**Course and Lab Allocation System**

**(Deliverable 4)**

(System Decomposition, Architecture Diagram, Component Diagram)

**Prepared for:**

**Sir Zeeshan Rana**

**Prepared by:**

Noor Fatimah(20l-0999)

Rabiya Kamran(20l-2097)

Sameera Adnan(20L -1101)

**April 4, 2023.**

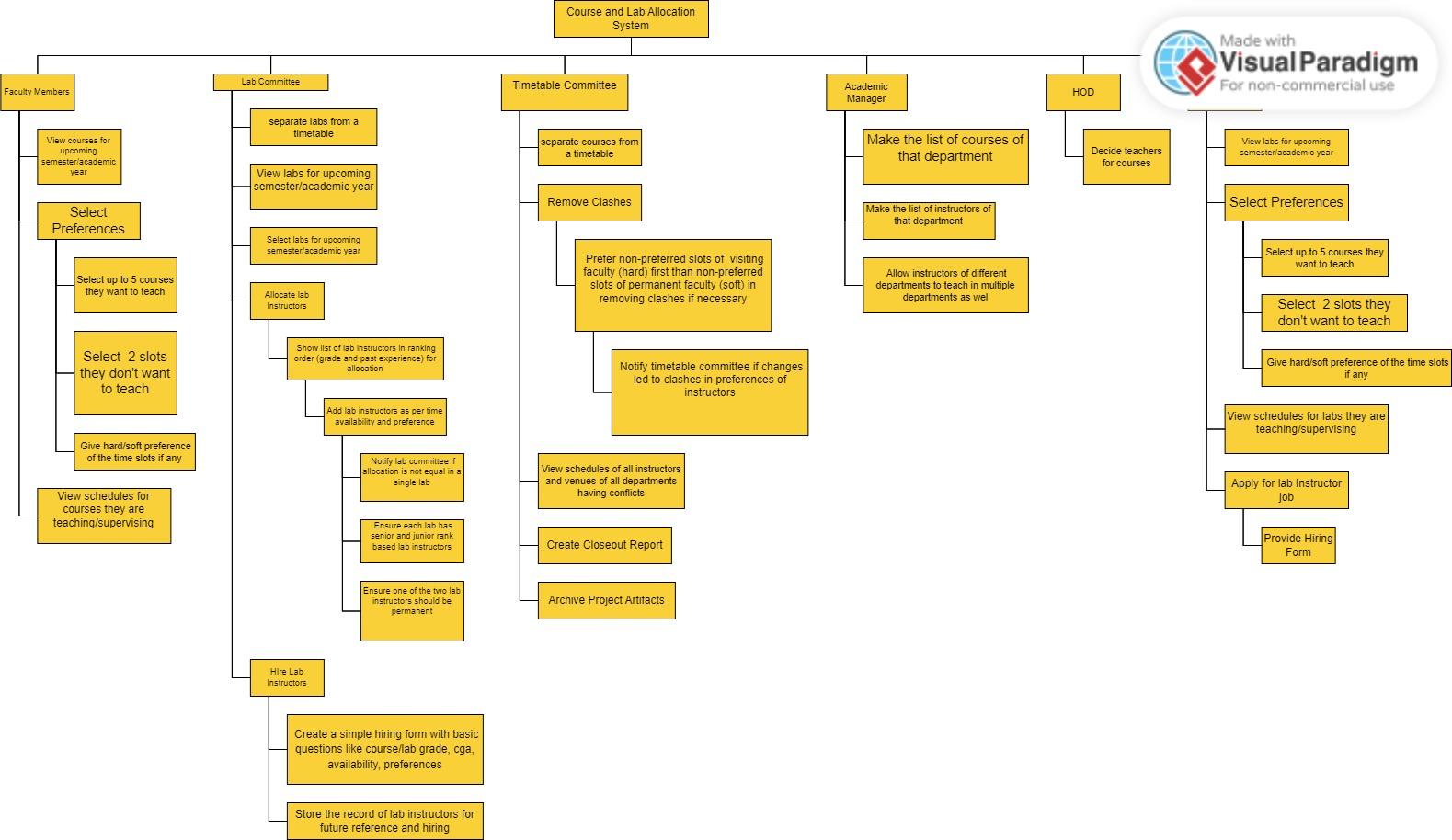
System Decomposition

**Functional Decomposition:**

The functional decomposition diagram’s image is present below but for better clarity the link is also provided.

<https://online.visual-paradigm.com/app/diagrams/#diagram:proj=0&type=FunctionalDecompositionDiagram&gallery=/repository/2347368b-a753-438f-9b7c-4a4c982b6b41.xml&name=Library%20Management%20System%20Decomposition>

**→** The following image contains a watermark. Under the watermark there is a heading of functional component as “lab instructors”.



Architecture Style

In the architecture of the Course and Lab allocation system we are using the combination of multiple architectural styles for the project. The architectural styles include Client Server and Repositories Architecture.

1. **Client Server Architecture:**

It is the most suitable architectural style that is used in this system. This architectural style divides the system into two components, one component is called client and the other is called server. Client makes the request to the server and the server sends the results back. Over here the client is playing the role of the interfaces which are used by the different stakeholders so that they can interact with the system. While the server handles the request of the stakeholders sent by the client. For instance servers may be handling requests from faculty members, HOD, lab instructors, timetable and lab commitee. It is a scalable architecture as it consists of many functionalities and many stakeholders which are most of the time, interacting with the system. Hence Client Server Architecture style is an efficent way for communication between the client and server component of the system. It ensures management of centralized data between different components of the system.

1. **Repositories:**

In Repositories architectural style data is stored in a central repository where all the stakeholders can access the repository and make changes to it. The storage of data can be done in a database or using file storage. The accessors of the repository are all the stakeholders, they can access the data from a centralized data store which proves the integrity and consistency of the data. The stakeholders may include the faculty members, HOD and lab instructors, lab and timetable committee which interact with the system by data management.

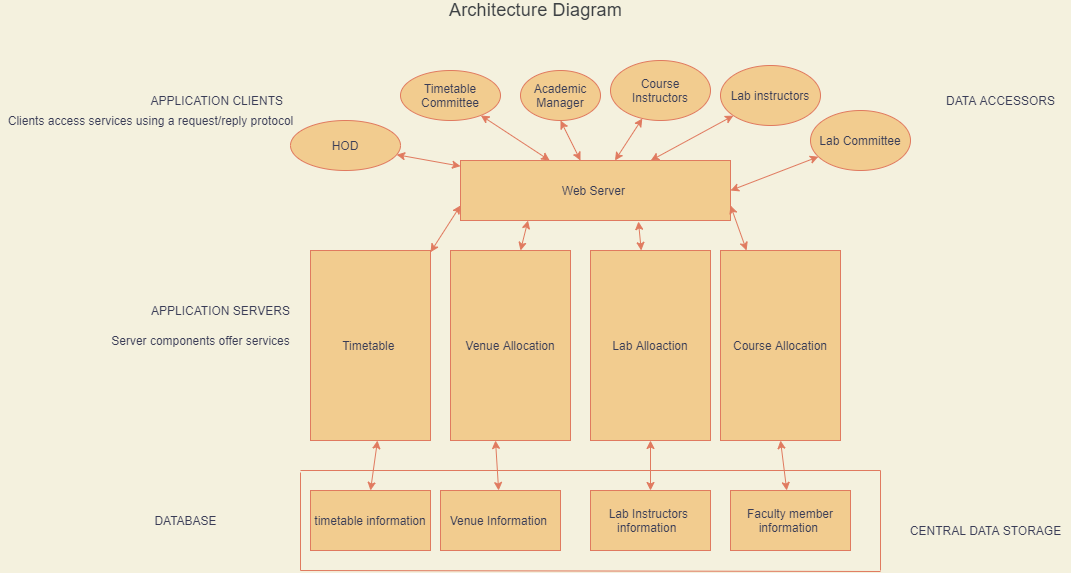
**Why not other Architectural Systems:**

Some of the architectural styles not used in our system include Pipe-and-Filter, Publish-Subscribe, Peer-to-Peer (P2P), Layering. Pipe-and-Filter Architectural architectural style is a simple style used for the system and is not used within our course and lab allocation system as it involves the complex interactions between different stakeholders and requires a complex way of the management of the data and resources. It does not provide a centralized component for data management, hence different stakeholders such as lab committee, timetable committee, HOD and lab instructors need to interact with the system and get access to the data. Our system has various requirements and needs which might not be handled by pipe and filter architecture which handles a linear and sequential flow of data. Publish-Subscribe architectural style is not suitable for our system as it does not align with the complexity of our system. Our system requires complex interactions between data stores and stakeholders whereas this architecture uses messages to be published to a queue and the subscribers receive the message. It is loosely coupled architectural style where each of it;s component is not dependent on the other The Peer-to-Peer (P2P) Architectural Style is also not efficient to be used in our system as it involves peers to communicate with each other in a decentralized manner where they do not rely on the centralized server. As it does not involve the centralized component hence efficient to handle the complex interactions of the stakeholders and data access within our system. Similarly Layering architecture is difficult to maintain as the change in a single layer can affect the entire system as the layers are interdependent on each other and hence this architecture is difficult to mainatin.

**Conclusion:**

The combination of these two architectural styles firstly separates different components of our system, mostly the components of our systems defined here are client part or server part in **client server** architecture and a centralized data store in **repositories**. The components of the client server are used as an interface which are accessed by stakeholders. The data store stores the data in a database or files whcih has the stakeholders as its accessors and users too. All these styles combine together and make the architectural style of our course and lab allocation system. It’s combination has many advantages which include the reduction of the complexity of the data management, the system to be scalable based on the requirements, the systems can easily be maintained and no big changes need to be done in the whole system if something is added or removed from the system, modification within the system can be easily done and it remains maintained. The combined form of architectural style ensures that the data used here has its integrity and consistency being ensured. The components used here are interfaces which are easily handled and maintained by the stakeholders. However it ultimately depends on the requirement of the project and the availability of the resources. If the project does not require a central data repository, then we can only use the "client-server" architecture. Whereas if data management and consistency are important requirements, then using both "repositories" and "client-server" architecture may be the better option.

Architecture Diagram



The link to the architecture diagram for better understanding is also provided:

<https://drive.google.com/file/d/14y_e303RpwoQUXeFBzPmZMT3GAhgEO9s/view?usp=sharing>

Component Diagram

**Component-to-component interface:**

FacultyMember:

- view\_All\_Avialable\_Courses()

- Choose\_Course(course: Course)

TimetableCommittee:

- getTimetable()

- setTimetable(timetable: Timetable)

- viewInstructorSchedules()

- viewVenueSchedules()

HOD:

- InstructorAssignment(instructor: Instructor, course: Course)

LabCommittee:

- view\_All\_Avialable\_Labs()

- Choose\_Lab(lab: Lab)

Instructor:

- Choose\_Course(courses: List[Course])

- selectUnavailableSlots(slots: List[Slot])

- selectPreferenceSlots(slots: Dict[Slot, str])

LabInstructor:

- hiringApplication(labGrade: str, cgpa: float, availability: List[Slot], preferences: Dict[Slot, str])

Course:

- setSections(numSections: int)

- getSections() -> List[Section]

Lab:

- setLabInstructors(numInstructors: int)

- getLabInstructors() -> List[LabInstructor]

Section:

- getEnrollment() -> int

- setEnrollment(enrollment: int)

- getInstructor() -> Instructor

- setInstructor(instructor: Instructor)

Slot:

- \_\_init\_\_(day: str, start: str, end: str)

Timetable:

- \_\_init\_\_(sections: List[Section], labs: List[Lab], slots: List[Slot])

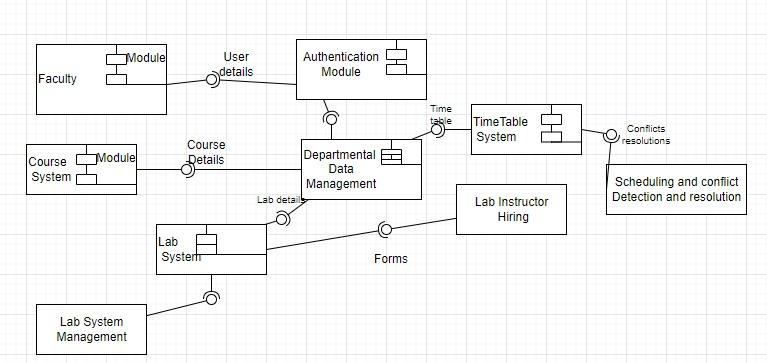
- setSectionsSchedule(section: Section, slot: Slot)

- getSectionsSchedule() -> Dict[Section, Slot]

- setLabsSchedule(lab: Lab, slot: Slot)

- getLabsSchedule() -> Dict[Lab, Slot]

**Component Diagram:**



The link to the component diagram is also provided:

<https://drive.google.com/file/d/1BTXxiBdVaF_lwidTfpUSRvlS8K_msKZ5/view?usp=sharing>