Iteration 1: Establishing an Overall System Structure

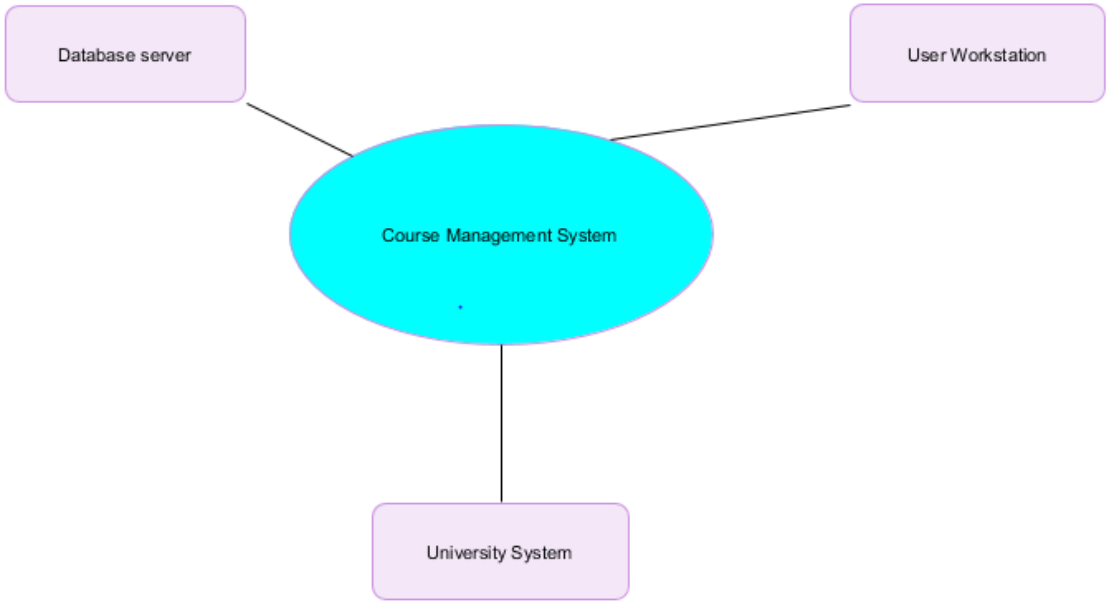
ADD Step 1: Review Inputs

<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

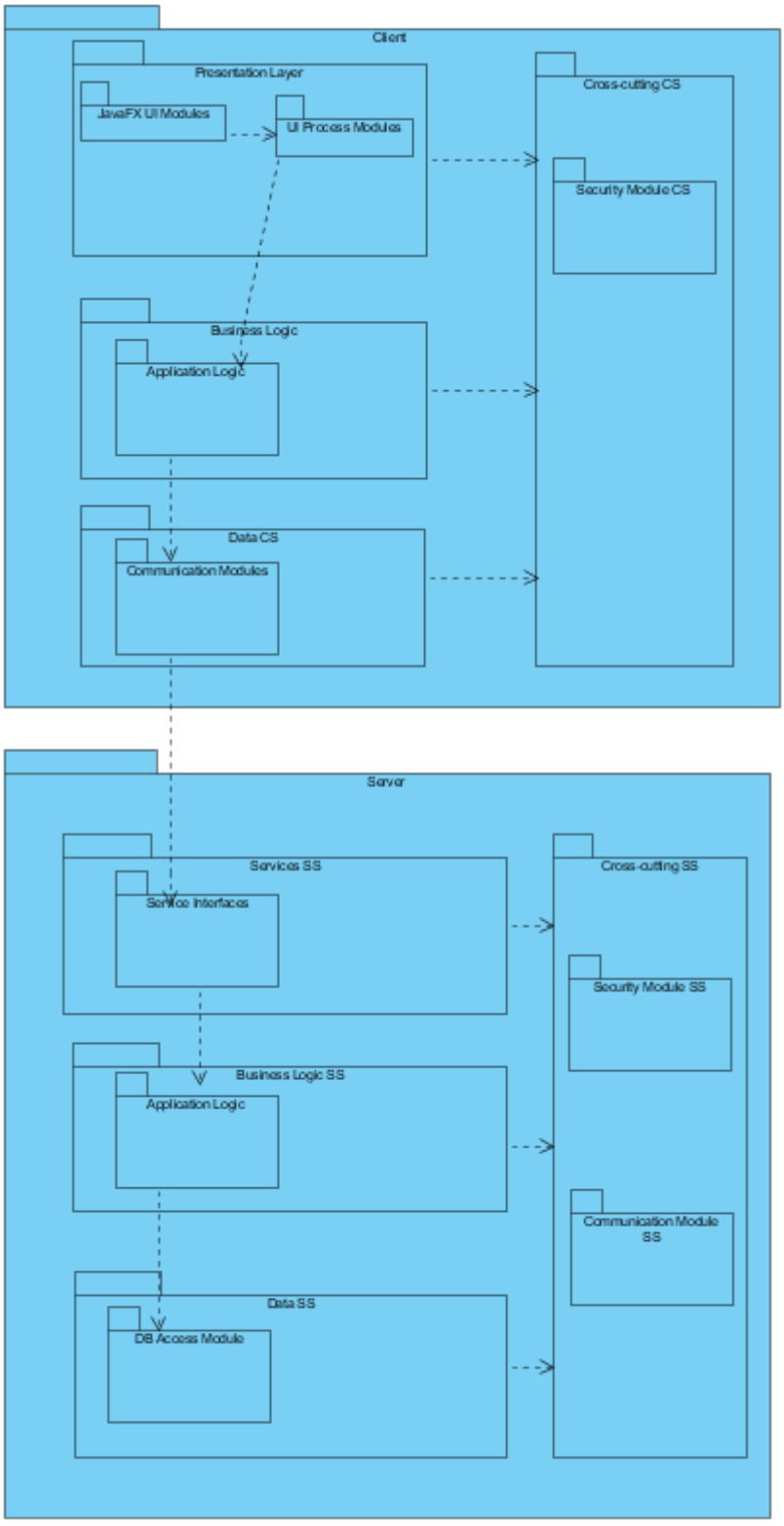
<table>
<tr><th>Design Decisions and Location</th><th>Rationale</th></tr>
<tr><td>Logically structure the client part of the system using the Rich Client Application reference architecture</td><td>“Rich client applications are installed and run on a user’s machine. Because the application runs on the 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>

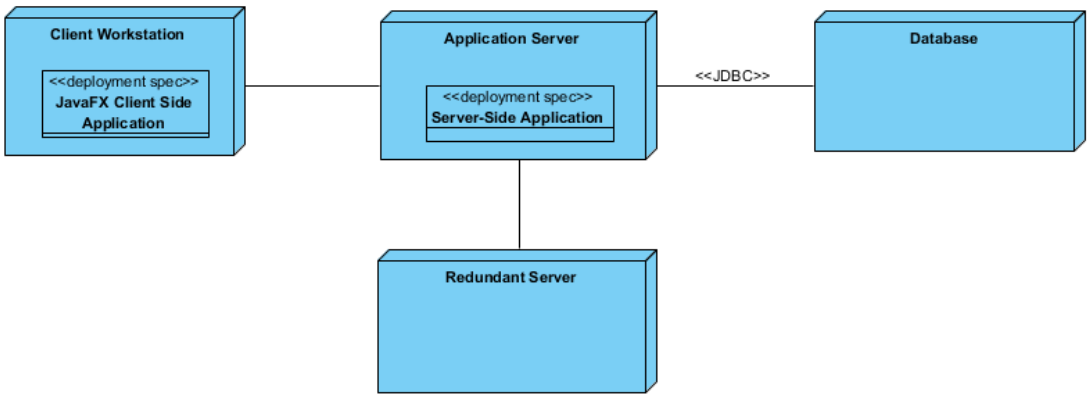
application using JavaFX	
Deploy the application using Spring	Although it can quite complex, Spring provides great tool support, easy integration with other frameworks and security (QA-1).

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.