#### **Modules:**

##### Container Runtime Module

*Purpose:*

This module is responsible for handling service requests that are sent by the user. The Container Runtime isolates running services by creating a specific container instance for the service based on the user’s security levels that it receives in the request. This module also handles shutting down its container instances once the user has disconnected from the service.

*Responsibilities:*

* Handles the REST API endpoint for service requests from the user.
  + POST request
  + Endpoint:
    - {base\_path}/service\_request
  + The specific *service* and the user’s *username*, as String values, are sent in the body of the POST request.
  + Example:
    - service=”course\_manager”
    - username=”student”
* Creates a container instance for the requested service
  + Each service will have its own Dockerfile and Docker image. These contain the operating system image, along with all the needed dependencies, configurations, and files for running this service.
  + Enforces the SELinux label of the IP address of this container based on this user’s authorization levels.
  + Starts the RESTful service on the container instance.
* Returns a response with a message, and the url where the service is running on the host machine (if successful):
  + Example:
    - message=”Service request was successful”
    - url=”127.0.0.1:8000”
  + Returns the session cookie for this user in the header of the response
    - The Container Runtime automatically logs in this user, to obtain this session, before sending the request back
* Shut downs running container instances
  + The container instance stops a running service when the user has disconnected from the service.
  + Destroys the container instance when notified by the container of the service being stopped
* Logs all transactions

##### Course Manager Service Module

*Purpose:*

This module is responsible for implementing the mock Course Manager system as defined in our Course Manager System requirements. It handles all REST requests from the user, communicating these requests to the Data Storage, and returning the response (along with any authorized data) back to the user.

*Responsibilities:*

* Establishes connection with the user to the Data Storage (PostgreSQL database) after startup
  + Connects to the database using statically configured credentials.
  + Sends a response to the user letting them know whether connection has been successfully established.
* Authentication of user via login REST API endpoint (UC1)
  + POST request
  + Handles logining in of all users
  + Endpoint:
    - {base\_path}/api/login
  + The user’s username is sent in the body of the POST request
  + Example request:
    - username=”student\_test”
  + (***NOTE:*** For the purpose of this mock system, the system only looks to see if a user exists with this username. In a real system, a password, that is stored with a secure hashing and a salt, would be used in conjunction.)
  + Returns a response containing a session cookie with a randomly generated value to maintain authentication with this user for the session.
* Viewing user account details via REST API endpoint (UC2-S1)
  + GET request
  + Handles request to view logged in user’s account details
  + Endpoint:
    - {base\_path}/api/user
  + Returns a response with a status code. If successful (200) the user’s account details, including their username, name, and id are contained in the response.
  + Example 200 status response:
    - id=”1”
    - name=”Student Tester”
    - username=”student\_test”
    - gpa=4.0
* Adding new user via REST API endpoint (UC4-S1)
  + POST request
  + Handles requests to add a new user to the system.
  + Endpoint:
    - {base\_path}/api/user
  + The user’s username, name, and role are sent in the body of the POST request
  + Example request:
    - username=”student\_test2”
    - name=”Student Tester2”
    - role=”student”
  + Returns a response with the status code and a message for whether adding a new user was successful or not.
* Deleting existing user via REST API endpoint (UC4-S2)
  + DELETE request
  + Handles requests to delete an existing user from the system.
  + Endpoint:
    - {base\_path}/api/user
  + The user’s username and role are sent in the body of the DELETE request
  + Example request:
    - username=”student\_test”
    - role=”student”
  + Returns a response with the status code and a message for whether deleting an existing user was successful or not.
* Adding new course via REST API endpoint (UC3-S1)
  + POST request
  + Handles request for adding a new course to the system
  + Endpoint:
    - {base\_path}/api/course
  + The course’s name, days, start\_time, end\_time, along with an existing instructor's id are sent in the body of the POST request.
  + Example:
    - name=”CSC316”
    - days=”MW”
    - start\_time=”11:15”
    - end\_time=”12:30”
    - instructor\_username=”jtking”
  + Returns a response with the status code and a message for whether creating a new course was successful or not.
* Removing existing course via REST API endpoint (UC3-S2)
  + DELETE request
  + Handles request to delete an existing course from the system
  + Endpoint:
    - {base\_path}/api/course
  + The course’s id is sent in the body of the DELETE request.
  + Example:
    - course\_name=”CSC316”
  + Returns a response with the status code and a message for whether deleting an existing course was successful or not.
* Adding student to an existing course via REST API endpoint (UC3-S3)
  + POST request
  + Handles request to enroll a student to an existing course
  + Endpoint:
    - {base\_path}/api/mapping
  + The student’s username and the course’s id are sent in the body of the POST request.
  + Example:
    - username=”student tester”
    - course\_name=”CSC316”
  + Returns a response with the status code and a message for whether enrolling the student in the course was successful or not.
* Modifying a student’s grade for enrolled course via REST API endpoint (UC5-S1)
  + PUT request
  + Handles request to update a student’s grade for a course
  + Endpoint:
    - {base\_path}/api/mapping/grade
  + The course’s id and the student’s username are sent in the body of the PUT request.
  + Example:
    - course\_name=”CSC316”
    - username=”student tester”
    - grade=”4.0”
  + Returns a response with the status code and a message for whether updating the student’s grade was successful or not.
* Viewing course schedule via REST API endpoint for Student (UC6-S2)
  + GET request
  + Handles request to view a student’s course schedule with grades
  + Endpoint:
    - {base\_path}/api/schedule
  + Returns a response with a list of course objects, along with an associated grade field.
  + Example response:

[

name=”CSC316”

days=”MW”

start\_time=”11:15”

end\_time=”12:30”

instruct\_name=”Jason King”

grade=4.0,

name=”CSC333”

days=”TH”

start\_time=”2:15”

end\_time=”1:30”

instruct\_name=”Sarah Heckman”

grade=2.0

]

* Viewing course schedule via REST API endpoint for Instructor (UC6-S3)
  + GET request
  + Handles request to view an instructor’s course schedule with list of enrolled students
  + Endpoint:
    - {base\_path}/api/schedule
  + Returns a response with a list of course objects, along with an associated list of student names.
  + Example response:

[

name=”CSC316”

days=”MW”

start\_time=”11:15”

end\_time=”12:30”

students=[ “Jeen”, “Caleb”, “Spencer”, “Daniel”, “Jonathan” ],

name=”CSC492”

days=”TH”

start\_time=”9:30”

end\_time=”10:45”

students=[ “Jeen”, “Caleb”, “Spencer”, “Daniel”, “Jonathan” ]

]

* Viewing all logged transactions for the Course Manager System (UC7)
  + GET request
  + Handles request to view all logged transactions
  + Endpoint:
    - {base\_path}/api/logs
  + Returns a response with a list of log objects.
  + Example response:
    - code=”401”
    - description=”Successful delete user”
    - logged\_in\_user=”coordinator”
    - secondary\_user=”user”
    - transaction\_type=”Delete”
    - timestamp=” 2020-03-06 20:11:53.0”
* Disconnection of user via user logout endpoint
  + GET request
  + Handles logging out of all users
  + Endpoint:
    - {base\_path}/api/logout
  + Disconnects from the database
  + Notifies Container Runtime of service shutdown
  + Shutdown running application
* Logs all transactions