



ACS5423: Software Development for Web – Project Phase 2

Dewayne Hafenstein

University of Oklahoma – College of Engineering

Table of Contents

Abstract.....	3
Design.....	4
Use Cases	5
Data Model	8
Wireframes (UI)	9
Structure	12
GitHub Repository.....	13
CI/CD Pipeline	13

Abstract

The project for ACS5423 was to develop a web application using the MEAN stack (Mongo, Express, Angular, and Node.js) technologies to explore an extract of the FDA's (Food and Drug Administration) branded food database. This database extract was loaded into a Mongo database cluster and accessed by a Node.JS server to support a browser-based UI, all developed using node and html.

In this phase, the use of GitHub actions to implement a CI/CD pipeline have been added, the application has been deployed to Heroku, and some feedback and planned upgrades applied to the code.

Key links and references:

GitHub repository: <https://github.com/dewayneh57/ACS5423>

Application Web Site: <https://hidden-wave-57588-cb129b54c25c.herokuapp.com/>

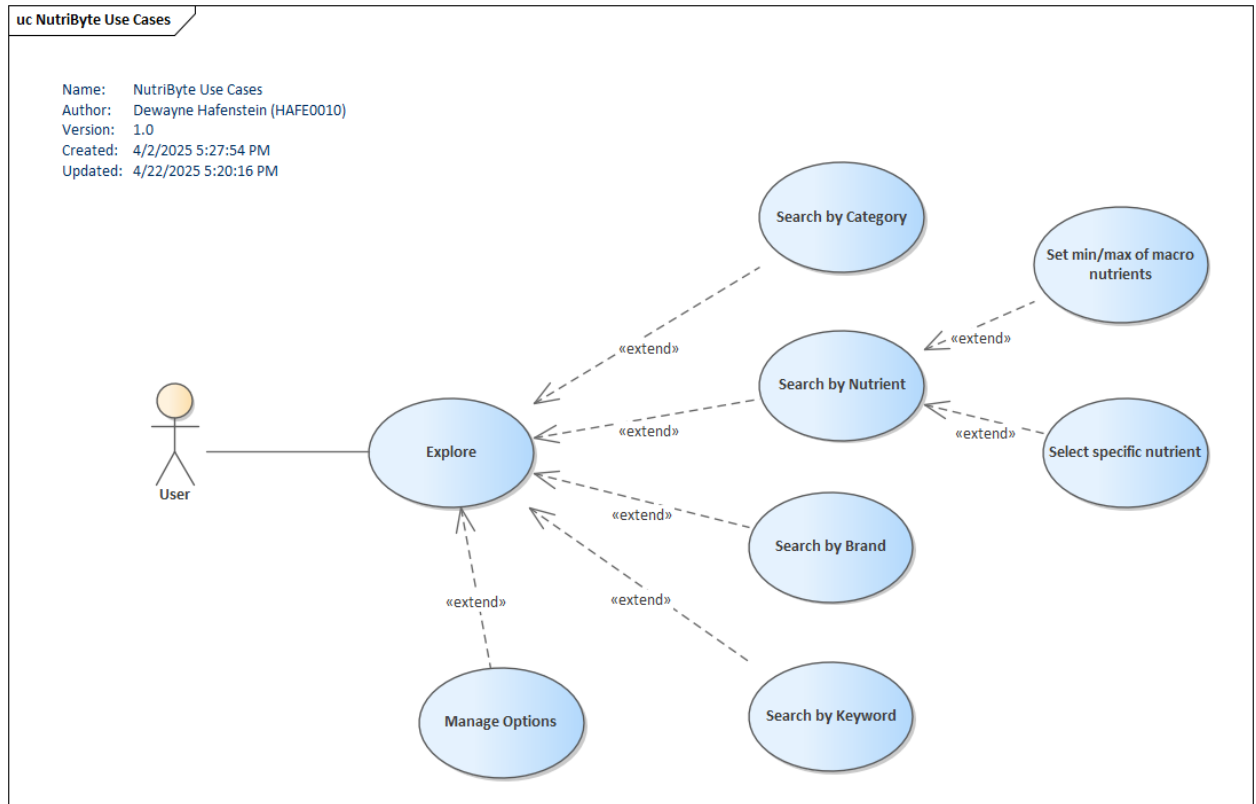
ACS5423 – Project Phase 2

This report details the design, structure, and implementation details of the project. This phase introduced the complete CI/CD pipelines using GitHub actions, automated deployment to Heroku, and updates to the functionality of the application.

Design

Design was performed using a professional architecture design tool which I happen to have a license for, named **Enterprise Architect** by Sparx Systems, Ltd. This tool is used heavily at AT&T and I am very familiar with its use. I did not, however, build out all artifacts that I would normally do in a project for work. Instead, I focused on the use cases, the data model, and the UI wire frames. A detailed report from the tool is provided and is titled “**NutriByte Design Report.docx**”. This report is generated from the tool and includes all the diagrams as well as comments, descriptions, and linkages between the various design artifacts. I have extracted those diagrams and inserted them into this document as well.

Use Cases



The use cases were designed to provide 4 different ways to query and obtain information from the database, as well as to set behavioral options and limits. These allow a user to:

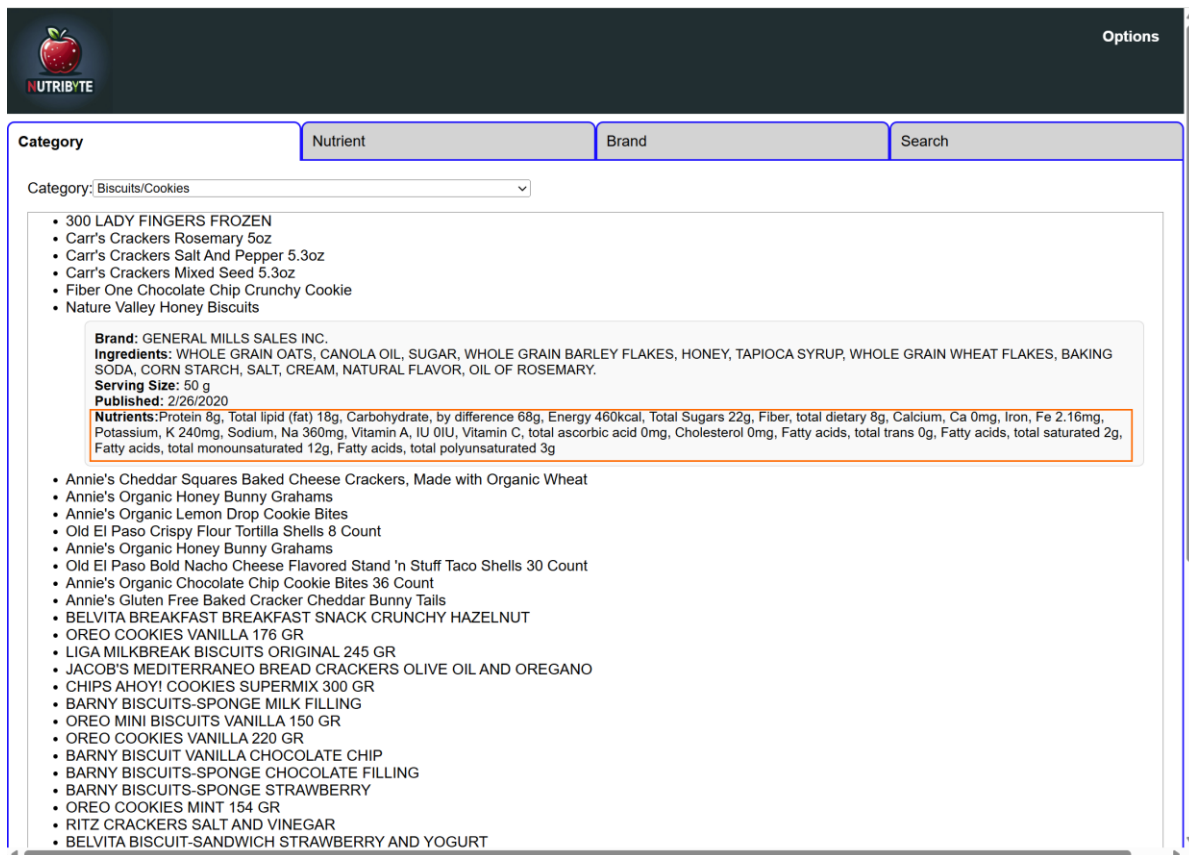
1. Search the database by the categorization of branded food. This returns a list of all foods that are categorized with the selected category. The categories are obtained from the database and used to fill a select list so the user can just pick one that already exists and need not type anything in.
2. Search by nutrient. This allows the user to search the database for all foods that contain specified nutrients. Again, the selection list is pre-populated with the list of all nutrients that exist in the database. This feature was enhanced to allow the user to alternately specify one or more macro nutrients, and to set minimum and maximum values, or a range, to match foods within those ranges of those nutrients. For example, the user can select to

- include carbohydrates between 10 and 200, in which case foods with carbohydrates within that range are returned.
3. Search by brand. This view allows the user to search the database for all foods that are produced by a specific brand, or marketed under the desired brand. Again, the list of all brands that exist in the database is used to pre-populate the selection list. This list is very large, and takes a few seconds to draw the selection list. However, the list is sorted, and the user can scroll to the brand by pressing the letter corresponding to the brands first letter to rapidly get to the appropriate grouping.
 4. Finally, the ability to search the foods using a keyword search. This searches not only the ingredients, but also the nutrients, brands, and descriptions of all branded foods and returns all foods that contain the specified keyword in any of those fields.
 5. The user can now also control the limits for searches as well as case sensitivity for keyword searches. If not specified, the default is a limit of 50 foods and no case sensitivity.

Once the list of foods has been returned, the user can select any of the foods to obtain detailed information about that food. For example, if the user selects the view by category (tab), and selects the category “Biscuits/Cookies”, the following (partial) display results. Scroll bars are used to allow the user to scroll through the contents. If the user selects any food, a detail area expands under the food to show the detailed information about that food.

This behavior has been enhanced to also return the nutrients for the food, in the exception of the nutrient search tab. When the nutrient search tab is used, only the nutrient(s) that are being requested in the search are shown.

More than one food may be expanded to see the detail and to compare different foods if desired. A screen shot of the application is shown below:

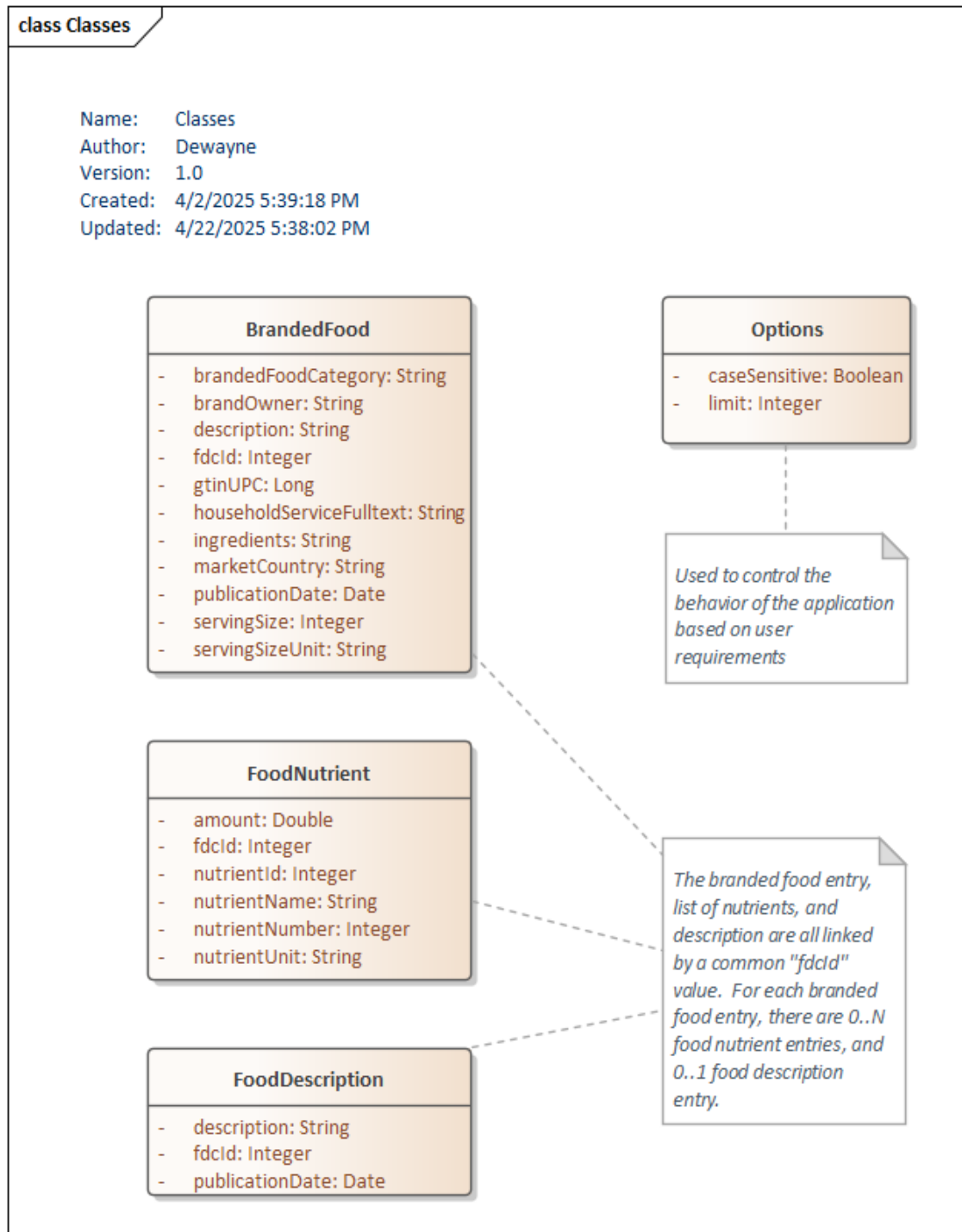


The areas of the detailed display which were enhanced are shown inside a red rectangle in the above diagram. Note, this rectangle DOES NOT appear in the UI, but is used to highlight the areas that have changed.

Data Model

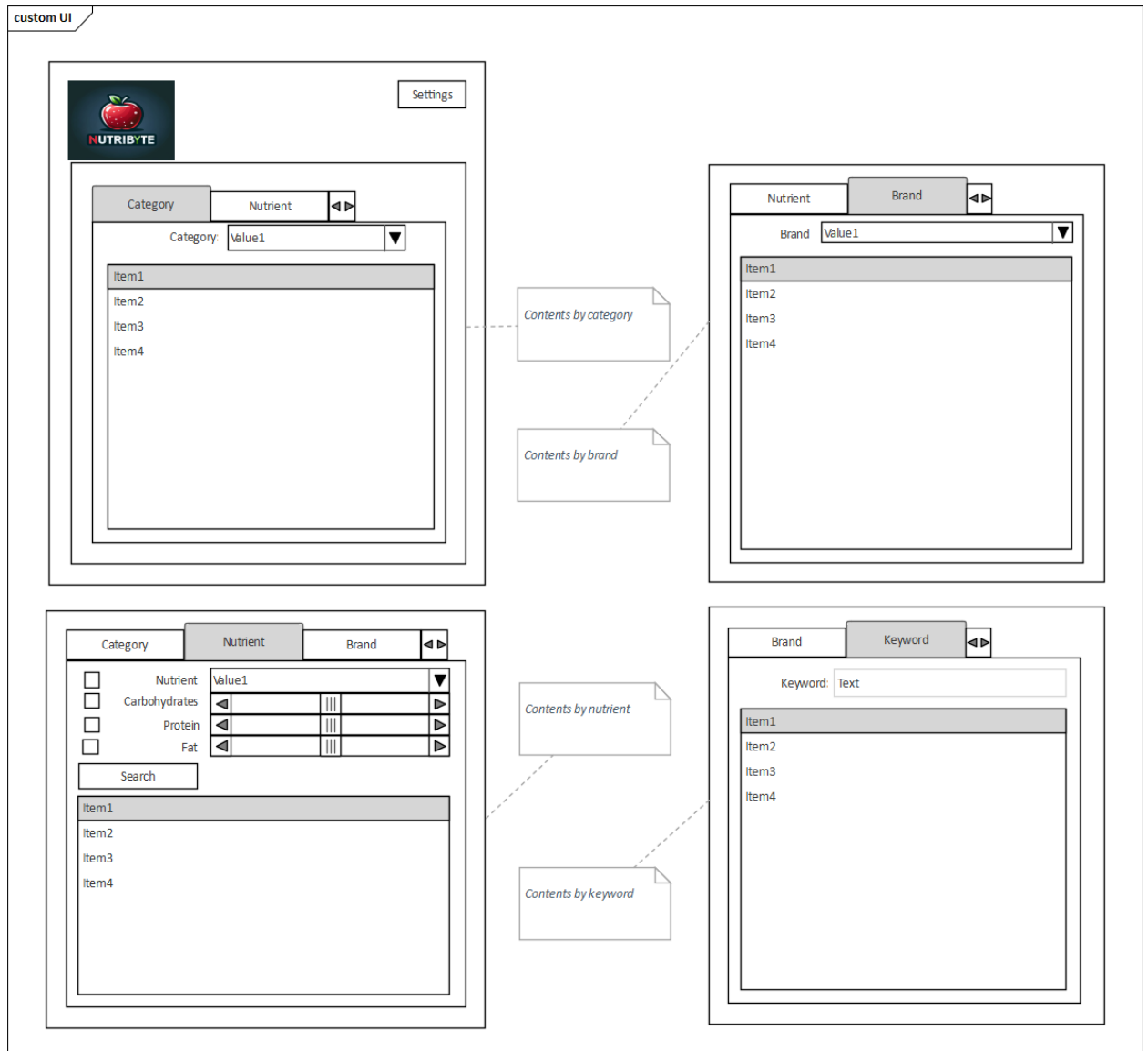
The data model is derived strictly from the FDA branded food database documents. There is one “class” defined for each document structure, and this is represented in the node.js project as schema definitions used by Mongoose.

In addition to the USDA data, an additional document is defined to track the operational settings for the product. These settings include a search limit (default is 50 if not set) and if keyword searches are case sensitive or not (default is not case sensitive).



Wireframes (UI)

The UI/UX design was started from wireframes that were created to meet the needs of the use cases. These wireframes are just mockups of the general idea behind the user interactions.









The nutrient search wireframe shows an enhancement that allows a user to not only specify a specific nutrient, but also a min/max range of one to three macro-nutrients. These macro-nutrients are carbohydrates, proteins, and fats. The min/max slider is a custom control that allows the user to either enter the min and max values as numeric quantities, or to use slider "thumbs" to set the minimum and maximum values.

An example of using nutrient sliders is shown below.

Filter Nutrient Search

Select a specific nutrient from the selection list and/or a range of the desired macro nutrients. Click the check box to include the macro nutrients in the search.

Nutrient:












<input checked="" type="checkbox"/>	Carbohydrates:	<input type="text" value="50"/>			<input type="text" value="250"/>
<input checked="" type="checkbox"/>	Protein:	<input type="text" value="10"/>			<input type="text" value="50"/>
<input checked="" type="checkbox"/>	Fat:	<input type="text" value="20"/>			<input type="text" value="75"/>

The minimum value is shown as a green “thumb” and the maximum value as a red “thumb”.


















The numeric value for the range is shown in the text input boxes on the left (minimum) and right (maximum) ends of the sliders. To include the nutrient in the search, select the check box to the left edge of the specific macro-nutrient(s) desired.

Structure

The project is structured using a hierarchical organization. This structure is defined as follows:

 nutribyte	/models contains the schema definitions .
 .git	/modules contains common server-side functions.
 models	/public contains all public resources used by the browser (client-side).
 modules	/public/css contains the cascading style sheet.
 node_modules	/public/images contains the images used on the web page.
 public	/public/scripts contains the scripts downloaded to the browser and used to define all dynamic behavior in the browser.
 css	/routes contains the definition of all server-side APIs used by the client-side javascript functions.
 images	/views contains all HTML files used to create the user interface presentation.
 scripts	
 routes	
 views	

The root of the project contains all directories, as well as the README.md file, server javascript file, package.json, and the design model and documentation. This is shown in the following image:

Name	
 .git	
 models	
 modules	
 node_modules	
 public	
 routes	
 views	
 .env	
 .gitignore	
 ACS5423 Project Report - Phase 1.docx	ACS5423 Project Report – Phase 1: this report document.
 ACS5423 Project Report - Phase 1.pdf	NutriByte Design Report: The design documentation produced by the Enterprise Architect product from the model.
 App.js	NutriByte Design.eapx : The Enterprise Architect design model itself. This file can only be processed by the Enterprise Architect product.
 NutriByte Design Report.docx	package.json: the project package file defining the required modules and structure of the node.js project.
 NutriByte Design.eapx	README.md: The read me markdown file that describes the project and which will appear in the GitHub repository main page.
 package.json	App.js: the main entry point for the Node.js project.
 package-lock.json	
 README.md	

GitHub Repository

All source code and design artifacts are committed to a personal public repository on GitHub named **ACS5423**. The URL to access the repository is: <https://github.com/dewayneh57/ACS5423>

CI/CD Pipeline

The actions are defined as CI (Continuous Integration) action and CD (Continuous Deployment) actions. The CI action is applied after a push to any branch in the repository. This action is defined below:

```
name: CI

on:
  push:
    branches: [ ** ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '22'

      - name: Install dependencies if package.json exists
        run: |
          if [ -f package.json ]; then
            npm ci
          fi

      - name: Run tests
        run: |
          if [ -f package.json ]; then
            npm test
          fi
```

This action is applied on a push to any branch. This means the CI actions are performed for developer branches as well as the main branch. The action performs a checkout of the code from the

GitHub repository, sets up Node.js version 22 on the runner, installs all needed dependencies for the application, then runs the test cases defined.

The continuous deployment action applies only to pushes or pull requests are applied to the main branch. The content of the GitHub action for CD is shown below:

```
name: CD

on:
  push:
    branches:
      - main

  pull_request:
    branches:
      - main
    types:
      - closed

jobs:
  build-and-deploy:
    if: github.event.pull_request.merged == true || github.event_name == 'push'
    runs-on: ubuntu-latest
    steps:
      - name: Checkout source code
        uses: actions/checkout@v4
        with:
          fetch-depth: 0

      - name: Set up Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '22'

      - name: Install dependencies
        run: npm install

      - name: Run tests
        run: npm test

      - name: Deploy to Heroku
        env:
          HEROKU_API_KEY: ${ secrets.HEROKU_API_KEY }
        run: |
          curl https://cli-assets.heroku.com/install.sh | sh
          heroku auth:token
          heroku git:remote -a ${ secrets.HEROKU_APP_NAME }
```

```
git push https://heroku:${{ secrets.HEROKU_API_KEY
}}@git.heroku.com/${{ secrets.HEROKU_APP_NAME }}.git
HEAD:refs/heads/main -force
```

This action has several key definitions:

1. The “ON” block indicates this action is executed whenever a successful push to the “main” branch OR a merge of a closed pull request is performed to the main branch. These actions generally indicate that production-level code has been pushed to the main branch and needs to be deployed. The condition “types: closed” is required to prevent the action of creating a PR from running the action.
2. Additionally, the “if” condition further checks that the repository is in the correct state to run the CD action. That is, the main branch has successfully been pushed OR a closed PR has been merged. This test is needed to keep the CD action from running if the PR were simply to be closed. Without this test, creating and then closing a PR on the main branch would have triggered the action.
3. This action also performs a checkout of the source code from the git repo, with the additional specification of “depth: 0”. Depth of 0 indicates to GitHub to checkout the entire revision history of the repository not a shallow copy. The default is “depth: 1” which is just the latest history. Heroku will not accept a deployment without the full history included, so “depth: 0” is required.
4. This action uses GitHub secrets to connect to Heroku and to deploy the application. These include the API Key and application name. The use of secrets keeps these values secured.

To access the application on Heroku, launch the following URL in a browser:

<https://hidden-wave-57588-cb129b54c25c.herokuapp.com/>