Iteration 2: Identifying Structures to Support Primary Functionality

The focus of this section is to present the results of the activities that are performed in each of the steps of the ADD in the second iteration of the design process for the CMS system. There were generic descriptions of the functionalities described in iteration 1. The goal was to establish an overall system structure. Iteration 2 provides detailed descriptions and decisions that result in the formation of development teams; its goal is to reason about the units of implementation that affect formation, interfaces and also the manner in which development tasks are distributed, outsourced and implemented in sprints.

We need to design in a systematic way, by showing discipline about the decisions we make as we cannot design everything from the start. We initially focus on the biggest risks and then focus on finer details.

Step 2: Establish Iteration Goal by Selecting Drivers

The goal of iteration 2 is to address the general architectural concern of identifying structures to support primary functionality. In this second iteration, the architect considers the system's primary use cases:

- UC-1
- UC-4
- UC-5

Step 3: Choose One or More Elements of the System to Refine

The elements that are to be refined in iteration 2 are the ones that were located in the different layers defined by the three-tier reference architectures in iteration 1. The collaboration of the components that associate with the modules located in different layers is required for the support of functionality in the system.

Step 4: Choose One or More Design Concepts that Satisfy the Selected Drivers

The following table summarizes the design decisions.

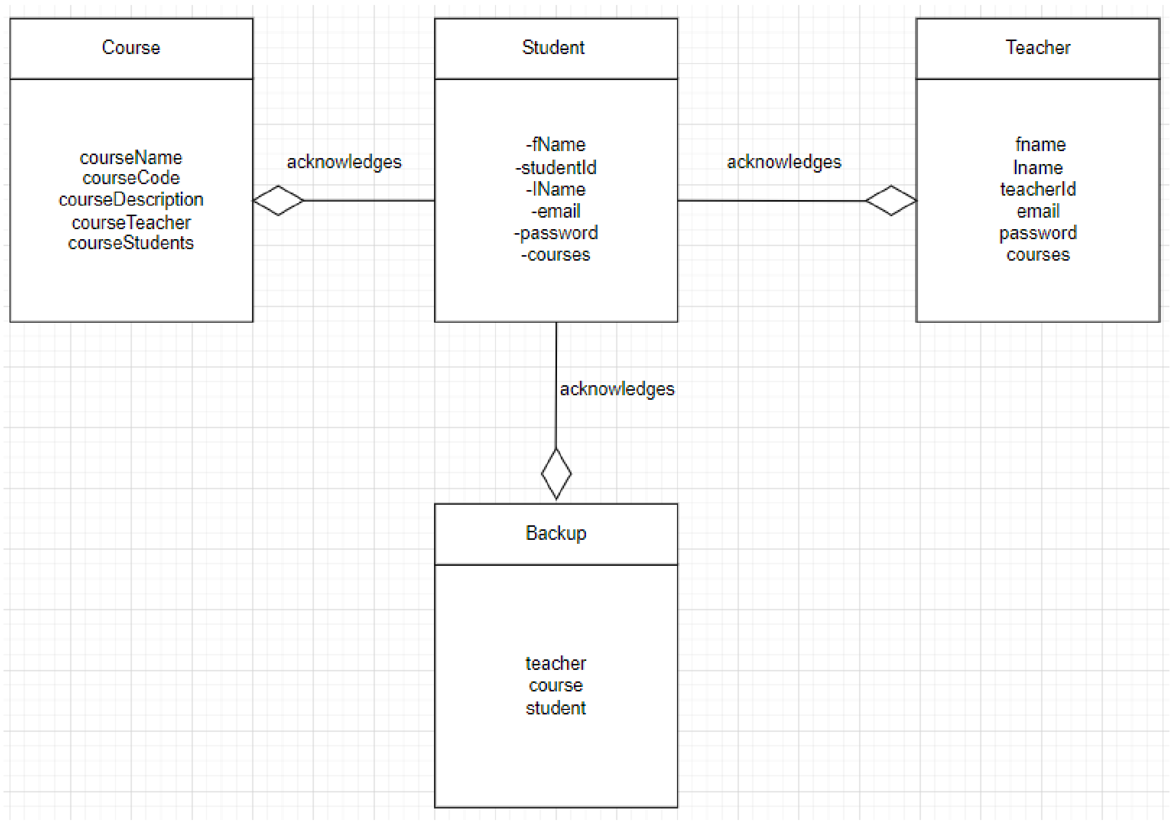
Design Decisions and Location	**Rationale and Assumptions**
Create a Domain Model for the application	An initial domain model is necessary for the system, which identifies major entities and their relationships.
Identify Domain Objects that map to functional requirements	A domain object, where each distinct functional element of the application needs to be encapsulated in a self containing building block.
Decompose Domain Objects into general and specialized components	Complete sets of functionality that are supported by finer-grained elements located within the layers are represented by domain objects.
Use React and Express framework	React is a widely used client side JavaScript framework. Express is a server side JavaScript framework. React and Express integrate well with each other. There are no alternatives considered for the application development. React and express were selected because they are easy to implement and the development team is familiar with it, resulting in greater and earlier productivity.

Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces.

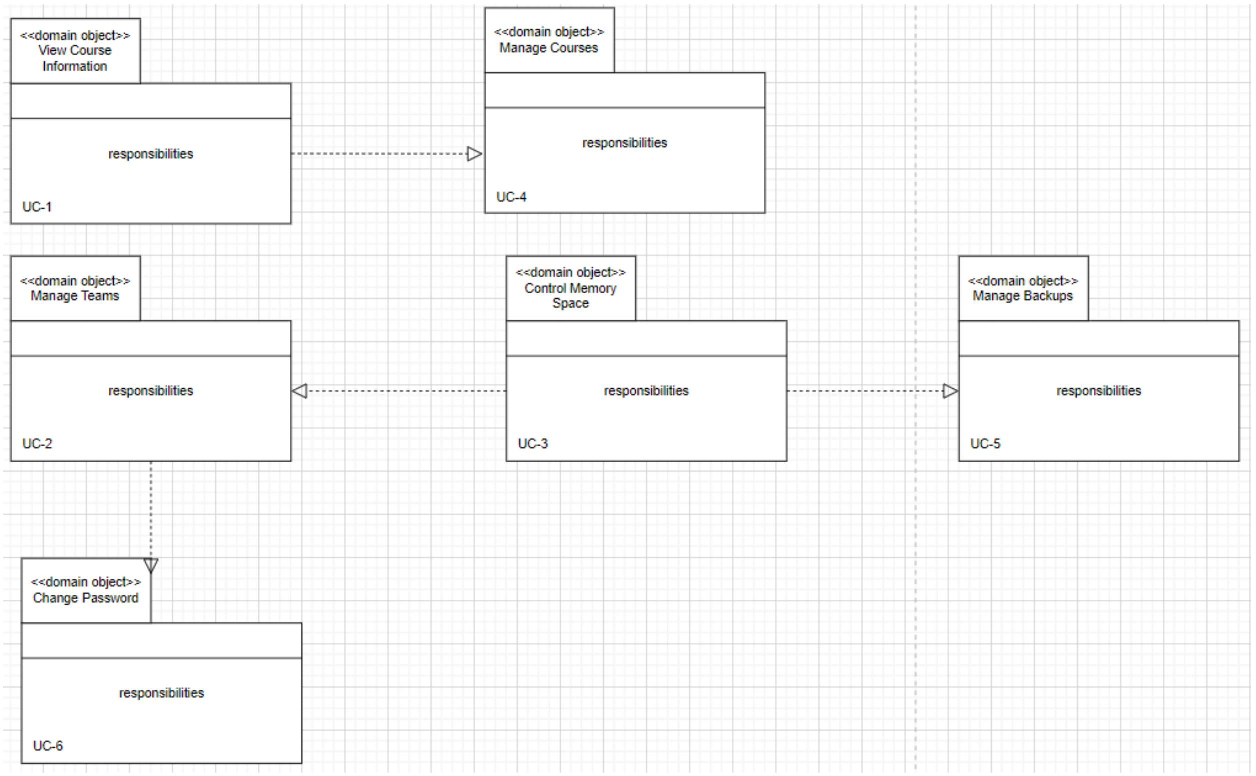
Design Decisions and Location	**Rationale**
Create only an initial domain model	To accelerate this phase of design, only an initial domain model is created.
Map the system use cases to domain objects	By analyzing the system’s use cases, an initial identification of domain objects can be made. For all of the use cases in the use case description, domain objects are identified.
Decompose the domain objects across the layers to identify layer-specific modules with an explicit interface.	This will ensure that the modules that support all of the functionalities are identified. Since the architect will perform this task for just the primary use cases, another team member is required to identify the rest of the modules. The architect discovers the necessity to test the modules, except the ones that implement UI functionalities therefore a new architectural concern is identified: CRN-6: There is a test required on most of the modules.
Connect components associated with modules using React	Since React uses the inversion of control approach, different aspects are supported and the modules are unit-tested.
Associate frameworks with a module in the data layer	The modules that are in the data layer summarize Object Relational Mapping.

Step 6: Sketch Views and Record Design Decisions

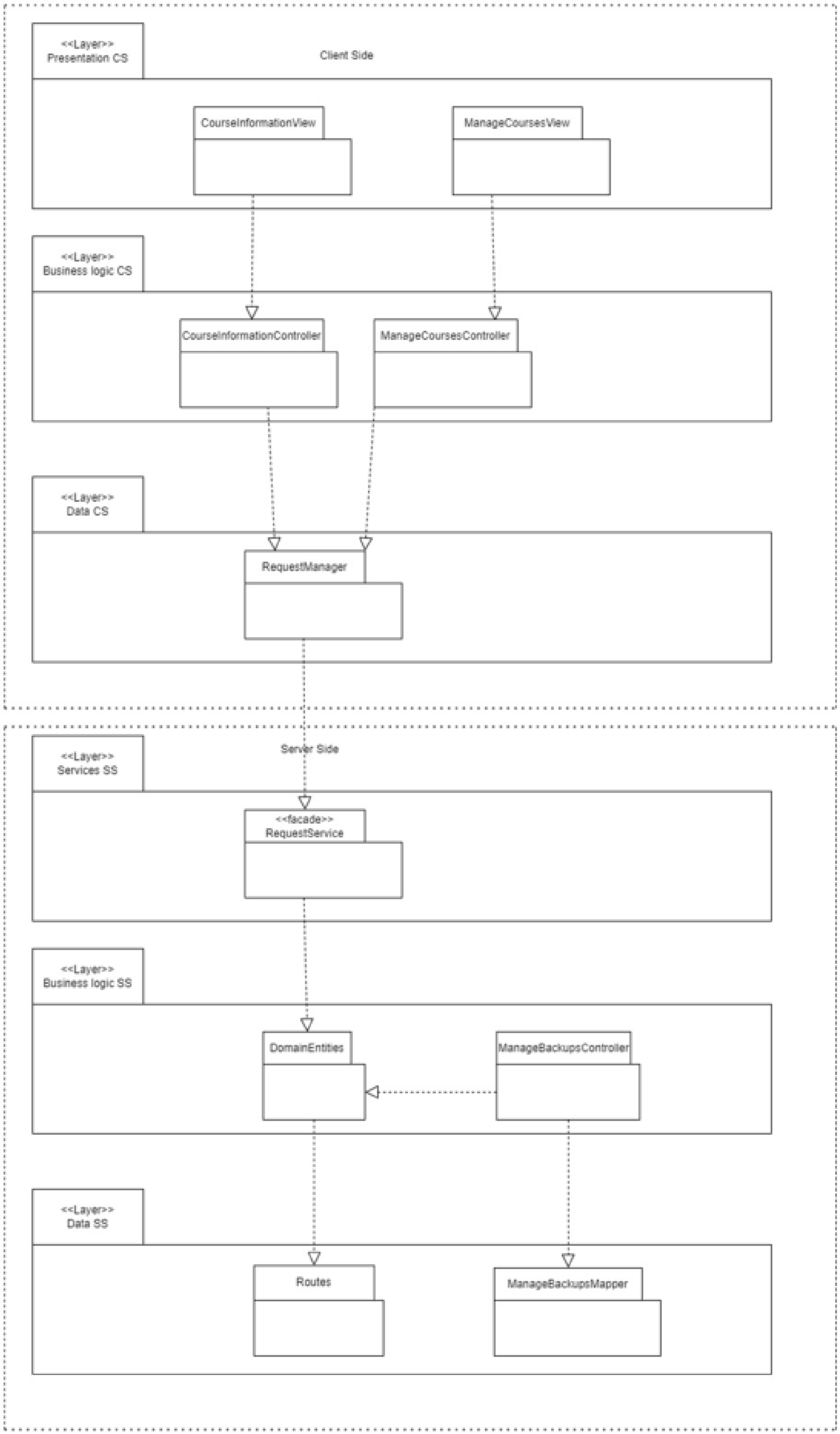
1. Figure shows the initial domain model for the system.



2. Figure below shows the domain objects that are to be instantiated from the use case model.

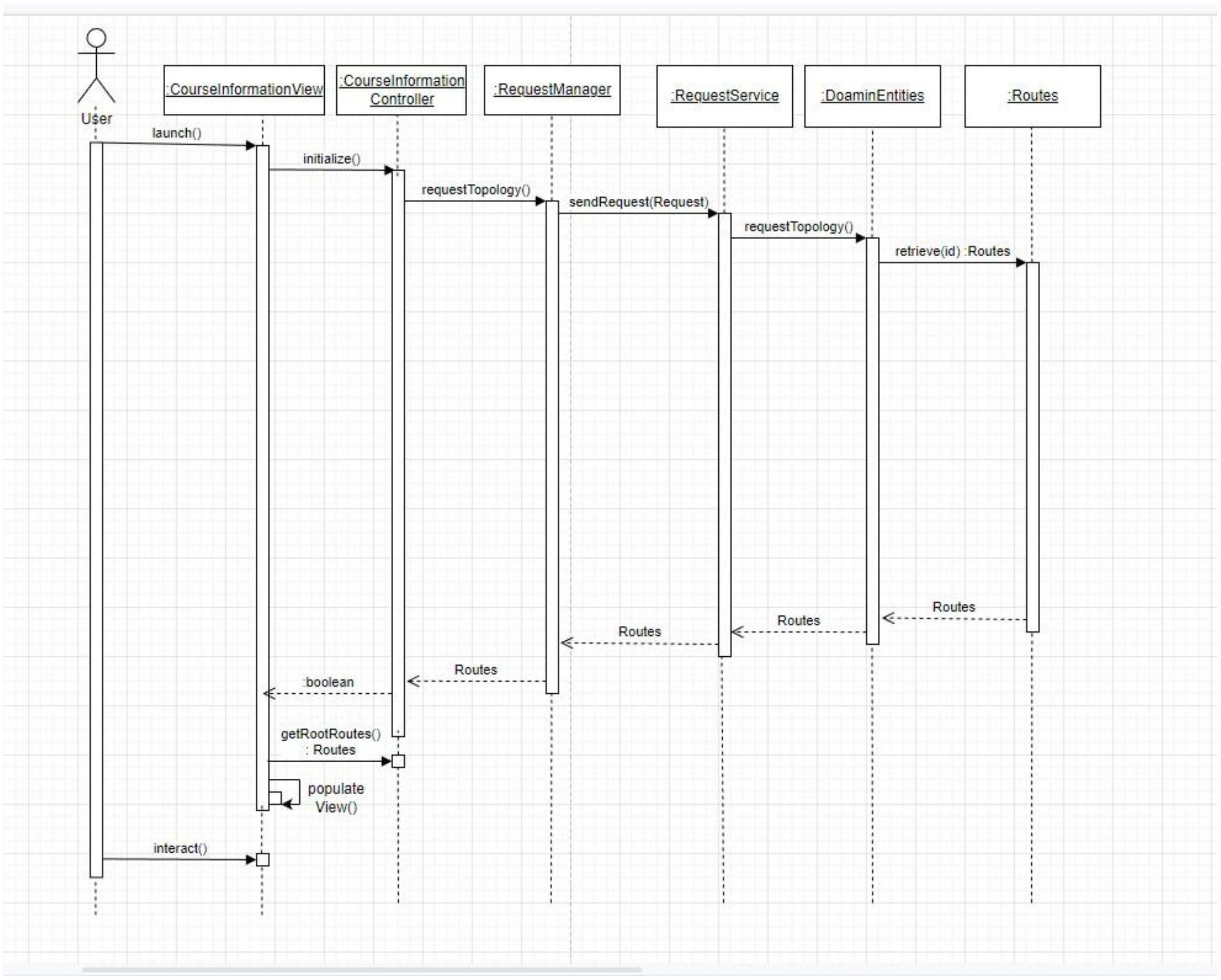

<<domain object>> View Course Information
responsibilities
UC-1
<<domain object>> Manage Courses
responsibilities
UC-4
<<domain object>> Manage Teams
responsibilities
UC-2
<<domain object>> Control Memory Space
responsibilities
UC-3
<<domain object>> Manage Backups
responsibilities
UC-5
<<domain object>> Change Password
responsibilities
UC-6

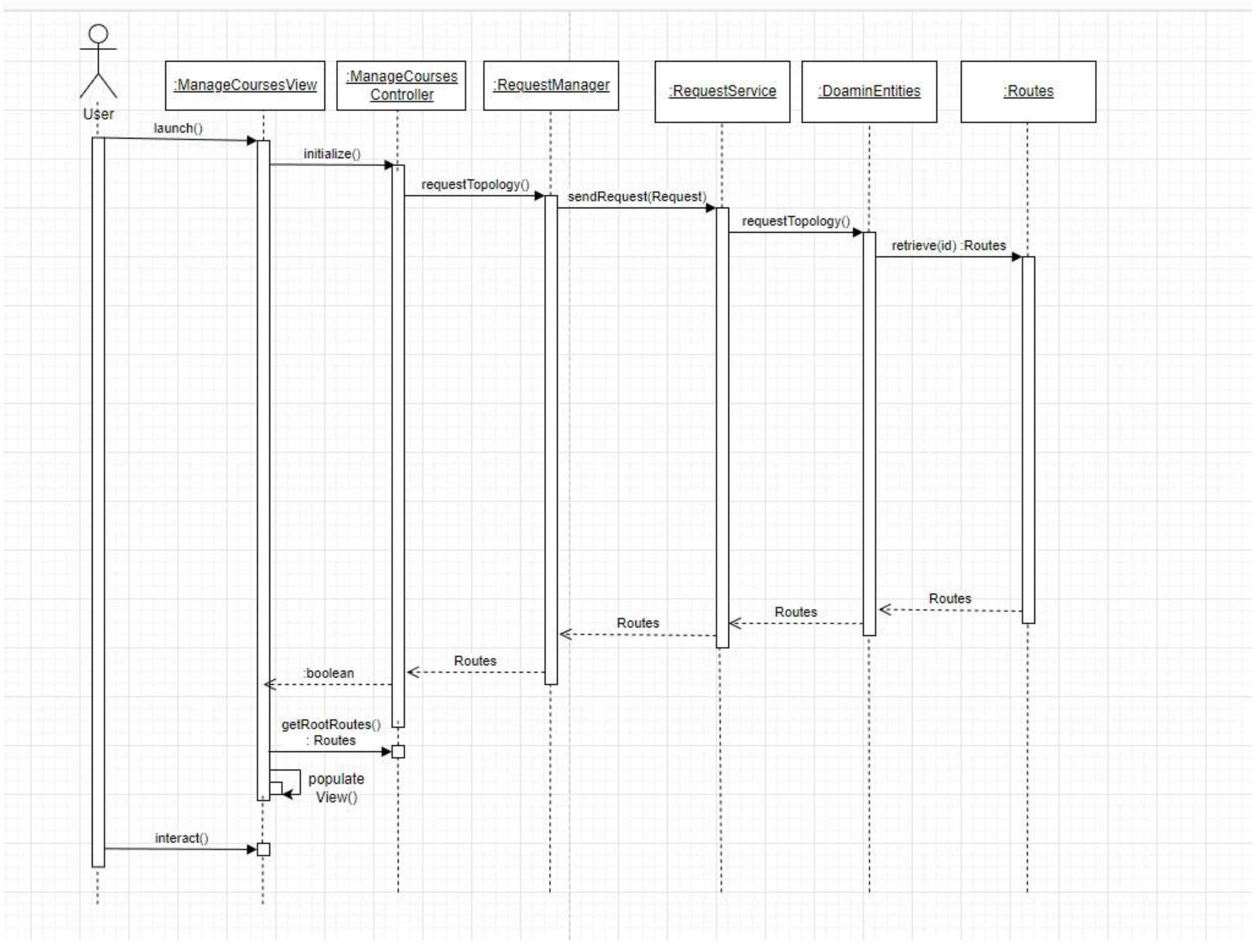
3. Figure below shows the module view with the modules that support the primary use cases.

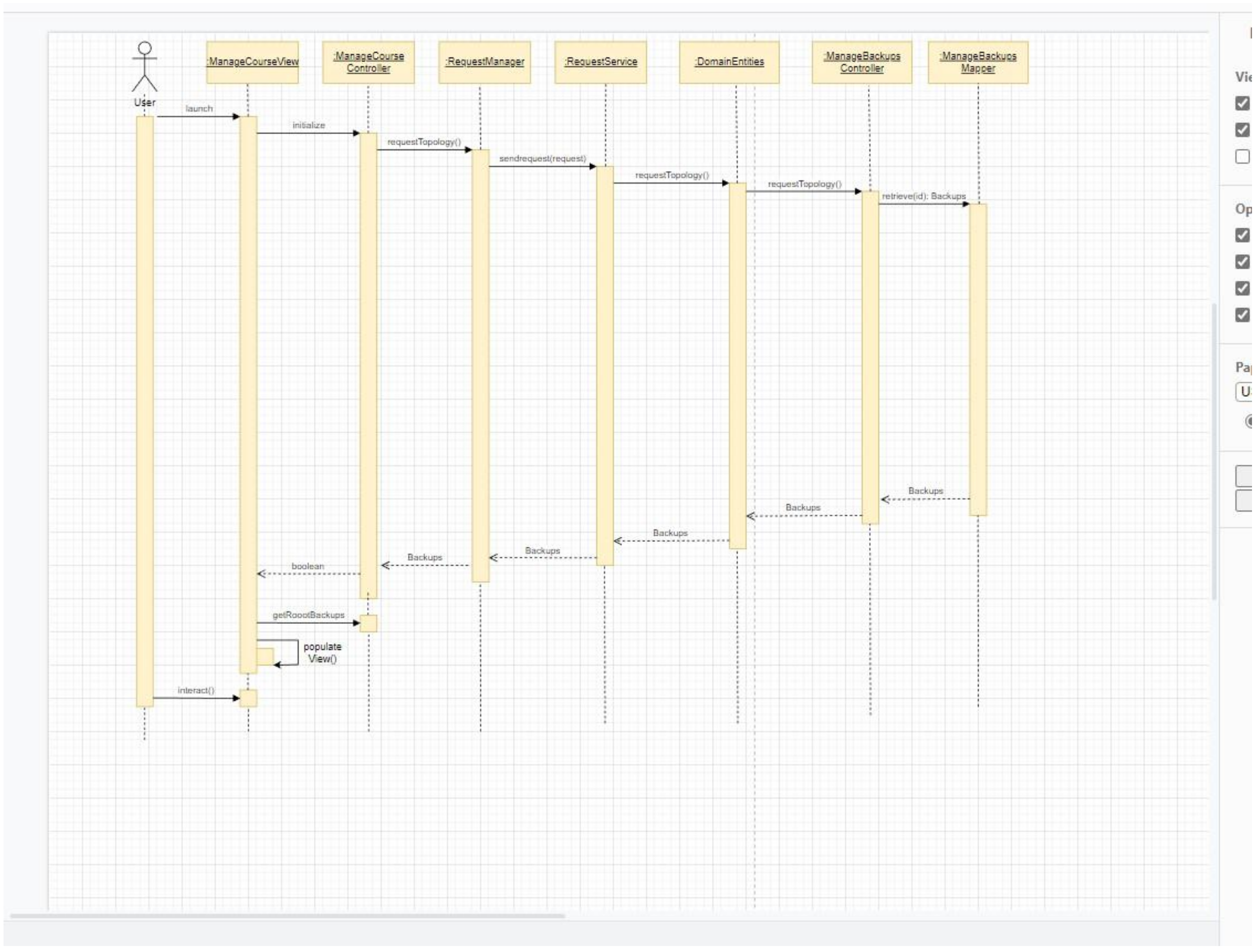


Element	**Responsibility**
CourseInformationView	Responsible for displaying the course information and updates it when events are received. UI components and process components from the reference architecture are embodied in the component.
ManageCoursesView	Responsible for displaying the manage courses view and updates it when events are received. UI components and process components from the reference architecture are embodied in the component.
CourseInformationController	Responsible for providing the necessary information to the presentation layer for displaying the course information functionality.
ManageCoursesController	Responsible for providing the necessary information to the presentation layer for displaying the manage courses functionality.
RequestManager	Communicates with the server-side logic.
RequestService	Receives requests from the clients.
DomainEntities	Consists of the entities from the server side domain model
ManageBackupsController	Responsible for the backups to the course management system
Routes	Responsible for the navigation among different routes in the application.
ManageBackupsMapper	Responsible for the operations that relate to backups.

4. The following sequence diagrams for the primary use cases: UC-1, UC-4 and UC-5 were created in the step of the method that defined interfaces.




User
:ManageCoursesView
:ManageCourses Controller
:RequestManager
:RequestService
:DoaminEntities
:Routes
launch()
initialize()
requestTopology()
sendRequest(Request)
requestTopology()
retrieve(id) :Routes
Routes
Routes
Routes
Routes
:boolean
getRootRoutes() : Routes
populate View()
interact()


User
:ManageCourseView
:ManageCourse Controller
:RequestManager
:RequestService
:DomainEntities
:ManageBackups Controller
:ManageBackups Mapper
launch
initialize
requestTopology()
sendrequest(request)
requestTopology()
requestTopology()
retrieve(id): Backups
Backups
Backups
Backups
Backups
Backups
boolean
getRoootBackups
populate View()
interact()

Step 7: Perform Analysis of Current Design and Review Iteration Goal and Achievement of Design Purpose

Through the decisions made in this iteration, an initial understanding of how functionality is supported in the system was determined.

Not Addressed	**Partially Addressed**	**Completely Addressed**	**Design Decision Made During the Iteration**
		UC-1	Modules across the layers and preliminary interfaces to support this use case have been identified.
		UC-4	Modules across the layers and preliminary interfaces to support this use case have been identified.
	QA-1		The elements that support the associated use case (UC-1) have been identified.
	QA-2		No relevant decisions made
	QA-3		No relevant decisions made
	QA-4		The elements that support the associated use case (UC-4) have been identified.
	QA-5		No relevant decisions made.
	QA-6		The elements that support the associated use case (UC-5) have been identified.
	CON-1		No relevant decisions made.
	CON-2		No relevant decisions made.
	CON-3		No relevant decisions made.
	CON-4		No relevant decisions made.
	CON-6		No relevant decisions made.
	CRN-1		No relevant decisions made.
	CRN-2		No relevant decisions made.
	CRN-3		Concern is expected to be addressed well through the capable and quality use of React and Express.
	CRN-4		No relevant decisions made.
	CRN-5		The architectural concern that all types of course information is provided was partially addressed in the third figure in iteration 2.

Iteration 3: Addressing Quality Attribute Scenario Driver

Step 2: Establish Iteration Goal by Selecting Drivers

For this iteration the focus will be on:

a. QA-2: Availability
b. QA-5: Security
c. CON-3: A database server must be used
d. CON-5: Must have a backup server for the system
e. CRN-2: Ensure that all information is kept private
f. CON-4: At least 100 users must be able to access the system at the same time.

Step 3: Choose One of More Elements of the System to Refine

In this iteration the elements will be refined are the application server and database server. Some attention will also be given to the security component.

Step 4: Choose One or More Design Concepts That Satisfy the Selected Drivers

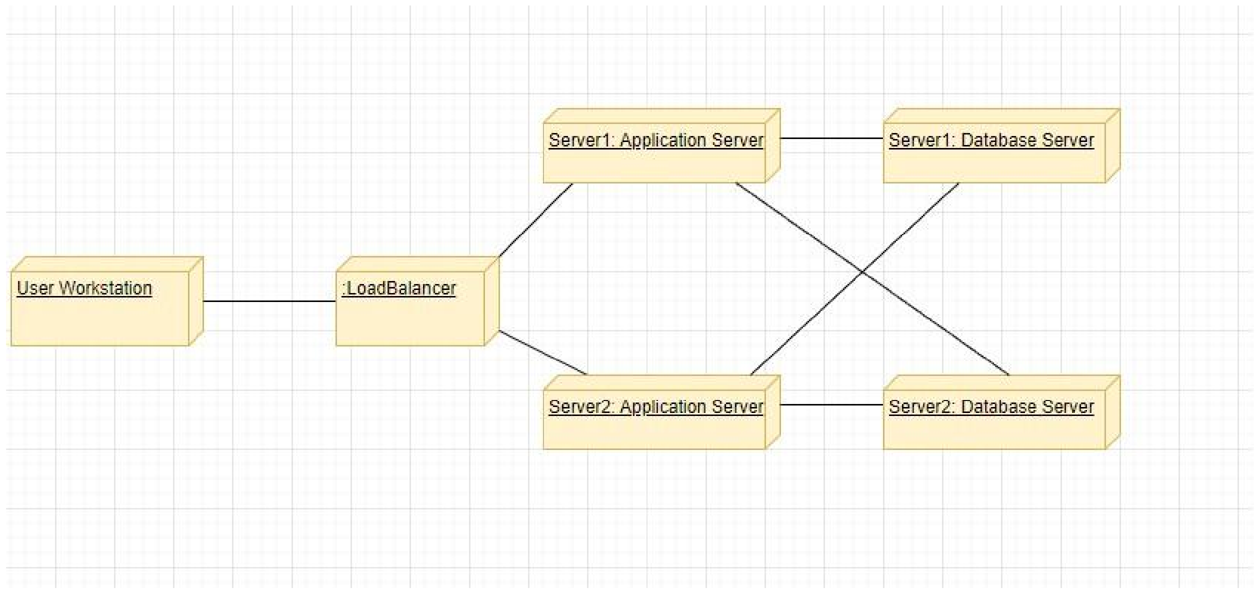
Design Decisions and Location	**Rationale**
Apply active redundancy in the application server	This allows the system to respond in the event of an application failure by switching to a replica of the current application. This fulfills QA-2 and CON-5.
Apply active redundancy in the database server	This allows the system to respond in the event of a database failure by switching to a replica of the current application. This fulfills QA-2 and CON-5.
Allow the system to be able to access data from the database	When new users join their login information will be stored in the database alongside their information on courses. This will ensure that only login information that matches can access the database. This fulfills CRN-2,CON-3 and QA-5.

Use a load balancer	A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. This fulfills CON-4.

Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

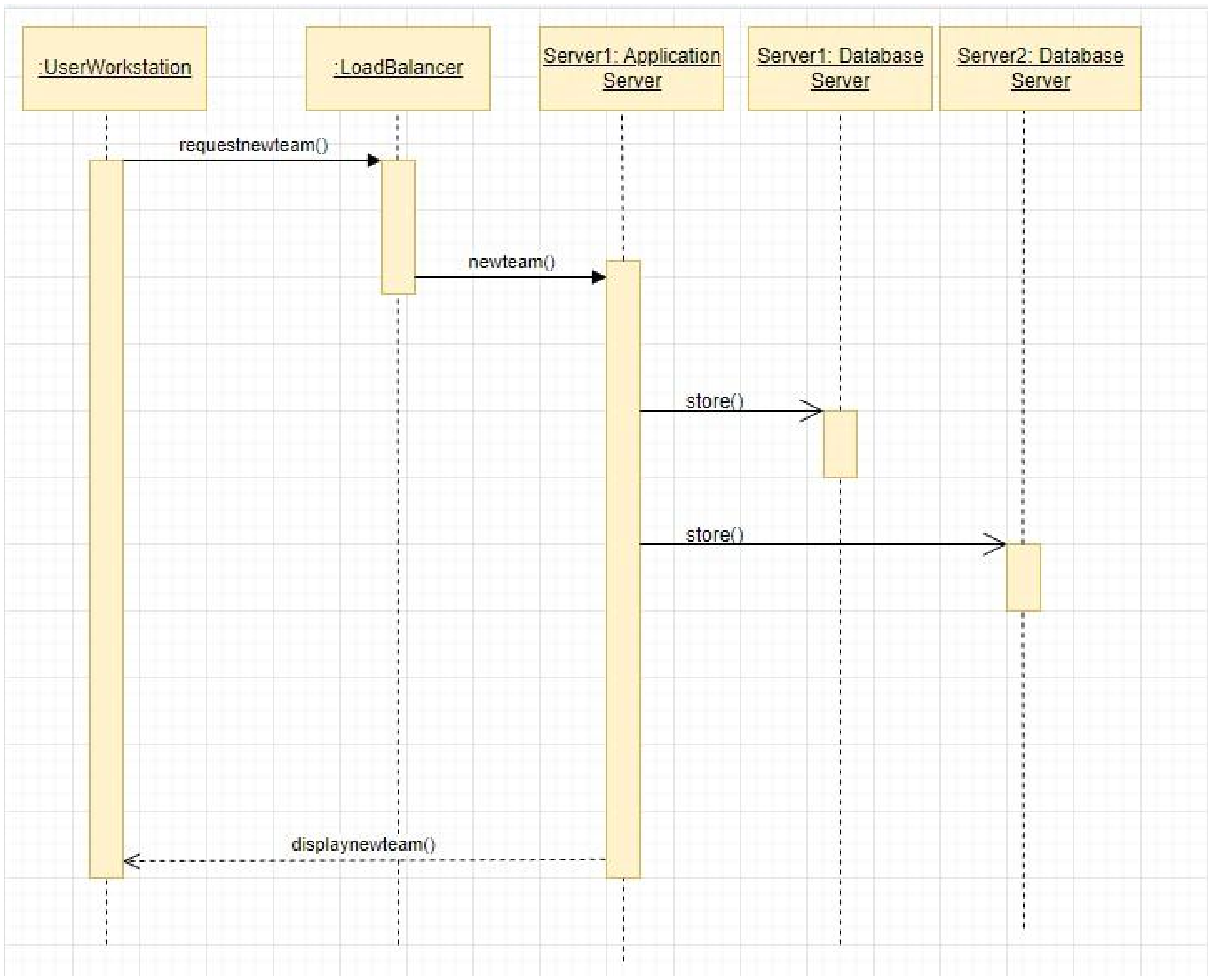
Design Decisions and Location	**Rationale**
Use MongoDB to store data	MongoDB is a document database. It stores data in a type of JSON format called BSON. We can use this store and retrieve user and course information. This fulfills CON-3.
Archiving backups	Backups that have been created and saved by previous users can be accessed and used while maintenance is being done. This fulfills CON-5.
Upload Load balancer	Load balancer used to control traffic on the two application servers. This fulfills CON-4.

Step 6: Sketch Views and Record Design Decisions



Element	**Responsibility**
LoadBalancer	Dispatches (and balances the load of) requests coming from clients to the application servers. The load balancer also presents a unique IP address to the clients.

The UML sequence diagram shown below shows how a lecturer can make a new team to support UC-2,which is also associated with QA-2 (availability) .



Step 7: Perform Analysis of Current Design and Review Iteration Goal And Achievement of Design Purpose

Not Addressed	Partially Addressed	Completely Addressed	Design Decision Made During the Iteration
	QA-1		If a student is denied access then he can't see information associated with that account
	QA-2		Duplicated servers to ensure that during a failure the system can still run even with a failure of one component.
	QA-3		No relevant decisions made.
	QA-4		No relevant decisions made
	QA-5		Login information stored in database has to match one entered to be given access to account
	QA-6		No relevant decisions made
	CON-1		No relevant decisions made
CON-2			No relevant decisions made
	CON-3		MongoDB was chosen as a database to store user and course information
	CON-4		Load balancer was introduced to make sure the system could handle lots of requests from multiple users
	CON-5		Backups were made and can be made for each system
	CON-6		No relevant decisions made
	CRN-1		The servers have duplicates that can be used if one part fails so the downtime of the system is very low
	CRN-2		Only users with the correct password and username that stored in the database can have access to information associated with that account
	CRN-3		No relevant decisions made
	CRN-4		No relevant decisions made
	CRN-5		No relevant decisions made