Faculty of Engineering and Applied Science
SOFE 3650U Software Design & Architecture
Course Project

Sunny Patel	100620076
Karan Jariwala	100619029

Step 1 - Review Inputs

<table>
<tr><td>Category</td><td>Details</td></tr>
<tr><td>Design Purpose</td><td>This is a greenfield system from a mature domain. The purpose is to produce a sufficiently detailed design to support construction of the system.</td></tr>
<tr><td>Primary functional requirements</td><td>From the use cases in Deliverable 1, the primary ones are as follows:
UC1: Because it supports the core business and concerns multiple stakeholders
UC2: Because it supports the core business.
UC3: Because it supports the core business.
UC6: Because it supports the core business.
UC9: Because it supports the core business.
UC11: Because it supports the core business
UC12: Because it supports the core business
UC17: Because of technical issues associated with the CMS.</td></tr>
<tr><td>Quality Attribute Scenarios</td><td>The scenarios were mentioned in the first deliverable, and will be prioritized as below:

<table>
<tr><td>Scenario ID</td><td>Importance to Customer</td><td>Difficulty of Implementation</td></tr>
<tr><td>QA 1</td><td>High</td><td>High</td></tr>
<tr><td>QA 2</td><td>High</td><td>Low</td></tr>
<tr><td>QA 3</td><td>High</td><td>Medium</td></tr>
<tr><td>QA 4</td><td>High</td><td>Low</td></tr>
<tr><td>QA 5</td><td>Low</td><td>Low</td></tr>
<tr><td>QA 6</td><td>High</td><td>High</td></tr>
<tr><td>QA 7</td><td>Medium</td><td>High</td></tr>
<tr><td>QA 8</td><td>Medium</td><td>High</td></tr>
</table>

Based on the above analysis, Quality Attributes QA1, QA2, QA3,QA5 and QA6 are selected as drivers</td></tr>
<tr><td>Constraints</td><td>All the constraints which are shown in Deliverable 1 are included as drivers.</td></tr>
<tr><td>Architectural Concerns</td><td>All of the architectural concerns as shown in Deliverable 1 are included as drivers. They form the basis for the iteration goals.</td></tr>
</table>

Iteration 1

Step 2: Establish a goal
The goal of this iteration is to achieve architectural concern CNR-1 of establishing overall system architecture.

Drivers:
- QA-1: Security
- QA-2: Usability
- QA-4: Availability
- QA-6: Security
- QA-7: Scalability
- QA-8: Maintainability
- CON-1: CMS must be executed via a web browser
- CON-5: Relational database server must be used

Step 3: Choose 1 or more elements of the system to refine
We will be refining the backend of the CMS system.

Step 4: Choose one or more design concepts that satisfy the selected drivers

<table>
<tr><th>Design Decisions</th><th>Rationale</th></tr>
<tr><td>Three-tier-deployment pattern</td><td>Since the users are required to use a web browser to use the system CON-1 and database is required to store and update data CON-5, three-tier-deployment is suitable.

Discarded Alternatives
<table>
<tr><th>Alternative</th><th>Reason for Discarding</th></tr>
<tr><td>Layered Pattern</td><td>A layered architectural pattern is best suitable for separating components of a system and each layer provides support to next layer. This design fails to address any of our constraints.</td></tr>
</table></td></tr>
<tr><td>The system tier (server-side) will communicate with the database tier using</td><td>PostgreSQL offers an easy backup solution, as well as many security features including concurrent user encryption and sanitized inputs which can prevent unauthorized access. This</td></tr>
</table>

<table>
<tr><td>PostgreSQL</td><td>design decision helps satisfy QA1, QA-8, CON4, CON-2, CON-3, and CON-5.</td></tr>
<tr><td>The system layer (server-side) will communicate with the client tier via a REST API reference design</td><td>This offers extensibility, security and can easily transmit authentication tokens. Additionally, this helps satisfy CON-1 by enabling the client tier to access data, while not forcing the use of a Standard Web Application (wherein only static pages are generated, like with PHP).

Discarded Alternatives:

<table><tr><td>Alternative</td><td>Rationale for Discarding</td></tr><tr><td>PHP</td><td>The use of PHP would force us to adopt the Web Application Reference Architecture for the front-end, which would mean we cannot satisfy QA-2 and QA-3.</td></tr></table></td></tr>
<tr><td>The system layer will be written using Node.js</td><td>This offers easy extensibility, good integration with JSON-based REST and CRUD API’s (a result of a previous decision process), and the asynchronicity helps satisfy the need for many concurrent users (CON-3). Lastly, because node runs on Google’s V12 JavaScript engine, access to system clocks are found within the server, removing the need for a clock component.

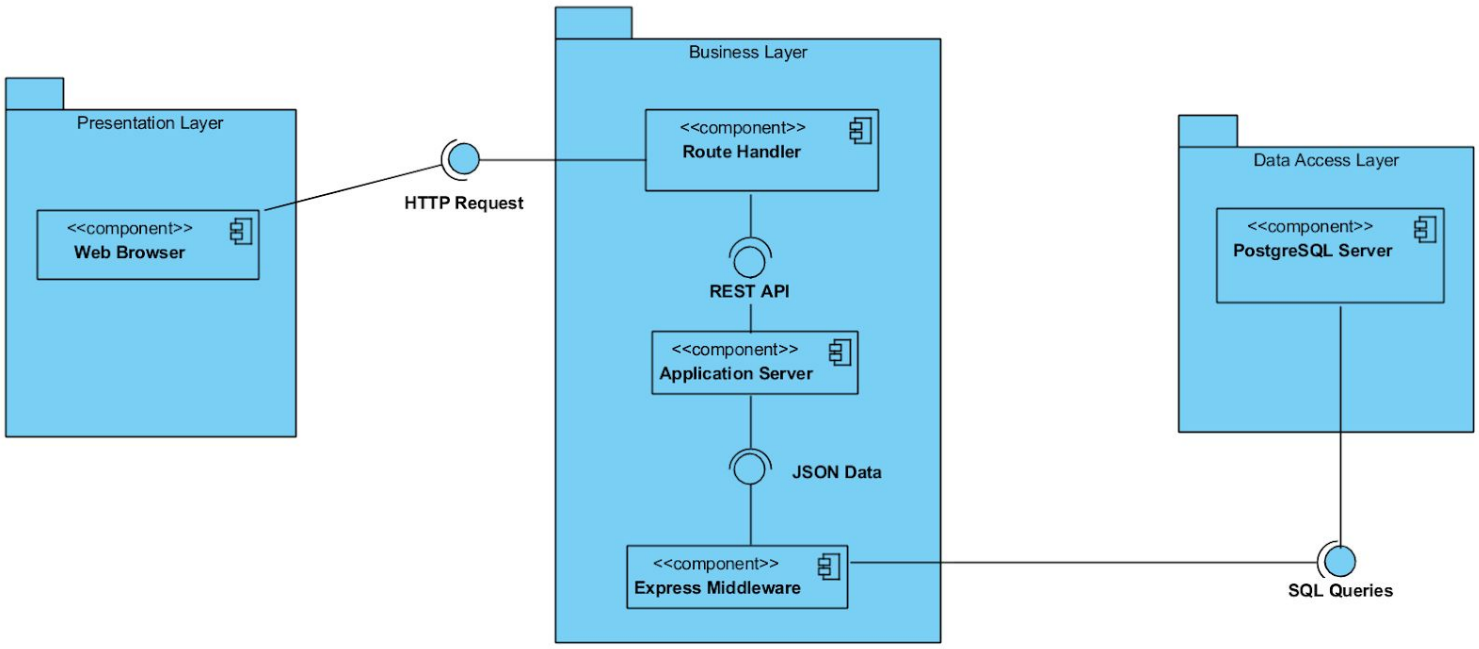
Discarded Alternatives:

<table><tr><td>Alternative</td><td>Rationale for Discarding</td></tr><tr><td>.Net</td><td>.Net was discarded due to its inability to natively handle JSON objects (json handlers are a component and therefore require packaged libraries to handle such objects.</td></tr><tr><td>Java</td><td>Java offers useful features, but implementing non-blocking IO (supporting multiple users) is more expensive.</td></tr></table></td></tr>
</table>

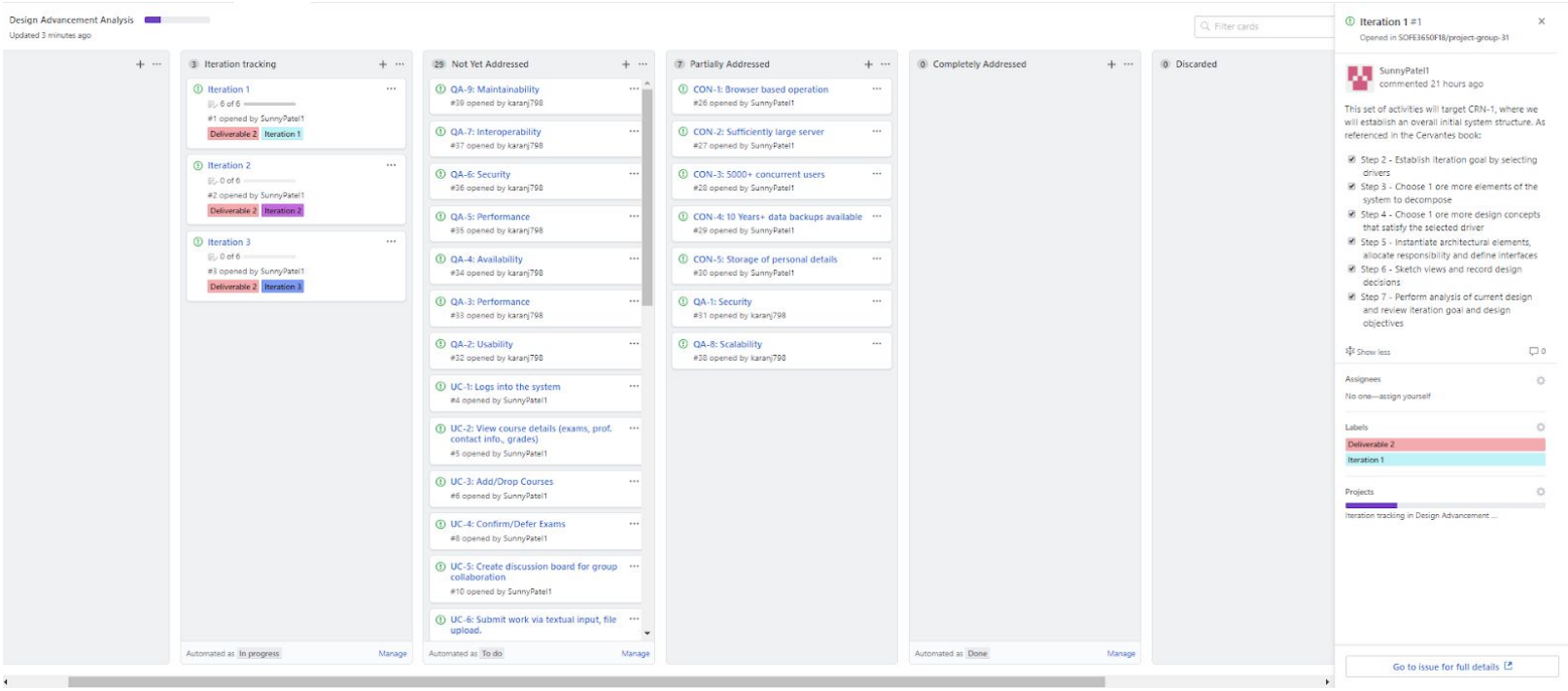
Step 5: Instantiate architectural elements, allocate responsibilities and define interfaces

Design Decision (Element + Responsibility)	Rationale
The system tier (server-layer) will utilize Express.js middleware for handling asynchronous I/O, and to create a REST API interface.	Express middleware offers various functionalities which are required to create a connection to PostgreSQL database CON-5.
The system tier will use promise/await structure instead of callbacks.	The usage of asynchronous function allows us to support over 5000 simultaneous connection and thus addressing CON-3.
The system tier will utilize Node-Postgres as the handler for PostgreSQL database connection.	The handler is used to create a connection from the server to the Postgres database server CON-5.

Step 6: Sketch views and record design decisions



Step 7: perform analysis of current design and review iteration goal and design objectives



Iteration 2

Step 2 - Establish a goal

The goal of this iteration is to achieve architectural concern CNR-2. This will involve designing the front-end of the system. Related artifacts are chosen as the drivers,even though all Quality Attributes, Use Cases and Concerns should be taken into consideration. Namely, the following are going to be focused on:

- QA-2 (Usability)
- QA-3 (Performance)
- QA-6 (Security)
- QA-7 (Interoperability)
- CRN-1
- CON-1
- CON-2

Step 3: Choose 1 or more elements of the system to decompose

This is a specific development effort for the client-side module of the deployable application. As such, the element to refine is the Client-Side Application (or the front-end). Refinement is performed through decomposition.

Step 4: Choose one or more design concepts that satisfy the selected drivers

<table>
<tr><td>Design Decisions and Location</td><td>Rationale</td></tr>
<tr><td>Logically Structure Client part of system using the Rich Internet Applications Reference Architecture</td><td>The Rich Internet Application (RIA) supports the development of dynamic or static client-side web pages that can be rendered in a browser. As such, constraint CON-1 is satisfied. Additionally, through the use of web sessions QA-6 is satisfied. QA-2 and QA-3 are satisfied through the use of dynamic menus. Lastly, QA-6 can be satisfied, as javascript can be used to generate and export google/apple calendar importable files.

Discarded Alternatives:

<table><tr><td>Alternative</td><td>Reason for Discarding</td></tr><tr><td>Rich Client Applications</td><td>Rich Client Applications (RCAs) are similar to RIAs, but they fail to satisfy CON-1, as they are run as OS platform applications</td></tr><tr><td>Web Applications</td><td>Rendering things server side fail to meet QA-2 and QA-3 requirements, as static web pages may require multiple navigations to get new data.</td></tr><tr><td>Mobile Applications</td><td>A mobile application would fail to meet QA-2 and QA-3, as the size constraints would ultimately result in hidden menus and modals. Additionally, mobile and handheld devices were not considered for accessing the system (as most project submissions and long-tasks require more powerful devices, anyway)</td></tr></table></td></tr>
<tr><td>Structure the logic of the client-side application to use an MVC pattern</td><td>For an RIA, implementing the MVC framework would generally involve using DOM manipulation and event listeners as controllers, a javascript program as a model (which uses AJAX and HTTP requests), and the view would be the rendered html document. This further satisfies QA-2, as it would reduce the requirement to navigate to different pages to access data, as data can be dynamically loaded into view from model once attained asynchronously via AJAX requests</td></tr>
</table>

<table>
<tr>
<td>Build the user-interface of the client application using Angular</td>
<td>One of two standard framework for front-end rich browser based applications, angular enables the use of Token Authentication (to satisfy QA-6, i.e. Security)

Alternatives were considered, and were discarded for the below reasons
<table>
<tr><th>Alternative</th><th>Reason for Discarding</th></tr>
<tr><td>React</td><td>React does not have the application structure enforcement that Angular does, nor does it comply to the level of security that Angular provides. It fails QA-6, and therefore is discarded.</td></tr>
<tr><td>Plain JavaScript</td><td>Security gaps would be far more difficult to analyze for an entirely custom application, while little benefit would be provided. Discarded for high development costs.</td></tr>
</table>
</td>
</tr>
</table>

Step 5: Instantiate architectural elements, allocate responsibilities and define interfaces

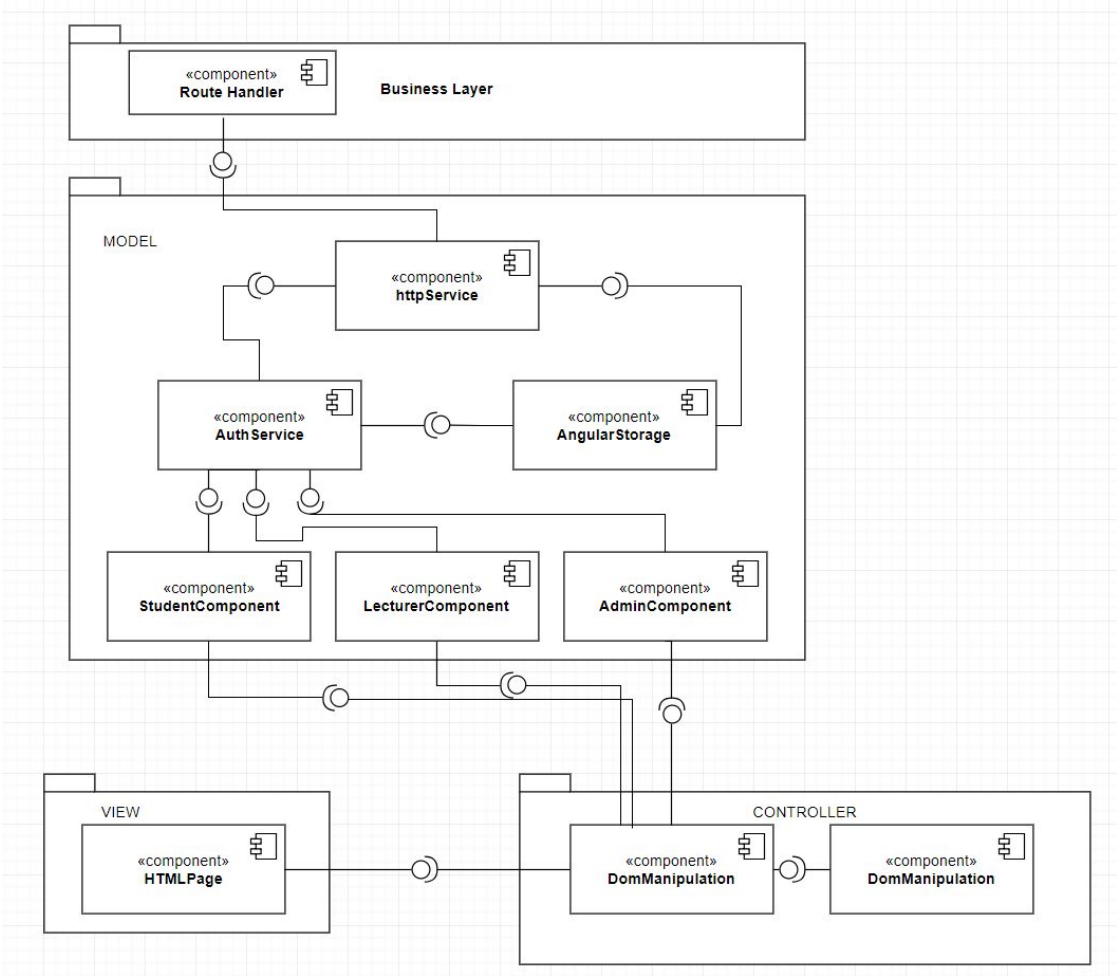
*The services referenced below are actually Angular services. In architectural terms, they resemble components, and not services.

Design Decision	Rationale
A single Javascript-based HTTP Service* is used for all communication to the server. Additionally, a single Service* is also used to hold all of the relevant data.	Since the data is refreshed all together, and all data is 2-way bound between the view and the model, having intermediary services (which are components). This helps to maintain security and performance, helping satisfy QA-2, QA-3, and QA-6.
No data is stored locally, except for session variables and authorization tokens. This is handled by the **AngularStorage** module	Because the web application is relatively quick, and for security purposes, data that requested and received via the HTTP service is never stored on the user's machine. However, security tokens for logging in are. This helps to satisfy QA-6 and QA-1
A single page is rendered, and what can be accessed is additionally restricted by a *Service module that handles authorization and encryption called the **authService** component	This will help ensure that QA-1 is satisfied, by restricting access to pages that the user does not have access to. Even if the server-side application does not serve any information to unauthorized users, the structure of the

	frontend may still provide hints towards possible security flaws.
The DomManipulation module interfaces with the independent modules for each user to change the browser render.	This will ensure that QA-2 and QA-3 are realized. Alternative is to hardcode these behaviors, but they are less secure and are far more expensive to implement.
The Student, Lecturer and Administration modules are implemented with views and logic manipulators for each user respectively. They are responsible interfacing with the DomManipulation module that renders the html, and will implement some basic business logic.	This addresses all of the use cases with the exception of UC17 and UC18, which are largely implemented in the previous iteration.

Step 6: Sketch views and record design decisions

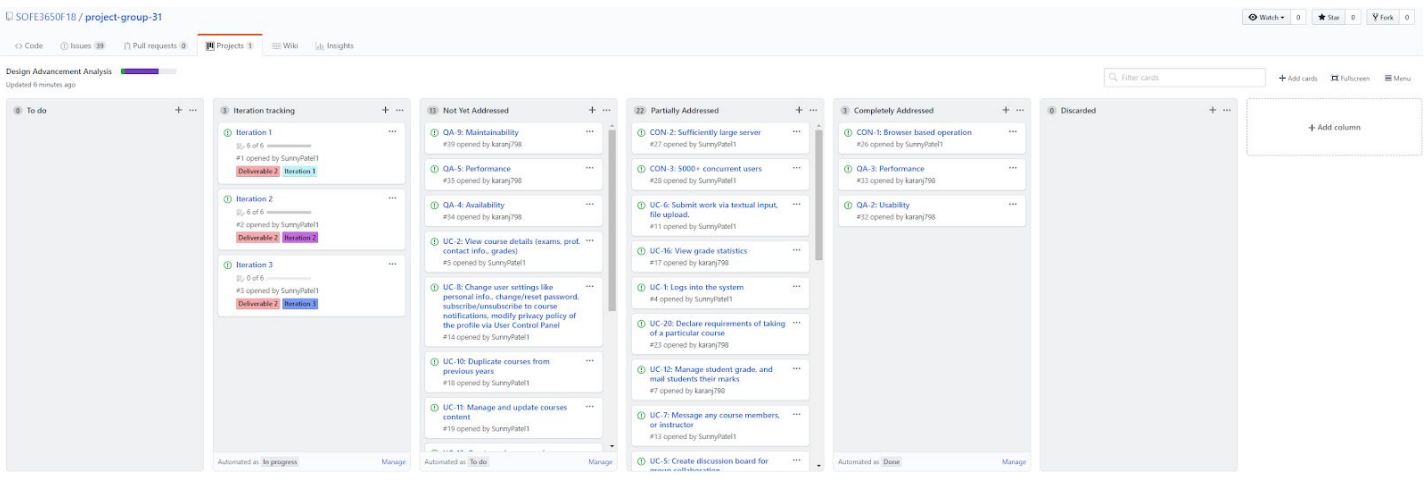
Elements within the front-end package are decided upon below. While they conform to Angular 6 component or service definitions, components that are packaged as part of angular are also



included if they are relevant to the selected drivers

Step 7: perform analysis of current design and review iteration goal and design objectives

An analysis is shown below in the form of the Kanban Board. In addition to the design decisions solidified through iteration 1, the design decisions that



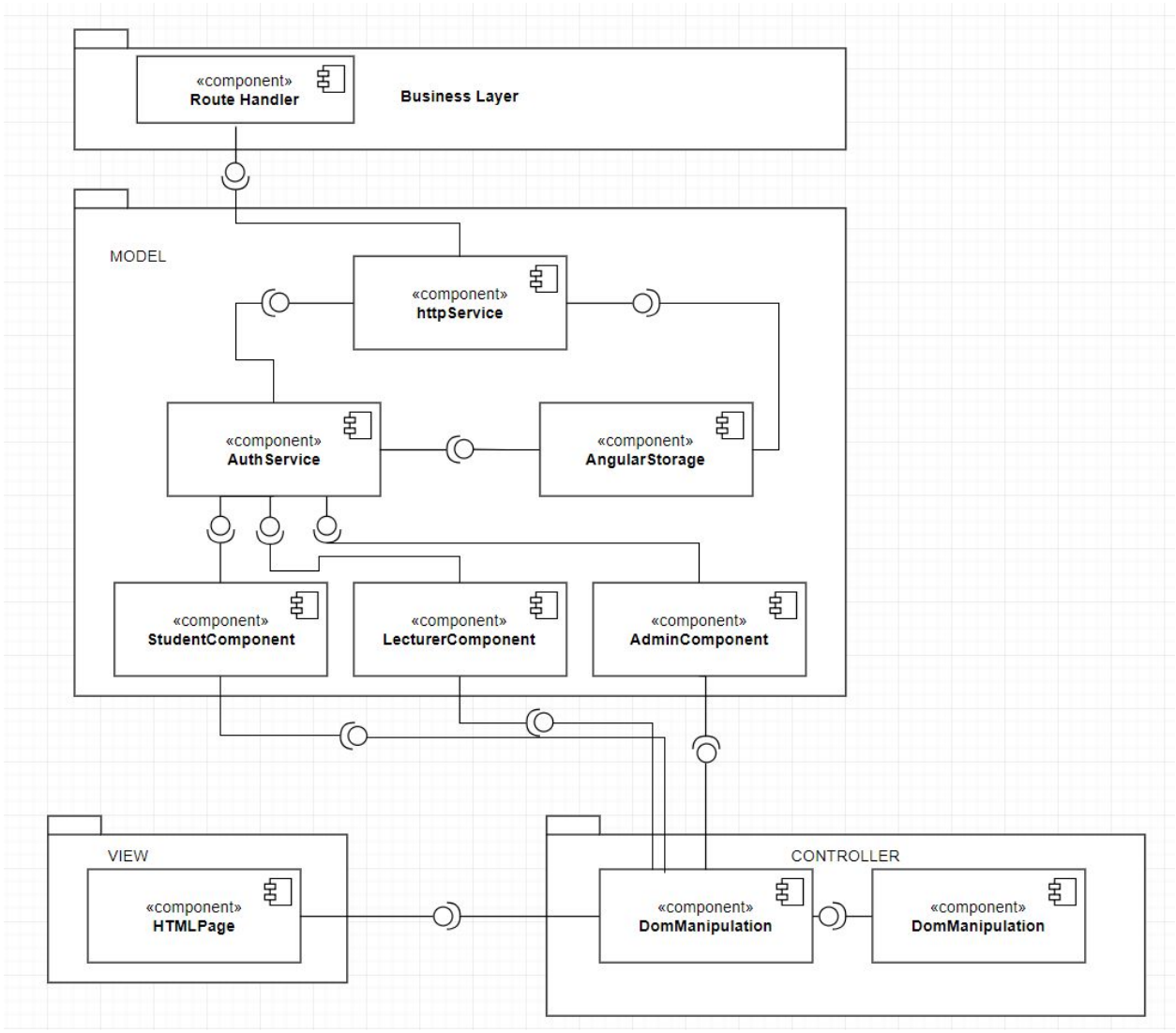
Iteration 3

Step 2: Establish a goal

The goal of this iteration to realize architectural concern CNR-3, of establishing a secure database logic and implementing security features.

Step 3: Choose 1 or more elements of the system to refine

We will be refining the database tier and the security components of the CMS System (back-end).



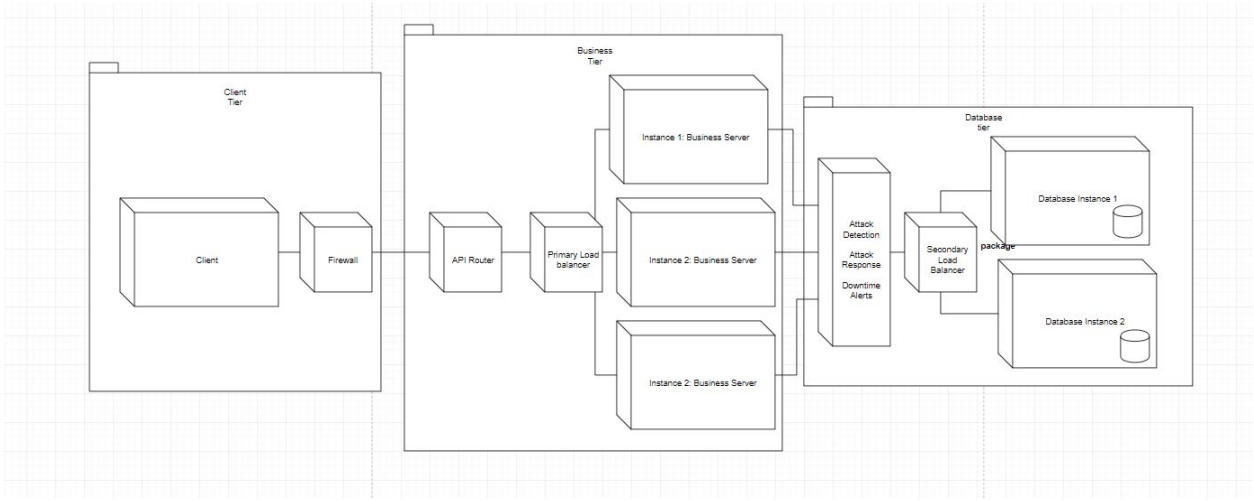
Step 4: Choose one or more design concepts that satisfy the selected drivers

Design Decision	**Rationale**
To prevent direct access to the database, we will be using a **3-tier architecture**, thereby separating the user from the database with an API layer that supports JWT Authentication	In previous iterations, this decision supported usability. It also enhances security by making it easy to prevent users from using malicious queries to directly address the database, thereby addressing QA-1, QA-6 and QA-5. This is because the login system is able to ensure only authorized persons have access to certain parts of the web application. This also sufficiently completes CON-5. This also provides the benefits of a **Distributed Deployment System** (Cervantes)
The **Database Tier** of the application will use **PostgreSQL** and users, backup tables and simultaneous access will be enabled	By enabling backup, we can address UC17, QA-9, and CON-4. Additionally, by enabling simultaneous access, we can allow more than 5000 users to be supported at once, thereby CON-3 is addressed. Additionally, because a relational-data model was required, this also addresses CON-5.
A distributed **Load-balanced cluster** is utilized to access the database simultaneously, and database clusters are used in RAID 10 to ensure quick, secure data and removes many data limitations	This design decision fully addressed CON-2 and CON-3, as it can now sufficiently handle large amounts of users. Additionally, large amounts of data can now be stored, as storage space is not sacrificed for redundancy in a design involving a distributed array of storage solutions. This decision also fully enables QA-8, as new clusters can be added to ensure more data. Lastly, QA-9 is also fully addressed here, as data backups can be taken.
Security tactics (Cervantes) are implemented to detect Denial of service attacks and intrusion, as well as identify actors.	This design decision fully addresses QA-4, by preventing downtime from attacks.

Step 5: Instantiate architectural elements, allocate responsibilities and define interfaces

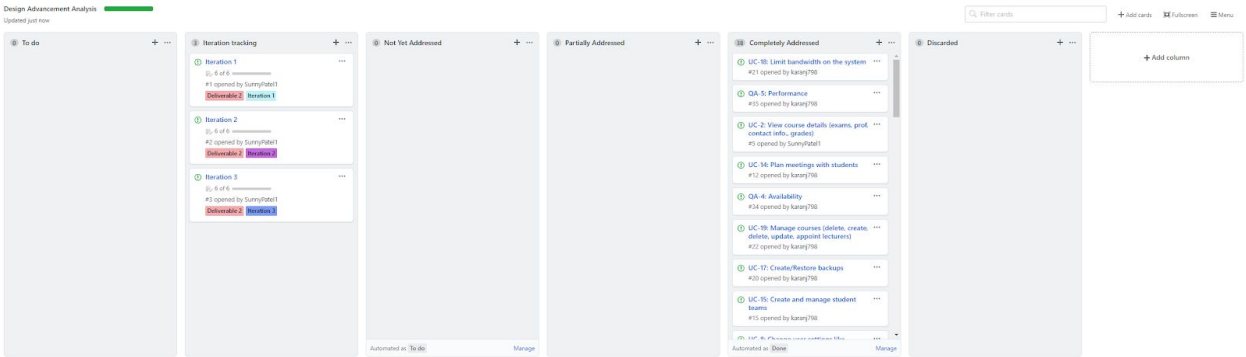
Design Decision and responsibility	**Rationale**
The database will employ multiple tables, views and backup tables to hold, backup and relate data	This decision realizes CON-5, by correctly implementing object-relational logic, as well as QA-9 and CON-4 by enabling data backup for tables
The database tier will have a **attackDetection** security module, which will monitor patterns in API use to detect malicious activity	This architectural element will interface with the front-end and serve as an enabler for the API tier. This completes QA-6, QA-8.
The database tier will have a **downtimeAlert** module, which will alert clients when outages will occur.	This architectural element will only interface with the client tier via business/server tier and accomplishes QA-4 and QA-5, ensuring there is no unexpected downtime.
The database tier will have an **attackResponse** security module, which will respond to attacks by removing access to the database.	This architectural element will interface with the attackDetection module as well as the business/server tier. This fully realizes QA-1 and QA-8.
A **loadBalancer** will balance the load for the multiple instances of the databases.	Realises QA-8 and CON-2, by enabling large amounts of data to be implemented by increasing the amount of physical hardware used.

Step 6: Sketch views and record design decisions



Step 7: Perform analysis of current design and review iteration goal and design objectives

The use cases were updated as they relate to the Quality attributes. At iteration 3, all of the use cases, Constraints, and Quality Attributes had been addressed.



Contributions

For Sunny

- Team's Commitment: 5.0, commitment was strong
- Team's Communication: 5.0, communication was strong
- Team's Knowledge, Skills and Abilities to perform the project: 5.0, used great deal of his skill and knowledge
- Team's Ethical Standards: 5.0, did no wrongdoings
- Teams Focus: 5.0, always was on focus

For Karan J

- Team's Commitment: 5.0, commitment was great
- Team's Communication: 5.0, communicated well through the use of Slack
- Team's Knowledge, Skills and Abilities to perform the project: 5.0, used an apt amount of knowledge as would be expected from a course enrollee
- Team's Ethical Standards: 5.0, Pulled own weight without prompts
- Teams Focus: 5.0, Focused well