SOFE 3650 - Fall 2018

Software Design and Architectures

Final Project

Prepared By
Arda Celik – 100596185 || Thomas Jansz – 100642111

Due: December 5, 2018
Instructor: Dr. Ramiro Liscano

Objective and Individual Contribution

Objective

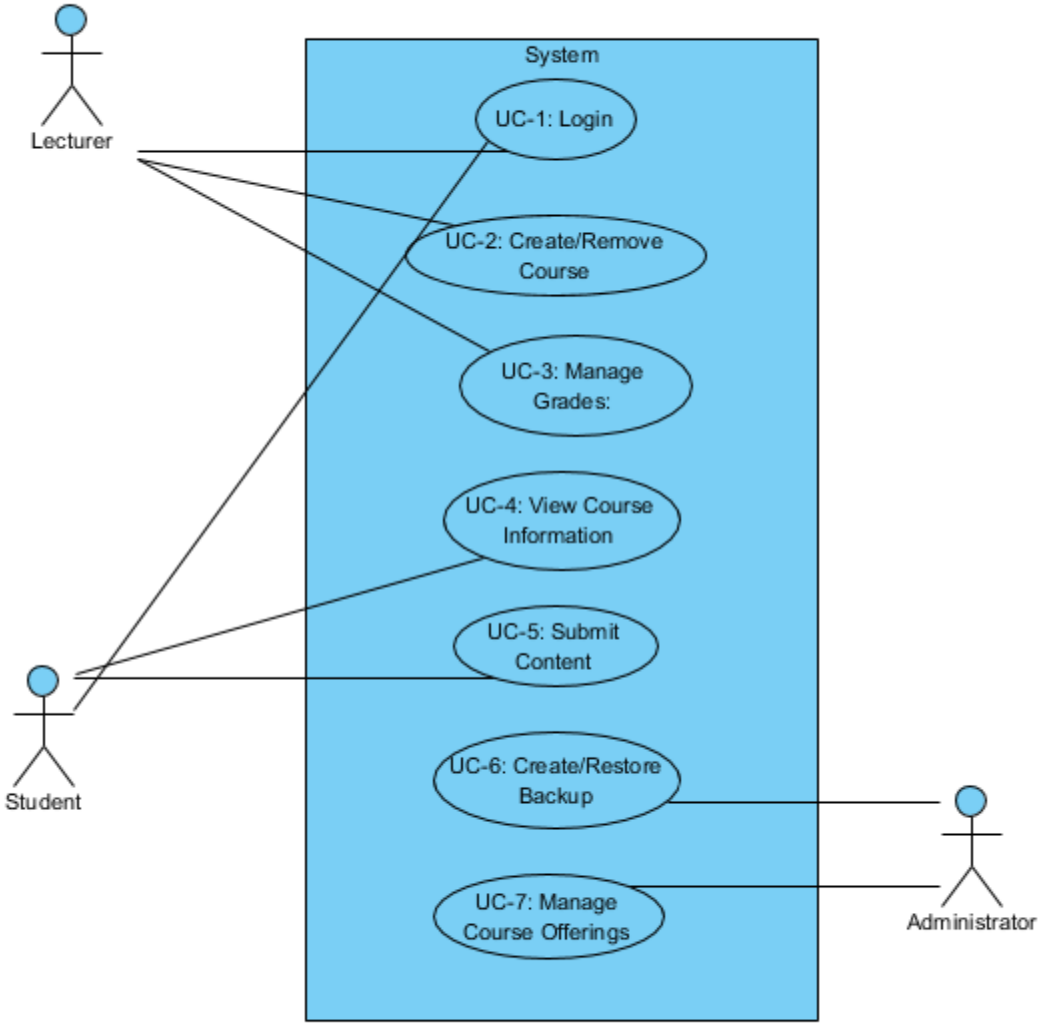
“The goal of this project is to demonstrate a methodological set of steps in the design of a software architecture for a set of requirements provided by your instructor. The expected design approach to take is the Attribute Driven Design (ADD) presented in some detail in the book “Designing Software Architectures: A Practical Approach” by Humberto Cervantes and Rick Kazman and covered in the course.”

Individual Contributions

Group Member	**Contribution**
Arda Celik	50%
Thomas Jansz	50%

SYSTEM REQUIREMENTS

Use Case Model



Use Case	Description
UC-1: Login	There are 4 types of users in the system and each user has different privileges. With a login functionality we are differentiating different type of users and create privacy in the system.
UC-2: Create/remove content	Lecturers will upload course related material such as lecture notes/slides, lab manuals and practice quizzes for students.
UC-3: Manage grades	Student's grades will be updated regularly depending on the course grade breakdown. This will be done by lecturers.
UC-4: View course information	Each course page should provide general information about the course.
UC-5: Submit content	Students need to submit their work (e.g. assignments, lab reports, final projects) for grading.
UC-6: Create/restore backup	It is really important to have a backup in case the system stops working due to some reason. This backup should be easily restored.
UC-7: Manage course offerings	Course offerings might change every year so the system should reflect these changes.

Quality Attribute Scenarios

ID	Quality Attribute	Scenario	Associated Use Case
QA-1	Security	The system contains sensitive data such as, first name and last name of a student that should not be seen by someone outside of the system. Also each users account needs to be private to make sure other users are not performing activities that they are not supposed to perform.	UC-1
QA-2	Availability	When the system needs to be maintained, it needs to be done during low-intensity hours. For instance, if a backup needs to be restored by maintainers it should not be done during school time where students and lecturers might need to access the system.	UC-6
QA-3	Interoperability	The system provides a convenient way for students to manage their calendar. They can add their scheduled lecture times to the calendar as well as events, deadlines and any other personal notes which makes it easy for them manage their time appropriately by keeping everything in one place.	UC-7
QA-4	User Friendliness	If the system's user interface is complicated and not intuitive, it will be difficult for users such as students and lecturers to perform certain activities. For this reason the system needs to be user friendly.	All
QA-5	Accessibility	The system provides support to its disabled users. It should be equally easy for blind students to navigate and perform the activities they are expected to perform.	All

Constraints

ID	Constraint
CON-1	Must have 99.999% availability
CON-2	Submission file size should be limited
CON-3	Student/Lecturer must be able to view/make changes to his/her own class
CON-4	Constrained to Java application compatibility

Architectural Concerns

ID	Concern
CRN-1	Establishing initial system structure
CRN-2	Leverage team's knowledge on Java and JavaFX
CRN-3	Allocate work to member of the development team

The Design Process

ADD Step 1: Review Inputs

Iteration 1: Establishing an Overall System Structure

Step 2: Establish Iteration Goal by Selecting Drivers

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

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

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

Step 6: Sketch Views and Record Design Decisions

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

Iteration 2: Identifying Structures to Support Primary Functionality

Step 2: Establish Iteration Goal by Selecting Drivers

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

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

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

Step 6: Sketch Views and Record Design Decisions

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

Iteration 3: Addressing Quality Attribute Scenario Driver

Step 2: Establish Iteration Goal by Selecting Drivers

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

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

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

Step 6: Sketch Views and Record Design Decisions

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

Iteration 1: Establishing an Overall System Structure

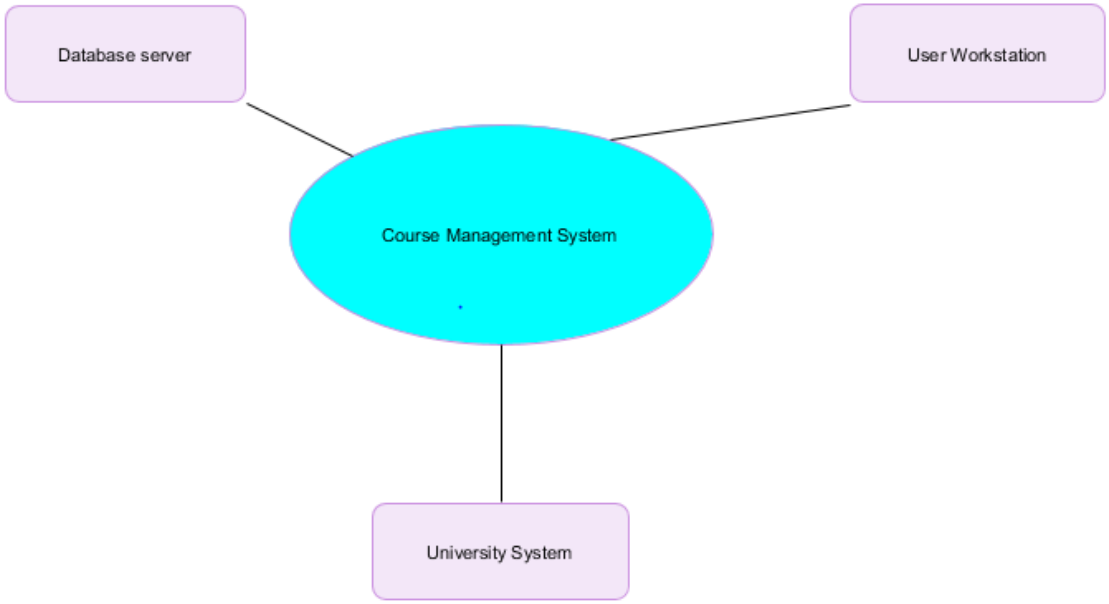
ADD Step 1: Review Inputs

<table>
<tr><th>Category</th><th>Details</th></tr>
<tr><td>Design Purpose</td><td>This is a greenfield system for a mature domain. The purpose is to produce a sufficiently detailed design to support the construction of the system.</td></tr>
<tr><td>Primary functional requirements</td><td>From the use cases presented, the primary ones were determined to be:
UC-2: Because it directly supports the core business
UC-3: Because it directly supports the core business
UC-4: Because it directly supports the core business
UC-5: Because it directly supports the core business</td></tr>
<tr><td>Quality attribute scenarios</td><td>The scenarios were previously described, they have now been prioritized as follows:
<table>
<tr><th>Scenario ID</th><th>Importance to the Customer</th><th>Difficulty of Implementation According to the Architect</th></tr>
<tr><td>QA-1</td><td>Medium</td><td>Low</td></tr>
<tr><td>QA-2</td><td>High</td><td>High</td></tr>
<tr><td>QA-3</td><td>Medium</td><td>Medium</td></tr>
<tr><td>QA-4</td><td>High</td><td>Low</td></tr>
<tr><td>QA-5</td><td>High</td><td>High</td></tr>
</table>
From the list, only QA-2, QA-4, and QA-5 are selected as drivers.</td></tr>
<tr><td>Constraints</td><td>All of the constraints discussed previously are included as drivers.</td></tr>
<tr><td>Architectural constraints</td><td>All of the architectural constraints discussed previously are included as drivers</td></tr>
</table>

Step 2: Establish Iteration Goal by Selecting Drivers

- QA-1: Security
- QA-2: Availability
- QA-4: User Friendliness
- CON-4: Constrained to Java application compatibility
- CRN-2: Leverage team's knowledge on Java and JavaFX

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



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

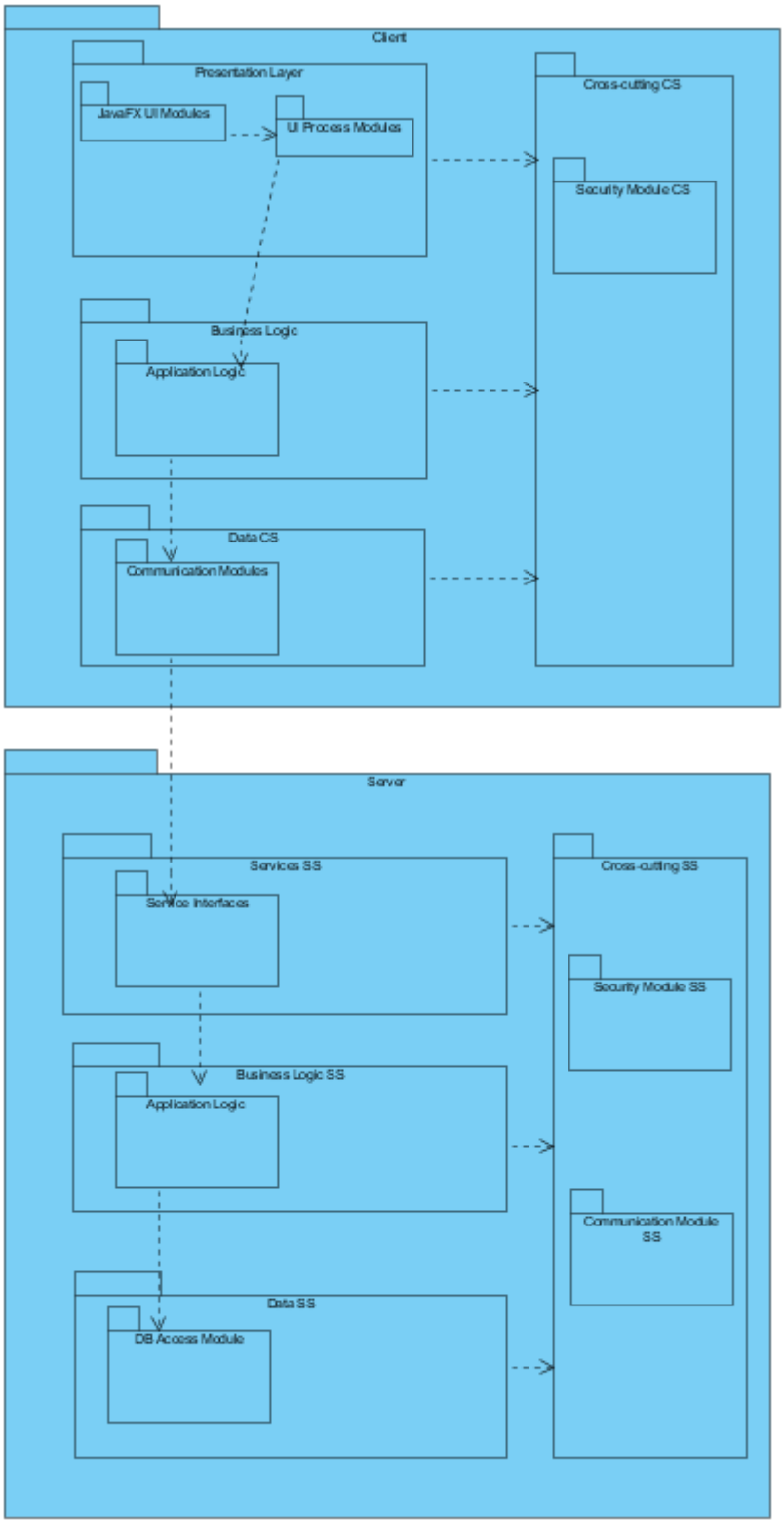
<table>
<tr><th>Design Decisions and Location</th><th>Rationale</th></tr>
<tr><td>Logically structure the client part of the system using the Rich Client Application reference architecture</td><td>“Rich client applications are installed and run on a user’s machine. Because the application runs on the user’s machine, its user interface can provide a high-performance, interactive, and rich user experience.” (Cervantes, 2016)
This decision allows us to leverage the familiarity with the Java technologies which addressed CON-4 and CRN-2. Since we are not using web technologies and our system is not accessible from a web browser using Java technologies is an effective solution.
Discarded alternatives:
<table>
<tr><th>Alternative</th><th>Reason for Discarding</th></tr>
<tr><td>Mobile Applications</td><td>This type of reference architecture is more suited for handheld devices. We want our system to be accessible from student laptop computers and the desktop computers located on campus.</td></tr>
<tr><td>Web Applications</td><td>This reference architecture is discarded due to unfamiliarity with designing and developing secure, full stack web applications that provide rich user interface experience.</td></tr>
<tr><td>Rich Internet applications</td><td>Just like web application reference architecture, this alternative is also discarded due to unfamiliarity with web technologies and the rich development capabilities provided by the Java environment.</td></tr>
</table></td></tr>
<tr><td>Logically structure the server part of the system using the Service Application reference architecture</td><td>“Service applications do not provide a user interface but rather expose services that are consumed by other applications” (Cervantes, 2016)
Since this part of the system does not need to be interactive, we are not worried about the presentation layer. Loose coupling that comes with using Service Application reference architecture also help us achieve high availability (QA-2) as system maintenance can be done during downtimes without having a negative impact on the client side of our system.</td></tr>
<tr><td>Physically structure the application using the three-tier deployment pattern</td><td>A three tier deployment is appropriate since the system requires the use of a database, a middle layer to establish the business logic and a client layer (e.g. student’s laptop). Other n-tier patterns are discarded because extra servers are not required (when n>4) and a 2-tier architecture does not include a database layer.</td></tr>
<tr><td>Build the user interface of the client</td><td>The developer team is already familiar with Java technologies (CRN-2) and a user friendly (QA-4) interface can easily be created with this decision.</td></tr>
</table>

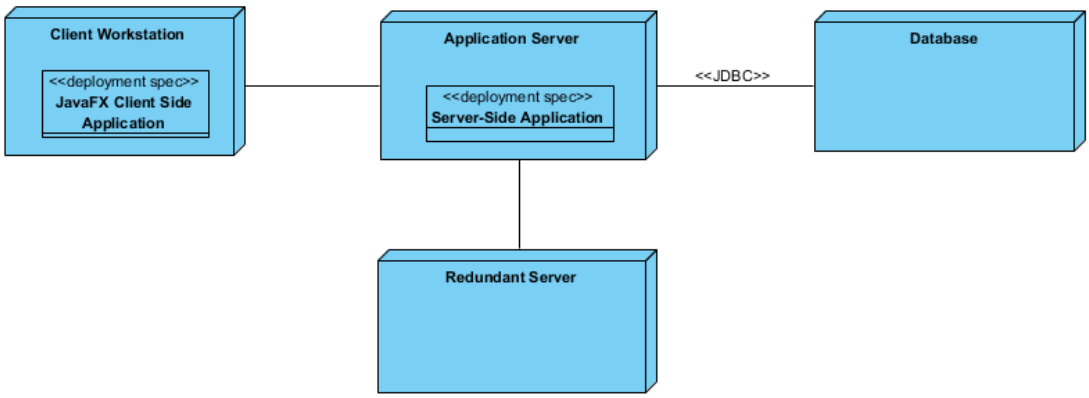
application using JavaFX	
Deploy the application using Spring	Although it can quite complex, Spring provides great tool support, easy integration with other frameworks and security (QA-1).

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

Design Decision and Location	Rationale
Local data sources not required for rich client application	To easier ensure data integrity, local data is not needed. All required data should be instantly updated in database/model. Network connection is generally reliable and not a large inconvenience.
Another redundant server to act as a load balancer/redundancy	Needed to ensure availability requirements in the event of a physical failure or other critical outage in one of servers.

Step 6: Sketch Views and Record Design Decisions





Presentation client side (CS)	**Contains modules that involve the user interface**
Business logic	Layer contains modules that perform the primary application logic on the client side
Data CS	Layer that includes modules that involve communication to server
Cross-cutting CS	Involves modules that span across many layers that involve security
JavaFX UI Modules	Java modules that render and receive input from the user
UI Process Modules	Modules are responsible for control flow
Business Modules	Application logic layer performed on client side
Communication Modules CS	Perform services to connect to server from client
Services server side	Layer has modules which allows server resources to be accessible by client
Data SS	Layer contains modules that are responsible for communication with database
Cross-cutting SS	Modules have functionality goes across different layers, such as security and communication
Service interfaces SS	These modules expose services consumes by client
Business modules SS	Contain application logic
DB access module	Module which communicates to external database

User Workstation	**User PC, which hosts the client**
Application Server	Server performs authentication and other logic of application
Database Server	Server that holds model – relational data

Relationship	**Description**
Between controller and database	Communicates using JDBC
Between controller and client application	Communicates using REST

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

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
UC-2			No relevant decisions made.
UC-3			No relevant decisions made.
UC-4			No relevant decisions made.
UC-5			No relevant decisions made.
	QA-1		Spring framework is introduced which provides great tool support, easy integration with other frameworks and security.
	QA-2		Service Application reference architecture is used to achieve high availability as system maintenance can be done during downtimes without having a negative impact on the client side of our system.
	QA-4		The goal of producing a modern, efficient, and fully featured rich client applications is achieved by using JavaFX on the client side.
		CON-4	Since both the client side and the server side of our system is written in Java technologies, execution under different operating systems (e.g. Windows, Linux, OSX) is supported.

	CRN-2	Technologies that have been selected so far were based on the team's knowledge and familiarity with that technology.

Iteration 2: Identifying Structures to Support Primary Functionality

The goal of this iteration is to address the general architectural concern of identifying structures that support the primary functionality of the application.

Step 2: Establish Iteration Goal by Selecting Drivers

The following primary use cases will be addressed:

UC-2: Create/remove content

UC-4: View Course Information

Step 3: Choose One or More Design Concepts That Satisfy the Selected Drivers

In this iteration we will be refining the server's architecture in the Rich Client Application Architecture.

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

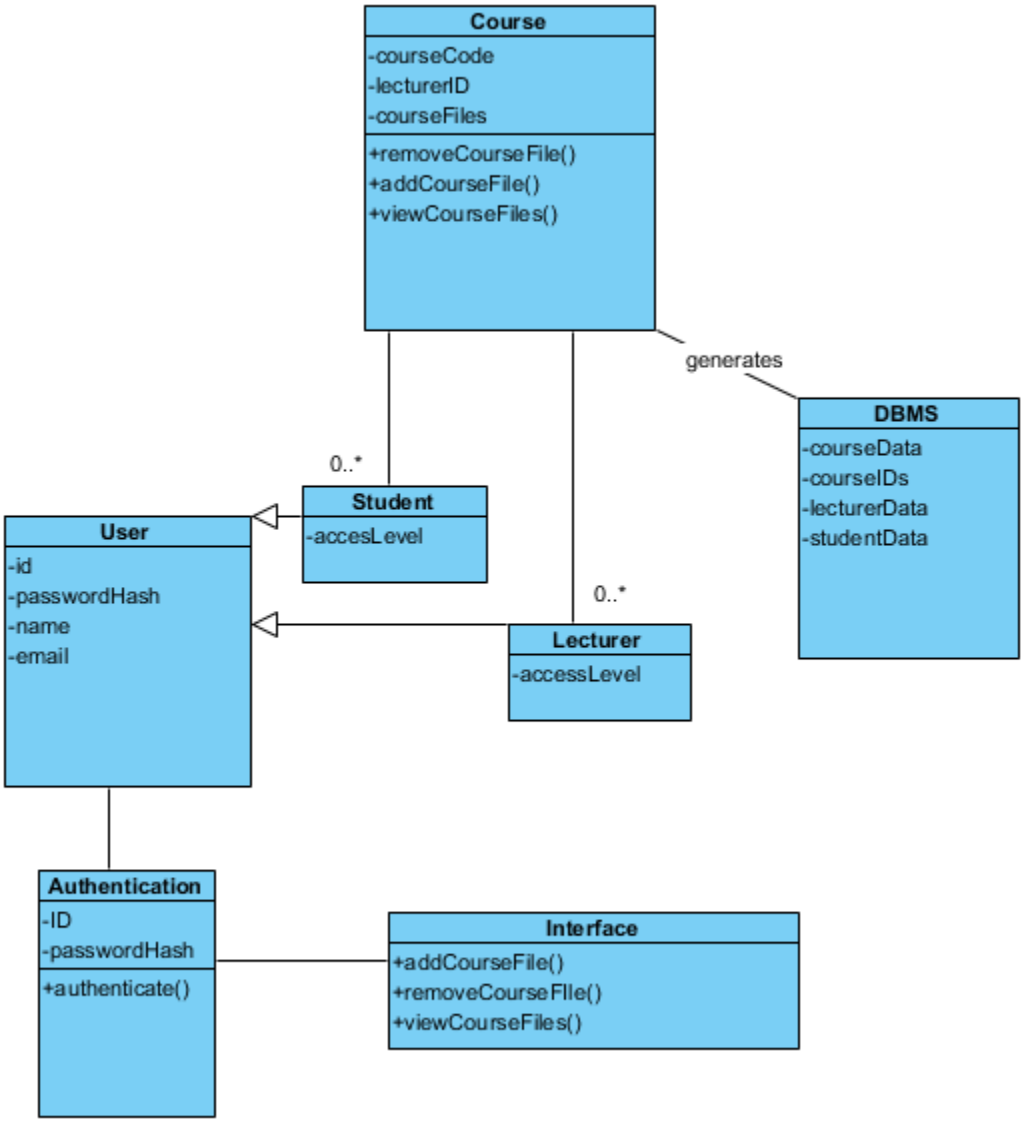
Design Decisions and Location	Rationale and Assumptions
Create a Domain Model for the application	In order to identify the major entities of the system and their relationship with each other in the domain, a domain model needs to be created. No alternatives exist for the domain model as it eventually needs to be discussed.
Identify Domain Objects that map to functional requirements	A domain object is a building block of the application as it encapsulates the functional elements.
Decompose Domain Objects into general and specialized components	Functionality is represented by domain objects but some small-scaled extra elements located within the layers might be needed. Each module is associated with a layer.
Use Spring framework	Spring is selected as the development team is already familiar with it. It is a popular Java framework that is light and it has a good tool support.

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

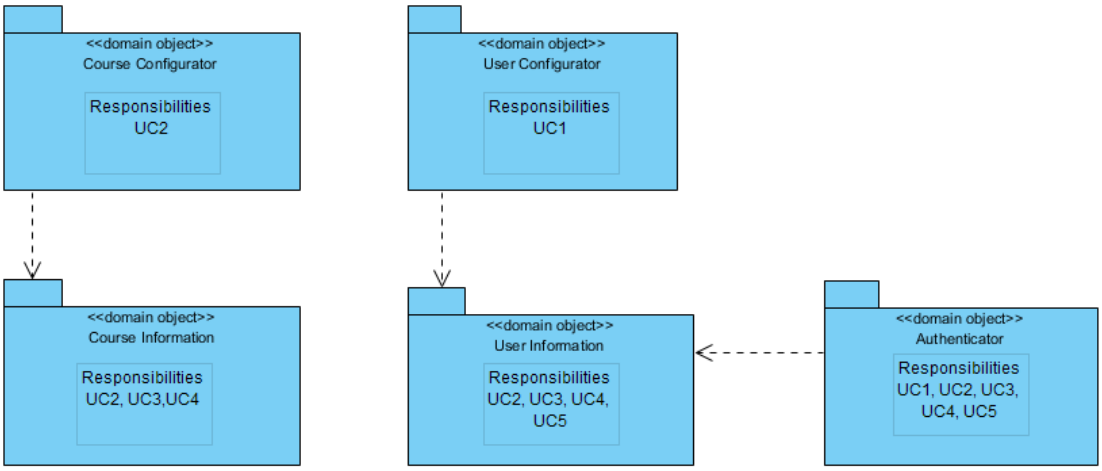
Design Decisions and Location	Rationale
Create only an initial domain model	In order to accelerate the design process only an initial domain model is created.
Map the system use cases to domain objects	The system's use cases need to be analyzed to identify the domain objects. For each use case, domain objects are identified to achieve CRN-3.
Decompose the domain objects across the layers to identify layer-specific modules with an explicit interface	The architect identifies the modules based on the primary use cases. It is recommended that other team members contribute to this process by identifying the rest of the modules. CRN-3 is addressed again by dividing the work between the members in the team. As modules are being identified, a new architectural concern is realized: CRN-4: Most of the module should be unit tested.
Connect components associated with modules using Spring	With Spring the modules can be unit tested and different aspects can be supported.
Associate frameworks with a module in the data layer	Spring is associated with the modules in the server side.

Step 6: Sketch Views and Record Design Decisions

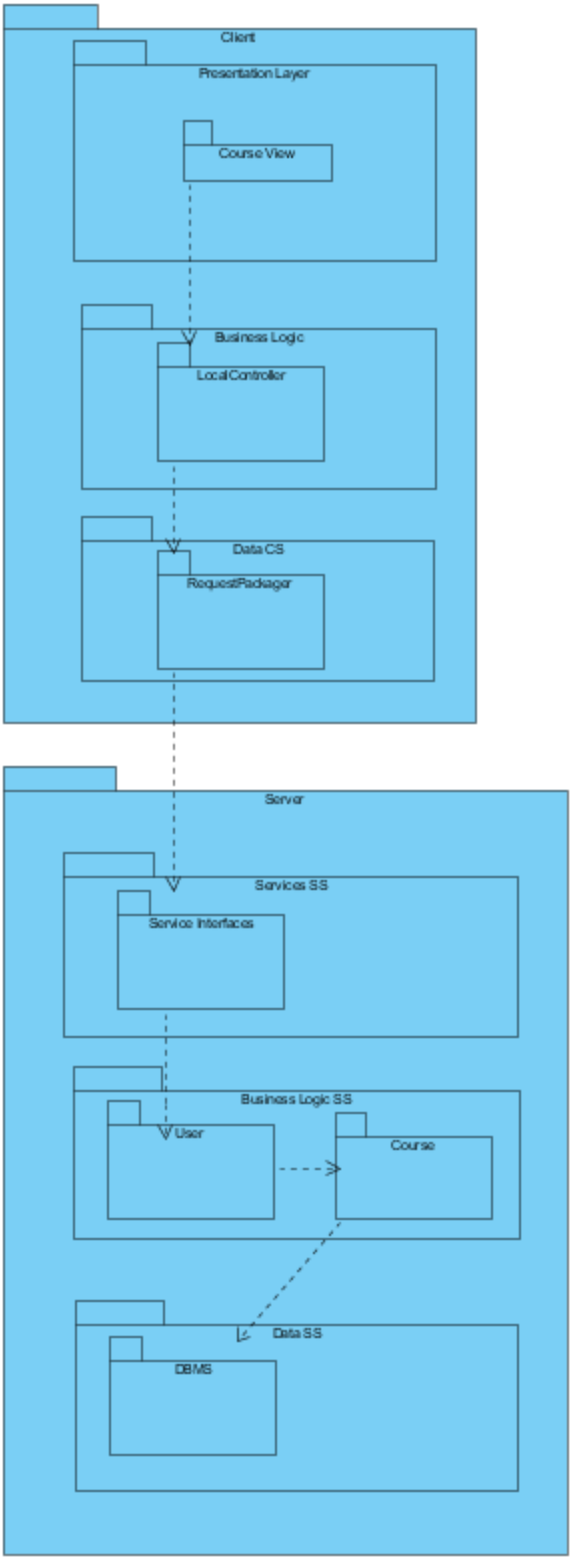
Initial Domain Model



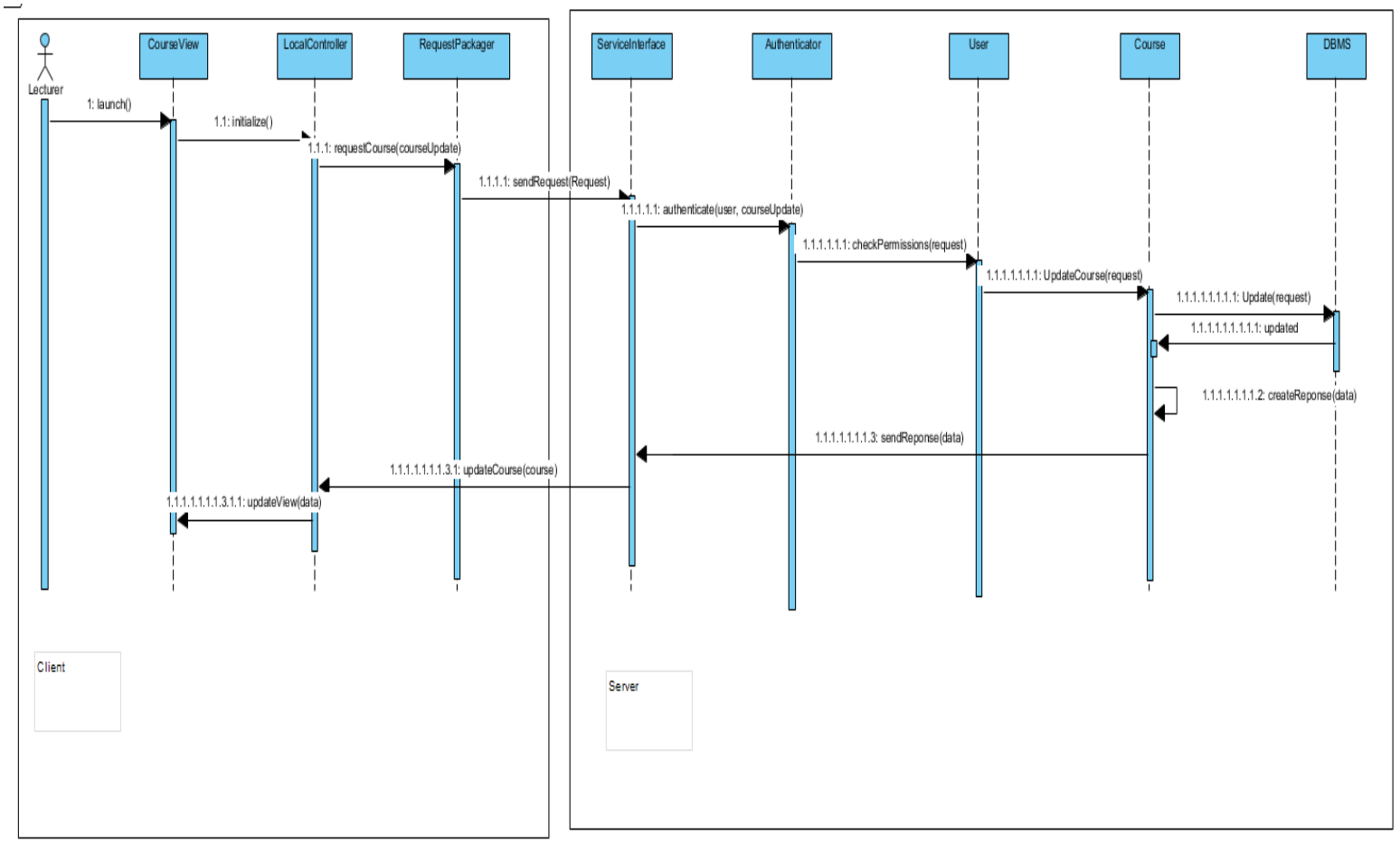
Domain Objects



Module View


Client
Presentation Layer
Course View
Business Logic
Local Controller
Data CS
RequestPackager
Server
Services SS
Service Interfaces
Business Logic SS
User
Course
Data SS
DBMS

Sequence Diagram



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

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
		UC-2	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		Some modules across layers and preliminary interfaces to support this have been identified
	QA-2		Some modules across layers and preliminary interfaces to support this have been identified
	QA-3		No relevant decisions made, but system is modular
		QA-4	Use of rich UI interface support this quality attribute
QA-5			No relevant decisions made
	CON-1		Some modules across layers and preliminary interfaces to support this have been identified
CON-2			No relevant decisions made
		CON-3	The elements that support the associated use case have been identified
		CON-4	The elements that support the associated use case have been identified
		CRN-1	Initial system structure have been established after iterations
		CRN-2	Team's knowledge of Java and related frameworks have been leveraged
		CRN-3	Application has been broken down into units to be distributed across members of team

Iteration 3: Addressing Quality Attribute Scenario Driver

In this section, we are building on the fundamental structural decisions made in the previous iterations. We can reason about the fulfillment of one of the most important quality attributes: this iteration focuses on quality attribute scenario QA-2 (Availability).

Step 2: Establish Iteration Goal by Selecting Drivers

For this iteration, the focus is on the QA-2 (Availability) quality attribute scenario. The system should keep functioning in case of a failure within a short period of time.

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

In this availability scenario, the elements that will be refined were identified during the first iteration:

- Application server
- Database server

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

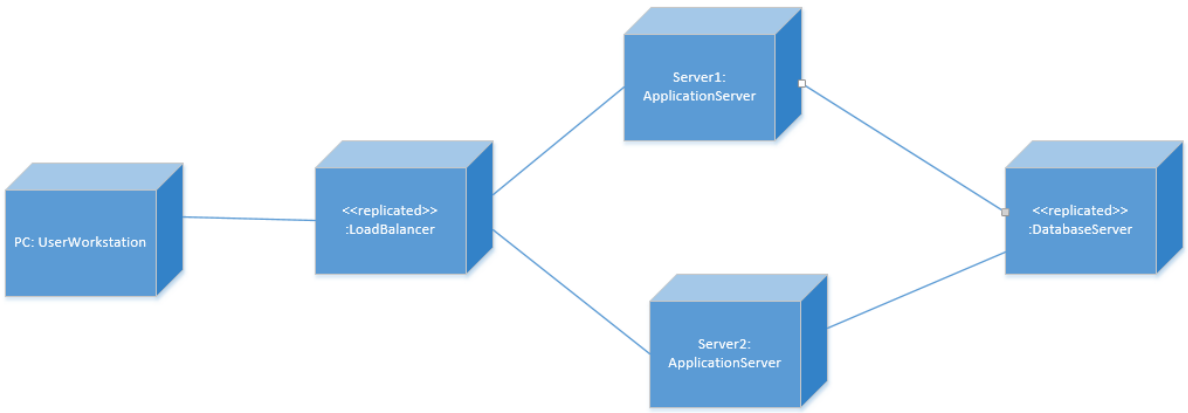
Design Decisions and Location	Rationale and Assumptions
Active redundancy is introduced by replicating the application server and other critical components such as the database	The functionality of the system will not be affected in case of a failure when the critical parts are replicated.
Create backup servers for both the application server and the database server	By having backups of the servers the system can be easily restored in case of an error.

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

Design Decisions and Location	Rationale
Use active redundancy and load balancing in the application server	Since there are two servers that are copies of each other are running at the same time, it is recommended to distribute the load between two servers so that the system can perform in an efficient way.

Step 6: Sketch Views and Record Design Decisions

The figure below shows the updated deployment diagram that introduces the active redundancy and load balancing.



Element	Responsibility
LoadBalancer	Balances the load of requests coming to the main application server
Server2:ApplicationServer	The copy of Server1: ApplicationServer. It helps balancing the load and it operates in case of a failure that might happen in Server1: ApplicationServer.

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

Not Addressed	Partially Addressed	Completely Addressed	Design Decisions Made During the Iteration
		UC-2	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		Some modules across layers and preliminary interfaces to support this have been identified
		QA-2	The load balancer is introduced as well as a replica of the application server. By using a load balancer we are dividing the requests coming to the server. Also, replicated server keeps the system functioning in a short period of time which improves the availability.
	QA-3		No relevant decisions made, but system is modular
		QA-4	Use of rich UI interface support this quality attribute
QA-5			No relevant decisions made
	CON-1		Some modules across layers and preliminary interfaces to support this have been identified
CON-2			No relevant decisions made
		CON-3	The elements that support the associated use case have been identified
		CON-4	The elements that support the associated use case have been identified
		CRN-1	Initial system structure have been established after iterations
		CRN-2	Team's knowledge of Java and related frameworks have been leveraged
		CRN-3	Application has been broken down into units to be distributed across members of team