

# Central Connecticut State University

**University Student Portal using the *Semantic Web Approach***

CS-595 Capstone Final Report

**Team Members:**

*Nagaharika Ogirala*

*Swetaa Ganesan*

*Sherin Cheriyan*

**Advisor:**

*Dr. Neli Zlatareva*

**Spring 2021**

**Table of contents:**

**Acknowledgement**

**List of Tables and Figures**

**Abstract**

1. **Introduction**
   1. **Domain**

**1.1.1** Existing System

**1.1.2** Proposed System

1. **Knowledge Gathering**
   1. **Domain – Department of CS**

**2.1.1** Courses Datasets

**2.1.2** Faculty Datasets

* 1. **Skillset**

**2.2.1** Semantic Web

**2.2.2** Spring Boot Micro-services

**2.2.3** Angular

1. **Design and Analysis**
   1. **Technical Stack Decisions**

**3.1.1** Semantic Web Technology

**3.1.2** Java Technologies

* 1. **Semantic Web Schema Models**

**3.2.1** Course Schema

**3.2.2** Faculty Schema

**3.2.3** Integrated Schema

* 1. **Course Op Service**

**3.3.1** Jena API Service

**3.3.2** Front-end JSP Design

* 1. **Login and Registration OP Services**

**3.4.1** Backend Design

**3.4.2** Front-end Angular Design

**3.4.3** Additional Services

* 1. **It all fits (Integration)**

**3.5.1** Role of Microservices

**3.5.2** Microservice Integration

**3.5.3** Wiring with Spring Boot

1. **Planning and Development Strategies**
   1. **Agile Development**

**4.1.1** Use Cases Diagram

**4.1.2** User Stories

**4.1.3** Sprint Planning and Implementation

* 1. **Version Control**

**4.2.1** GitHub

**4.1.2** Multiple Repositories

1. **Implementation**
   1. **Control Flow**

**5.1.1** Application Flow

**5.1.2** Sequence Models

* 1. **Services and Endpoints Used**

**5.2.1** SPARQL Endpoint

1. **Application**
   1. **User Interface**

**6.1.1** Course Op Pages

**6.1.2** Login Page

**6.1.3** Registration Page

1. **Possible Extensions**
2. **Conclusion**
3. **References**

**Acknowledgements**

We are grateful to Dr. Neli Zlatareva for providing our team with guidance and support.

# List of Tables and Figures

**Figure 3.2.1a** Asserted Class Hierarchy of Course Schema 11

**Figure 3.2.1b** Inferred Class Hierarchy of Course Schema 11

**Figure 3.2.1c** Object Properties of Course Schema 12

**Figure 3.2.1d** Data Properties of Course Schema 12

**Figure 3.2.1e** Individuals and Properties of Course Schema 13

**Figure 3.2.2a** Asserted Class Hierarchy of Faculty Schema 13

**Figure 3.2.2b** Inferred Class Hierarchy of Faculty Schema 14

**Figure 3.2.3a** Usage of Pellet Reasoner in Integrated Schema 15

**Figure 3.4.1a** Login and Registration Services Application Architecture 15

**Abstract**

A Student Information Management System (SMS) is a way for schools to provide a simple interface to maintain records of students and administration. Many student information management systems manage all types of data from student grades, information about university courses to faculty details. In these systems, a university student portal serves as a way for students to easily access their data. The information is easily accessed by those that are authenticated access. A university student portal allows students to manage their administrative tasks for school all in one place in a simple and effective way. The system maintains up-to-date information about the student, for example the courses they are taking throughout their academic years at the university.

Our team has designed a university student portal, where a student accesses a login page by providing their username and password to access information related to computer science courses during each academic year: Fall, summer and spring. There is one backend service which includes information about the courses by the computer science department, such as timings, name, semester, type, prerequisites, and faculty teaching during each academic semester. There are also details about the faculty of the computer science department. On the other hand, the other backend service provides details about the courses that the student has registered and already completed. The user interface allows the student to login and register for courses after accessing the information of the models, controllers and services provided by the backend services.

This university student portal allows students to login, access course and faculty details and register for courses. Services maintain confidentiality and security of student information, as well. Angular, Spring Boot, MySQL and Protégé were four main tools used in the creation of the university student portal.

1. **Introduction**

The university student portal is developed using the Angular User Interface and back-end is developed with Java using Spring Boot. Here the data is created and updated through the Ontology model developed using Protégé. The ontology creates a relationship between various entities and classes, it also describes how these entities can be grouped according to their similarities and differences. In our ontology model, we are creating two different ontologies: Faculty Ontology and Courses Ontology. The Faculty Ontology contains details of the faculty and staff members. The Course Ontology contains the course details, time, days and the semester, in which the courses are provided by the department for both graduate and under-graduate students. These ontologies are then combined, thus providing a relationship between the faculty and the course, where we can then obtain the details of the professors teaching the different courses.

Jena API is used to maintain the schema of the ontology using various Inference and Ontology models. This application is built in Java SE 10 using the Eclipse IDE. The Angular framework is used to build the User Interface for the data available in the Ontology models. Spring Boot works at the middle layer, wiring the Angular User Interface with the ontology models and/or database for student login.

The intent of the web application is to help students log into their University account, as well as register for courses and view their registration status. We are using two possible data-sources, one being Protégé to populate domain information (turtle files) and a database to store the login information and student registration status. Jena API is used to integrate the models in runtime.

**1.1 Domain**

**1.1.1 Existing System**

**1.1.2 Proposed System**

This application is used for course registration for students, where students can login to the portal using their University username and password. This application mainly focuses on the Computer Science Department for both undergraduate and graduate students. The user can register for the courses available for the current semester, as well as browse courses for any other semesters. They can also add and drop courses depending on their registration status. The user will be displayed with the courses, faculty who teaches the course, semester, and time when the course is offered.

A Microservice is created for integrating the services from the CourseOp model, registration and login services. This framework maintains a database for all the student login and course details, where data can be added and removed based on the changes made by the user. Spring Boot is a Java-based framework used to create microservices. We use the student login and registration data that is maintained in the relational DB, as well as the RDF data maintained in the CourseOp services.

Programs: type of programs like Honors, Graduate etc.

Courses: Courses available in each program.

Faculty: Browse the Faulty from Department.

Features of Proposed System:

* Login/Logout Feature for student
* Show list of Registered and completed courses.
* Browse Programs, Courses and Faculty from the Department.
* Register for new courses if pre-requisites are met.
* Email the advisor if the pre-requisites not satisfied but still wants to enroll to a course.
* Forgot password to reset the old password through email.

**2. Knowledge Gathering**

**2.1 Domain – Department of CS**

This system is designed for the courses and the faculty who teaches the courses. This helps the students to register for courses according to their requirements in that particular semester. In this application we are providing the registration service only for the students in the Computer Science department. These datasets are represented in a RDF triple format using the Protégé editor.

**2.1.1 Course Datasets**

The Course data is collected from the CCSU websites for all the courses under the CS department. This data is collected based on the different semesters i.e., Spring/Summer/Fall. It contains details like when the course is provided, timings/days for different courses, semester when the course is offered, its description and its prerequisites.

**2.1.2 Faculty Datasets**

The faculty datasets contain the data of different faculty and staff from the CS department. The data is obtained from the official website for CCSU. This contains the name, email, phone number and website for all the faculty and staff. This data is combined with the courses to define who teaches what so that the students can choose courses based on the different faculty available.

**2.2. Skillset**

**2.2.1 Semantic Web**

The Semantic Web is a vision about extending the existing World Wide Web. The data in World Wide Web is unstructured, therefore if this data is made a part of the new Semantic web, it can be referenced directly by its unique identifier, called the Universal Resource Identifier (URI). URIs are interconnected in a graph infrastructure, thus comprising a huge library of information that can be easily and uniformly accessed by Semantic Web applications.

The benefit of using Semantic web over the existing web is that the Semantic web facilitates the machines to understand the information allowing the users to perform task effortlessly. To enable the encoding of semantics with the data, technologies such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL) are used. These technologies are used to formally represent metadata.

**2.2.2 Spring Boot Micro-services**

Microservices are increasingly used to create larger, more complex applications that are better developed and managed as a combination of smaller services that work cohesively together for larger, application-wide functionality. Spring Boot is a Java-based framework, which builds a standalone environment which communicates with many microservices available in our application.

The benefits of using Microservice architecture is it would allow the developers to scale up or down the size of the project independently. This feature is also useful in decreasing the expenses for the overall application, as well as the failure of one module would not affect the whole operation of the application.

**2.2.3 Angular Framework**

Angular is a component-based framework for building scalable web applications. The basic building blocks of the Angular framework are Angular components that are organized into NgModules. NgModules collect related code into functional sets and an Angular application is defined by a set of NgModules. An application always has at least a root module that enables bootstrapping and typically has many more feature modules.

Angular applications are built using the Typescript language, a superscript for JavaScript, which ensures higher security as it supports types (primitives, interfaces, etc.). It helps catch and eliminate errors early on when writing the code or performing maintenance tasks. With Angular, we do not need any additional getter and setter functions. Since, every object it uses is a Plain Old Java Object (POJO), it enables object manipulation by providing all the conventional JavaScript functionalities.

**3. Design and Analysis**

**3.1 Technical Stack Design**

In this section, we will discuss the different technologies used in this application. They are mainly Semantic Web Technologies and Java Technologies along with Angular UI for Front End services.

**3.1.1 Semantic web Technologies:**

The data in the World Wide Web is unstructured and if this data is made a part of the new Semantic Web, it can be referenced directly by its unique identifier, called the Universal Resource Identifier (URI). URIs are interconnected in a graph infrastructure, thus comprising a huge library of information that can be easily and uniformly accessed by Semantic Web applications.

**Turtle:**

A Turtle file allows writing down an RDF graph in a compact textual form. An RDF graph represents information using semantic triples consisting of a subject, predicate and object. Each item in the triple is expressed as a Web URL. Turtle provides a way to group three URIs to make a triple and allows ways to abbreviate such information. Subjects are referenced by a number of predicates.

**Protégé:**

Protégé is an editor and a knowledge-based framework, which helps us to create ontology models. The Protégé version we are using is Protégé 5.5.0. It is an ontology editing environment with full support for the OWL 2 web ontology language and direct in-memory connection to the description logic reasoner like Pellet.

**Apache Jena:**

Apache Jena API is a Semantic web framework for Java. It provides extensive java libraries for helping developers to develop code that handles RDF, RDFa, OWL and SPARQL. The API version used here is API 3.15.0. The graphs are represented as an abstract model. Jena includes a rule-based inference engine to perform reasoning based on OWL and RDFS ontologies and a variety of storage strategies to store RDF triples in memory or on a disk. These models are queried using SPARQL.

**Openllet:**

Openllet is a plugin to implement the Pellet reasoner in the Jena API. The version of Openllet is 2.6.5 and is currently only supported by Jena-API version of 3.15. For further API support to work with Openllet, Java SE 10 is required. Dependencies required for Jena are:

* + Jena-core: It is a Jena framework for building Semantic Web. It provides a pragmatic environment for RDF, RDFS and OWL.
  + Jena-arq: ARQ is a SPARQL 1.1 query engine for apache Jena.
  + Jena-tdb: It is a storage system for Jena and ARQ.
  + Jena-cmds: It is a command line tool
  + Jena-iri: Provides implementation for IRI and URI specifications.

**3.1.2 Java Technologies:**

Java is an object-oriented programming language. This implies that the execution of a Java program creates objects, modifies them and updates object references. The program behavior during the execution is therefore described by the evolution of its object list. The semantic specification of our subset of Java is composed of about 850 inference rules. Some modules are expressed in a Natural Semantics style. The Java version used here is Java SE 10. This version is only compatible with both semantic and openllet reasoner.

**Apache Maven 3.6.3:**

Maven is a build automation tool used primarily for Java projects. Maven addresses two aspects of building software: how software is built and its dependencies. An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories and required plugins. It comes with pre-defined targets for performing certain well-defined tasks, such as compilation of code and its packaging. The maven version used here is Maven 3.6.3.

**Spring Boot:**

Spring Boot makes it easy to create stand-alone, production-grade Spring based applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries, so we can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration. Spring Boot provides opinionated 'starter' dependencies to automatically simplify the build configuration, as well as configure Spring and 3rd party libraries whenever possible.

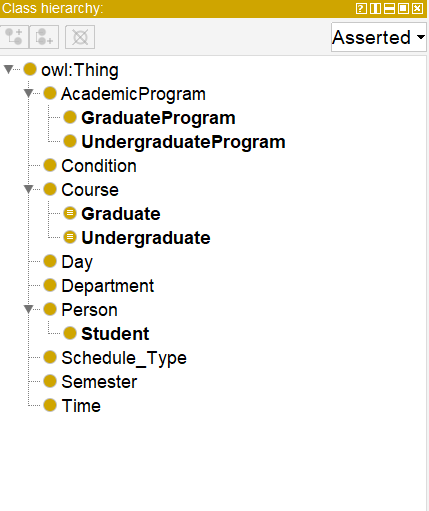
* 1. **Semantic Web Schema Models**

This Ontology model contains two different ontologies: Faculty Ontology and Courses Ontology. The faculty ontology contains details of Faculty and staff members. The Course Ontology contains the course details, time, days and the semester in which the courses are provided by the department for both graduate and undergraduate students. The two ontologies are combined to provide a relationship between the faculty and the course, where we obtain the details of the professors teaching the different courses.

**3.2.1 Course Schema**

**Class Hierarchy: Asserted Class Hierarchy**

The asserted class hierarchy view is one of the primary navigation devices for named OWL classes. In the asserted class, there are various domains like academic program, courses, department, schedule-type, semester and time in the Course Ontology.

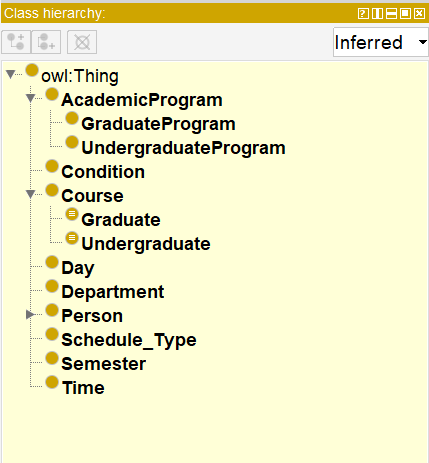


**Figure 3.2.1a**

Asserted Class Hierarchy of Course Schema

**Inferred Class hierarchy:**

The inferred class is obtained after running the reasoner. The inferred hierarchy helps to classify entities depending on the values that are been inferred. The inferred hierarchy can be obtained with the Protégé application using the pellet reasoner. The following image shows how the reasoner is done in Protégé.

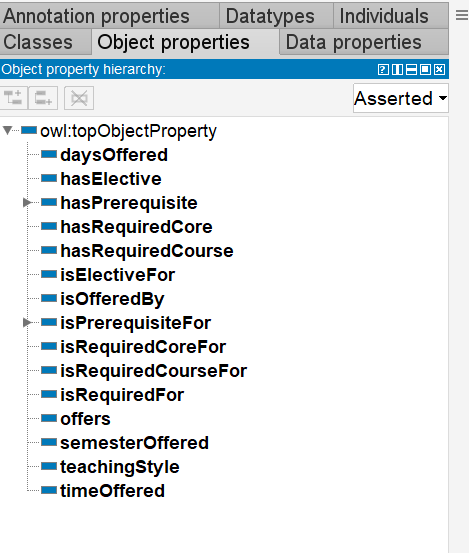


**Figure 3.2.1b**

Inferred Class Hierarchy of Course Schema

**Object Property:**

The object property helps to create relationships between various individuals in a class. The object properties used here are hasElective, which is the inverse of isElectiveof. Similarly, the figure below shows the other object properties.



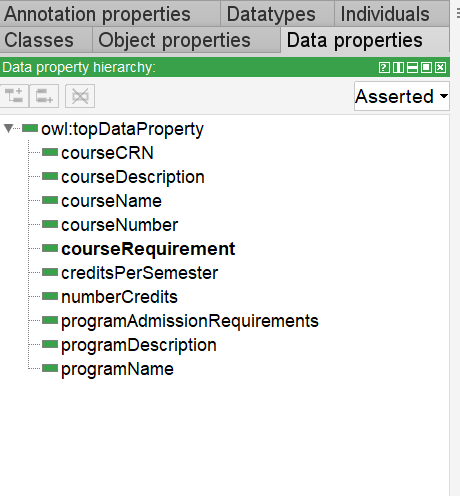
**Figure 3.2.1c**

Object Properties of Course Schema

**Data Property:**

  Data Property describes what kind of values a triple should have. Data property relates individuals to literal data (e.g., strings, numbers, date, times, etc.)

* Few data properties are defined for literal and integer values
* The data property hierarchy view displays the asserted and inferred data property hierarchies

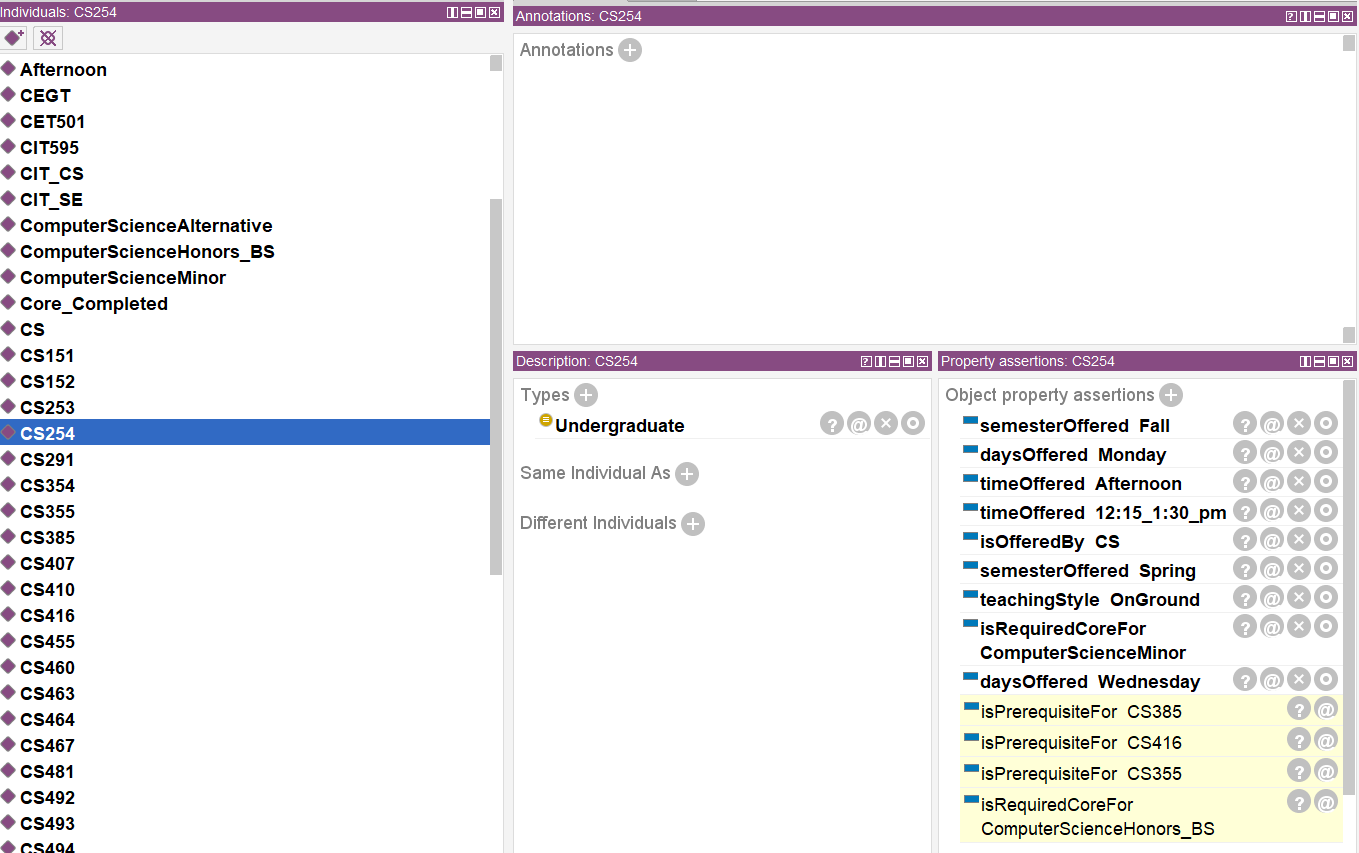


**Figure 3.2.1d**

Data Properties of Course Schema

**Individuals and Properties:**

The named individual helps to create a set of individuals and it also defines an object or data property for each entity. The figure below shows the list of individuals defined in the faculty and course schema.



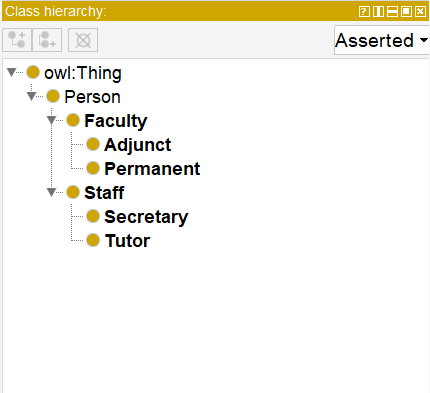
**Figure 3.2.1e**

Individuals and Properties of Course Schema

**3.2.2 Faculty Schema**

**Class Hierarchy: Asserted Class Hierarchy**

The asserted class hierarchy view is one of the primary navigation devices for named OWL classes. In the asserted class, there are various domains like name, email, phone number and website for faculty and staff for the Faculty Ontology.

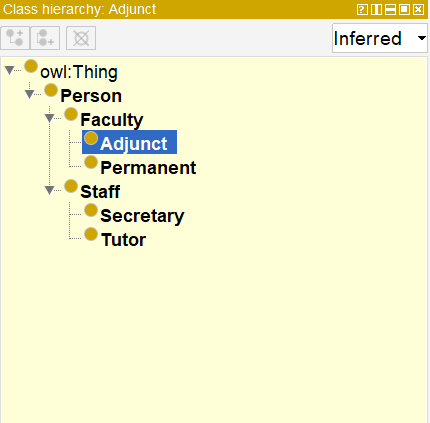


**Figure 3.2.2a**

Asserted Class Hierarchy of Faculty Schema

**Inferred class hierarchy:**

The inferred class is obtained after running the reasoner. The inferred hierarchy helps to classify entities depending on the values that have been inferred. The inferred hierarchy can be obtained with the Protégé application using the pellet reasoner. The following image shows how the reasoner is done in Protégé.



**Figure 3.2.2b**

Inferred Class Hierarchy of Faculty Schema

**3.2.3 Integrated Schema**

The integrated Schema is where both the Faculty and Course models are integrated together to provide a coherent relationship between them. The relationship we generate is to provide the users with the data of which faculty teaches what course. The data is updated and then integrated with a pellet reasoner using Jena.

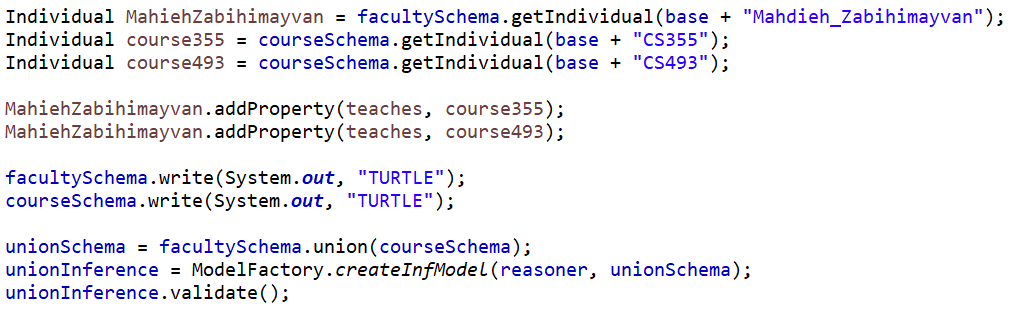
**Usage of Pellet Reasoner in Integrated Schema:**

unionSchema = facultySchema.union(courseSchema);

unionInference = ModelFactory.createInfModel(reasoner, unionSchema);

The pellet reasoner is used to validate and obtain results for the course schema, faculty schema and read the integrity schema. The execution and the output of the reasoner running using the pellet reasoner is then obtained.

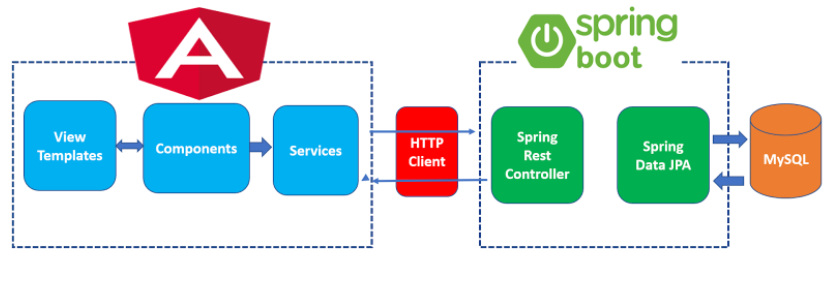
* The two schemas StudentProgarm and Faculty are merged after adding a property teaches in the Jena API and the reasoner runs to validate the new integrated schema
* UnionSchema is now considered to contain the merged data from both of the model schemas
* The Faculty and course ontologies are integrated together where classes, resources, individuals and properties are created
* A relationship can be developed by adding triples to these properties



**Figure 3.2.3a**

Usage of Pellet Reasoner in Integrated Schema

* 1. **Course Op Service**
     1. Jena API Service
     2. Front-end JSP Design
  2. **Login and Registration Services**



**Figure 3.4.1a**

Login and Registration Services Application Architecture

**3.4.1 Backend Design** **\*need to include figures of package structure and code after finalizing code**

The CourseRegistrationOpServices project developed REST APIs for the courses that students registered. The REST API was also created for the courses students completed to verify prerequisites before the student registered for new classes, as well as an API for student login, which is authenticated for access to the student portal according to the username and password entered in the login page. This projected developed these REST APIs in the Java language using Spring Boot, JPA and a connection to MySQL to store the student information. Using Spring initializer, a maven project was initially generated using Java and Spring Boot. All in all, tools that were used for the running of the program was Maven and IntelliJ as an IDE for the Spring Boot API development. To implement these services, the pom.xml file included dependencies for the Spring framework, Spring Boot starter data JPA, Spring Boot starter web, Spring Boot development tools, Spring Boot starter test, MySQL connector, Spring Boot starter mail and Spring Boot maven plugin.

The package structure includes a constants directory with an enum class for the course’s status of registered or completed. Next, model classes were created for courses, login, course details, faculty, course response and student courses. Then, controllers were generated for courses, exceptions, forgot password and login. To implement security, a BcryptGenerator served as a helper to encrypt login password details, which were then stored in a table of the MySQL database. CorsFilter and SecurityConfig classes implemented Spring security as well with the information of login and student registration. Services were included for course registration, login, mail, REST template and user details. For all services a JPA entity, a Spring data repository and a Spring REST controller were created.

Exception handling for RESTful services were also implemented, such as resource not present with a ResourceNotFoundException class. There were also an ErrorDetails class, which customizes the error response with a message, timestamp and details. Then, there was a GlobalExceptionHandler class, which handles exceptions and specific global exceptions in a single place. The CourseOpRegistration.java file is the entry point Java class used with the main method to start the application. The main() method uses Spring Boot’s SpringApplication.run() method to start the application in a localhost address at a specific port number.

**3.4.2 Front-end Angular Design** \***need to include figures of tables in UI after finalizing code**

Front end technologies include Angular with HTML and typescript for languages. Visual Studio Code and Angular CLI were tools used for the angular app development. JQuery and bootstrap was integrated with Angular. The angular app architecture for the user interface is a log in page, where the user can enter their username and password, which will then be authenticated for access to the student portal. A forgot password functionality is provided to allow the user to change their password by submitting their email address for a password reset. When entering the student portal, there is a table of student courses completed and courses registered. With the courseop-iframe component, the student has the ability to access the courseop service, where they can search courses, time, faculty teaching and more on. The student clicks the register button, which through the integrated mail service in the backend, verifies if the student can register for the class requested based on the prerequisites completed. The students can also deregister and see the course details of their registered classes. The database of MySQL will store information about the student’s courses: the courses registered, courses completed and their login information, such as password which is encrypted. The app has a router, which provides the buttons clicked in the reactive forms of the user interface to direct to the typescript files of the requested component, therefore providing the service requested by the user to view in the user interface.

Components of course-list, course-details, corseop-iframe, header, footer, create-courses, login, logout, forgot-password and reset-password were created to design the user interface of the university student portal. Services were created for courses, authentication for login, error interceptor, as well as models for courses, login and forgot password. Services interact with the HTTP client. This allows for the view that the user is requesting. The authentication service checks if the username and password is accurate and then sets it in session storage. The method authenticate() will authenticate the username and password. The method isUserLoggedIn() checks if the session storage of the user name exists. If it does, then it returns true. The logout() method clears the session storage of the username. The login component includes the checkLogin() method, which checks if the credentials are valid by calling the AuthenticationService. The AuthGaurd service activates a particular route only if the user is logged in. This service implements the CanActivate interface causing the user to only access the student portal if the user is logged in.

**3.4.3 Additional Services** \***need to include figures of MySQL tables after finalizing code**

The backend services of login and registration implemented with Spring Boot and user interface service implemented with angular created a university student portal application that has CRUD (create, retrieve, update, and delete) functionality for the user.

An additional service was Spring security. This was integrated using mail service when verifying the student’s eligibility to register for a new course based on their completed prerequisites and when resetting their password. To achieve this, the application.properties file was configured to send email via STMP. Another service was when the password was encrypted using the BCryptPasswordEncoder. Lastly, MySQL was an additional service. The application.properties file of the backend used a JDBC URL, username, and password per environment of the team members to connect to the MySQL database. MySQL was therefore configured to serve as a database to record student information in their university student portal. The MySQL tables populate the information of the student using the backend service and controls of the user in the user interface. One of the database tables is a courses table with course id, course number, course name, semester, type, prerequisites, part of the day, time, day and faculty of the course. Also, there is a student courses table which includes the student’s CCSU id, course id and status of completed or registered for the course. There is also a user table after the student’s login that stores the student’s id, username, password in encrypted form, first name, last name and email.

* 1. **It all fits (Integration)**

**3.5.1 Role of Microservices**

**3.5.2 Microservice Integration**

**3.5.3 Wiring with Spring Boot**

**4. Planning and Development Strategies**

* 1. **Agile Development**

**4.1.1 Use Cases Diagram**

**4.1.2 User Stories**

**4.1.3 Sprint Planning and Implementation**

**4.2 Version Control**

**4.2.1 GitHub**

The team used GitHub to keep track of the source code. We used Git functionality on our IDE’s, such as in Eclipse, IntelliJ, and Visual Studio Code to create and clone repositories, create branches and push our changes from our local machines to the remote repository. GitHub was chosen because of the familiarity of all team members with Git commands and the effective features in GitHub that allows the team to easily collaborate and review each other’s code by following a feature branch pattern. With this pattern, we had a master branch, which then had feature branches off of the master branch, where changes made in the feature branches after team review were pushed up to the master branch as final.

**4.1.2 Multiple Repositories**

Three separate GitHub repositories were created, one for the front-end with the user interface: CourseRegistrationOp and two for backend: CourseOp and CourseRegistrationOpServices. We chose to include separate repositories to maintain the proper microservice architecture.

Their locations are as follows:

1. CourseRegistrationOp: <https://github.com/HarikaOgirala/CourseRegistrationOP>
2. CourseOp: <https://github.com/HarikaOgirala/courseop>
3. CourseRegistrationOPServices: <https://github.com/HarikaOgirala/CourseRegistrationOpServices>