

# 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**

**1.1 Domain**

**1.1.1** Proposed System

**2. Knowledge Gathering**

**2.1 Domain – Department of CS**

**2.1.1** Courses Datasets

**2.1.2** Faculty Datasets

**2.2 Skillset**

**2.2.1** Semantic Web

**2.2.2** Spring Boot Micro-services

**2.2.3** Angular

**3. Design and Analysis**

**3.1 Technical Stack Decisions**

**3.1.1** Semantic Web Technology

**3.1.2** Java Technologies

**3.1.3** Relational Database

**3.2 Control Flow**

**3.2.1** Application Flow

**3.2.2** Sequence Models

**4. Planning and Development Strategies**

**4.1 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

**4.1.2** Multiple Repositories

**5. Implementation**

**5.1 Semantic Web Schema Models**

**5.1.1** Course Schema

**5.1.2** Faculty Schema

**5.1.3** Integrated Schema

**5.2 Course Op Service**

**5.2.1** SPARQL queries

**5.2.2** Jena API

**5.2.3** Spring Boot wiring

**5.2.4** Front-end JSP Design

**5.3 Login and Registration OP Services**

**5.3.1** Database Schemas

**5.3.2** Springboot Services

**5.3.3** Front-end Angular

**5.3.4** Additional Services

**5.4 It all fits (Integration)**

**5.4.1** Role of Microservices

**5.4.2** Microservices wiring with Spring boot

**6. Application**

**6.1 User Interface**

**6.1.1** Course Op Pages

**6.1.2** Login Page

**6.1.3** Registration Page

**7. Deployment on AWS cloud**

**8. Possible Extensions**

**9. Conclusion**

**10. References**

**Acknowledgements**

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

# List of Tables and Figures

**Figure 3.1.1a** Semantic Web Stack 11

**Figure 3.1a** Application Technical stack 13

**Table 3.1b** Technology versions used 14

**Figure 3.1.2a** Microservices Architecture

**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 3.2.2c** Individuals of Faculty Schema

**Figure 3.2.3a** Adding a property to integrate schemas, validating with Pellet Reasoner. 15

**Figure 3.2.3b** Integrated schema validation at runtime using @PostConstruct . 15

**Figure 3.3a** CourseOp architecture 21

**Figure 3.3.1a** Jena classes for SPARQL queries

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

**Figure 3.5a** ApplicationFlowchart

**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 and 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 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(CourseOp).

A Microservice(RegistrationOp) 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 for RegistrationOp, 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 that 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**

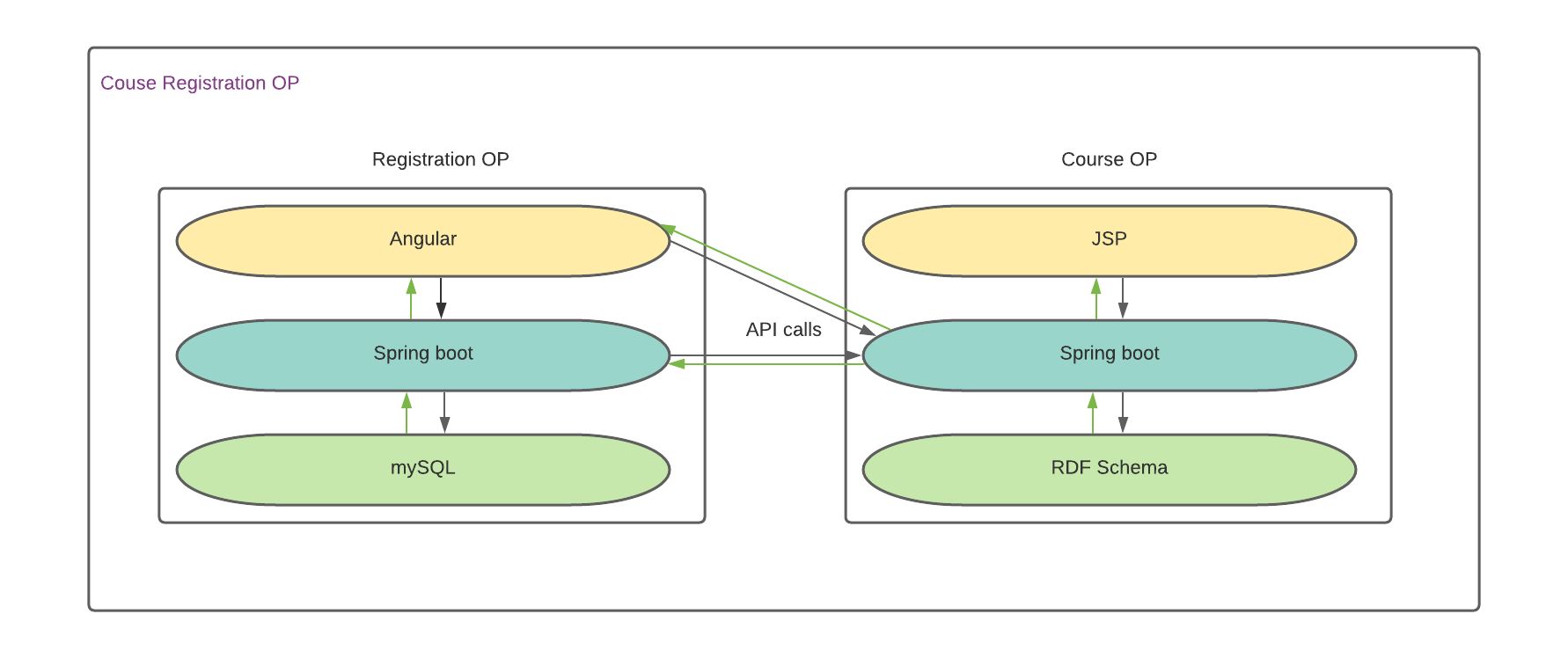
Angular is a component-based framework for building scalable web applications. The basic building blocks of the Angular framework are Angular components that are organized into NgModules. NgModules collect related code into functional sets and an Angular application is defined by a set of NgModules. 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.



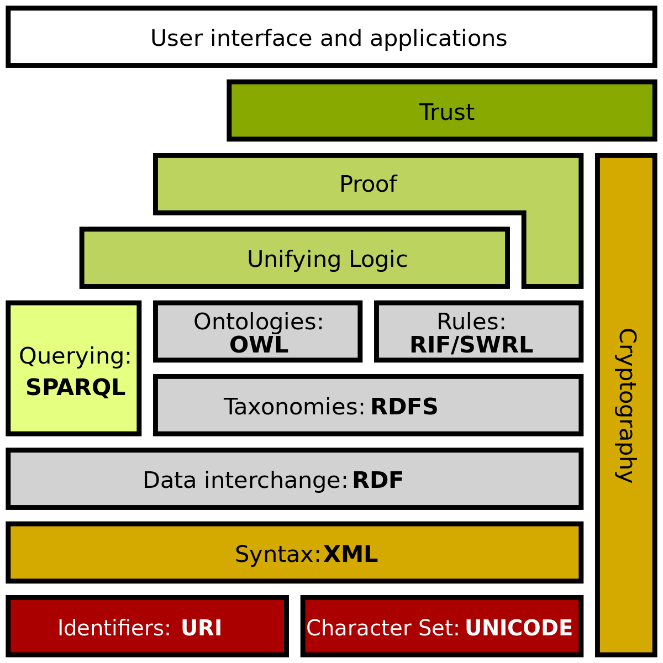
**Figure 3.1a** Application Technical stack

|  |  |  |
| --- | --- | --- |
| Application | Technology | Version |
| C.R.O.P | Java SE | 10 |
| Apache Maven | 3.6.3 |
| Spring boot | 2.3.4 |
| CourseOP | Bootstrap | 3.3.7 |
| JSP and JSTL | 3.0 |
| Protégé | 5.5.0 |
| Apache Jena | 3.15.0 |
| Openllet | 2.6.5 |
| RegistrationOP | mySQL | 8.0 |
| Angular |  |

**Table 3.1b** Technology versions used.

**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.



**Figure 3.1.1a** Semantic Web Stack

**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.

**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.

Spring Boot reduced lots of spring configuration and kick o fast development. Spring Boot offers a new paradigm for developing Spring applications with minimal friction. With Spring Boot, you’ll be able to develop Spring applications with more agility and be able to focus on addressing your application’s functionality needs with minimal. The spring boot helps to connect all the microservice to produce one single application. The data from different port is redirected to a single localhost port.

A screenshot of a cell phone

Description automatically generated

**Figure 3.1.2a** Microservices Architecture

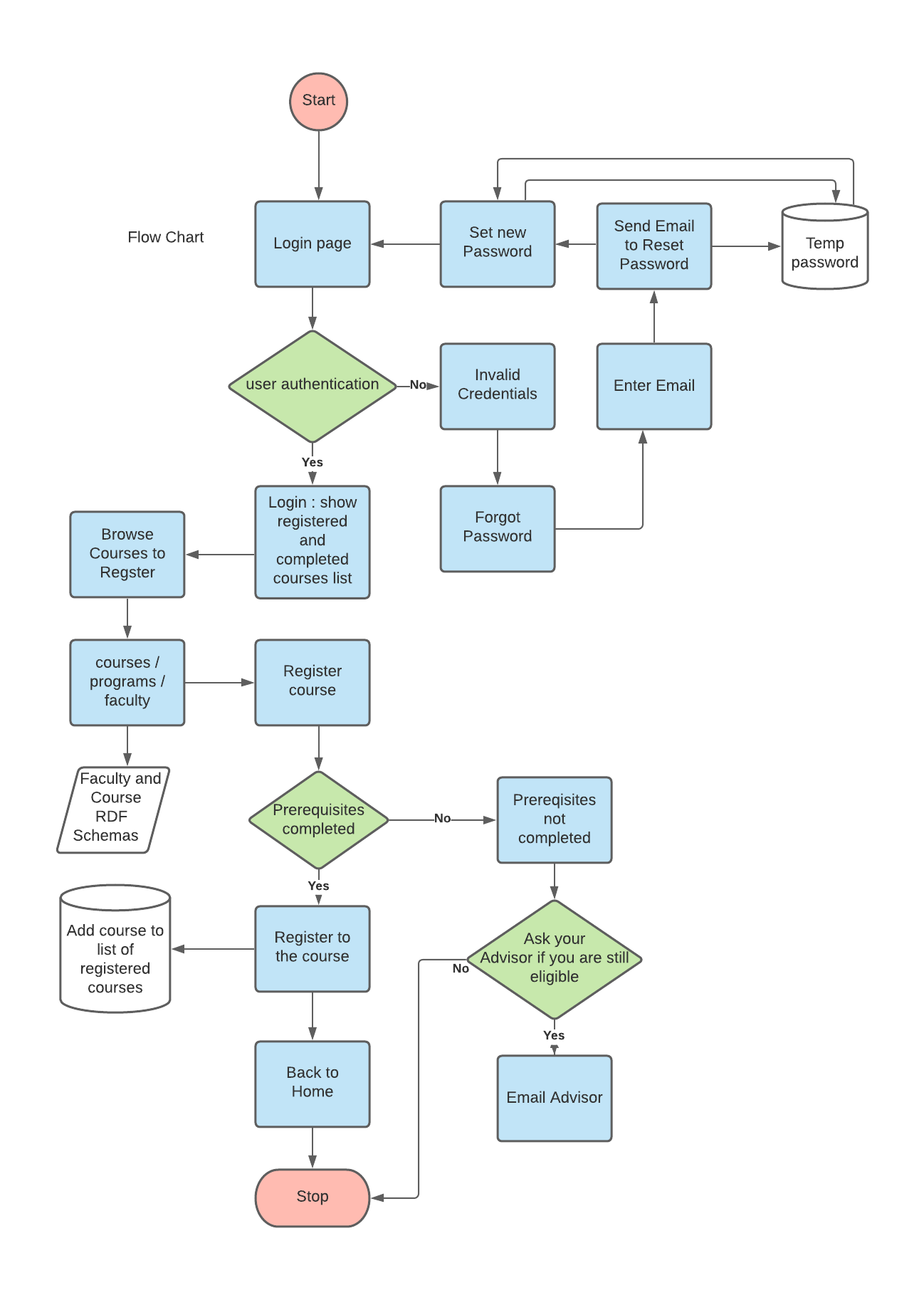
**3.1.3 Relational Database:**

**MySQL:**

MySQL is a Relational database management system based on SQL – Structured Query Language. We use Relational DB to maintain the User login credentials and courses registered/completed. The RegistrationOp Microservice directly interacts with the DB to create, Modify and delete courses according to the user operations performed after logging in.

**3.2 Control Flow**

**3.2.1** Application Flow



**Figure 3.2.1a** ApplicationFlowchart

**3.2.2** Sequence Models

**4. Planning and Development Strategies**

In this section we can see various planning and development strategies that we incorporated in our application. We followed agile development process and maintained git repository throughout our entire development.

* 1. **Agile Development**

  Agile methods or Agile processes is a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals. Agile development refers to any development process that is aligned with the concepts of the Agile Manifesto.

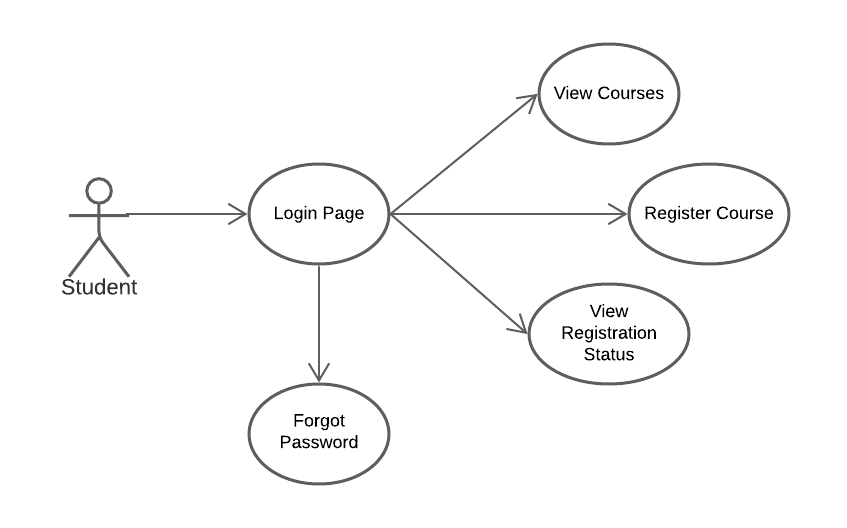
Uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

* Individuals and interactions over processes and tools
* Working software over comprehensive documentation
* Customer collaboration over contract negotiation
* Responding to change over following a plan



**4.1.1 Use Cases Diagram**

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. In our application the student is the user and here we are showing how the user will communicate with the system and how the system should be handled by the user. The diagram shows how the user login the system and view courses and manage the registration status of his courses.



**Figure 4.1.1a** Use-case diagram

**4.1.2 User Stories**

This is the user story board. In this story board the defines all the story that needs to be implemented in different sprints. The story points have been defined based on how difficult the user story is. The story points are defined based on the Fibonacci numbers .

**Story Board:** We have totally 89 story points to be completed which we divided the work for this entire semester.

|  |  |  |
| --- | --- | --- |
| **Story No** | **User Story** | **Story Points** |
| 1 | As a CCSU student, I would like to login the student account, so that I can view my portal. | 21 |
| 2 | As a CCSU student, I would like to view the courses in my portal, so that I can register to courses. | 21 |
| 3 | As a CCSU student, I would like to view my registration status, so that I can register for more courses. | 13 |
| 4 | As a CCSU student, I would like to reset my password, because I forgot my existing password. | 13 |
| 5 | As a CCSU student, I would like to view my completed courses, so that I can register for other courses based completed prerequisites. | 8 |
| 6 | As a CCSU student, I would like to protect my password, so that my password won’t get hacked. | 5 |
| 7 | As a CCSU student, I would like to request permission to register courses, because I have not satisfied the required prerequisites. | 8 |

**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 are merged to the master branch.

**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>

**5. Implementation**

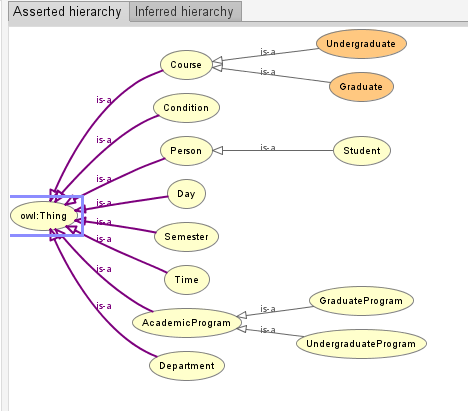
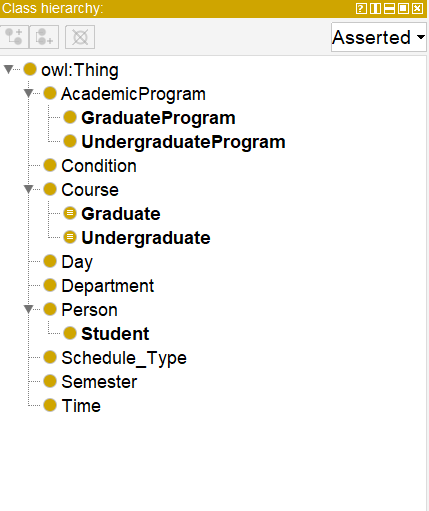
* 1. **Semantic Web Schema Models**

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.

**5.1.1 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.

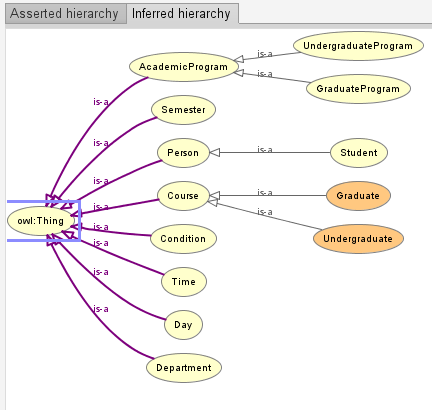
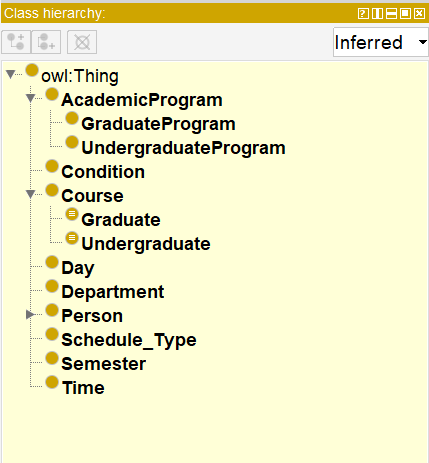


**Figure 3.2.1a**

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.

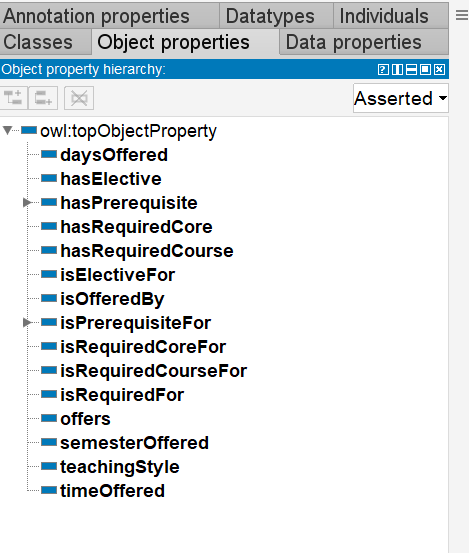


**Figure 3.2.1b**

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.



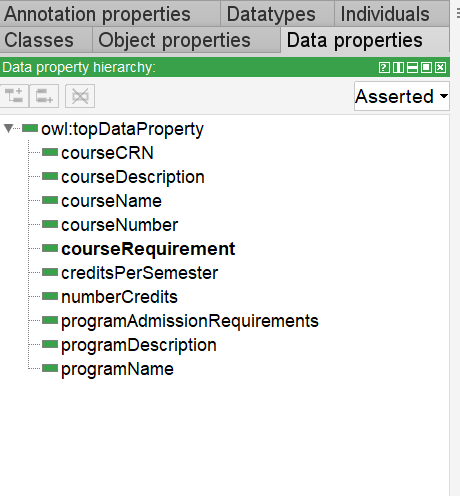
**Figure 3.2.1c**

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

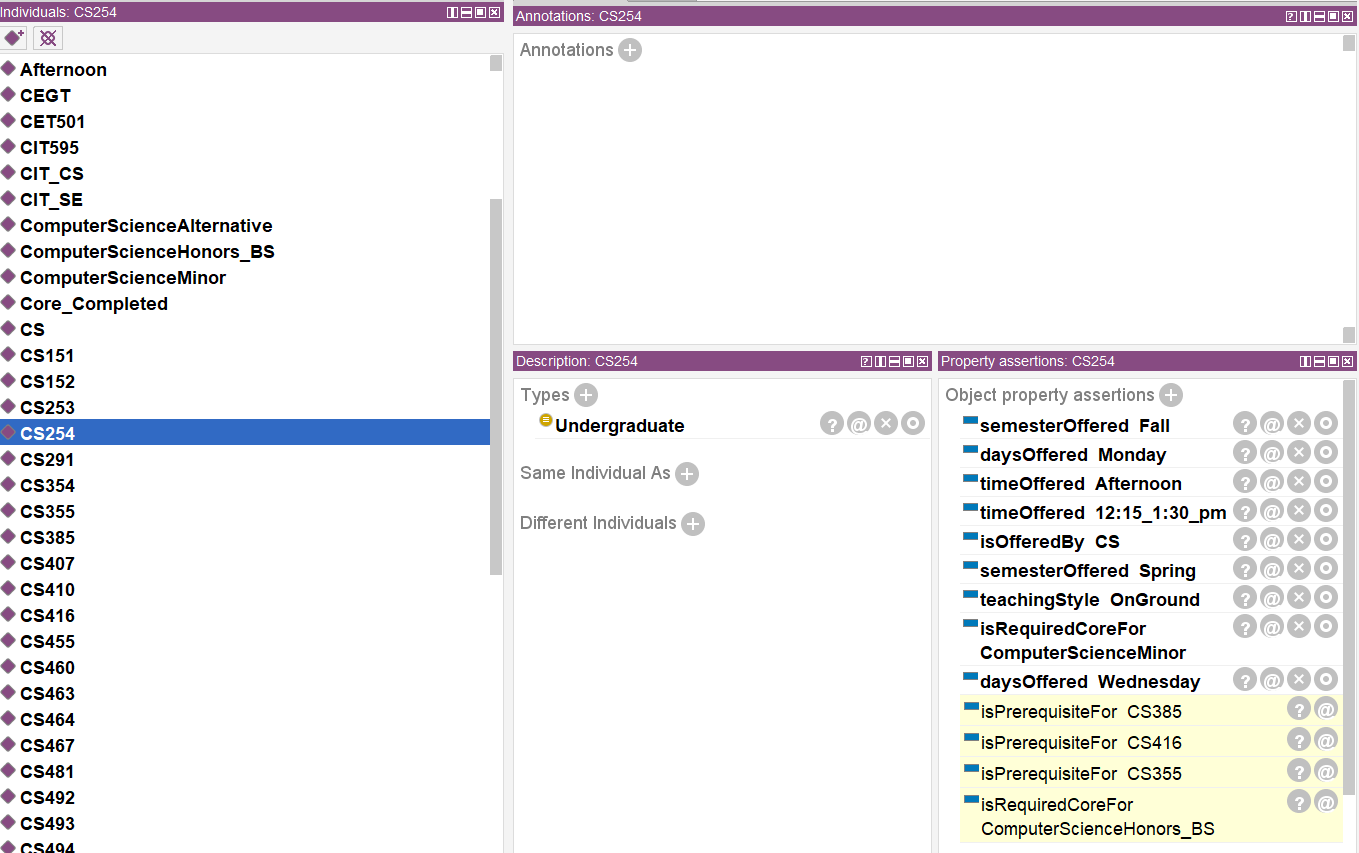


**Figure 3.2.1d**

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.



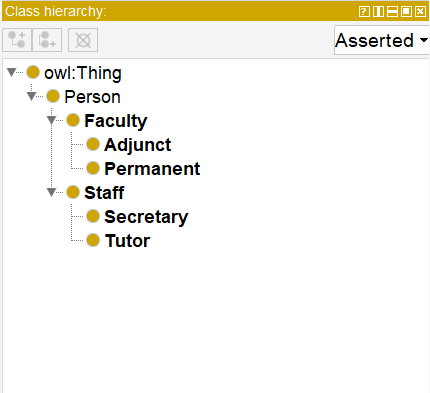
**Figure 3.2.1e**

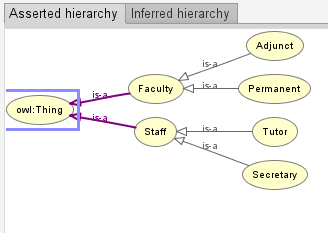
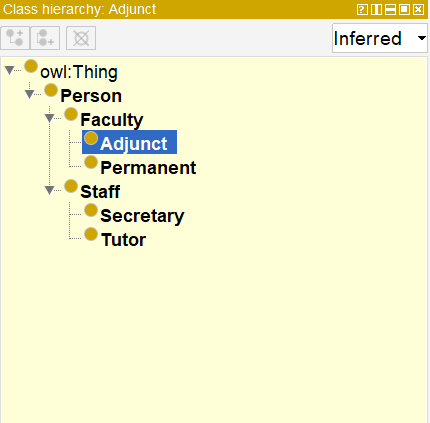
Individuals and Properties of Course Schema

**5.1.2 Faculty Schema**

**Class Hierarchy: Asserted Class Hierarchy**

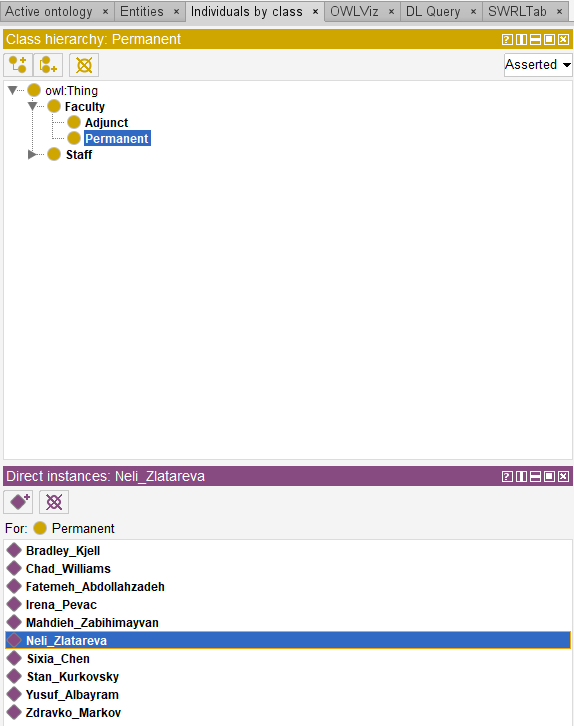
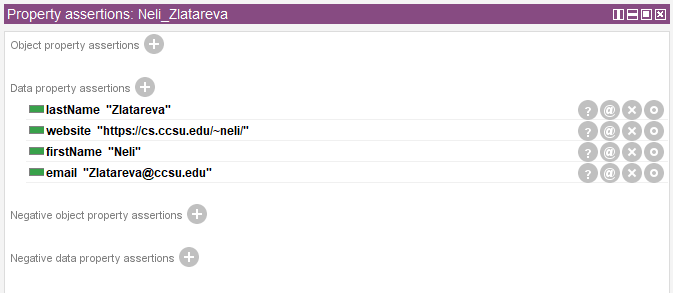
The asserted class hierarchy view is one of the primary navigation devices for named OWL classes. In the asserted class, there are various domains like name, email, phone number and website for faculty and staff for the Faculty Ontology.

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



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

**Individuals and Properties:**

**Figure 3.2.2c** Individuals of Faculty Schema

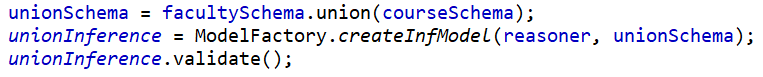
**5.1.3 Integrated Schema**

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.

**Configuring and invoking the Pellet Reasoner in Jena:**

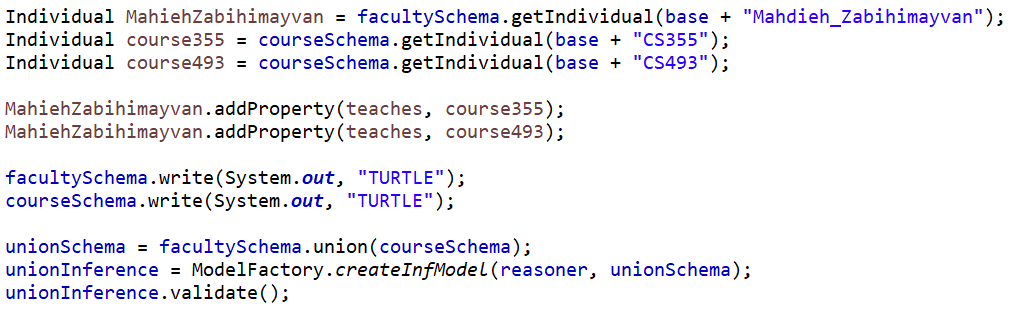
 

**Usage of Pellet Reasoner in Integrated Schema:**



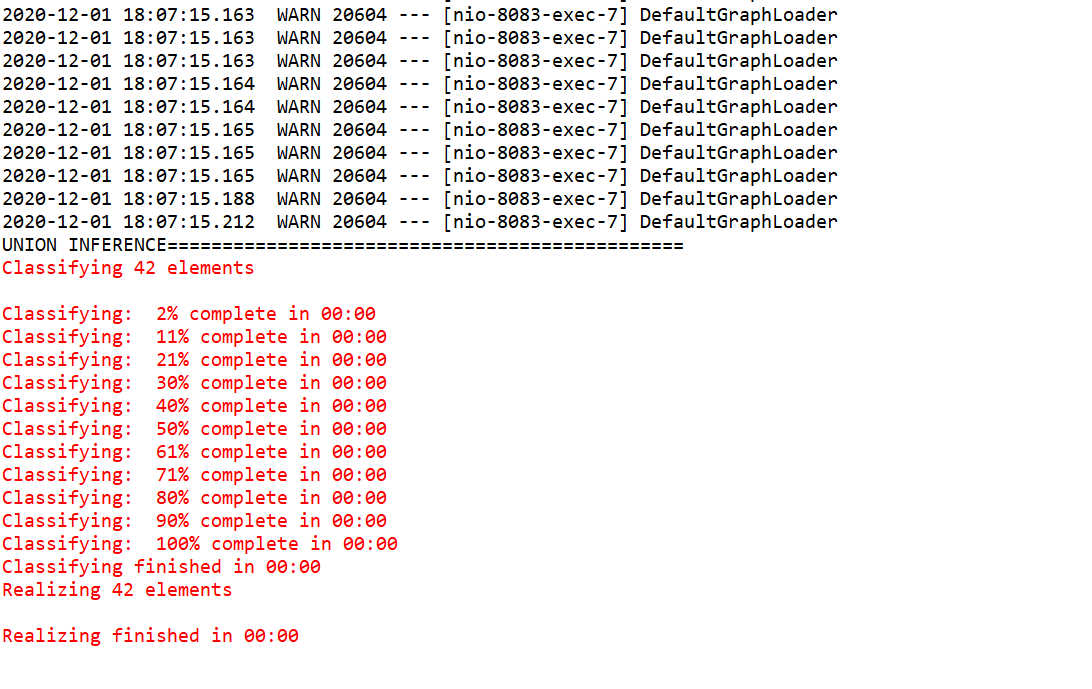
The pellet reasoner is used to validate and obtain results for the course schema, faculty schema and read the integrity schema. The execution and the output of the reasoner running using the pellet reasoner is then obtained.

* 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 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** Adding a property “teaches” to integrate schemas and validating with Pellet Reasoner.

**Validating the Schema after integration:**



**Figure 3.2.3b**

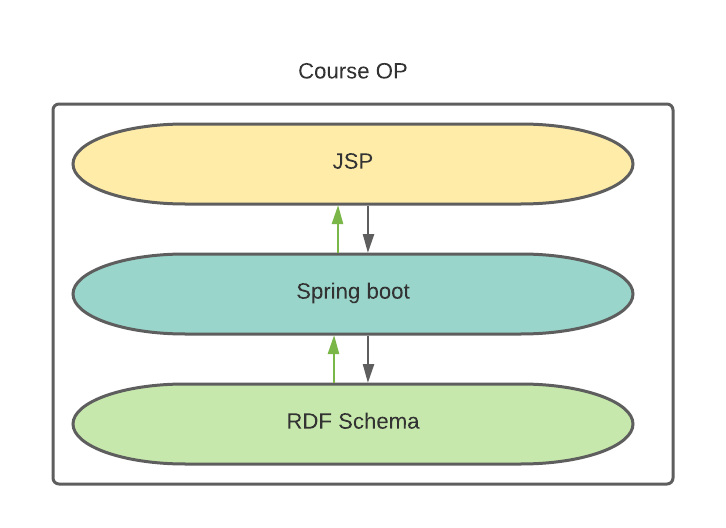
Integrated schema validation at runtime using @PostConstruct .

**PostConstruct Schema Integration:**

@**PostConstruct** is a spring boot annotation to execute a method after the spring bean is initialized. We can have only one method annotated with @PostConstruct annotation. In a method called SchemaGenerator, We are integrating the course and Faculty schemas and running the reasoner to validate this integrated schema to reduce the response time and not integrate it before the request. After Refactoring using PostConstruct, all the schemas are readily available even before user input.

* 1. **Course Op Service**

This service provides the list of courses and faculty based on CS department. The data is obtained from RDF triple store and the data is retrieved as json objects through SPARQL(ARQ) queries in Jena model. Dependencies are managed by Maven and spring boot is used to wire the result-set with Front-end JSPs. The Course Op API is working on the port 8083 in local. The data retrieved is then displayed in the UI through JSP pages.



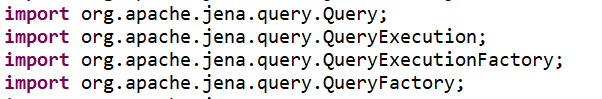
**Figure 3.3a** CourseOp architecture

**Tools used:**

* **Maven:** 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.
* **Protégé**
* **Eclipse IDE**

**5.2.1 SPARQL queries:**

SPARQL is a recursive acronym, which stands for SPARQL Protocol and RDF Query Language. We write SPARQL queries to retrieve data from the RDF schemas. This data is further mapped to JSP pages through spring boot wiring.



**Figure 3.3.1a** Jena classes for SPARQL queries

**Query for Course Schema**

The query populates all the courses its title, Course Number, Pre-requisites for both graduate and undergraduate courses.

**Example query:**

prefix : <http://www.cs.ccsu.edu/~neli/AdvisoryBot.owl#>

prefix owl: <http://www.w3.org/2002/07/owl#>

prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

prefix xml: <http://www.w3.org/XML/1998/namespace>

prefix xsd: <http://www.w3.org/2001/XMLSchema#>

prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

prefix AdvisoryBot: <http://www.cs.ccsu.edu/~neli/AdvisoryBot.owl#>

base <http://www.cs.ccsu.edu/~neli/AdvisoryBot.owl>

SELECT ?course ,?course\_name ,?course\_number ,?prerequisite

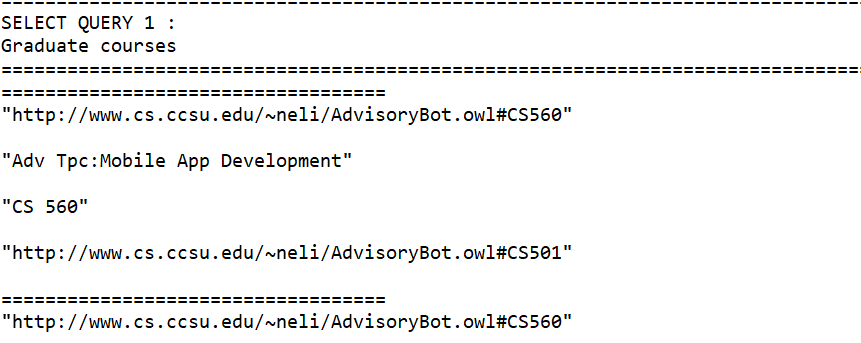
WHERE {?course rdf:type AdvisoryBot:Graduate ?course\_name.

OPTIONAL { ?course AdvisoryBot:hasPrerequisite ?prerequisite .}

?course AdvisoryBot:courseName ?course\_name .

?course AdvisoryBot:courseNumber ?course\_number .}

**Output for the Query:**

****

**Query for Faculty Schema:**

The query helps to create permanent and adjunct faculty and providing all the details of each faculty like name , phone website and email .

**Example Query for Faculty Schema:**

prefix owl: <http://www.w3.org/2002/07/owl#> prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> prefix xml: <http://www.w3.org/XML/1998/namespace>

prefix xsd: <http://www.w3.org/2001/XMLSchema#>

prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

prefix AdvisoryBot: <http://www.cs.ccsu.edu/~neli/AdvisoryBot.owl

base <http://www.cs.ccsu.edu/~neli/AdvisoryBot.owl>

SELECT ?faculty ,?first\_name ,?last\_name ,?email, ?website

WHERE {

?faculty rdf:type owl:NamedIndividual AdvisoryBot:Permanent.

?faculty AdvisoryBot:firstName ?first\_name .

?faculty AdvisoryBot:lastName ?last\_name .

?faculty AdvisoryBot:email ?email .

OPTIONAL {?faculty AdvisoryBot:website ?website .} }

**Output for the Query:**

****

**Query Integrating Faculty and Course Schema:**

The Faculty and course ontology are integrated together where classes, resources, Individuals and property are created.

A relationship can be developed by adding triples to these properties.

**For Example:**

Property teaches = facultySchema.createProperty(base + "teaches");

Individual profChad = facultySchema.getIndividual(base + "Chad\_Williams");

Individual course502 = courseSchema.getIndividual(base + "CS502");

profChad.addProperty(teaches, course502);

From the above code example we have to add property ‘teaches’ which is used to link two individuals ‘profChad’ who ‘teaches’ a particular course ‘course502’.Thus a link can be created between these two individuals in two different ontology in Jena and the result can be obtained using queries.

**Example Query:**

prefix : <http://www.cs.ccsu.edu/~neli/AdvisoryBot.owl#>

prefix owl: <http://www.w3.org/2002/07/owl#>

prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

prefix xml: <http://www.w3.org/XML/1998/namespace>

prefix AdvisoryBot: <http://www.cs.ccsu.edu/~neli/AdvisoryBot.owl#>

base <http://www.cs.ccsu.edu/~neli/AdvisoryBot.owl>

SELECT ?course, ?courseNumber, ?courseName , ?type, ?prerequisite ,?timings,

?requiredCourse, ?days ,?semesters ,?courseCRN ,?courseDesc , ?faculty

Where {

?course AdvisoryBot:courseNumber ?courseNumber

?course AdvisoryBot:courseName ?courseName .

?course AdvisoryBot:timeOffered ?timings .

?course AdvisoryBot:semesterOffered ?semesters .

?course AdvisoryBot:daysOffered ?days .

?course AdvisoryBot:courseCRN ?courseCRN .

OPTIONAL {?course AdvisoryBot:hasPrerequisite ?prerequisite .}

OPTIONAL {?course AdvisoryBot:isRequiredCoreFor ?requiredCourse .}

OPTIONAL {?course AdvisoryBot:courseDescription ?courseDesc .}

?course rdf:type ?type .

?faculty AdvisoryBot:teaches ?course .

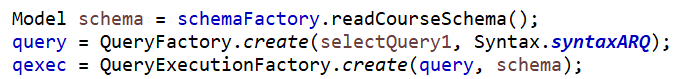
FILTER (?type =graduate ) .}

**Output for the Query:**

****

From the above result we can see the course and the details associated with the course along with the faculty name associated with the course.

**SPARQL query API:**





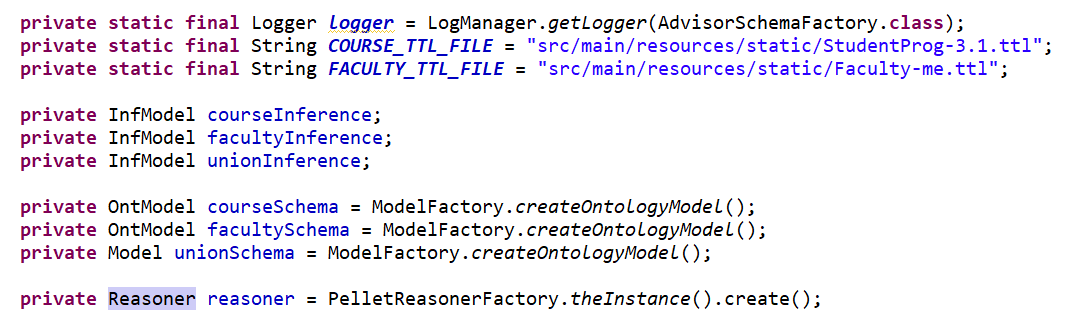
**5.2.2 Jena API:**

Jena API is used to Capture the validated Schema from the reasoner and can be considered as Inference Model after reasoning . Jena API uses various Models to maintain the Schema in a relevant way to User. Some of them are: InfModel, OntModel.

Likewise, Reasoner also have different flavors.

•RuleReasoner is available with API and ReasonerRegistry class is used to obtain the same

•Pellet reasoner needs an external plugin(openllet) and PelletReasonerFactory is the class used to get this reasoner instance.



**Reasoner:**

Jena allows of three types of reasoners:

* Internal reasoner integrated into Jena framework. CourseOP uses this reasoner.
* External reasoners offered as external Java (.jar) files. I was able to find “old” references on how Pellet reasoner can be integrated in Jena.
* External reasoners remotely offered via the Description Logics Information Group (DIG) interface. We will not discuss those.

We are using pellet reasoner for Jena, to integrate different schemas - course and faculty schema in this case. We create a union of these schemas to create one integrated schema.

**5.2.3 Spring boot wiring, annotations used and responsibilities:**

Spring Boot is an open source, microservice-based Java web framework. The microservice architecture provides developers with a fully enclosed application, including embedded application servers.

* JSP Pages(User Interface)
  + Browser hit.
* Controller : @RestController, @GetMapping, @Autowired, @PathParam.
  + initiated in JSP.
  + calls the service.
* Service (where queries are processed) : @Service, @Autowired.
  + Uses Util to fetch Inference Models.
  + Sets query results to Model.
* Model (Plain old Java objects) : @Component.
  + Courses, Faculty, Programs, Timings
* Util (supplies the model to execute queries) : @PostConstruct.
  + Runs the reasoner and integrate schemas.

**Services:**

|  |  |
| --- | --- |
| Service class | Responsibilities |
| CourseService | * Get the list of all Courses. * Get the list of Graduate courses. * Get the list of Under-graduate courses. * Get course details. * Get Schedule. |
| FacultyService | * Get the list of Permanent Faculty. * Get the list of Adjunct Faculty. |
| ProgramsService | * Get program information for CS or CIT streams. * Get program information for Honors, Graduate, Under-graduate or CS Minor. |

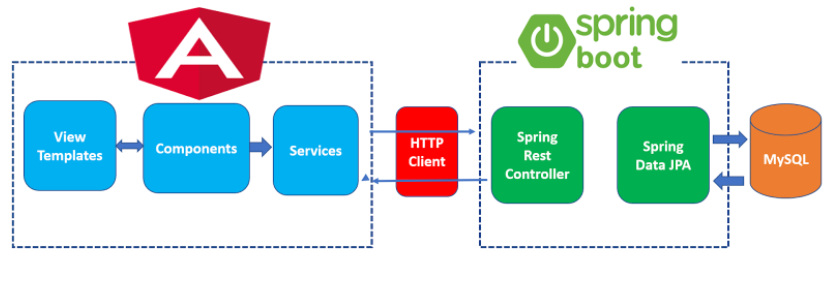
* + 1. **Front-end JSP Design**

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. Java Server Pages (JSP) is a Java standard technology that enables you to write dynamic, data-driven pages for your Java web applications. JSP is built on top of the Java Servlet application. The JSP is wired to the data retrieved from the Jena and it is then formatted and displayed in the UI screens. Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.

In Course OP service we are using JSP and Bootstrap to design the frontend UI screens. This contains the list of all the courses and faculty who teaches the course. This information is retrieved from the Jena application after running the pellet reasoner where the course and faculty schema is incorporated to provide the integrated schema. The queries are written in Jena depending on the data to be retrieved. Three different pages are created to display data of course, Faculty and Programs from the Department of CS.

* 1. **Login and Registration Services**

This service provides the user to login to their student portal and register to the courses for the current semester. The services used here is Java for backend and Angular for Front-end UI service. Here Mysql database is used to store student details and their course registration details. Java service and Angular UI service are different microservice which is created using spring boot.



**Figure 3.4.1a**

Login and Registration Services Application Architecture

**Tools and IDE’s used:**

* Maven Dependencies:
  + Spring boot persistence - Spring Boot starter data JPA.
  + Spring boot starter web.
  + Spring boot started security.
  + Spring boot Devtools.
  + Spring boot started mail.
  + Mysql connector java
* IntelliJ – for Services
* Visual code studio – for Angular

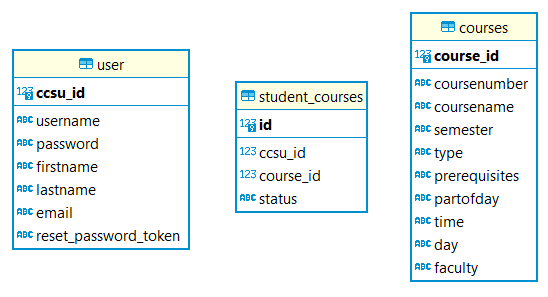
**5.3.1 Database Schemas:**

The Database is used to store the login details and the registration update made by the students. The students can register and deregister to courses based on the choice. They need to satisfy the prerequisites to register to courses. The Database will store the data based on their registration status i.e., Registered/Completed. The Students can view the course data and register accordingly. The password for different users is encrypted and stored in the database. The MySQL tables prepopulate certain student information and can be controlled in the user interface.

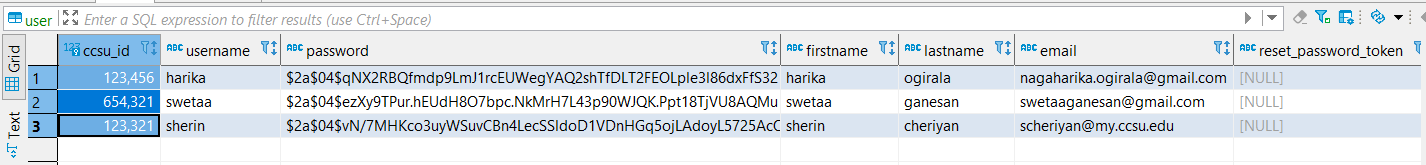
Tables:

* User
  + Maintains Login details with ccsu\_id as primary key.
  + Prepopulated and updated for Forgot password.
* student\_courses
  + Maintains the list of registered and completed courses for specific user.
  + Completed courses are prepopulated and registration with user operations from UI.
* Courses
  + Maintains the list of all courses currently registered or completed.
  + Retrieved from RDF schemas from CourseOP service.

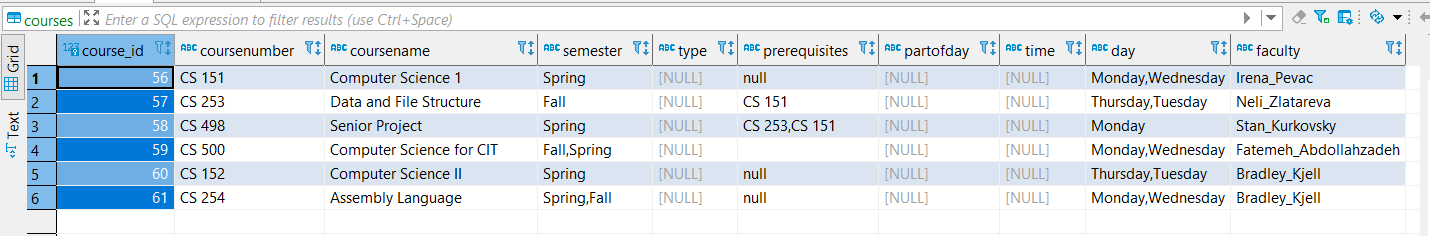
ER Diagram:



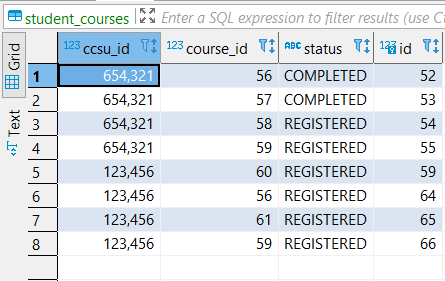
User :



Student\_courses:



Courses:



* + 1. **Spring boot services**

The CourseRegistrationOpServices project developed REST APIs for the courses that students registered. 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.

**Services:**

A service is created for the courses that verifies 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.

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.

|  |  |
| --- | --- |
| Service class | Responsibilities |
| LoginService | * Get User Details * Update Reset Password Token in the Database User table * Get Reset Password Token from the Database User table * Update Password |
| CourseRegistrationService | * Validate and register to a course. * Call Email service * Build Student courses * Build course entities |
| MailService | * Send Email to the Advisor * Build email content |
| UserDetailsServiceImpl | * Load user by username * Get the role of the user |
| RestTemplateService | * Get Course details * Get all course details |

**5.3.3 Front-end Angular Design**

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 RegistrationOP, 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 username 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.

**5.3.4 Additional Services**

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

**1. Registration Override via Email**

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. This service is provided to allow users to send an automated mail to the CS faculty to request permission for registering courses which requires prerequisites which is yet to be completed. This service is provided to avoid separate request to be raised by the students.

**2. Forgot Password – Reset Password via Email**

This service is created to allow students to reset a new password if they do not remember their current password. This service is provided in the login page where the user is authenticated with a reset link to their email the users can use this link to reset their password. The spring security which provides mail service for resetting their password. To achieve this, the application.properties file was configured to send email via STMP.

**3. Password Encryption**

Another service is when the password is encrypted using the BCryptPasswordEncoder. This is a standalone class that generates the encrypted password for the given password. This encrypted password can be manually added into the user database. The application.properties file of the ResgistrationOP uses 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.

**4. Exception Handling:**

Exception handling for RESTful services were also implemented, such as resource is 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 class CourseOpRegistration.java 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.

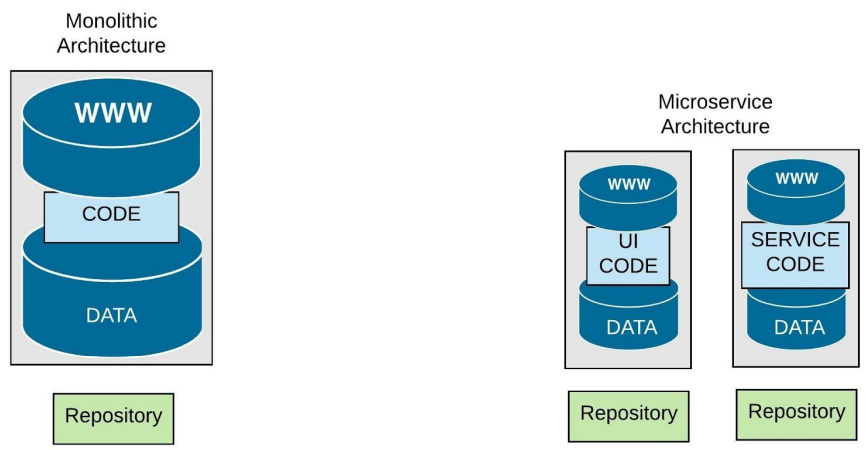
**5. Cors Filter:**

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

Here the entire application is broken down into simple microservice which is running in different port. We are using Spring boot service to integrate all the microservices running in different ports. In our application we have 3 microservice.

**5.4.1 Role of Microservices**

Microservices are increasingly used in the development world as developers work 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. Microservices are a form of service-oriented architecture style wherein applications are built as a collection of different smaller services rather than one whole app. These smaller programs are grouped together to deliver all the functionalities of the big, monolithic application.



There are several benefits to using microservices. For one, because these smaller applications are not dependent on the same coding language, the developers can use the programming language that they are most familiar with. That helps developers come up with a program faster with lower costs and fewer bugs.

* + 1. **Microservices wiring with Spring Boot :**

In our application we will be wiring port 8083 and 8080 to the port 4200. The CourseOP runs on the port 8083 and the RegistrationOP page runs on the port 8080 and the angular login page runs on the port 4200. To run the entire application, we need to run the servers on three different ports. Thus, data from three different port will run on the single port. Error Handling Spring Boot provides an /error mapping by default that handles all errors in a sensible way, and it is registered as a ‘global’ error page in the servlet container.

For machine clients it will produce a JSON response with details of the error, the HTTP status and the exception message. For browser clients there is a ‘whitelabel’ error view that renders the same data in HTML format. In our app the Spring boot is using annotations like Component, Autowired, Controller, Service, RequestMapping(“/path”), PathParam(variable) which is used to map the requests from the Java API to the Front-end services.

|  |  |
| --- | --- |
| Service | Port on localhost |
| Angular | 4200 |
| RegistrationOP | 8080 |
| CourseOP | 8083 |
| MySQL | 3306 |

**Table 3.5a** services and the ports on localhost

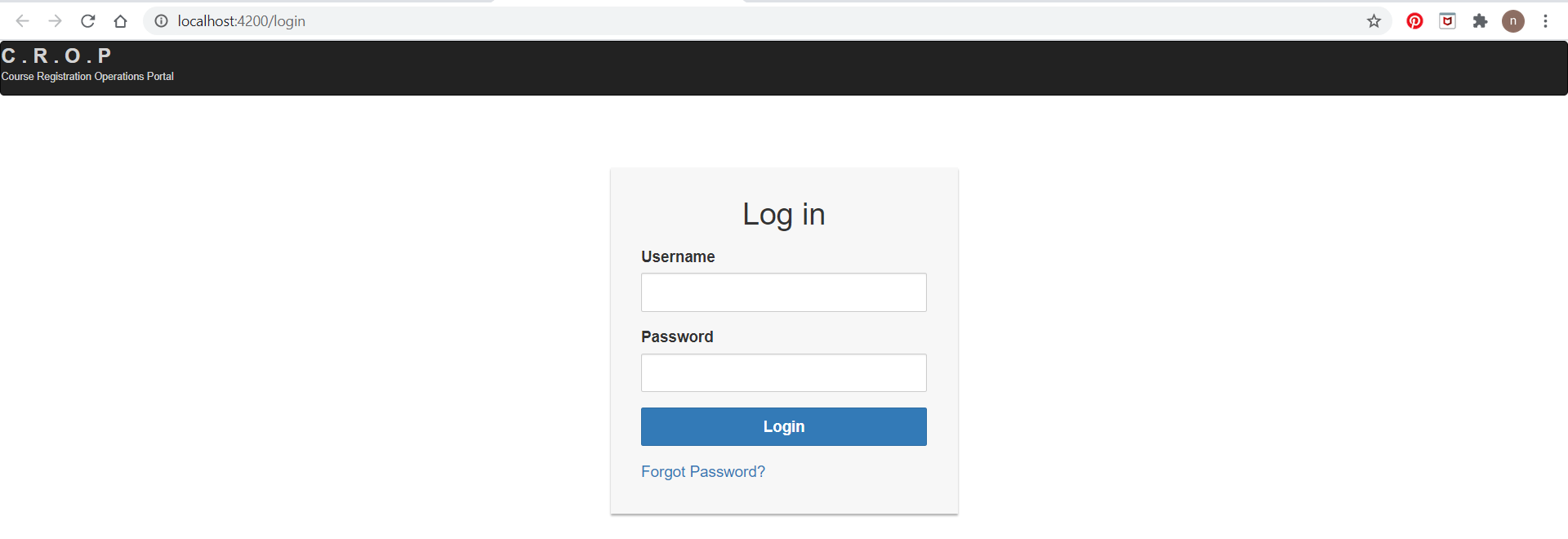
**6. Application**

**6.1 User Interface**

**6.1.1** Course Op Pages

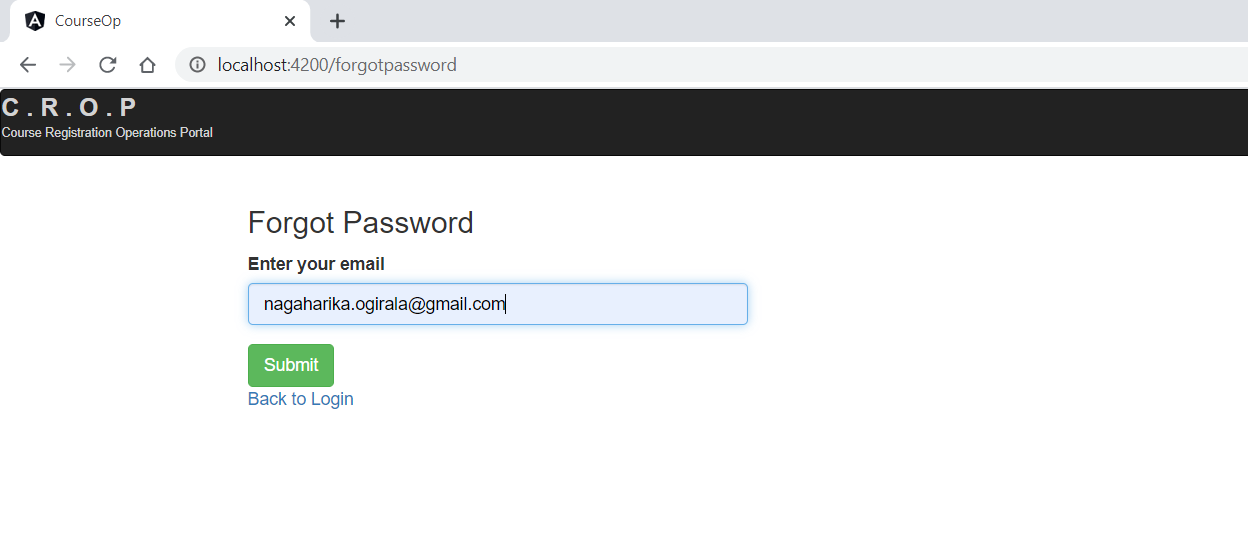
**6.1.2** Login Flow Pages

**Login page:**

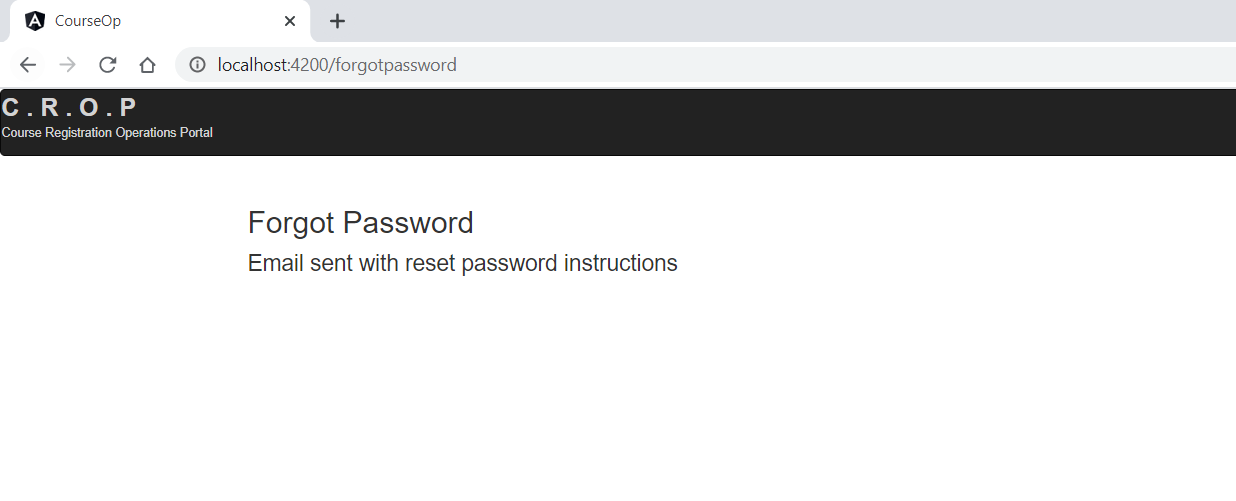


**Forgot password pages:**

**Step 1:**

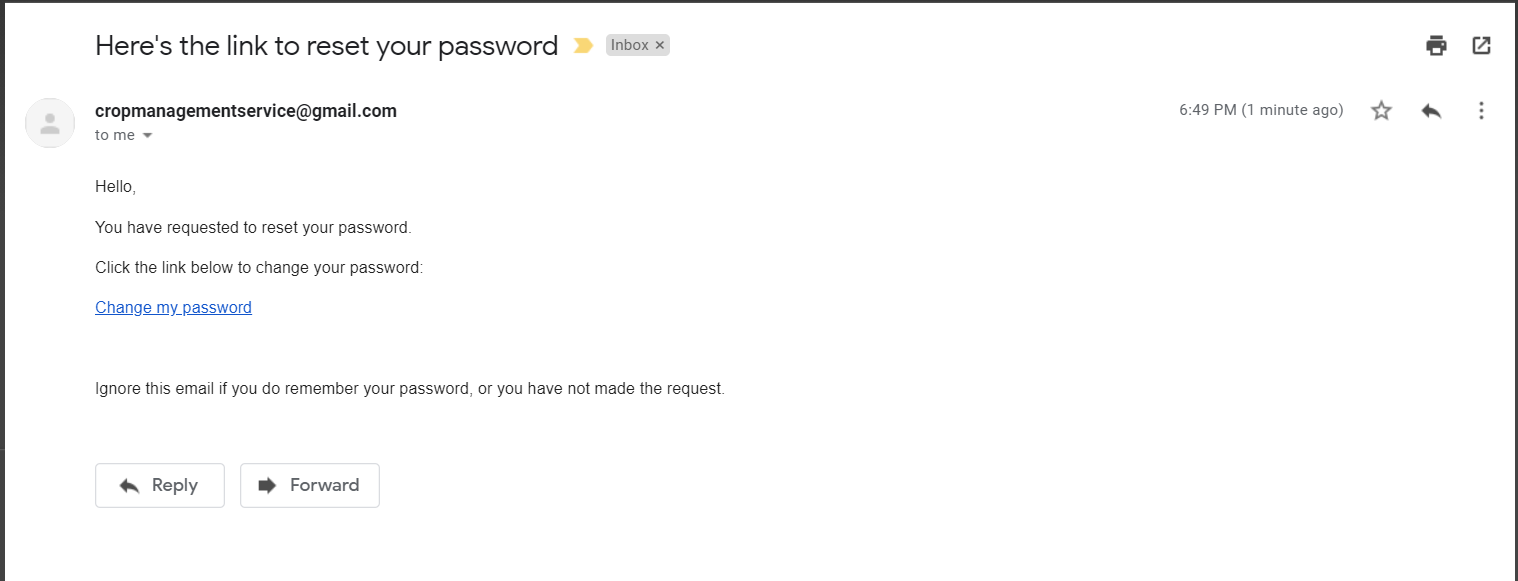


**Step 2:**

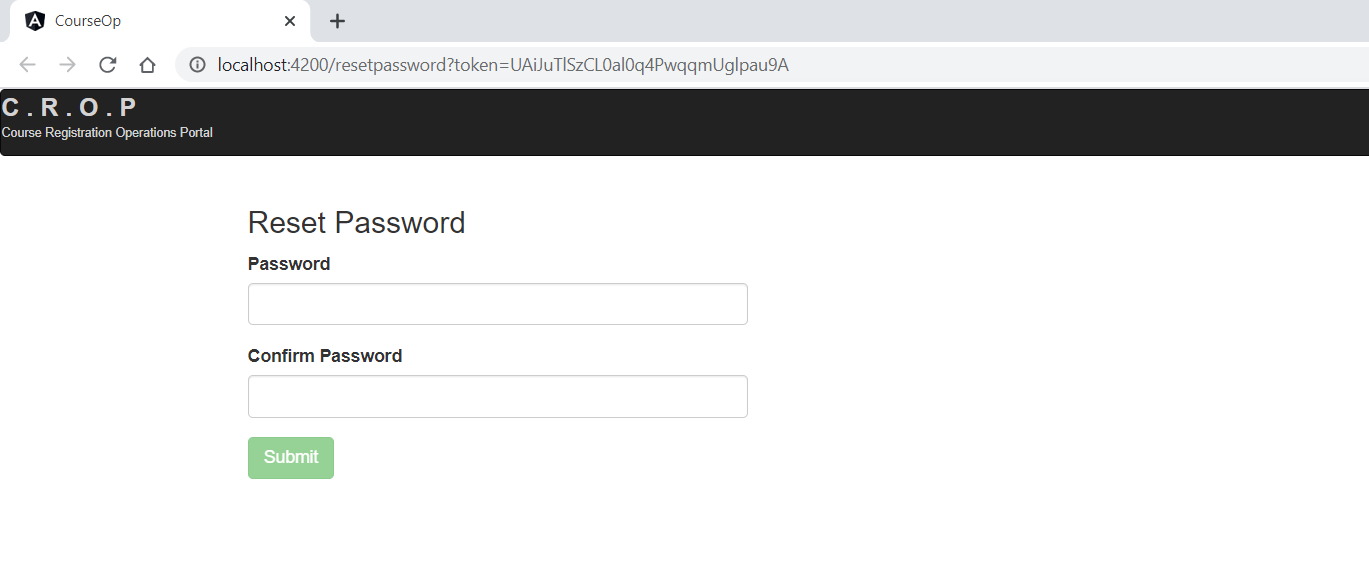


**Reset password pages:**

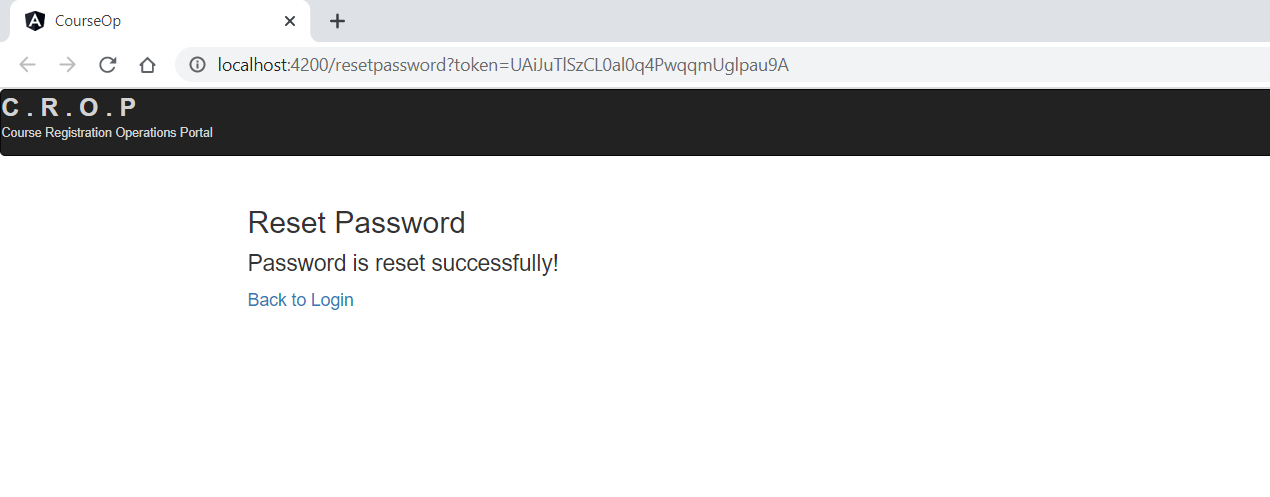
**Email sent:**



**Step 1:**

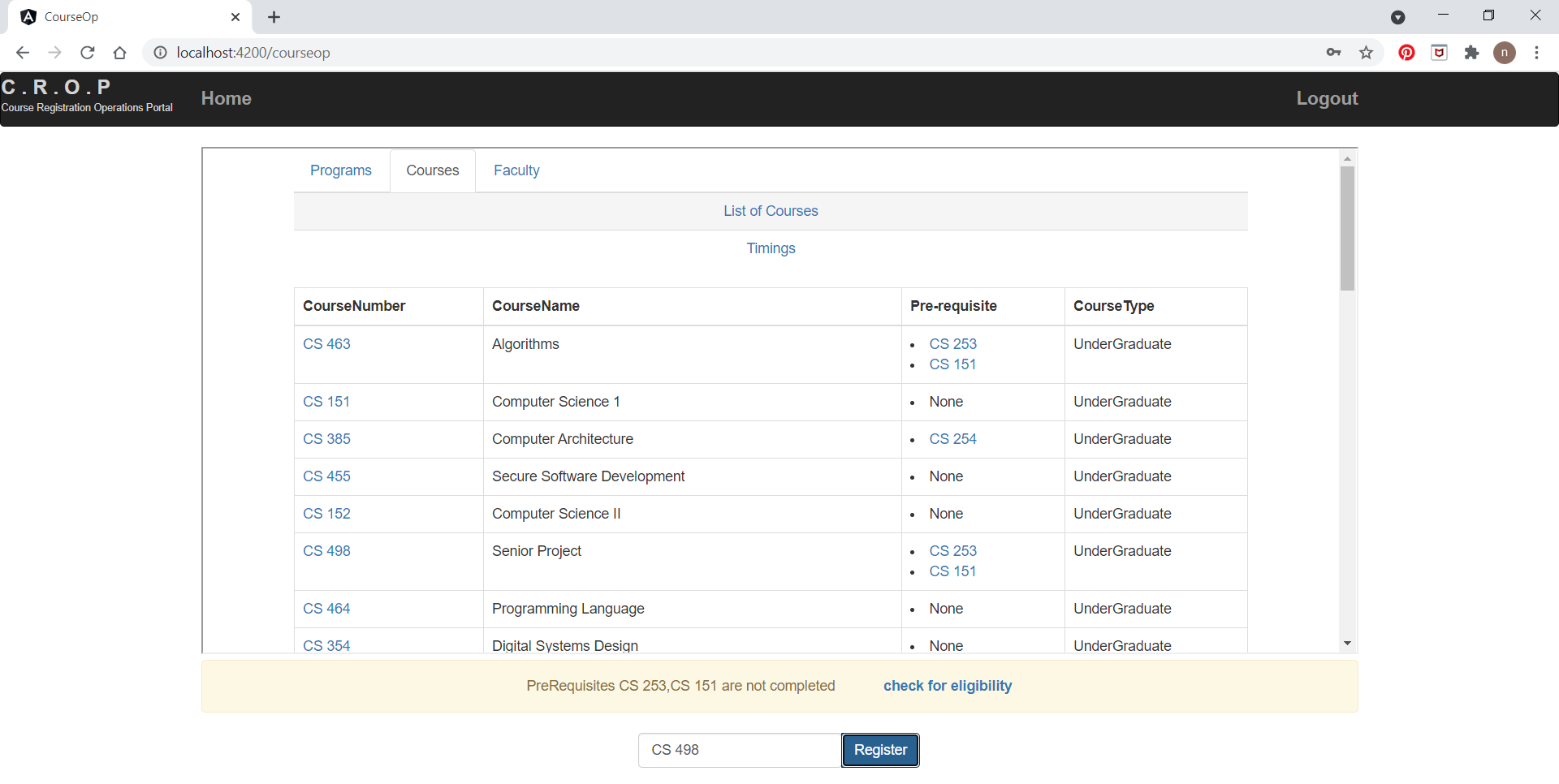


**Step 2:**

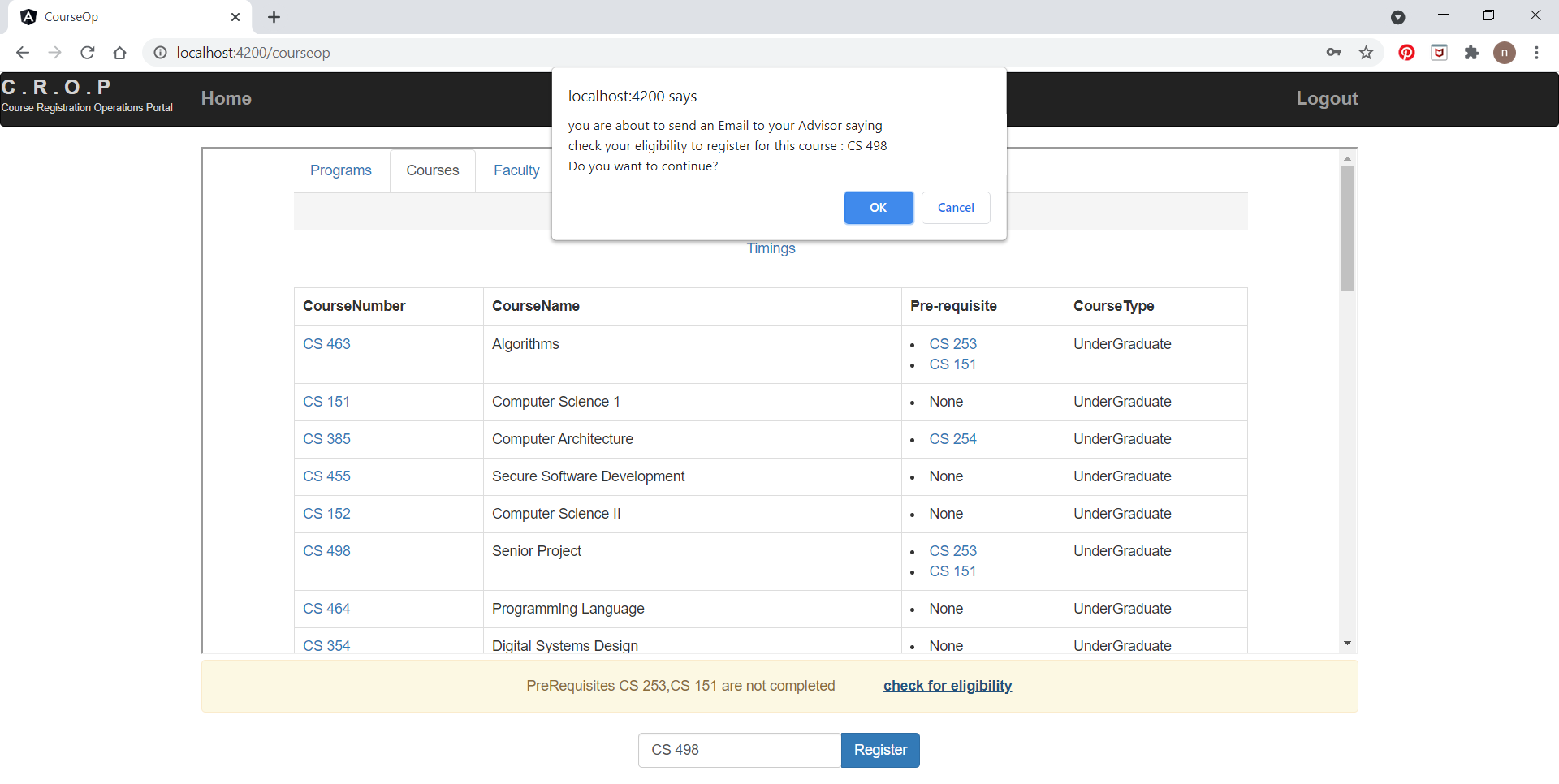


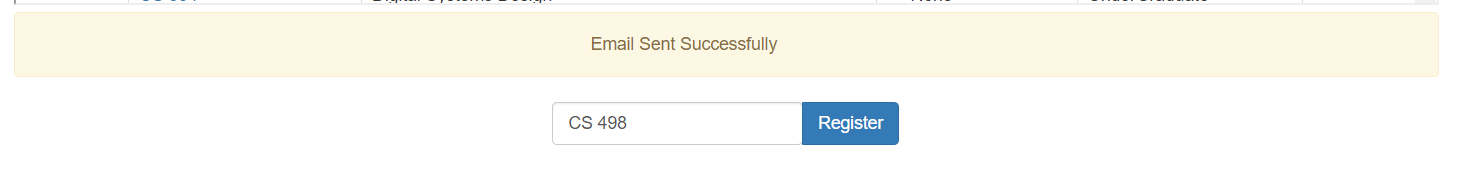
**6.1.3** Registration Page validations:

**Prerequisites:**

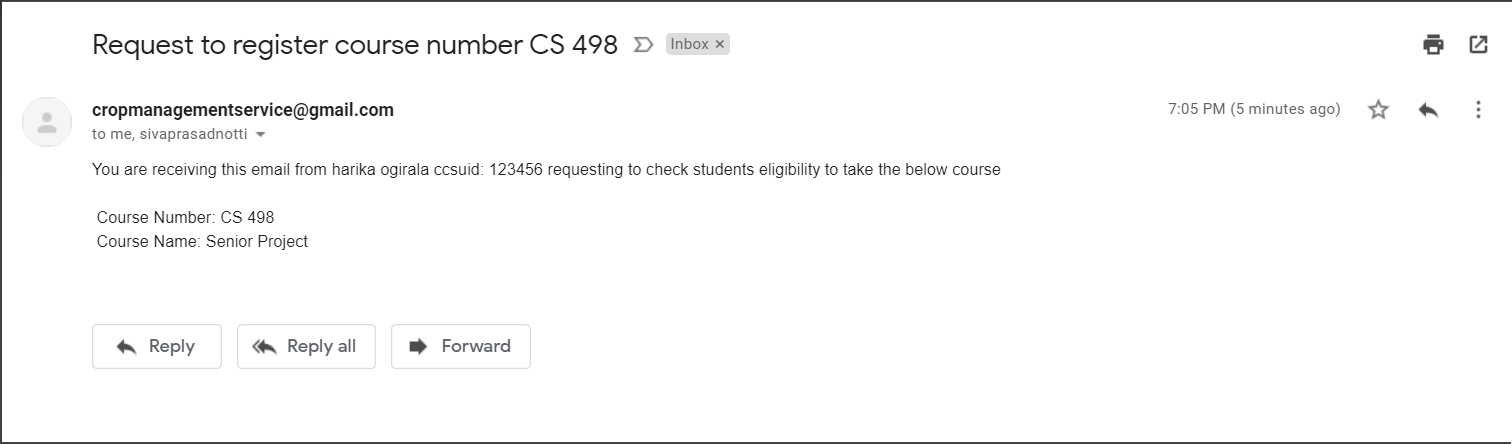


**Check for Eligibility confirmation:**

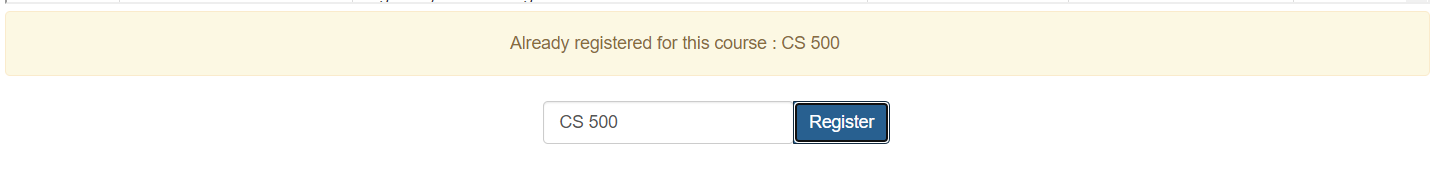


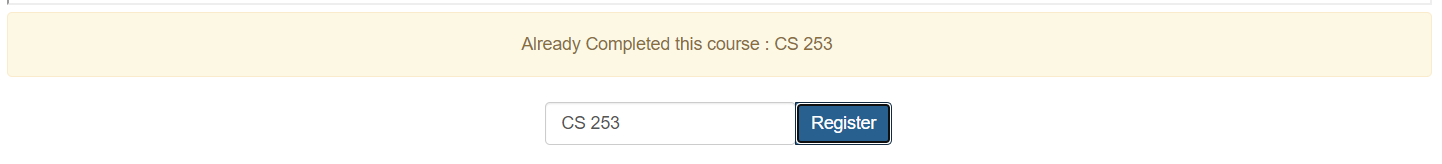


**Email sent to advisor:**



**Register completed or existing course:**





**7. Deployment on AWS cloud**

**8. Possible Extensions**

**Admin login:**

**9. Conclusion**

**10. References**