[**Final Project Requirements**](https://online.oit.edu/webapps/assignment/uploadAssignment?content_id=_1964826_1&course_id=_21621_1&group_id=&mode=view)

In general, you need to develop a full stack Web application.  Feel free to use your lab work or my example for guidance.  
It needs to include the following:

* A domain other than education or my example domain.
* At least 3 domain entities.  For example, in the labs the entities where student and person.  You can choose  
  whether or not the entities have relationships defined by primary/foreign keys.
* A frontend implemented using Vue with the following:  
    
  + A "home page" that describes the application.
  + A "navbar" allowing the user to navigate around the application.
  + A login page that uses the Auth0 SASS service for authenticating user logins and redirecting  
    authenticated users to the application's "home page".  Add validation checks to make sure the input fields  
    are not empty.
  + For each entity, include a way to list, add and delete instances of the entity.  You can use my example  
    application for an example of how to implement the entity add and delete.  For adding an entity, add  
    validation logic to make sure the required fields are not empty.
  + A way to logout of the application.
  + Use CSS styling to give the application some "style".
  + Call the backend REST API to get/add/delete entities using a secure token retrieved from the Auth0  
    SASS service.
* A REST API for allowing the frontend to access the application's data from a database.  The API should  
  be secured using the Auth0 SASS service.  Implement this API using ASP.NET Core MVC.  You should  
  have one MVC controller per domain entity.
* Use Postgres for the database.  You'll need so include scripts for creating the tables and some sample data.
* Use Entity Framework/LINQ for accessing the data from the database using "repository" classes to encapsulate  
  the data access code.
* Use "service" classes to implement at least one business rule for each of the entities.
* Use dependency injection for injecting the "service" classes into the MVC controllers for providing application business  
  logic/rules.  Also use dependency injection for injecting the database repository classes into these service classes.
* Include at least one unit test for each business rule you implement in the service classes.
* Include a Dockerfile for the frontend application and backend REST API.  Push Docker images for both the frontend application  
  and the backend REST API to your Docker Hub account.
* Include a docker-compose.yml file that references the Docker images form your Docker Hub registry account.  I should be able  
  to take this and "spin up" your application and use it.
* Extra credit (20 points):  Use a 3rd party REST API, much like the weather service I used, to add additional functionality to the application.
* No credit but worth trying:
  + Deploy the application containers to Azure "Container Instance" or Kubernetes container based services.
  + Create a build and deploy pipeline for you application using Azure DevOps.