Sprint 2 Backend Documentation

Web Scraping Functions

Scrape.go file

func miamiDade(cats []cat) []cat{}

This function goes into Miami-Dade Animal Services and collects all the cats in their database. It uses the colly external library to extract the HTML tags and information with the corresponding detail of the cat object requested. The parameter for this function takes in a list of cat objects and returns that list filled with the cats in the Miami-Dade Animal Services database. This will then be used to display all the cats on our website.

func lakeCounty(cats []cat) []cat{}

This function goes into the Lake County Animal Shelter and extracts all the cats they have displayed in their database. It takes in a list of object cat which includes Name, Gender, Breed, Age, and ImageURL. The colly external library allows the extraction of HTML tags to assign the data to the certain cat variable details for the cat object. Then it returns a cat list with all the cats and their properties from the Lake County Animal Shelter database.

func peggy(cats []cat) []cat{}

This function has the same functionality as miamiDade(cats []cat) and lakeCounty(cats []cat) but it accesses the information from Peggy Adams Animal Shelter in West Palm Beach, FL. It takes in the same parameter as the other two functions above and returns the cat object list with the cats from that shelter.

func keyWest(cats []cat) []cat{}

This function serves the same functionality as the above three functions. However, it extracts cat information from the Key West Animal Shelter. It has the same parameters and the same return type as the above functions which then assigns all the cat properties of the cat object in the cat list which will then be used on the website to display the cats for the user.

func marathon(cats []cat) []cat{}

This function has the same functionality as the above functions, but it extracts the cat information from the Marathon Animal Shelter. Using the colly external library, it is able to extract the HTML tag information that corresponds to the cat properties of the cat object that goes in the cat struct.

API Backend

*func getCats(w http.ResponseWriter, r *http.Request)*

The getCats function writes a method header of statusOK when the endpoint has been successfully reached and helps convert all of the list of cats into a JSON struct to be sent to the front end. The getCats function uses the GET method to portray all the cats on the screen.

*func getCat(w http.ResponseWriter, r *http.Request)*

This function is only slightly different from getCats in that it still uses the GET method, but it only returns a singular cat object after specifying the id of the cat. It also writes the method header of statusOK when the endpoint has been reached.

*func deleteCat(w http.ResponseWriter, r *http.Request)*

The deleteCats function writes a method header of statusNoContent when the endpoint has been successfully reached and finds the cat object in question by looking at its id and deleting it from the slice of cat objects. Afterwards it then encodes it as JSON to send the frontend a new updated list of cats. The deleteCats function uses the DELETE method to help get rid of the cat in question.

*func updateCat(w http.ResponseWriter, r *http.Request)*

The updateCats function writes a method header of statusOK when the endpoint has been successfully reached and finds the cat object by looking at its id and updating its content. After decoding the JSON and updating its content. It will put the cat object back into the list and then encode it to send to the frontend. The updateCat function uses the PUT method to help channel this process of updating the cat to the frontend.

*func createCat(w http.ResponseWriter, r *http.Request)*

The createCat function writes a method header of statusOK when the endpoint has been successfully reached and creates a new cat object to display to the frontend. It does this by decoding the information from the JSON and then writing it back to show to the frontend.

Backend Work completed for Sprint 2

For backend, we were able to web scrape across five Animal shelters in Florida to be able to get our cat object list which we will be displaying on our website. This was a difficult task since many animal shelter websites had HTML tag names that were not correctly read by the functions we were writing to extract the information. However, we were able to get enough cat information and for some cat properties that could not be extracted, we just gave it the “In

Website” property that the user can later go and check when they select the cat they want. We were also able to fully connect the backend and frontend together to display the webpage on our local machine. This task wasn’t necessarily difficult but the main issue was finding the correct folder to help display the information. In addition to hooking up the backend and frontend, we were able to create five unit tests to help verifying the routing process and to make sure that each endpoint is working by creating a dummy router to validate the endpoints and by putting in example cat JSON structs.

Sprint 2 Frontend Documentation and Work Completed

Services functionality

This folder in the front end branch serves to establish routing from the pages into a formatted layout the conveniently displays information such as an image of the cat (so cat images are place holders while we attempt to fix a recent issue with the frontend), the shelter that they are located in, and their price of adoption.

AdoptionPage page

This newly added page for the front end branch is a page that inherits from the shelters component page with ngOnInit as this would relay data to the secondary page and the routing would be shared between the pages so that navigation would not have any collisions just as we have done with the navigation bar, footer, and pages.

Unit and End-to-End testing with Cypress

Instead of performing our unit tests with Jasmine (the default for Angular), we decided to perform both our unit and end-to-end testing with cypress to provide both convenience for developers as well as reducing any collision with tests switching between Jasmine and Cypress, therefore we decide to stick only with Cypress and utilize its end-to-end and component testing. The end-to-end testing cases focused on the functionality of the routing between components such as navbar and its pages (home, about, contact, etc…), while the component testing focused mainly on buttons and form filling as mentioned in the Sprint 2 description.

Last Minute Issues

In a recent pull from github, a section of our components began having issues that we recently havent had as when attempting to compile using “ng serve” in terminal, compilation fails due to components being “The class 'AppComponent' is listed in the declarations of the NgModule 'AppModule', but is not a directive, a component, or a pipe.” however, this error only appeared recently despite this format being in our sprint 2 submission. However, our frontend team believes this could be an issue relating to possible absolute path and relative path collision and we strongly believe we can have this resolved for sprint 3, but since we can’t have the frontend

webpage running on a local host for unit and end to end testing, all the tests will fail do to the compilation error as cypress won't be able to access any components on the website.

Cypress Issues

As explained in the situation mentioned above, we were able to install and create both unit and end-to-end tests with cypress but due to the website not being able to compile, the tests are unable to pass due to not being able to access specific components and end-to-end tests on the website. As mentioned before we are certain of the source of the error, but the timing of when this issue appeared was unfavorable and we can ensure the frontend to be fully functional once again just like in sprint 1 but with integrated backend functionality.

Backend Unit Tests

Backend Unit tests

*func TestInWebsiteMiamiDade(t *testing.T)*
*func TestInWebsiteLakeCounty(t *testing.T)*
*func TestInWebsitePeggy(t *testing.T)*
*func TestInWebsiteKeyWest(t *testing.T)*
*func TestInWebsiteMarathon(t *testing.T)*

The above five unit test functions test if the correct cats are being updated into the cat list. We created separate functions that convert the cat list into a string to be able to compare them in the unit tests. However, this can change because the shelters update their cats every day.

*func TestInWebMarathon(t *testing.T)*

This unit test function checks if the Age property of the cat object in Marathon Animal Shelter returns "In Website" since the HTML tags do not return the age of the cat for this shelter.

*func TestInWebLakeCountyBreed(t *testing.T)*

This unit test function checks if the Breed property of the cat object for Lake County Animal Shelter returns "In Website". This test is needed because the Lake County Animal Shelter does not display the breed of the animal on the website.

*func TestGetCats(t *testing.T)*

This unit test creates a router and checks for a 200 ok status to see if it pings to the correct routing of getCats. The test also checks for an expected body response of getting all the cats.

*func TestGetCat(t *testing.T)*

This unit test creates a router and checks for a 200 ok status to see if it pings to the correct routing of getCat. The test checks for an expected body response of a singular cat.

*func TestUpdateCat(t *testing.T)*

This unit test creates a router to help test for the updateCat method and compares if it connects by testing the connection status. The test then checks for an updated cat JSON struct.

*func TestCreateCat(t *testing.T)*

This unit test creates a router to help test for the createCat method and compares if it connects by testing the connection status. The test then checks to see if the cat struct has been created.

*func TestDeleteCat(t *testing.T)*

The unit test creates a router to help test for the deleteCat method and compares if it connects by checking the connection status, the status should be NoContent. If the correct status shows up, then the test has passed.

Backend Unit Tests

Cypress Component Test - Home

This unit test for a valid routing when the user clicks on the “Home” button on the navigation bar on the webpage.

Cypress Component Test - Shelters

This unit test for a valid routing when the user clicks on the “Shelters” button on the navigation bar on the webpage.

Cypress Component Test - Contact

This unit test for a valid routing when the user clicks on the “Contact” button on the navigation bar on the webpage.

Cypress Component Test - About

This unit test for a valid routing when the user clicks on the “About” button on the navigation bar on the webpage.

Cypress End-to-End Testing

This unit test for a valid routing when the user clicks on the MAIN Page of website as well as other default end to end testing provided by Cypress.