Step 1: Review Inputs

Primary functional requirements:

Use Case 1: Account Management	Because it enables user access to the CMS
Use Case 2: Course Management	Because it directly supports the core functionality
Use Case 5: Security and Permissions	Because it facilitates the segregation of user-levels and their access/modification of the system

Quality Attribute Scenarios:

Quality Attribute ID	Importance	Difficulty
QA-1	M	L
QA-2	H	H
QA-3	M	L
QA-4	M	L
QA-5	H	H
QA-6	H	H

From this list, only QA-2, QA-5, and QA-6 will be selected as drivers, due to their high difficulty but high importance to the system. In the overall architecture decision stage, high difficulty can be lowered depending on chosen structures.

All constraints are included as drivers

All concerns are included as drivers

Iteration 1: Establishing an Overall System Structure

Step 2: Establish Iteration Goal by Selecting Drivers

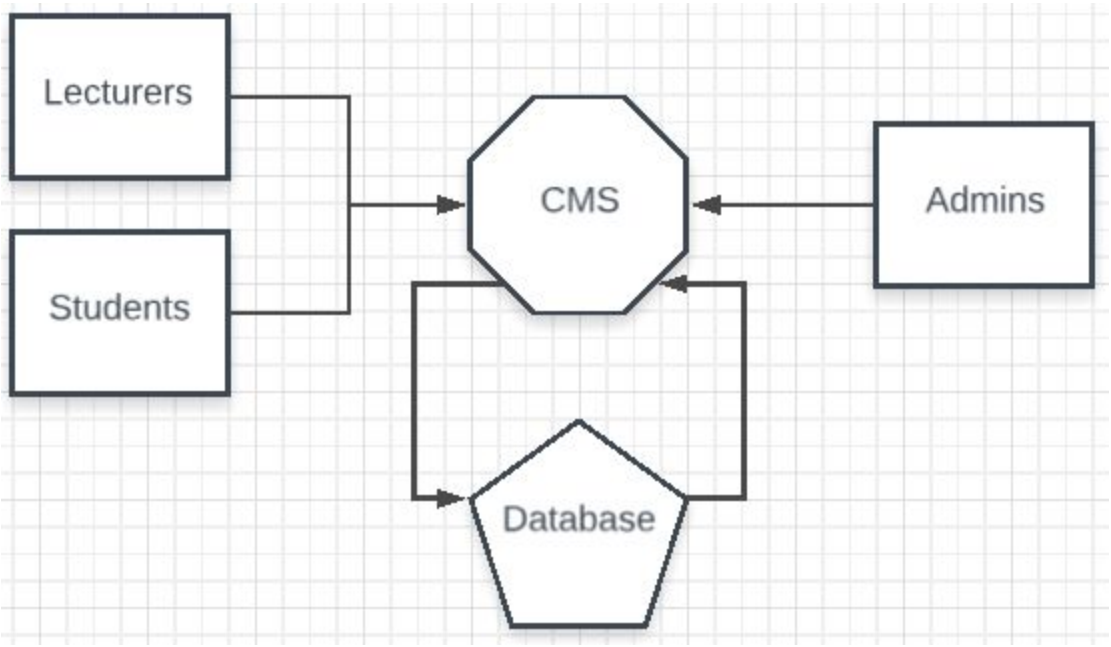
This being the first iteration in the design of a greenfield system (mature domain), our goal will be to satisfy CNR-1 “Establish an Overall System structure”. While deciding on a system architecture, we should be mindful of the impact it will have on our Quality Attributes, especially those we’ve selected as drivers: Availability, Security, and Extensibility. We should also work within out constraints of:

- **CON-1** (50 000 simultaneous users)
- **CON-2** (web browser useable)
- **CON-3** (MySQL relational database)

as well as be aware of our concern **CNR-3** (Javascript competency).

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

Since this is a greenfield system iteration 1, we have nothing to refine, so we must refine the entire CMS system. We will refine through decomposition.



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

<table>
<tr><th>Design Decision</th><th>Rationale</th></tr>
<tr><td>Logically structure the client part of the system using the Web Applications reference architecture</td><td>The Web Applications reference architecture suits our system’s aim for a lightweight browser-accessed application which supports deployment and updating to reinforce availability (QA-2). The minimized client-side intensive processing helps increases accessibility (QA-4), with little sacrifice due to unneeded isolated storage or feature-rich user interfaces in the application.

<table>
<tr><th>Discarded Alternatives</th><th>Reasoning</th></tr>
<tr><td>Rich Client Application</td><td>RCA reference architectures supports the development of applications that reside on the client’s computer. They boast robust User Interface capabilities, but do not run in a web browser (CON-2). Our CMS client-side should not sacrifice the availability of a lightweight and portable app for simple interface and feature improvements when they are not necessary.</td></tr>
<tr><td>Rich Internet Application</td><td>RIA reference architectures present a more robust and feature-heavy alternative to other web-browser embedded applications(CON-2). Like RCA though, much more load is put on the client-side with additional features such as plugin support, isolated storage, and client-side business logic. These features are not our top priority.</td></tr>
</table>
</td></tr>
<tr><td>Logically structure the server part of the system using the Service Applications reference architecture</td><td>The Service Applications reference architecture suits our server-side needs as it supplements a non-existent interface with services used by other applications. Due to the tight requirements on availability (minimized downtime), the separation of concerns exhibited by this architecture allows maintenance on server-side systems with minimized impact on the user-end’s functionality.</td></tr>
<tr><td>Physically Structure the system using the Distributed Deployment Pattern</td><td>In the Distributed Deployment Pattern, the three components (Web, Application, Database) reside on separate physical tiers. This pattern facilitates scalability to increased load (CON-1) as well as increased security checkpoints between the mentioned tiers (QA-5). This separation of the components also synergizes well with our selection of Service Applications reference architecture, as</td></tr>
</table>

	well as allows us to configure each tier to meet the requirements of the hosted component, fine-tuning performance as load changes (QA-6).
Build the user interface using ejs-complimented html, and css.	Due to our development teams familiarity with Javascript and web-based technologies (CRN-3), a familiar web-stack should be implemented. These W3 compliant languages are also the standard among web applications, which is the only feasible option to circumvent compatibility issues(QA-6).
Web-Deployed Application	Once launched, the application becomes immediately available online, hosted on a server. This ensures no downloads are required for installation or updates, and the user is always seconds away from accessing their account and course information (QA-2).

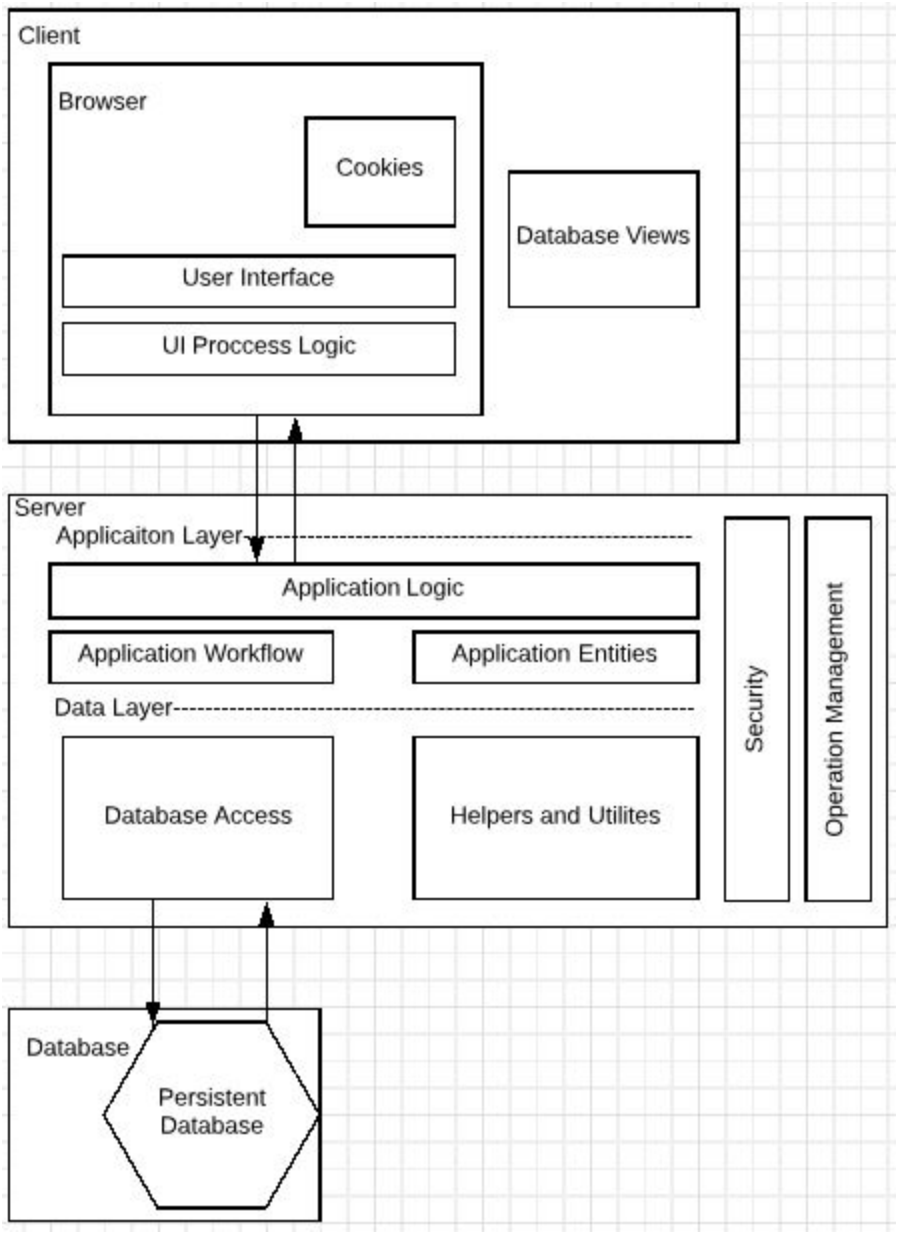
Step 5: Instantiate Architectural Elements, Allocate Responsibilities, and Define Interfaces

Decision and Location	**Rationale**
Cache most recent database views online and enable cookies to save page presets and details	This application is to be used frequently and massively, so we want to be able to support this, by minimizing the effect of downtime (QA-2, 3,4). This will be done by storing small amounts of data on the client-side, such as the most recently available database views (grades, courses, coursework). Persistent cookies allow for useful app-use statistics and allows faster persistent user-customized UI changes.

Due to this being the first iteration, many physical elements and tools have not been decided on.

Step 6: Sketch Views and Record Design Decisions

In adherence to the reference architectures selected in step 5:



Element	Responsibility
Cookies	Save page presets and user details for statistical analysis.
Database Views	Small-scale data persistence off-server. Privacy maintained by only holding pre-opened and cached information.

User Interface	HTML and CSS designed UI
UI Process Logic	Simple logic and querying
Application Logic	Query handling, user accounts management, course management, generates views
App Workflow	Works with Application Logic
App Entities	Database views, objects, users, etc.
Database Access	Required retrieval and Storage interface for persistent database.
Helpers & Utilities	Maintenance work tools for updating, debugging, error output, etc.
Security	Cross-cutting security layer checking access permissions and privileges, Firewall
Operation Management	Administrator and Maintenance-level user tools and features
Persistent Database	Database holding information

Step 7: Perform Analysis of Current Design and Review Iteration

Not Addressed	Partially Addressed	Addressed	
	UC-1		Account Management features are facilitated by our system structure, but lack specifically stated implementation.
	UC-2		Course Management features are facilitated by our system structure, but lack specifically stated implementation.
UC-5			No relevant decisions made
		QA-2	Availability was a central focus for this iteration, with **web applications** reference architecture being leveraged to better exhibit and application with high availability.
	QA-5		While the **distributed deployment pattern** allows security at every tier of the system, but

			specific implementation has not yet been conceptualized.
		QA-6	Extensibility and interoperability have been primarily focused on for this iteration, and the **distributed deployment pattern** have been picked specifically to allow better future modification and upgrading of the system.
	CON-1		The system planning has considered and made decisions in mind of a wide user base with continual requests, but CON-1 has been left as a secondary objective in this iteration.
		CON-2	Due to our high focus on availability in this iteration, browser compatibility was at the forefront of our design plans using the **web applications** reference architecture.
CON-3			No relevant decisions made
		CNR-1	Established the system's architectures and deployment pattern in this iteration.
CNR-2			No relevant decisions made
		CNR-3	EJS (Embedded Javascript)-complimented HTML and CSS were chosen for our front-end presentation, both of which are highly compatible with our team's familiarity with Javascript.