# CO Webapp Initial Design

## Architecture

Using: Layered n-tier Architecture

The current barebone design can be done traditionally in a single server (or separate into 2: one for backend and one for the database).

We can change it to a microservice architecture later but since we are not planning to scale the deployment, this is not ideal.

## Database

Using: Microsoft SQL Server

Backup: [Ask IT about backup and whether they have RAID arrays]

Tables:

* users
* [Details needs to be added as we get the data descriptions]

## Backend

We need to organize data input and output so that it can be easily query-able and maintain hierarchy. Therefore, an object-oriented approach should be used.

### UML Diagram

[Details needs to be added as we get the data descriptions]

### Class Descriptions

[Details needs to be added as we get the data descriptions]

### Endpoints

GET /q:

Main query-ing endpoint. Consider using different parsing methods and sanitize input(remove any SQLi-related symbols). Limit accepted SQL commands only to query-related commands e.g. no DELETE, DROP, UPDATE, etc. Implement stored query where possible.

POST /register:

Registration endpoint for user. As per Prof. Chelsea’s request, only an institutional email is needed and notify Prof. Chelsea to approve the account creation for that person. If approved, add the person to the user database with a secure-password hash-ed and email the person with the link to reset password (the below endpoint).

POST /resetPassword:

Update password endpoint. Can be done using OTP token generated and then keep it in a URL that expires in a few minutes.

POST /login:

Authentication endpoint for verifying someone trying to update. Use password-username authentication. Make sure the password stored in the users table is hash-ed. Do not store unencrypted password—sensitive data-at-rest needs to be encrypted. Launch generateToken endpoint below if successful.

POST /generateToken:

This endpoint is used if login is successful. Check either OAuth2 or JWT implementation—whichever is easy to implement for you.

POST /update:

* token

Main update endpoint. Use token as an additional input in order to verify that the person is logged in.

Expansions:

* Data verification endpoints: Add on to the update endpoints—can be auto-checking or graphical interface to assist Prof. Chelsea in verification.
  + Check bounds and domain-based knowledge verification.
  + Construct simple graphical models (linear or polynomials) from the current database and plot the new points on. [Ask Prof. Chelsea for domain-based checking e.g. useful plot comparisons]
* Analytics endpoints: To determine any insights that Prof. Chelsea might deem useful—can add a house own ML model, etc.