Team Information

Github Repo Link: https://github.com/UT-SWLab/TeamE3

Team Members:

- Harrison Berrier
  - EID: hlb962
  - Email: harrison.berrier@utexas.edu
  - Github username: harrisonberrier
  - Estimated completion time for each member: 8.5
  - Actual completion time for each member: 11
- Mohit Gupta
  - EID: mg58629
  - Email: mohit.gupta@utexas.edu
  - Github username: mohitg17
  - Estimated completion time for each member: 12
  - Actual completion time for each member: 15
- Nikhil Jalla
  - EID: nj5473
  - Email: nikhiljalla17@utexas.edu
  - Github username: nikhiljalla17
  - Estimated completion time for each member: 11
  - Actual completion time for each member: 12
- Silas Strawn
  - EID:
  - Email: strawnsc@gmail.com
  - Github username: StrawnSC
  - Estimated completion time: 15
  - Actual completion time: 15

Canvas Group: E3

Project Name: US Colleges & Universities Internet Database

Motivation and Users

We envision our site being used by prospective students to assess which college or university is right for them, as well as which major or field of study they should pursue. Our database lets students view universities in the US filtered by their city, or by the majors they offer. For example, if a student knows they are only interested in schools that offer a particular program, they can go to the major page for that field of study, and browse the colleges that offer that major. On the other hand, if the student is interested in colleges only in a particular location, they can pull up all the colleges for a particular city. Once they're on a university or college's

page, the prospective student will be able to see crucial information for making their choice. For instance, they will see facts on the cost of attendance, the acceptance rate, average SAT scores, the size of the school, etc. If they want to learn more, they can also follow a link to the institution's web page.

Stories:

Phase i:

- As a high school student, I want to be able to look through a list of universities in the United States so that I can decide where I want to go to college.
    - Estimate: 2.5
    - Actual: 2.5
- As a high school student that wants to move away from home, I want to be able to look at university education statistics for different cities so that I can decide where I want to move.
    - Estimate: 2
    - Actual: 3
- As a high school student that hasn't decided what I want to major in, I want to be able to look through a list of different majors and their associated statistics so that I can decide what I want to major in.
    - Estimate: 2.5
    - Actual: 3
- As a user, I want to read about the website that I am using so that I know where my information is coming from and what the creator's motivation is.
    - Estimate: 2
    - Actual: 3
- As a user that isn't very good with technology, I want the website's interface to be simple and navigation to be easy so that I can access the information that I want quickly and easily.
    - Estimate: 2.5
    - Actual: 3

Phase ii:

- As a user with limited time and a general idea of what I'm looking for in a university, I want to be able to view some attributes of a university without having to navigate to that university's instance page.
    - Estimate: 4
    - Actual: 4
- As a high-school student, I want to be able to view a list of all colleges offering a 4-year degree in the US so that I don't have to go elsewhere to find statistics on certain schools.
    - Estimate: 4
    - Actual: 3.5
- As a user with limited time, I want to be able to navigate by more than one page at a time so that I don't have to waste time clicking through pages of instances.
    - Estimate: 3

    - Actual: 2.5
- As a user looking through a long list of cities, universities, and majors, I want the list to load into pages rather than one large screen so that it makes it easier to navigate.
    - Estimate: 2.5
    - Actual: 3
- As a user that is scrolling quickly through a long list of universities, I want each university to be listed with a picture to help me sort quickly through them.
    - Estimate: 3
    - Actual: 5
- As a user that is scrolling quickly through a lot of cities, I want each city to be listed with an image to help me sort quickly through them.
    - Estimate: 3
    - Actual: 5
- As a user that is scrolling quickly through a long list of majors, I want each major to be listed with an image to help me sort through them quickly.
    - Estimate: 3
    - Actual: 2.5

Testing

GUI Testing

Our GUI testing uses the Selenium IDE to reduce the potential points of error. All we have to test in our frontend right now is a collection of links. We have not yet added any text entry fields or extra functionality, but we will implement tests for those when they are added. We test all of the links on the navbar by clicking them and asserting the title of the page. The model page links are then tested in the same way. Each page's pagination is tested as well. The test makes sure that both navigation arrows work as well as the early and late page navigation. To test the page navigation, the script checks the text of the active tag, confirming that the page changed. It also checks to make sure that the first page and last page elements exist, regardless of which page is active.

Unit Testing

We used python unittest to test our database queries and data processing. The database test queries the database for university, major, and city objects, and compares the returned objects with the expected object. The objects are queried using varying combinations of their unique fields to ensure that all combinations work. The data processing test constructs data dictionaries that are built in the main file and sent to the model pages. The dictionary that is built from sample instance data is compared to the expected data in the dictionary. These tests together validate that the database is returning the expected objects when queried, and that the expected data is being sent to the model and instance pages.

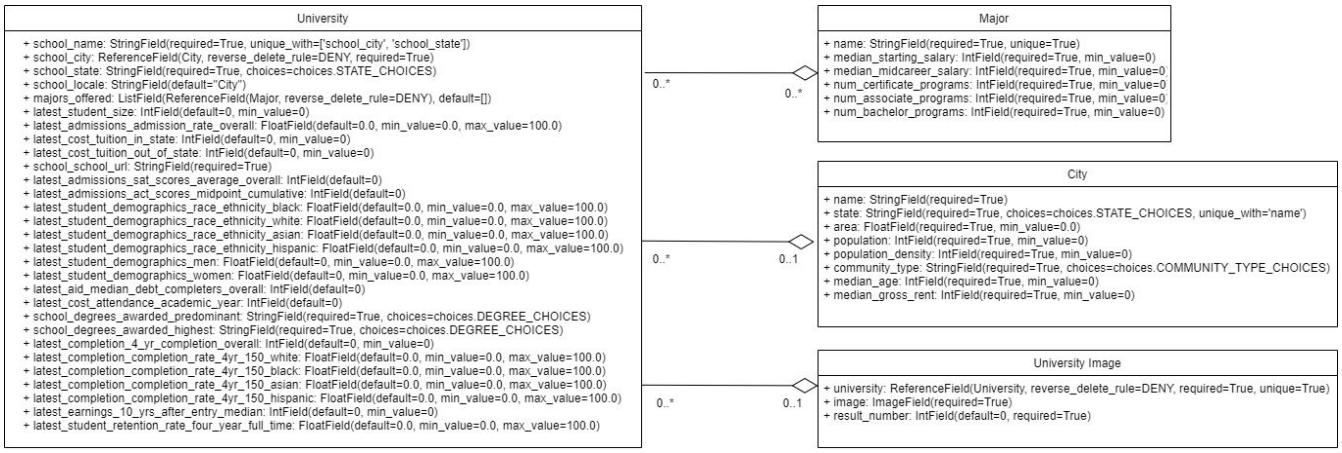
Design

Stack
We are using MongoDB, Flask, Jinja, and Bootstrap.

Pages
Every page inherits the base.html file, which includes the navbar. Each of the model base pages uses the model.html. Each instance uses the [model]_instance.html file. The landing page (index.html) is a central page where users can navigate to any of the model's base page. It contains a carousel and pictures that link to the models. The about page (about.html) contains information about the site and the developers.

Flask
The home page is at the root path ('/'). Each model is at the path './[model]'. Each instance is at the path '/[model]/[instancename]'.

Database UML



Models

University Model
Each university instance contains relevant statistics about the university along with its location and the majors that it offers. The page has an overview at the top that includes the school size, acceptance rate, and in-state tuition. Below that, there are sections for school info, majors, admissions, demographics, and tuition and aid. The left panel has links that navigate to each section. The right panel includes extra data about degrees awarded, completion rate, average earnings for graduates, and overall retention rate. All of the data is pulled from the CollegeScorecard API.

We would like to present some of the data graphically to make the page more visually appealing. Demographics data can be easily represented in a pie chart. More data about admissions and completion rates can also be displayed in intuitive ways. We can also make the pages more interactive for the user by adding more responsive elements such as collapsibles and buttons. We also need to link the university instances to the city and major instances.

City Model
Each city instance is described by statistics that would be relevant to students who would like to attend a school in the respective city. Currently, the page lists the area of the city, population, population density, community type, median gross rent, and of course the schools that are in the city. This data was acquired from this website - http://www.city-data.com.

The next step for this page would be to find an api with more relevant information and a better way to present the information. The core point of this page should be the information about the city, not the schools that can be found in the city.

Major Model
Each major instance page contains a list of universities that offer that major, as well as some aggregated statistics on the number of programs in the US for that field of study. We currently list the number of programs that offer a bachelor's degree, an associate's degree, and certificate (after less than one year of study) in the given major. All of this data was acquired from the Department of Education's College Scorecard API. Finally, we list the median starting salary and median mid-career salary for graduates of that major. Since the Department of Education didn't provide this information, we had to acquire it from an article posted on "Visual Capitalist."

Ideally, we would like to sort the universities in each major instance's list by the ranking for that particular major, but we have not found open datasets with such information. On the other hand, we will be able to order the Universities/Colleges by percentage of degrees awarded for a given major. However, since the site content is currently static, and we don't have many University pages up yet, this was not implemented in Phase I. In future phases, we may also consider augmenting the major data with other datasets because the Department of Education data is occasionally overly-broad. For example, all engineering majors are grouped together as one field of study.

Tools, Software, and Frameworks

Flask
Our backend framework is flask, a microservices framework for python. Flask is useful in that it is lightweight: it was trivially simple to set up the server and start running it locally. When we want to add new pages, we simply add a route for the new page. We also took advantage of Flask's template mechanics. Using flask templates, we dynamically generate instance pages for our major, university, and city models. This allows us to reuse the same HTML, instead of adding hundreds of nearly identical HTML files for each university.

Bootstrap
Our frontend framework is bootstrap, which we use alongside HTML and CSS to design our web pages. Bootstrap has been crucial for allowing our pages to be reactive. The pages in our site dynamically resize based on the size of the viewport, allowing our format to remain robust and look clean, despite the window size limitations of the user.

Google Cloud Platform
Our site is deployed to Google Cloud Platform. We used GCP because it is simple to set up and use with flask applications. All we have to do to update the deployment of our application is run the shell command “gcloud app deploy” in the /idb_app/ directory, using the gcloud CLI.

MongoDB and Mongoengine
All of our data is dynamically pulled from MongoDB, including the images for every model instance. Our models are defined with mongoengine, an open-source Python Object-Document Mapper. Mongoengine will allow us to define “schemas” (mongoDB does not technically have schemas since it’s NoSQL) with python objects, so we can interact with our database purely in terms of University, Major, and City python objects that we design. Mongoengine’s ODM capability is useful for not just defining the schema for our model’s fields, but also enforcing constraints on certain fields, such as uniqueness constraints. For instance, mongoengine forces our model instances to have university names to be unique with respect to the university’s city and state.

Sources

Flask documentation: https://flask.palletsprojects.com/en/1.1.x/quickstart/
Bootstrap reference guide: https://www.w3schools.com/bootstrap4/default.asp
Footer guide: https://www.w3schools.com/howto/howto_css_fixed_footer.asp
CSS variables: https://www.w3schools.com/css/css3_variables.asp
Major salary data: https://www.visualcapitalist.com/visualizing-salaries-college-degrees/
Google Maps API documentation:
https://developers.google.com/maps/documentation/maps-static/usage-and-billing

Reflection - Phase I

What we learned:

We’ve learned how to work together and divide work evenly among us. We were able to distribute tasks early on and each person completed their parts when convenient for them. We also learned how to build a website from ground up using html, css, bootstrap, and flask. Through designing the site, we’ve learned more about bootstrap and how to style components. We’ve also learned how to use the APIs selected to acquire the data that we need.

Five things that went well:

1. We quickly moved to the use of templates so that multiple pages can be changed quickly.
2. We have been able to make decisions quickly and then adapt as problems arise, given our heavy usage of slack to coordinate and communicate.
3. Our API's are well documented, making them easier to interact with. In particular, the Department of Education's College Scorecard API has extensive higher education data, and provides a spreadsheet explaining all the possible fields one can request from the API.
4. Since flask is a lightweight backend, we were able to get started putting pages together and making routes for them easily.
5. Flask also allows us to pass in parameters to the html, and we used this to dynamically generate different instance pages off of the same base html. Although the data is currently hard-coded in the python flask app, we will be able to easily migrate this over to database queries without having to change the frontend code.

Five things that went poorly:

1. We haven't been able to add a stylesheet other than bootstrap so we are stuck writing styling into each file at the moment.
2. Formatting pages to fit requirements has been very difficult versus formatting a page without specifications because of the way that html elements and they're styling interact with each other.
3. At first, the college scorecard API was overwhelming because of the sheer amount of data and the complexity of crafting queries to get the needed information.
4. Often, the styling of elements with HTML, CSS, and Bootstrap does not work as we would expect. For example, when we were making the about page, we had to put in elements with our photos and bios together. For some reason, adding more text to the paragraph element next to an image would increase the image's size. We ended up using different bootstrap elements to get around this.
5. When we were first setting up the deployment on Google Cloud Platform, the deploy would seem to work in the CLI, but when we tried to visit the site, we would get a 500 error. Apparently, if you don't have the exact right directory structure and naming of certain modules, GCP doesn't know how to deploy your flask app.

Reflection - Phase II

What we learned:

We learned how to set up, access, update, and query a MongoDB database. To set up the database, we learned how to create collections and set up data models. We learned how to use

a python object data mapper, mongoengine, to manipulate data in the database. To populate the database, we learned how to write ingestion scripts that call APIs to collect data and store them in the database. We also learned how to test both our frontend and backend code using Selenium and unittest respectively. To collect images, we learned how to use existing APIs to automatically retrieve relevant images. We learned how to use pagination to limit the number of instances that are loaded at a time.

Five things that went well:

6. We started testing both our frontend and backend early on, which helped us resolve bugs much quicker than before. We used Selenium to confirm our frontend functionality and we used unittest to validate our database queries and data manipulation.
7. We were able to easily communicate with the database using python objects through mongoengine. The object-based model allowed for logical data access and manipulation. It also allowed for greater flexibility in the structure of the data.
8. Since we used templates for phase I, we were able to transition from hard-coded data to dynamically accessed data very easily. Our data requires very little backend processing once it is received from the database, so large amounts of data can be aesthetically displayed in the model and instance pages. Data access is also very fast.
9. We wrote ingestion scripts to consume data from our APIs and populate our database very efficiently. After running our ingestion script initially, we were able to easily adjust the data and fields as needed.
10. We were able to divide responsibilities efficiently so that each person's work was not entirely dependent on another person's work. This allowed us to work on our own schedules and make good progress on all components of the project simultaneously.

Five things that went poorly:

6. Currently, all of the instances are loaded upfront when a model page is accessed, so there is a short loading time on the page. We will need to dynamically load instances for a given page in order to decrease loading time.
7. We used a python package that uses the google images API to get images for our instances. While it mostly worked very well, we need to go through and replace images that are not accurate.
8. We needed to adjust our data models slightly after we had already run our data ingestion scripts, so we had to re-run some of those scripts with the updated data model. This was time consuming since we had to query an API for some of the new data.
9. After adding the database code, we weren't able to deploy our app to Google in its existing form. We had to make changes to the deployment process to make it possible to deploy to App Engine.
10. Some of the APIs we had planned to use were hard to work with so we had to find alternative APIs with more easily accessible data.