Semester Project Phase 3 - The Final Phase

CS 3200 | Spring 2025

Overview

In Phase 3, you will pull the entire application together through realistic data generation, planning and developing your REST API, and implementing a set of UI screens in Streamlit. Finally, you will package this all up into your GitHub repos and prepare to WOW us!!!

Phase 3 Requirements

In Phase 3, your team needs to accomplish the following:

1. Create a REST API Matrix for your application
    a. Develop a list of *resources* needed to satisfy each user story of each persona.
        i. You can refer to your SQL queries developed for each user story for help identifying these as well.
        ii. Note that it is completely reasonable that one user story might need two or three resources. It is also reasonable that two different user stories might use a common resource. (The relationship between user stories and resources is not 1:1.)
    b. Determine the appropriate HTTP methods/verbs to associate with each resource based on the user story
    c. Organize these into a REST API Matrix as demonstrated in class. In each cell, give a brief synopsis of the action that will happen for that route. Also, indicate which user story will make use of that route.
2. **Team Repository Owner**: If you haven't done so already, create a public fork of the project boilerplate template repository. When you fork the repo, give it a name related to your project (rather than leaving the default name). Note: In the rest of this document, I will refer to this repo as the **Project Repo**.
    a. Ensure that the repository has **public visibility** so it can be accessed for grading purposes.
    b. Make sure that each team member has been given collaborator permission in GitHub to your Project Repo.
3. **Other Team Members:** Each team member should now clone the project repository to their laptops.
    a. You can clone the project repo via the command line, Git client in VS Code, or via Github Desktop. (If you're new to collaboration via Github, there are tons of tutorials online and videos on YouTube that cover the topic in various levels of detail. One I might recommend is here by FreeCodeCamp starting about ⅓ down with the section titled GitHub Workflow.)
4. **All Team Members:** Work on the project repo.
    a. **Important: Every team member will be required to have commits to the project repository. You can see commits by team members in Github by clicking *Insights > Commits***. The commits must be made through use of a Git Client (such as command-line git or Github Desktop); they may NOT be directly uploaded to GitHub through a browser. We will NOT accept excuses such as "I made my commits from my team member's computer" as a reason for your GitHub user having no commits.
    b. As you work on the project, update the README.md in the main repo folder to include an overview of your project and information about how to start the docker containers (see other GitHub repos for models of how to write a good README).
    c. As part of developing your project, *please remove any sample code/files from the Project Repo that you are not directly using in your project*. This includes Python code as well as sample SQL files.
5. Generate realistic sample data for **all** your database tables using Mockaroo, the Python Faker library, or a similar tool.
    a. One exception of using Mockaroo to generate data for a particular table is if you are using a data set found elsewhere online that you plan to import. That is totally fine. If you need to link to this data via foreign keys when generating the remainder of data in Mockaroo, you can upload datasets to Mockaroo.

b. How much sample data do you need to create? This is hard to answer, but here are some general rules-of-thumb for your project:
    i. Remember the goal of this stage of the project: make the data in your application feel real and not contrived.
    ii. For tables representing strong entities (for Northwind, these might include employee, customer, product, etc), aim for 30 - 40 rows.
    iii. For tables representing weak entities, aim for 50 - 75 rows.
    iv. For bridge tables (the ones used to resolve M:N relationships), aim for at least 125 rows.
    v. Be cognizant of participation constraints. If a table has mandatory participation, make sure that is reflected in the generated data.
c. Add your generated data to the project repo's **database-files** folder.
    i. If your data isn't in SQL Insert statement format, you can write a simple Python script that will convert a csv file with a header row into Insert statements.
    ii. You can have a single file with both the DDL for the schema and your mock data OR you could separate them into two or more separate files. If you do split them, remember that the files will be executed in alphabetical order by MySQL.
    iii. Any **.sql** files in the **database-files** folder of your repo are automatically executed when you ***create*** a new database container; simply restarting an existing one won't do the trick. So, if you make changes to your database schema or sample data, you'll need to recreate the database container in docker so the new files will be executed. You can use Docker Desktop to delete the mysql container using the trash icon. You can then create a new database container using terminal command `docker compose up db -d`

6. Implement the routes based on your REST Matrix inside the **api/** code base.
    a. Build out the routes in Flask. You should separate the routes into Flask Blueprints based on some logical organization, for example, by persona or by database table. Follow the model of the *customers* and *products* sample blueprints in the project repo. Feel free to reuse the code in those routes.
    b. Your project should have at least 4 Flask Blueprints, and every Blueprint must have at least 5 routes. You are absolutely welcome to have more routes.
    c. Your implementation must use all 4 HTTP verbs we covered. You must have at least 2 Post routes, 2 Put routes, and 2 Delete routes, and you can have no more than 1 of each in a single blueprint.
    d. How many routes in total should your project have? I think ~20 is a good target.
7. Build your Streamlit UI.
    a. The sample code for the Streamlit UI is found in the **app/** folder of the project repo.
    b. Remember, you are NOT implementing user login or account creation.
    c. General Overview of what you need to implement in the front end:
        i. Build/update the app's landing page to mimic one user from each archetype (persona) logging in (achieved by clicking the button). You can find the sample code for this in **app/src/Home.py**. Change the logo - feel free to use an AI generated logo.
        ii. For each of the sample users from 7.c.i above, create or update their landing page. In the sample app, you can find these in
            1. app/src/pages/00_Pol_Strategy_Home.py,
            2. app/src/pages/10_USAID_Worker_Home.py, and
            3. app/src/pages/20_Admin_Home.py.

            You'll have to add a 4th one for your 4th persona
        iii. For each of the 4 users, add 3 streamlit pages containing features/functionality pertinent to your project. You can of course add more if you'd like.
            1. Strive for a cohesive collection of functionality for each persona; your implementation should cover at least 3 of the 6 user stories you created for each persona.
            2. Somewhere among the 12 feature pages, you should use all the routes you implemented in

Flask, including the POST, PUT, and DELETE routes.
3. Note that neither the Main home page nor the individual user home pages count towards these minimums.

d. Check out the Streamlit Documentation. It is well-written.
   i. Streamlit Basic Concepts
   ii. Streamlit Additional Features (includes info on theme tweaks, etc).
   iii. Streamlit API Reference (includes all the various widgets available and functionality for app logic)
   iv. Check out the App Gallery for inspiration.

Phase 3 Deliverables:

1. Project Repo (make sure it is **public**)
   a. Update the ReadME file to describe your project, any information the user needs to build/start the containers (such as adding the secrets passwords files), etc. Include each team member's name somewhere in the README.
   b. When updating the README, use Github Markdown to add formatting (headings, bold, etc.).
   c. We should be able to create a valid **`.env`** file, and run **`docker compose up -d`** to start all services, sql files should be executed, etc.
2. A 6-8 minute Pitch and Demo video recorded by your team about your project. Take note of the following:
   a. General:
      i. 6 - 8 minutes is a hard min and max. 10 points will be deducted for every minute over or under the limits.
      ii. There's a lot of technical material to cover in 6 - 8 minutes. It will require your team to coordinate, plan, practice, edit, and practice again.
      iii. Every team member must appear in and meaningfully contribute to the video with their camera on for the entire presentation. You don't all have to be on the same camera. The Pitch and Demo video is in place of an in-person presentation. Each team member that does not appear in the video with video on will lose 15 points on their phase 3 score (points will be deducted on an individual basis, not for the entire team).
   b. How to record:
      i. You can use Zoom or Teams to record your demo. Other options are fine as well.
      ii. All team members must appear in the video and meaningfully contribute to it.
      iii. Do not record each member's part separately and then splice them together.
      iv. Remember that if you're including code or diagrams, make sure the font size is readable in the video.
   c. What to include:
      i. Brief intro of team members (you really only need to give your names)
      ii. Your product's "Elevator Pitch" (30 - 45 seconds)
      iii. **Quick** review of the routes (code) you've created and how you've organized them in blueprints. (2 mins max). *You don't need to go line-by-line*. Go through high level organization, point out the different types of routes (get, post, put, delete, etc)
      iv. Demo of the front-end Streamlit application your team has created. (~ 5 mins)
         1. Be sure to show that for any POST/PUT/DELETE routes, the database reflects the results of the operation by looking at the data in DataGrip or another tool.

You can host the video anywhere you'd like (YouTube, Dropbox, etc). Please put a link to the video in your Project Repo's README.md file and your Phase 3 Submission doc. It is **vitally important** that your video be **publicly available** to anyone with the link. If we have to "request permission" to view the demo video, you risk getting no credit for this part of the

project.

Officially Due: Tuesday April 15, 2025 by 11:59 pm EST. However, I will accept submissions until Thursday April 17 at 12-noon EST with no late penalty.