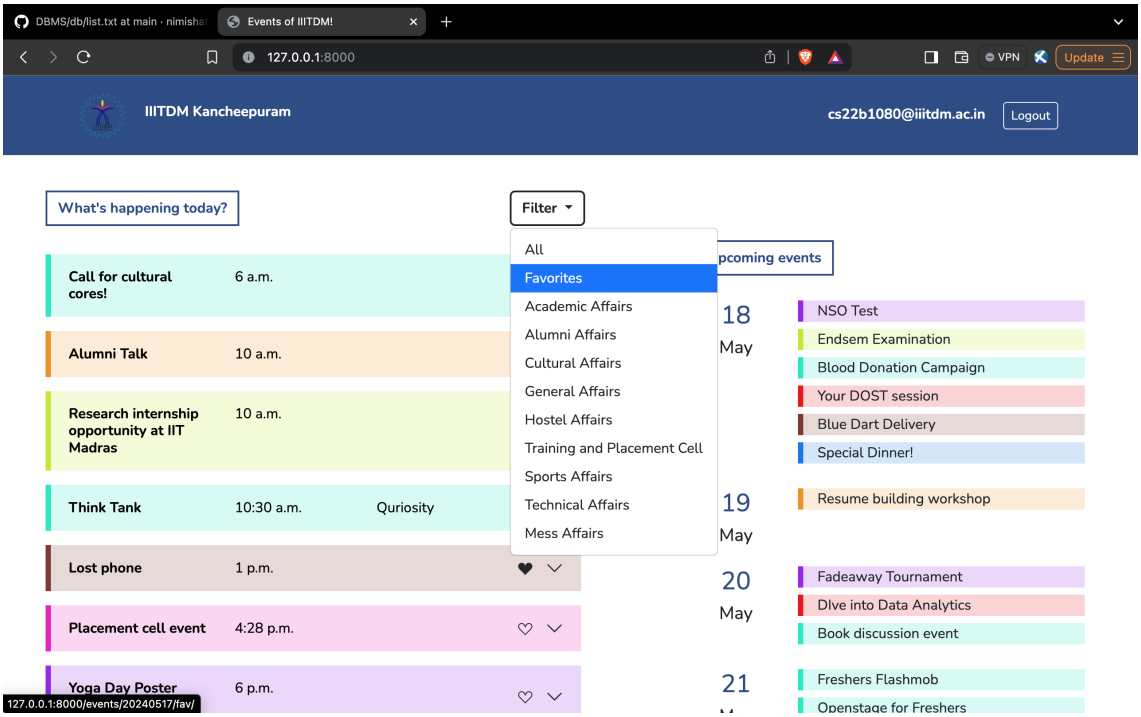
(b) After clicking 18th May



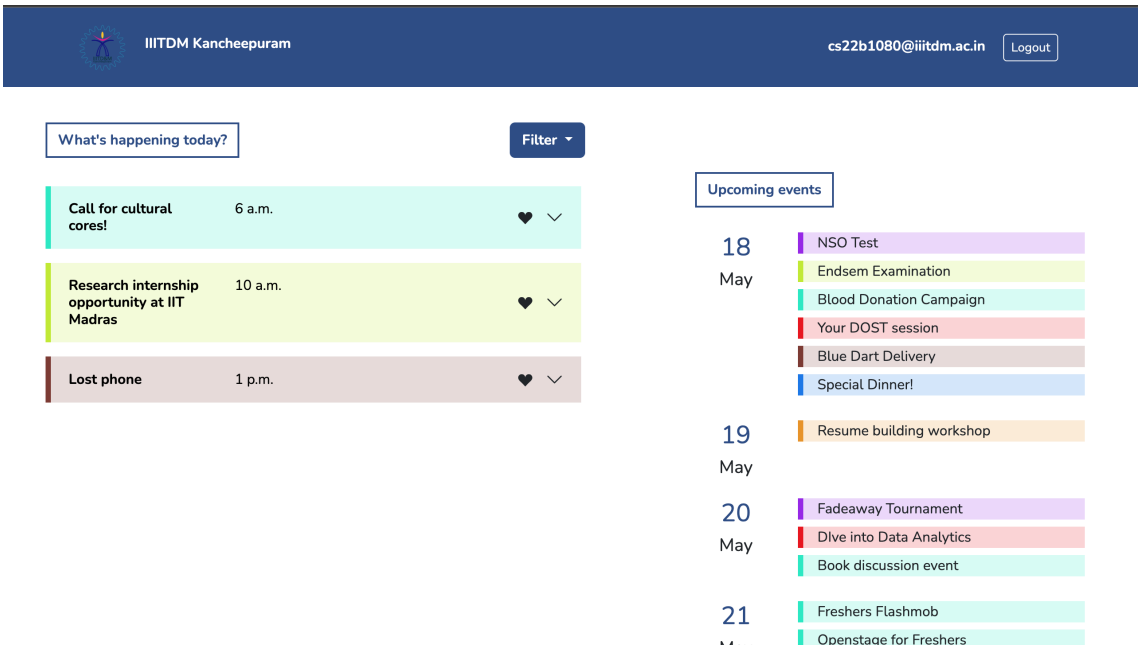
(c) Back to today button clickable to return to current date

Figure 16: Displaying events of another day

- Fetching Events Favourited by user - First we query the Favorites table to retrieve event_ids of events favourited by the user. Then we query the events table to retrieve events whose event_ids match.
  Query -
  my_fav_event_ids = list(Favorites.objects.filter(user=request.user, event__in=events_qs.all()).values_list("event_id", flat=True))
  events_qs = Event.objects.filter(id__in=my_fav_event_ids)



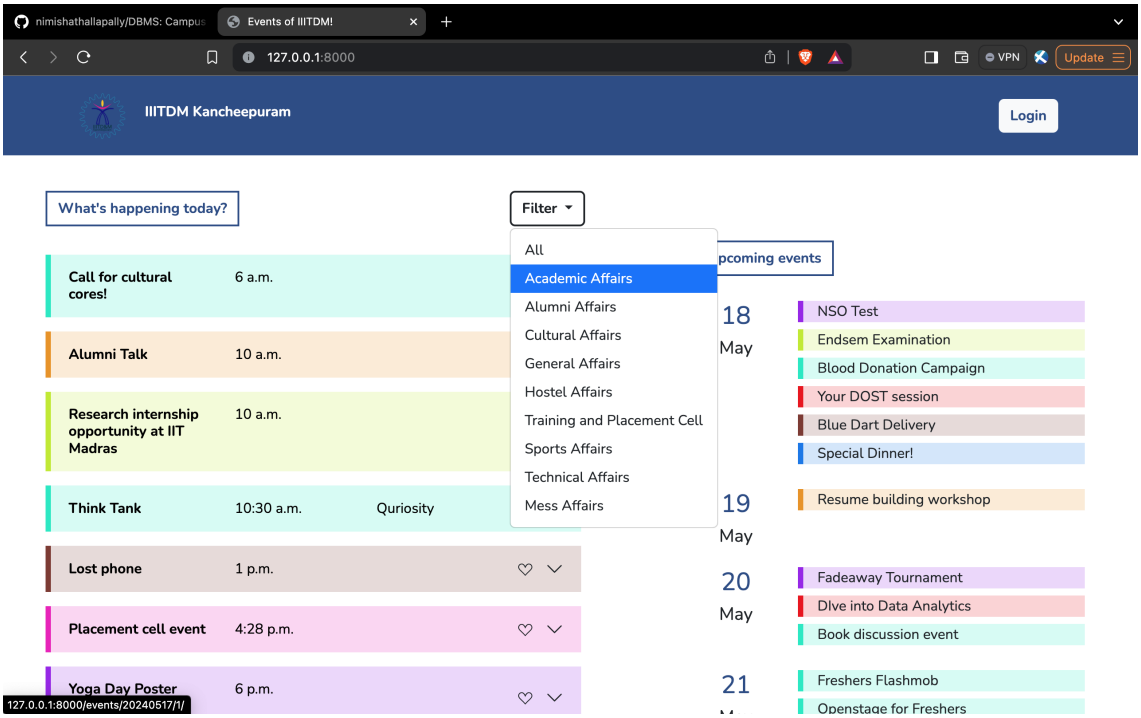
(a) showing filter for favorite



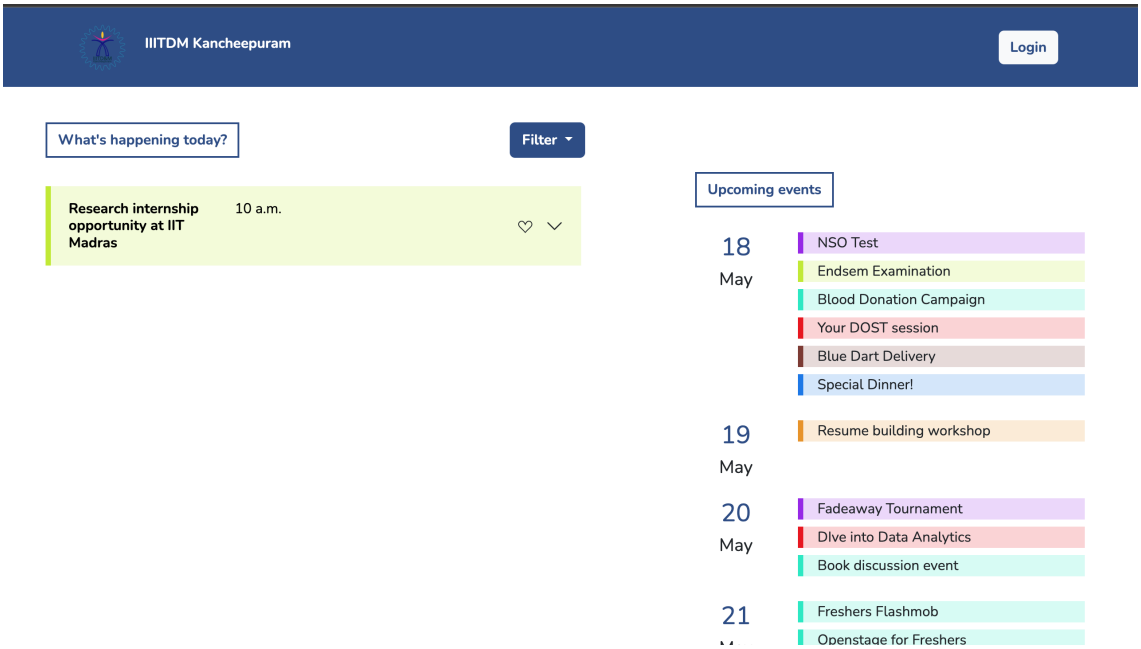
(b) After applying favorite filter

Figure 17: Applying filter for favorites

- Fetching Events of a particular department - We query the events table to retrieve events whose dept_id match the dept_id given.
  Query - events_qs = events_qs.filter(dept_id=selector)



(a) Applying filter for Department Academics

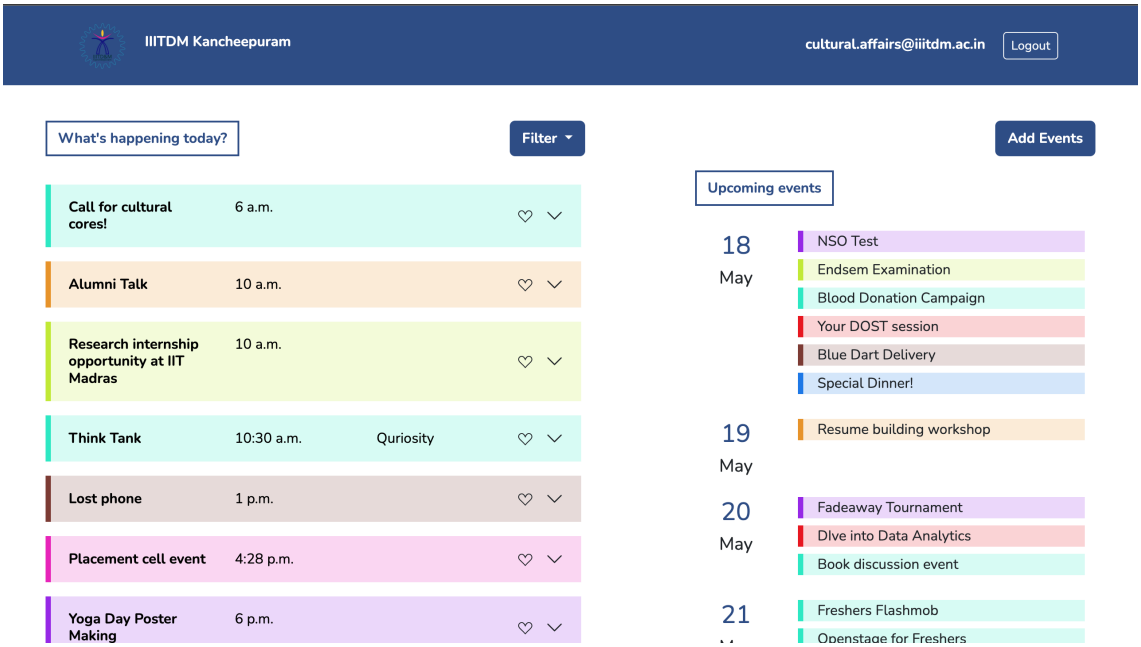


(b) After applying filter

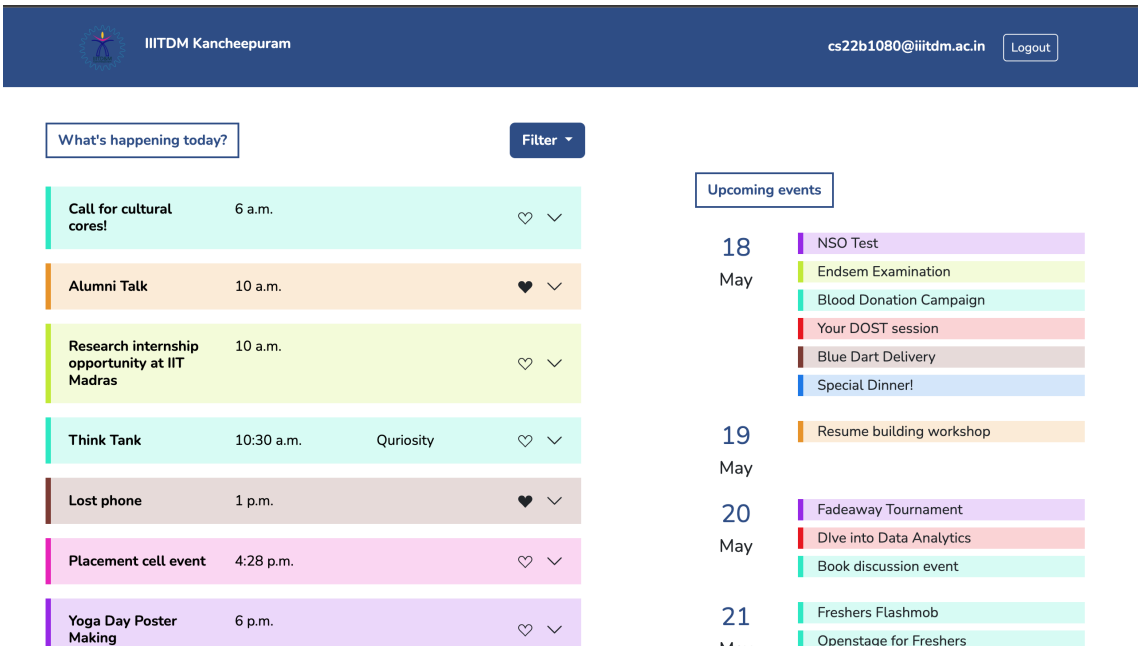
Figure 18: Applying filter for department

- The events fetched are ordered by their start date and time.
  Query - events_qs = events_qs.order_by('event_start_date_time')
- Checking Authorization to add events - Checks if there exists a row whose name is authorized in the list of all groups the user belongs to.

Query - request.user.groups.filter(name='authorised').exists()



(a) Add Event button available for authenticated user



(b) Add event not available for normal user

Figure 19: Add event button only for authorized users

- For adding event : To retrieve which department the event belongs to, we query the department table with head=user or club_head=user and club_dept_id=dept_id (for the later part there is a join with the department table and club table).
  Query - Department.objects.filter(head=request.user) — Department.objects.filter(club__head=request.user)

- For adding event : To retrieve list of clubs to which user can add events from we query the club table and check if head=user or clubs dept_id=dept_id and dept_head=user (for

the later part there is join with the club table and department table).
Query - Club.objects.filter(head=user) — Club.objects.filter(dept__head=user)

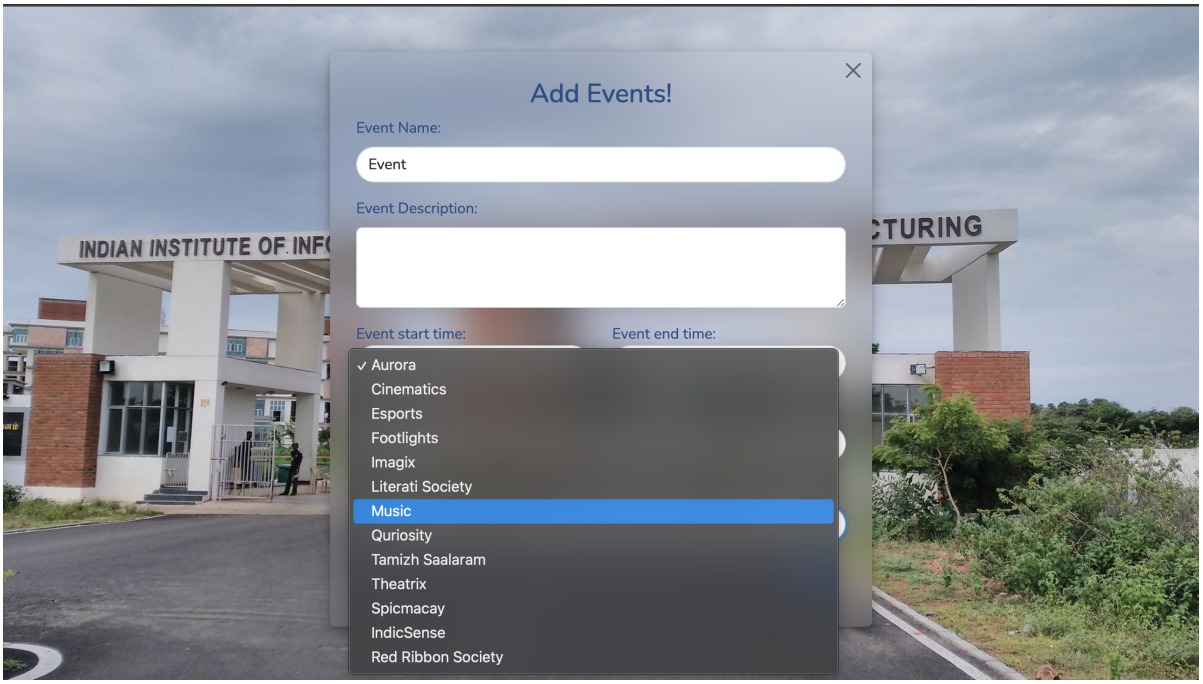


Figure 20: Clubs which are only available to Cultural Affairs have been listed

- For adding event : To retrieve list of venues available in the given time slot we query the venues table to find venues which are not booked by events in the given time slot(There is a join on tables Venue and Events).
  Query - Venue.objects.exclude(event__event_start_date_time__lte=et,event__event_end_date_time__gte=st)

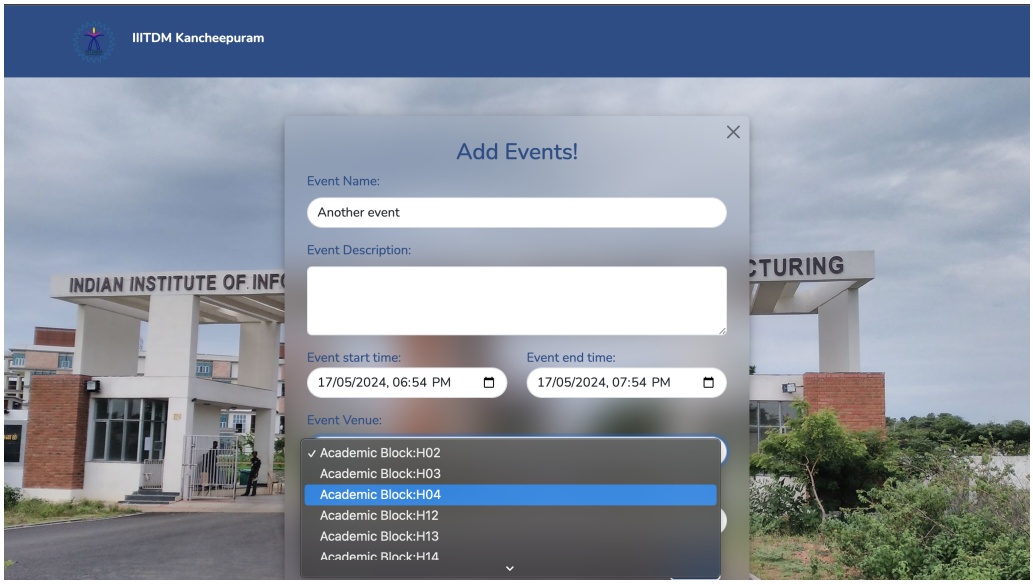
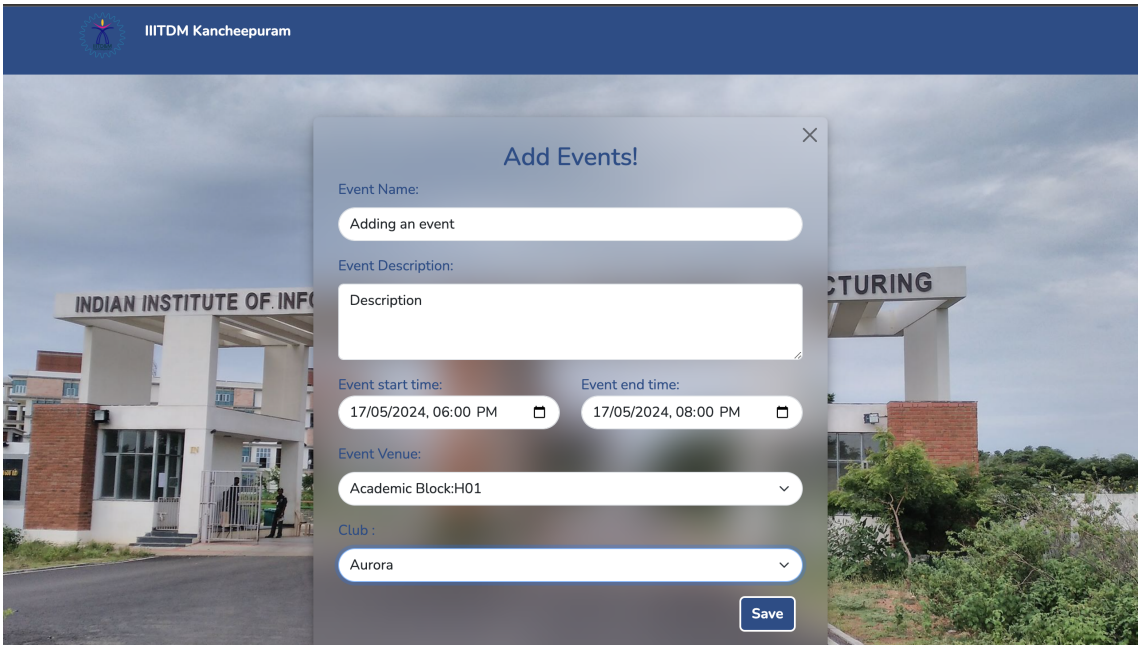
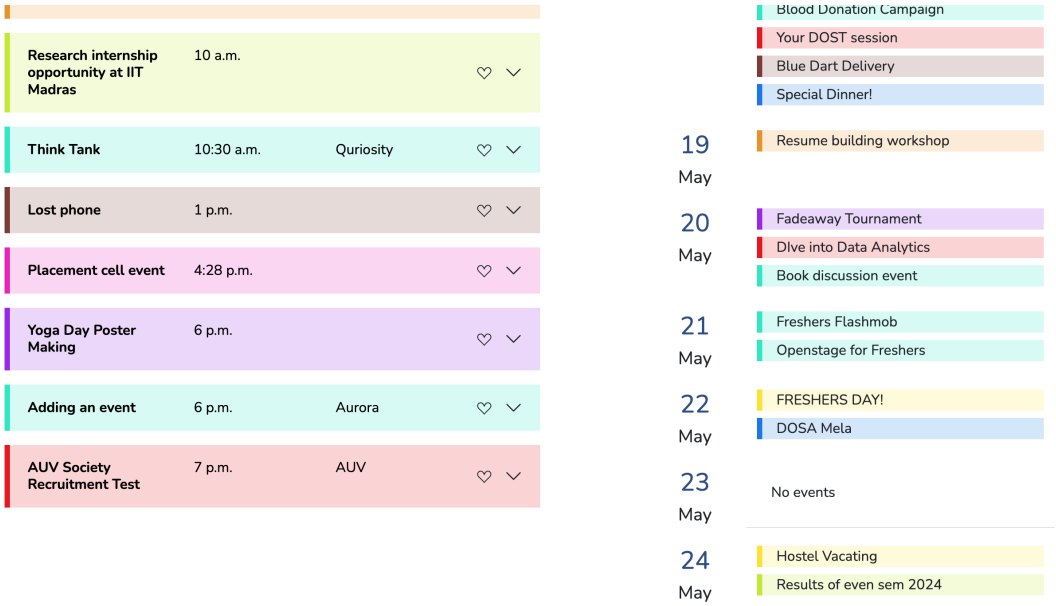


Figure 21: Venues available at the time clot between 6.54 and 7.54 PM on 17th May are listed

- Adding event : Event details are retrieved from the front-end using a Django Model form which after validation is saved in the database in the Events table.
  Code -
  event = NewEvent(ne)
  if event.is_valid():
  event.save()



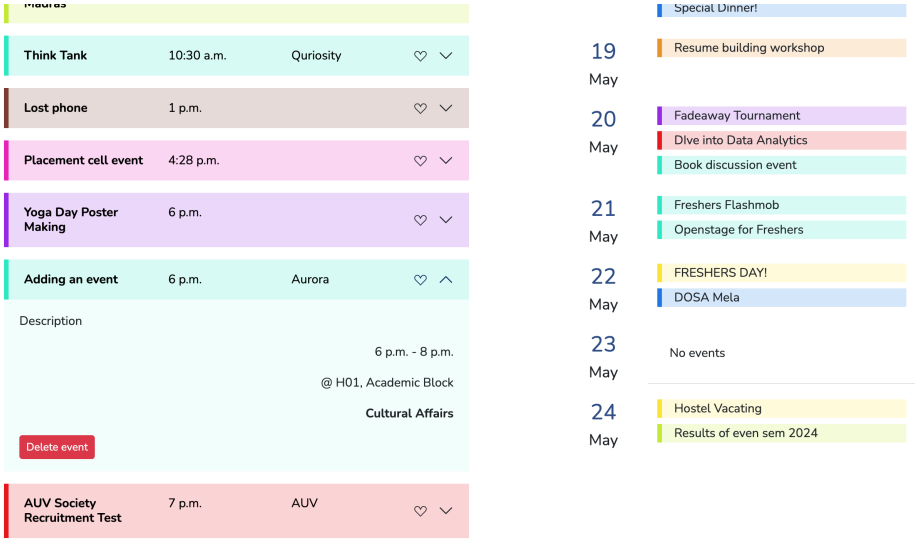
(a) Filling Event details for event



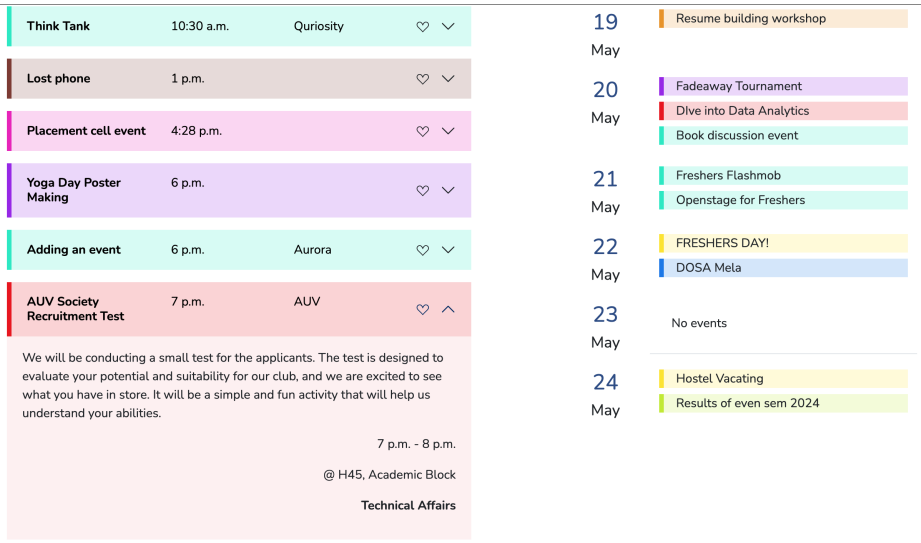
(b) Newly created Event listed in Events page

Figure 22: Adding Events

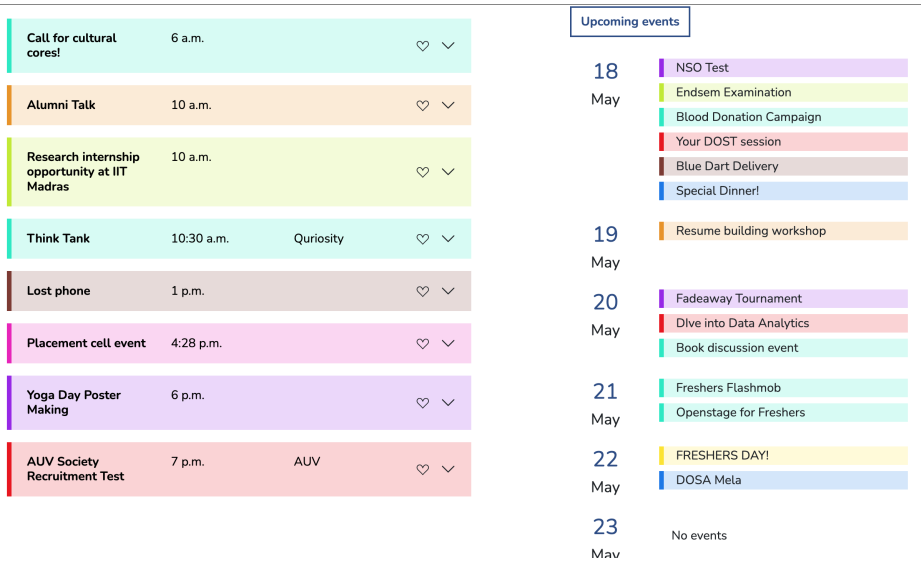
- Deleting event : Events are deleted from the events table after a match in event_id and user_id.
  Query - Event.objects.filter(user=request.user, id=eventid).delete()



(a) Delete option shown for Authorized user



(b) Delete option not shown for an event not of the department



(c) Event not listed after Deletion

Figure 23: Deleting an Event

- Favouriting events : An event is added to the Favourites table after retrieving the user_id and event_id.
  Query - event = Event.objects.filter(id=eventid).first()
  fav, created = Favorites.objects.get_or_create(user=request.user, event=event)

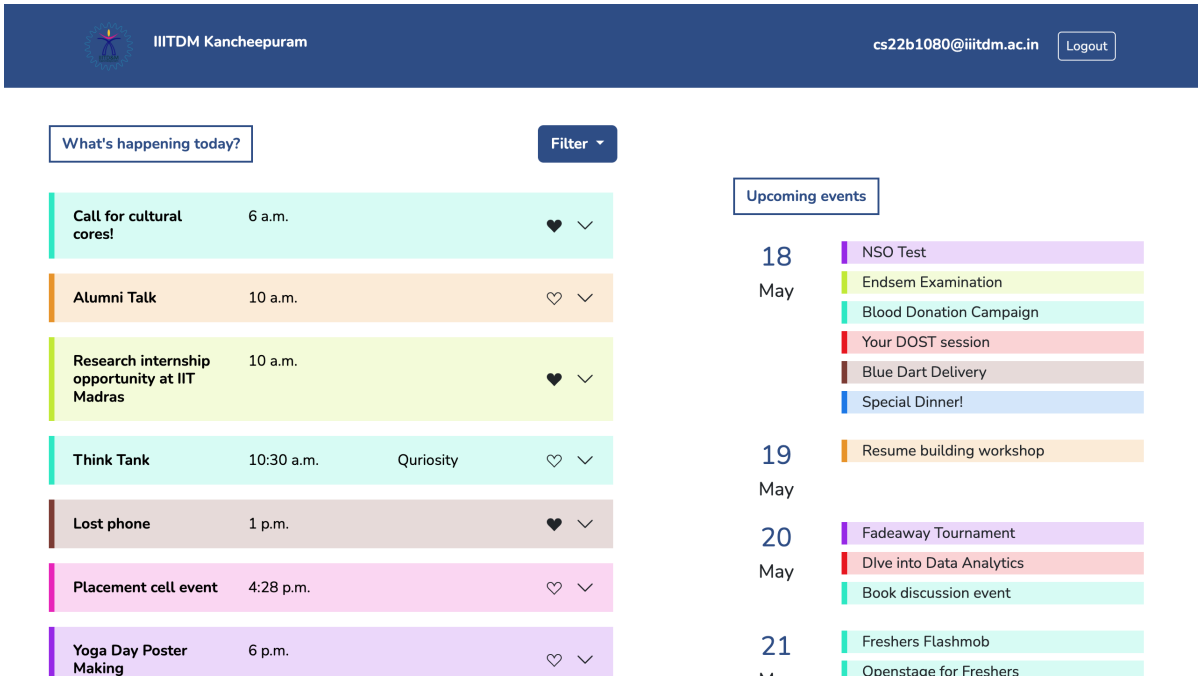
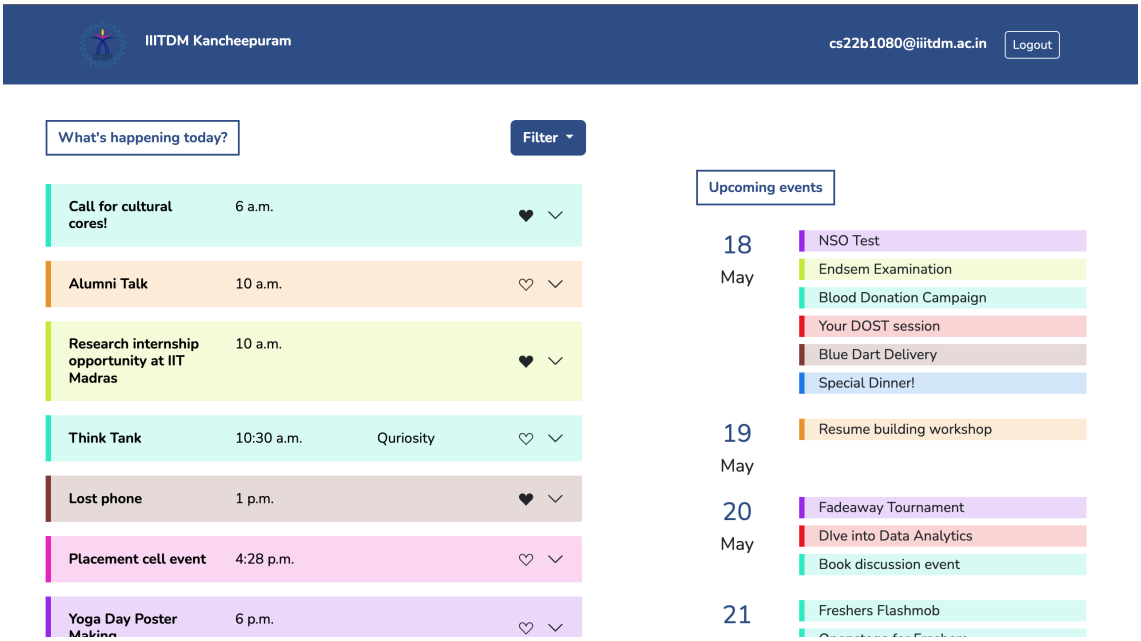
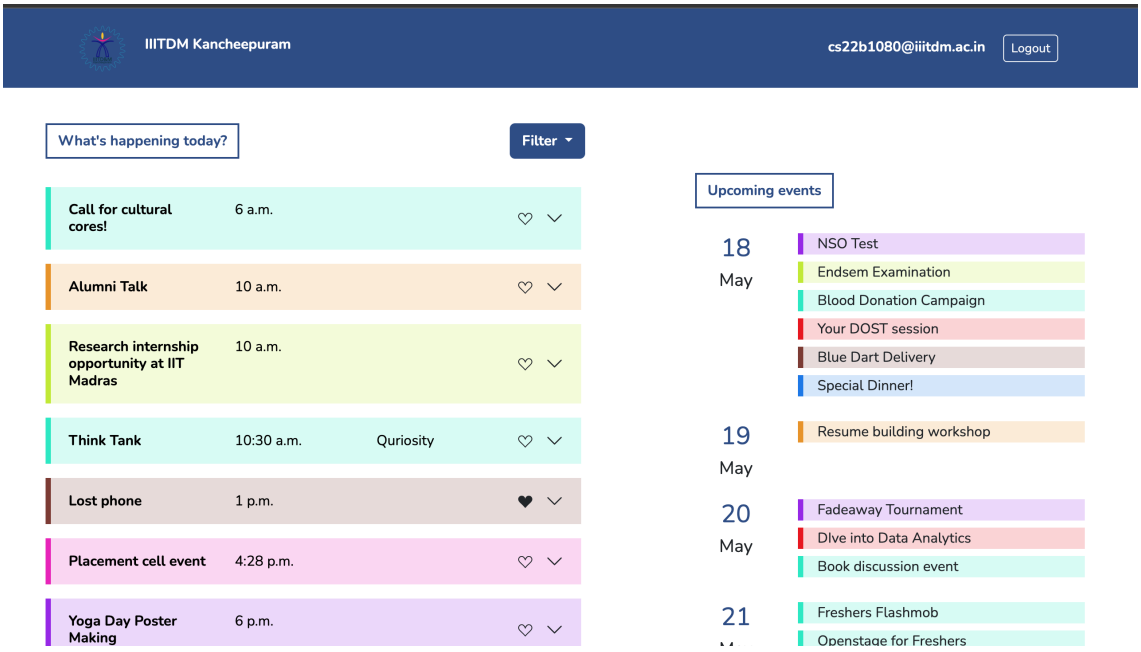


Figure 24: Favourited Events have the heart icon darkened

- Unfavouriting events : An event is removed from the Favourites table after retrieving the user_id and event_id.
  Query - event = Event.objects.filter(id=eventid).first()
  Favorites.objects.filter(user=request.user, event=event).delete()



(a) Favorited events



(b) The heart icon is not darkened anymore because the event is unfavorited

Figure 25: Unfavoriting Events

4.4 Frontend

4.4.1 UI Design

We used the Figma application to first visualize and develop the aesthetics and overall look and feel of the website. Figure 26 shows the initial design of the website.

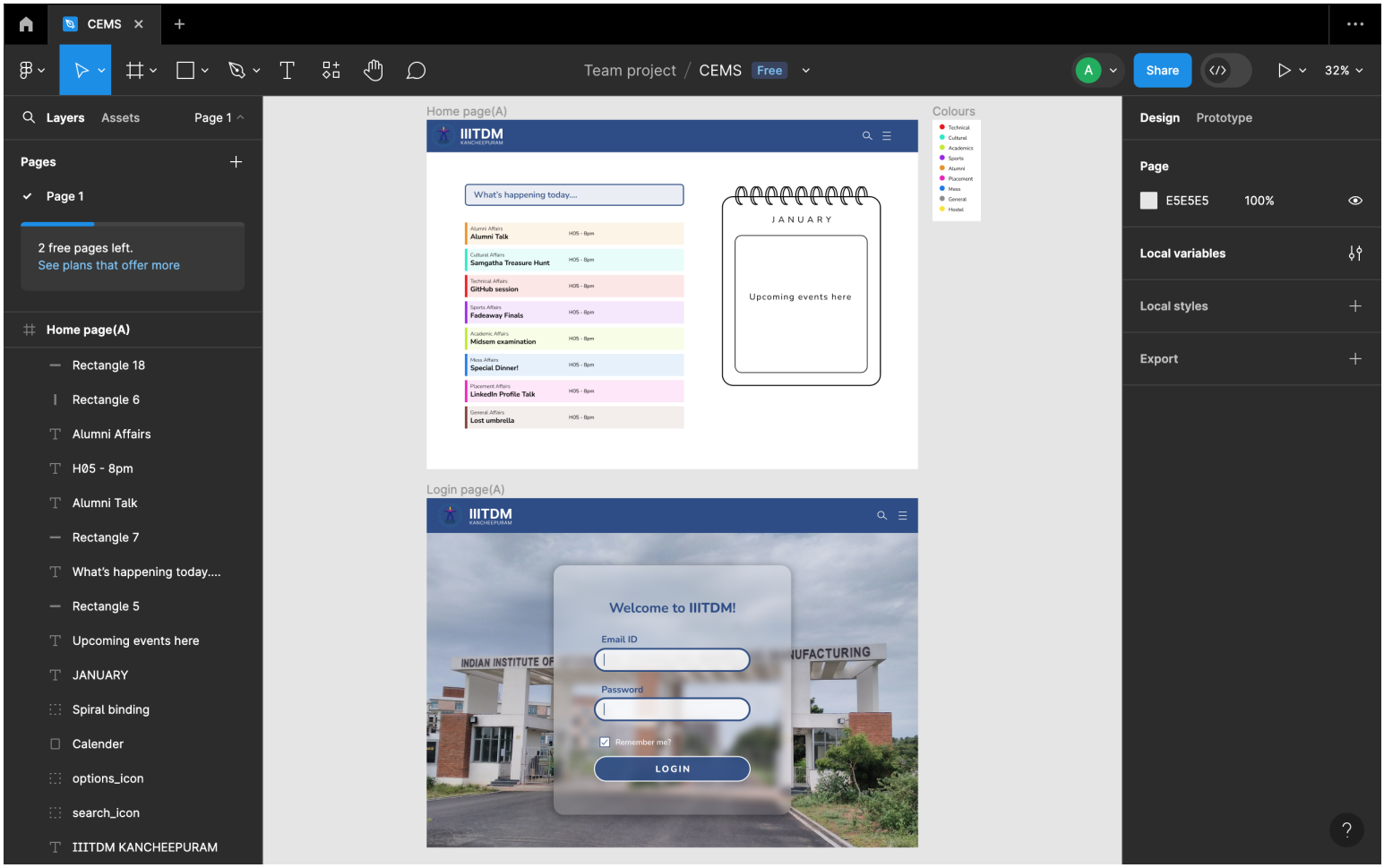


Figure 26: Figma file containing the initial design of website

- Simplicity: Our user interface is simple, and easy to use with minimal complexity.
- Colors: In our user interface, we utilized colors to distinguish between events conducted by various departments. This visual differentiation allowed users to quickly identify events associated with different departments, enhancing the usability and clarity of our application. The colors used are listed down below in their hex values.
  - Academic : #C0EB15
  - Alumni : #E99311
  - Cultural : #11E9C2
  - General : #7D3B32
  - Hostel : #FDE400
  - Placement : #E911BA
  - Sports : #9711E9
  - Technical : #E91111
  - Mess : #1175E9
- Consistency: We maintain consistency in the design elements (layout, fonts, colors) throughout the interface which creates a sense of familiarity and predictability for users.

- Use of base.html and base_display.html : This file provides the basic layout including background,header,font and colors to maintain uniformity among all webpages.
- Font-family : Nunito
- Text color : #2C4B87 (hex-value)
- Header : The header is a navigation bar containing IIITDM logo and the Institute name. The log in and log out button are displayed depending on the status of the user.

4.4.2 HTML,CSS,BOOTSTRAP AND JAVASCRIPT

We have used HTML,CSS,Bootstrap and Javascript to implement the frontend of our website.The following listsdown how we have incoprated these technologies in our website.

- HTML : Every webpage in our website is a .html file. All these files are stored in the templates folder in the root directory.
- Bootstrap : We have linked Bootstrap by referencing to Bootstrap5 links in our header element. We have used Bootstrap5 for CSS as well as Javascript elements.Some include the navigation bar,Time Date dynamic input in add event page, Form box and many more.
- Javacript : We have used javascript for the following functionaities :
  - Dynamically assigning values to venue drop down button : For adding events, only those venues are listed down which are available during the time slot given by the user. Based on the time entered by the user a query is made to list down the available venues. This data is then publised as a JSON response which is taken and fed to the dropdown list of the venue input box.
  - Dynamically assigning values to club drop down button : Users can put up events by a particular club if they are either the head of the club or if they are the head of the department the club belongs to.Based on the logged in user, a query is made and the valid club list is published as a JSON response.This data is fed to the dropdown list of the club input box.
  - Dynamically showing description of an event after clicking on it.

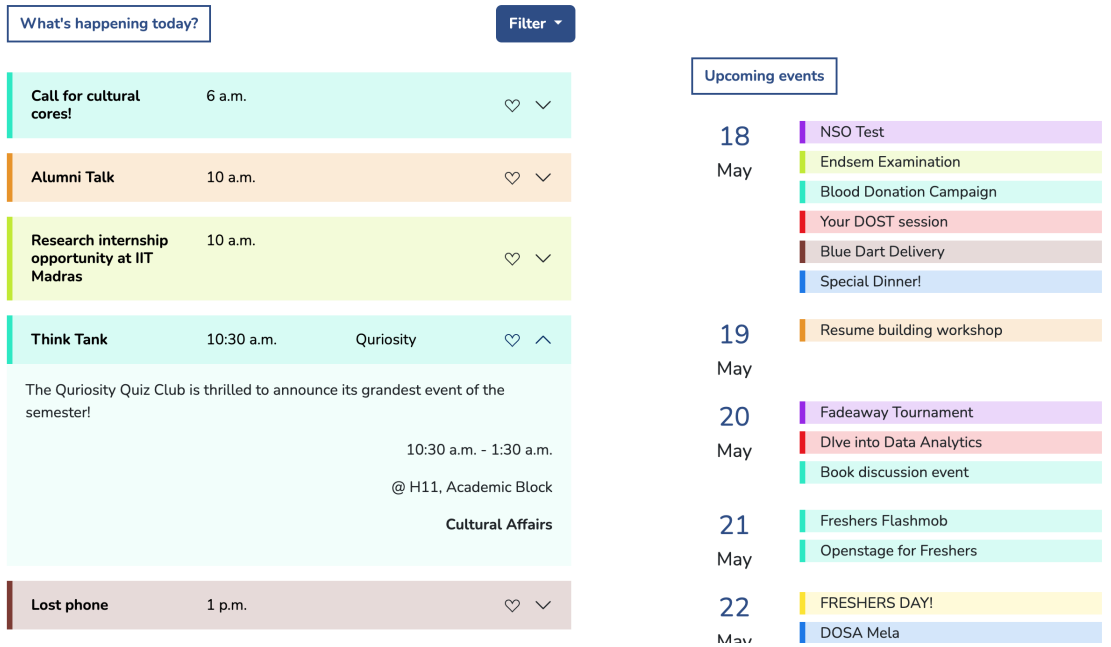


Figure 27: Clicking on an event shows the description of the event

– Confirmation for event deletion : Using a pop-up message to ask confirmation for event deletion. Once the confirmation is given the event id is sent to a url which deletes the event with the help of a view function where the appropriate query is given.

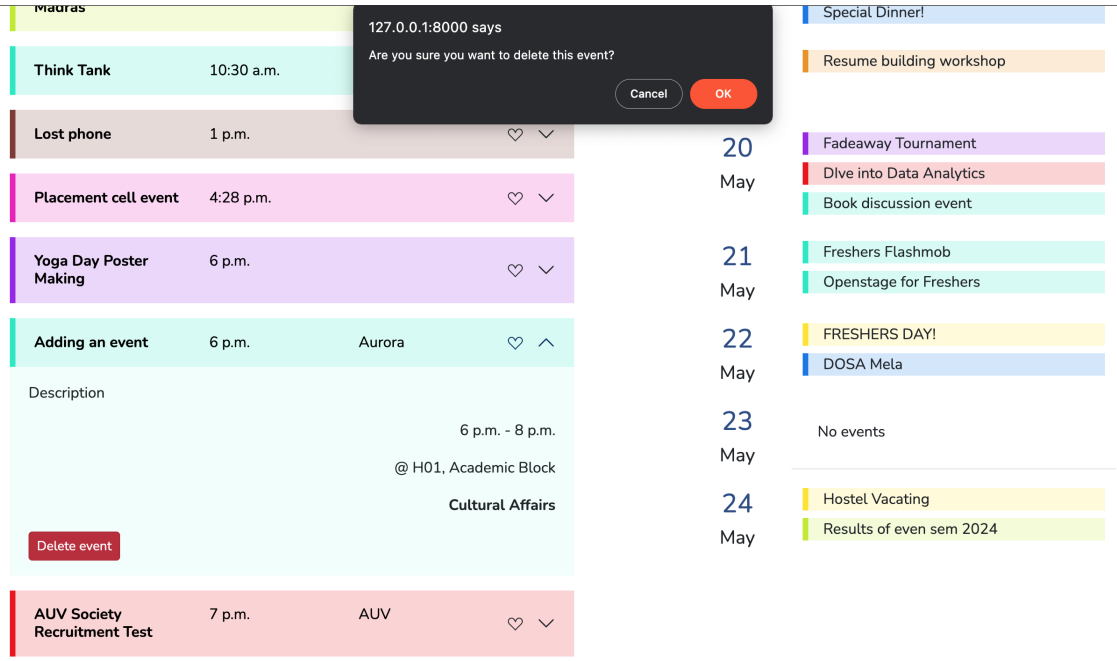


Figure 28: Confirmation Box for deletion of event

– Users not logged in are not permitted to favorite events : Using a pop-up message to alert the user that they have to sign in to favorite events. Once they get the alert they can either log in to favorite events or can just continue with a view-only interface.

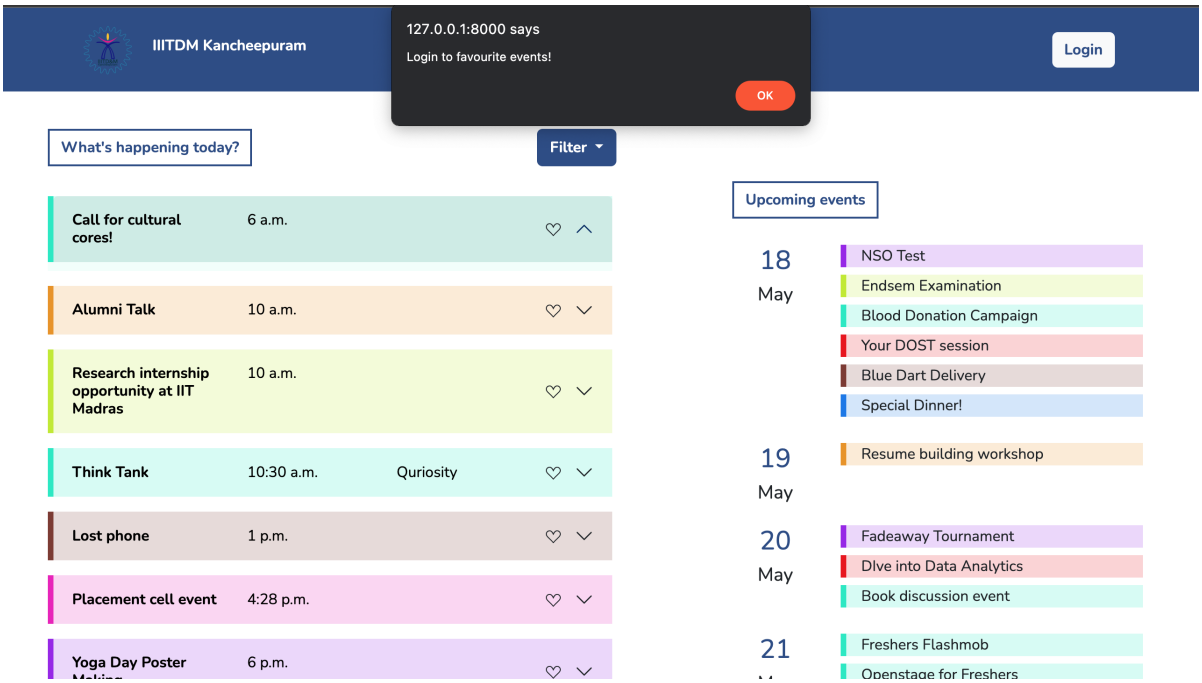


Figure 29: Notice box saying you havent logged in for favoriting event

– Adding favorite events to the Favorites table in our database : Once a user favorites an event, along with the user id of the user logged in, the event id is inserted into the database to keep track of the favorite events of each user. Similarly when a user unfavorites an event the same record is removed from the database. This is done by calling the rest API at the url fav/user id and at unfav/user id to the fav and unfav javascript function respectively to perform the mentioned functionalities.

– While filtering the favorites, we are taking all events from the favorites table and matching the current user id logged in with the user id in the favorites table and displaying the ones which match.

- Static Folder : We have used CSS to style elements of the webpage. The CSS files are stored in the Static folder of the root directory.They are referenced by the html file using the address "% static 'display_style.css' %".The static folder also contains images used for the front-end.

5 Challenges and Solutions

Challenge: Responsive Website Layout

Initial frontend development using HTML and CSS posed challenges in ensuring a responsive layout that adapts well across different devices and screen sizes. Manual adjustments for various viewport dimensions were time-consuming and prone to inconsistencies.

Solution:
We addressed this challenge by adopting the Bootstrap framework, which specializes in responsive design. Bootstrap's grid system and pre-built components facilitated the creation of layouts that automatically adjust to fit various screen sizes. Leveraging Bootstrap's grid system ensured proper alignment and sizing across different devices, significantly reducing the need for manual adjustments. Additionally, Bootstrap's CSS classes for responsiveness streamlined the adaptation process, enhancing development efficiency and ensuring a consistent user experience across devices.

Challenge: Efficient Authorization Management

Coordinating user access and permissions effectively presented a challenge, necessitating a streamlined approach to grant appropriate privileges based on user roles.

Solution:
We optimized authorization using Django's authentication and permissions system. This system facilitated user authentication, login, and password management, while also enabling precise access control for designated actions. Leveraging Django's authorization capabilities, we simplified access management, ensuring that only authorized users could execute relevant actions within the system.

Challenge: User Interface Consistency

Maintaining consistency in UI design across different pages and components presented a challenge. Inconsistencies in styling, layout, and interaction patterns could disrupt user experience, impacting usability and navigation.

Solution:
To ensure UI consistency, we began by developing a comprehensive style guide using Figma incorporating all the proposed ideas for the layout of the website before proceeding with the execution. This style guide outlined the design principles, typography, color schemes, and component usage. Regular UI reviews and feedback sessions were conducted to promptly identify and address inconsistencies, ensuring adherence to the established style guide.

Challenge: Real-time Venue and Club Listing

Implementing a feature to dynamically list venues and clubs in real-time based on user inputs, such as selected time and user profile, posed a challenge. Ensuring seamless updates of available options without page reloads required an efficient and responsive solution.

Solution:
To address this challenge, we employed jQuery to dynamically list venues and clubs based on user input. By leveraging jQuery's DOM manipulation capabilities, we created interactive interfaces that responded to user actions in real-time. Using asynchronous requests, we fetched data from the backend based on specified parameters, such as time and user profile. This enabled us to dynamically update the list of venues and clubs without requiring page reloads, enhancing user experience and system responsiveness.

Challenge: Database Optimization

As the database scaled with increasing event and user data, performance issues like slow query execution and high server load emerged. Ensuring smooth system operation required optimizing database queries and enhancing overall performance.

Solution:
To address database optimization challenges, we implemented strategies such as query optimization. We reviewed and optimized complex queries to minimize unnecessary database operations

and enhance performance. This involved restructuring queries for efficiency and improving indexing where applicable.

Challenge: Collaboration and Version Control

Coordinating development tasks among team members and maintaining version control posed a challenge. Without an efficient system in place, conflicts in code changes, version discrepancies, and lack of visibility into project progress could hinder productivity and collaboration.

Solution:

To address collaboration challenges, we utilized Git and GitHub extensively. Git provided version control capabilities, allowing team members to work on code independently and merge changes seamlessly. GitHub served as a centralized repository, facilitating collaboration by enabling code sharing, issue tracking, and pull request management. Regular communication and coordination using Git Bash ensured alignment on project goals and tasks, mitigating conflicts and ensuring smooth collaboration among team members.

6 Future Enhancements

- Implementing a calendar view of events for better visualization.
- Incorporation of a feedback mechanism for event organizers and attendees.
- Implementation of real-time notifications for events favorited by users.
- Providing support for event organization processes by incorporating procedures such as permission seeking by automating the approval workflow, allowing organizers to submit requests electronically and track their status in real-time.
- Implementing a budget management module that will enable organizers to create and manage budgets efficiently, with features for expense tracking, allocation, and reporting.
- Each student will get customized events based on their year of study, programme and department. For example a second year student studying CSE will get an event for submitting their DBMS course project by the given deadline.

7 Conclusion

The development of the Campus Event Management System was a collaborative effort that resulted in a robust platform for viewing and managing campus events efficiently. Our team worked together to design and implement features that meet the diverse needs of campus event organizers and attendees. The platform offers intuitive interfaces for users to easily view upcoming events, submit event proposals, and manage event details.

This report explored the implementation details and highlighted the chosen technology stack, including HTML, CSS, JavaScript, Bootstrap on the front-end and Django and MySQL as the relational database management system on the back-end.

8 References

- Django documentation
- Django CRUD
- Bootstrap documentation