

**Clubs & Events**

System Design

29.3.2019

Version 1.0

**Begüm Bilgin**

**Murat Akalın**

**Murat Yiğit**

**Mert Çetinkaya**

Prepared for

CSE490 Project



Table of Contents

[**1.** **Introduction** 5](#_Toc4599970)

[**1.1** **Purpose of the System** 5](#_Toc4599971)

[**1.2** **Design Goals** 5](#_Toc4599972)

[**1.3** **References** 5](#_Toc4599973)

[**2.** **Current Software Architecture** 6](#_Toc4599974)

[**3.** **Proposed Software Architecture** 6](#_Toc4599975)

[**3.1** **Overview** 7](#_Toc4599976)

[**3.2** **System Decomposition** 7](#_Toc4599977)

[**3.3** **Hardware-Software Mapping** 9](#_Toc4599978)

[**3.4** **Access Control and Security** 13](#_Toc4599979)

[**3.5** **Global Software Control** 13](#_Toc4599980)

[**3.6** **Boundary Conditions** 14](#_Toc4599981)

[**4.** **Subsystem Services** 15](#_Toc4599982)

[**5.** **References** 17](#_Toc4599983)

SYSTEM DESIGN DOCUMENT [1]

# **Introduction**

## **Purpose of the System**

As mentioned in the Requirements Analysis Document, the aim of the Clubs & Events System is developing a social network that is completely free. The web site will be a user-friendly platform and the main aim is creating a communication between the university clubs and students and the people who are not a student of the specific university. The information of the clubs and events will be available for everyone. The system will provide users to see and comment on the events and provide moderators and admins to be able to make changes on the events and clubs.

## **Design Goals**

The Clubs & Events System is designed as an online system where users can get information about university clubs and events. Thus, the main target is to implement all offered functionalities, functional requirements and non-functional requirements which are mentioned in Requirement Analysis Document.

Clubs & Events System provides a reliable, performance, supportable, usable, implementable, packaged and legal platform. It provides also a user-friendly interface.

**Definitions, Acronyms and Abbreviations**

**User**: The user of the system who can login, logout, see the clubs, events and comment on them.

**Admin**: The type of user who can login, logout, see the clubs, events and comment on them, also add clubs/events, edit clubs/events, deactivate clubs/events/comments, delete clubs/events/users/comments, accept/reject comments and update a user as moderator.

**Moderator**: The type of user who can login, logout, see the clubs, events and comment on them, also add clubs/events, edit clubs/events and deactivate clubs/events/comments.

**Visitor:** The type of user who can register and show clubs.

## **References**

* RAD of Clubs& Events
* BlackBoard
* Campus Online

# **Current Software Architecture**

While there is no existing system dedicated to school Clubs, we took references from similar systems used in our school and other school which is Marmara University. Two systems are used in our school’s bachelor program, BlackBoard and Campus Online. These systems have been implemented a long time ago, in 2000’s. Architecture used to build the current systems are unknown. In terms of user-interfaces, security, response-time, speed and other functionalities these systems are outdated. We want to make Clubs & Events more appealing and usable for both students and clubs, so these functionalities are actively used. Newer technologies should be used for a faster, more secure and use-friendly system. In terms of database, especially in Campus Online, the database infrastructure is complicated. More relations among different tables cause slower operations, sometimes it can cause to deadlock. While Clubs& Events will not have a database as big Campus Online, database relations, operations and complexity shall be aimed to be straightforward and fast as possible. The increasing population of members of the web-site also causing more network traffic, which is no longer sufficient enough to support a large number of users for the current software.

# **Proposed Software Architecture**

Our proposed design pattern of Clubs&Events is MVC. Model subsystems maintain domain knowledge, and does not depend on any view or controller subsystem (Entity Objects). View subsystems display it to the user (Boundary Objects), and Controller subsystems manage the sequence of interactions with the user (Control Objects). We used MVC model because, in our system, entity objects and data will be in Model, and Controller can be called bridge. Controller provides communication and interaction between Model and View. View can be called interface between users and the system, thus it has boundary objects. These project outputs are logically separated, which reside under on the same machine. MVC pattern is provided by the Django Framework ( it includes MVC module views and paths).

In our component diagram, we’ve used Layered Architectural Style to represent MVC, for readability and reusability. We support a system that has been implemented with the layered architecture. According to this layered architecture, there is a hierarchy of layer, each layer using services are offered by the lower layers. We implement our project with the layered architecture, all of the functions are called from services.

Essential requirements are to develop the features of Clubs & Events efficiently. Clubs & Events needs to have its own unique implementation, with functionalities alike BlackBoard & Campus Online and more functionality added. Optional requirements, in other words desired features, are going to be implemented. We are going to develop the system with Django Framework that is a framework that uses Python infrastructure. Our system is an object-oriented system. This is a technique of coding and we will use database in the part of back-end. The front-end part will include interfaces for users. The interface pages are using web services.

## **Overview**

We are going to develop the system with Django Framework, which is a framework that uses Python programming language and has model-view-controller design pattern infrastructure. The complete system is divided into smaller sub-systems with own activities.

Clubs & Events consists of three actors, namely visitors, registered user, moderator and admin. Each actor in the system has different authorities. Operations are divided into actor authorities. All four actors share some common functionalities related operations, such as showing clubs, clubs details, events and events details. For each of these four actors, plus the account operations, we have four subsystems.

To provide a bird's eye view of the software architecture used in the system;

***Admin Subsystem:*** Provides services to admin to manage the system. It allows admin to perform CRUD operations in the database for clubs, events, users and comments tables. Admin can access all things in the system. For example, only admin can delete anything in the system. He/ she can see all details of the users such as e-mail, username, firstname and lastname of the users. If admin want to delete user, he/ she can deletion process on admin panel.

***Registered User Subsystem:*** Provides services that allow adding any comments on events. They can show all clubs and clubs’ details. Their comments must be apply by admin or moderator for the publish on the web-site.

***Moderator Subsystem:*** Provides services to access to moderator panel on the web-site. He / she can add club on the web-site. He/ she can edit the any club which is on the system. Also he/ she can same things for events. Also, he/ she can reach the all comments on the system. Because, all comments must need admin or moderator confirmation for the publish in the web-site.

***Visitor Subsystem:*** Allows the showing clubs and their details of the system. And also, visitors can see events and their details of the events. If they want to join our web-site, they can register in the system.

## **System Decomposition**

Clubs & Events is made of three major actors that are named registered user, moderator and system administrators. Each actor in the system has different authorities. Operations are divided into actor authorities. We have three actors to implement. We also have common account functionalities for all the actors, such as showing all clubs and clubs details also , events and events details. So, we have four different sub-systems for interfaces, corresponding to each actor and one of their common functionalities.

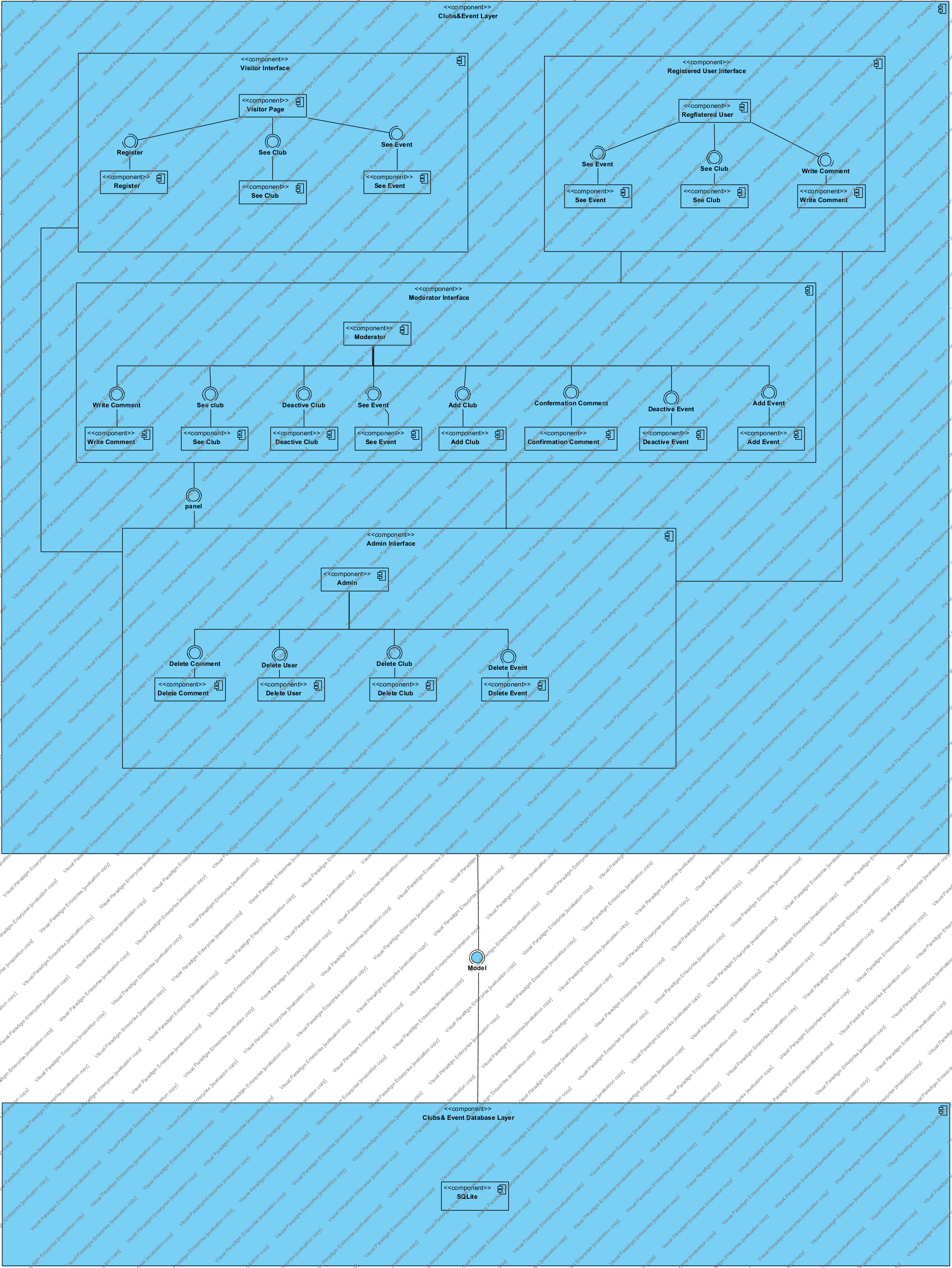


Figure 1: Coupling view of Subsystem Decomposition

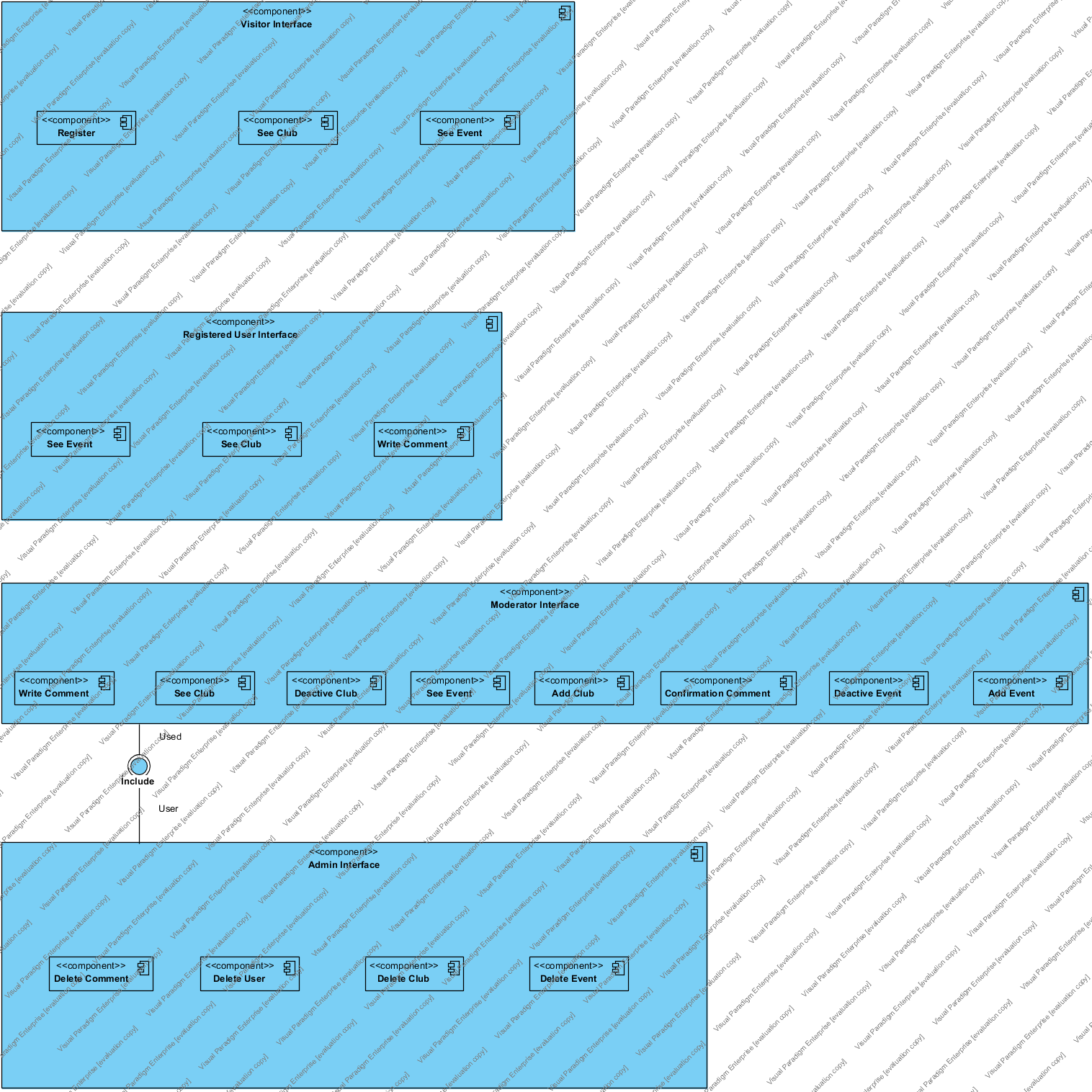


Figure 2: Cohesion view of Subsystem Decomposition

## **Hardware-Software Mapping**

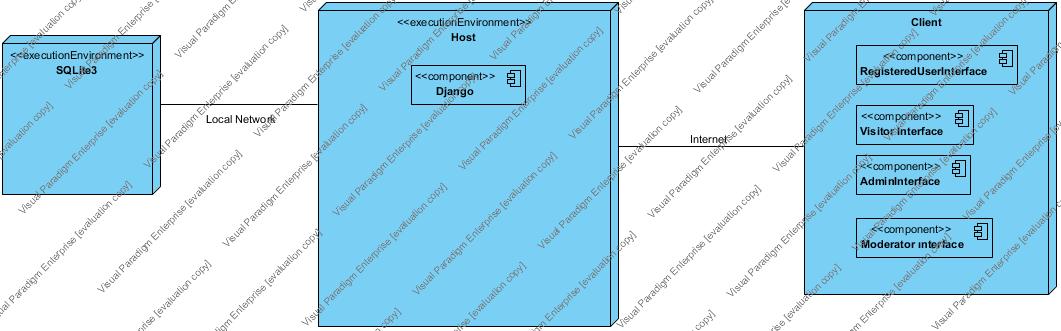
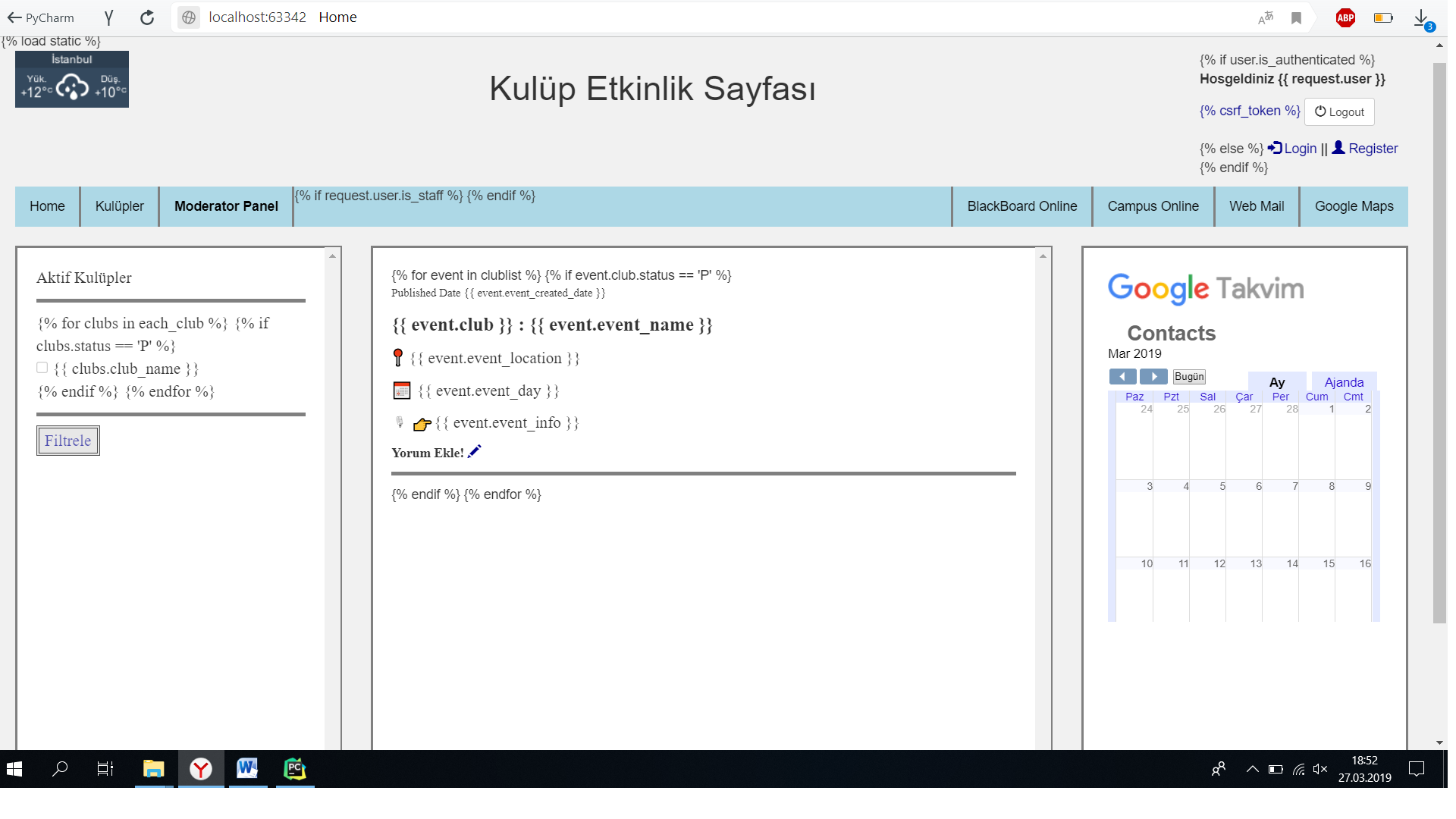


Figure 3: Hardware-Software Mapping Deployment Diagram



****

We are able to examine hardware-software mapping by two sides that are actors and system.

* In terms of actors, there is no any constraint on hardware-software mapping. Clubs & Event is a web-based application that developed by Python Django infrastructure. All different type actor is able to reach to the system with ordinary computer, or mobile devices with internet-connection.

**Persistent Data Management**

We are going to work with Django Standard database which is SQLite. This database simple interface, and its very simple to use. Now we have not too much users for the web-site. So we can use this database for our project. SQLite is a relational database management system, or RDBMS, that supports a wide variety of transaction processing, business intelligence and analytics applications in corporate IT environments.

We are going to work with object relational model, in shortly ORM. Object relational model provides more power, greater flexibility, better performance and greater data integrity then those that came before it. Some of the benefits that are offered by the Object-Relational Model include:

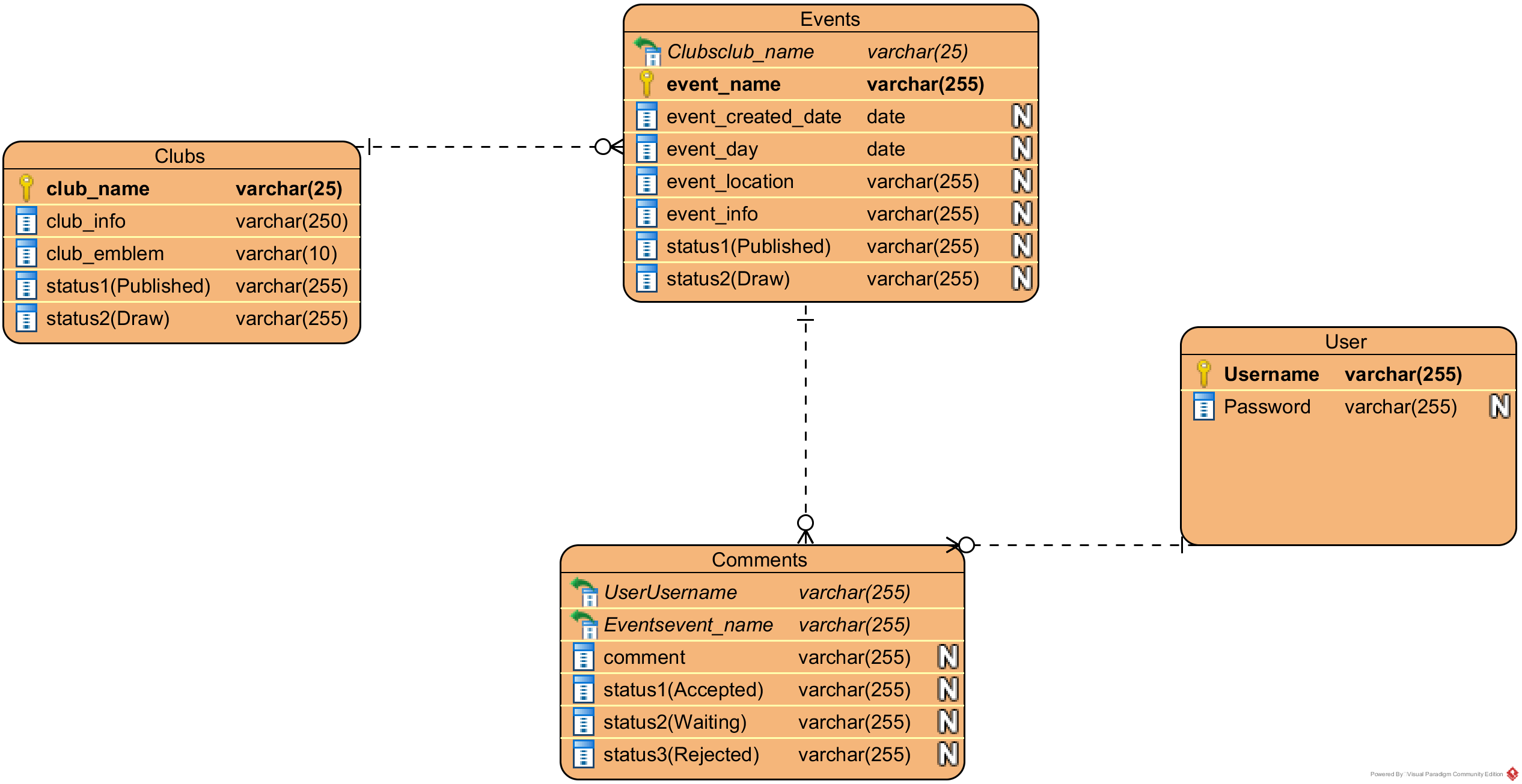
* Extensibility – We are able to extend the capability of the database server; this can be done by defining new data types, as well as user-defined patterns. This allows the use to store and manage data.
* Complex types – It allows us to define new data types that combine one or more of the currently existing data types. Complex types aid in better flexibility in organizing the data on a structure made up of columns and tables.
* Inheritance – We are able to define objects or types and tables that procure the properties of other objects, as well as add new properties that specific to the object that been defined. It also complements object-oriented programming, which we are using.

Entity Framework (EF) will be used as ORM tool, and we will be using a Code-First approach. Code-First is mainly useful in Domain Driven Design. In the Code-First approach, we focus on the domain of your application and start creating classes (model of MVC) for our domain entity rather than design our database first and then create the classes which match your database design. We use this approach for speeding up development, cleaner code generation and database relations.

Entity Framework provides:

* Provides dedicated functionality for CRUD operation (Create, Read, Update, Delete). Easy to implement CRUD operations.
* If we want to replace the data store, it is very easy to replace without modifying the data access logic since all data access logic are present in higher level.
* Easy to manage one to one, one to many and many to many relationships between tables.
* Conceptual model can be represented in a better way.
* It enables developers to visually design models and mapping of database.
* Developer can reduce the code in classes and sub-classes for data access.
* It provides auto generated code, reduces development time and development cost.

**Description of Encapsulation of the Database**



'User' is an abstract user, which indicates a defined user in the system. All the other actors except the visitor, are inherited by this user in the model. With the help of EF ORM, only 'User' table is seen as a user in the database, and all user has a status on the system like staff, user , moderator or admin. We will set these statuses on the admin panel.

* **User Model:** Holds the common attributes and information shared by all users in the system, namely First-Middle Name, Last Name, E-mail, password. In the models; Admin, Moderator and Registered User inherit from this model.
* **Admin Model:** Holds the same attributes as User table and is inherited from it.
* **Registered User Model:** Inherited from User Model.
* **Moderator Model:** Inherited from User Model.
* **Clubs Model**: Clubs model stores all the information about clubs. It needs club\_name (unique) varchar, club\_infos varchar, club\_emblem imagefield and status varchar(multiple choice).
* **Event Model**: Event model stores all the information about events. It needs club ( its Foreign Key), event\_name(unique) varchar, event\_created\_day date, event day date, event\_location varchar, event\_info varchar and status varchar.
* **Comments Model:** In this table contains all users’ comments. A comment need an event ( Foreign Key) , user ( Foreign Key) , comment varchar and status varchar ( multiple choice )
* **Some attributes will not shown in database. Because they are just setting on django admin panel ( moderator staff, user, admin )**

If any main row will delete on the system , with foreign key other tables will delete ( with this code: on\_delete= models.CASCADE). With this code, tables or attributes will safely delete on the system.

All uniqueness check is controlled in models.

## **Access Control and Security**

In case of security, user’s access to the system with a uniquely username and password at login page. Django Framework Identity are used for system security. Each user has a role in the system, and a model inherited from the User table according to their role, then defined in the database via ORM. User accesses services by the actor role, and redirected to their respective pages according to their role.

Each user, which are registered user, moderator and admin, has their permissions and interfaces defined. After login, each actor is redirected to same home pages, providing them an interface to perform their operations. If an actor tries to reach another actors function or page, a function or page that they’re not allowed or authorized to do and redirect to home page, they are thrown back to the login page and cannot reach that page. System administrator has full management authentication, and can control and manage the system and the database. Thus, because of front-end structure, an authenticated user in a specific role cannot reach another actor’s permissions.

## **Global Software Control**

Clubs & Events is a web-based system that uses HTTP GET/POST requests. Thus, having an internet connection is required to access the system. The system uses cookies in authentication, and keeps the user’s username, e-mail and role on a thread. After successfully authenticating, the user is redirected to their respective page according to their role. Users role is detected on login, so that the user is redirected correctly to their respective home page, and cannot access to a page that he/she does not have permission to display.

Accessible pages are synchronized over internet-connection. The synchronization is satisfied among queries on server. To succeed this, database connection is also required. After database connection has been done, synchronization operations are satisfied.

## **Boundary Conditions**

**Start-up:**

* The system must be run in Django platform, via IIS Express, through any browser.
* Clubs & Events needs to have a stable internet connection to work properly.

**Shutdown:**

Shutdown might occur with two-ways;

* Physical hardware might be interrupted.
* IIS might be stopped.

**Error behavior:**

* Internet-connection might be unconnected.
* Physical hardware might be interrupted.
* IIS Express might have problems.
* Database might be lost.
* Database connection might be unconnected.
* User might have attempted to login with invalid credentials. (These exceptions are getting caught by the system and handled properly.)

While we’re not implementing start and stop server functionalities, when the project is live, admin will need server functionalities in order to do maintenance in the system.

Clubs & Events is initialized by the system admin invoking the ‘Initialize System’ use case. Once the initialization of the system is completed, the departments and the curriculum information are registered onto system by the system admin. In addition, the system admin initialize the server by invoking the ‘Start Server’ use case. After that, when the system is brought from non-initialized state to steady-state, the system is opened and free for the system users (Students, Instructors, and Admin) to login and perform their tasks. When the system is terminated by system admin invoking the ‘Shutdown Server’ use case. All users that currently logged in to the system will be disconnected. This situation occurs for instance, the system admin closes the system for a maintenance before the registration day(s). No user can login to the system until the system is initialized by the admin again.

# **Subsystem Services**

The subsystem decomposition of Clubs & Events System, we divide the system into smaller subsystems with strong coherences.

The subsystem separation shows the entities of following subsystem

- Visitor Subsystem

- Registered User Subsystem

- Admin Subsystem

- Moderator Subsystem

**Registered User Subsystem**

This subsystem gives the ability to see clubs and make comment on their events for users.

This subsystem provides:

- Login

- Make Comment

- Show Clubs, clubs details

- Show Events, events details

- Logout

**Moderator Subsystem**

This subsystem provides managing event and club activities for moderators.

This subsystem provides:

- Login

- Make Comment

- Show Clubs, clubs details

- Show Events, events details

- Add Club

- Add Event

- Edit Club

- Edit Event

- Deactivate Club

- Deactivate Event

- Deactivate Comment

- Logout

**Admin Subsystem**

This subsystem provides managing adding, editing, deleting, deactivating events for admins.

This subsystem provides:

- Login

- Make Comment

- Show Clubs, clubs details

- Show Events, events details

Add Club

- Add Event

- Edit Club

- Edit Event

- Deactivate Club

- Deactivate Event

- Deactivate Comment

- Delete Club

- Delete Event

- Delete User

- Delete Comment

- Accept Comment

- Reject Comment

- Update a User to Moderator

- Logout

**Visitor Subsystem**

This subsystem provides visitors to be able to register and show clubs.

This subsystem provides:

- Register

- Show Clubs, clubs details

- Show Events, events details

# **References**

1. Bruegge B. & Dutoit A.H.. (2010). *Object-Oriented Software Engineering Using UML, Patterns, and Java*, Prentice Hall, 3rd ed.
2. Campus Automation System (2002) University of Işık,[**Campus Online**](http://campus.isikun.edu.tr/)
3. <https://searchsqlserver.techtarget.com/definition/SQL-Server>
4. [Implementing Inheritance with the Entity Framework 6 in an ASP.NET MVC 5 Application](https://docs.microsoft.com/tr-tr/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/implementing-inheritance-with-the-entity-framework-in-an-asp-net-mvc-application)