Spring 2022

Cook-it!: A Web Application for Easy Meal Planning

Carol Juneau
University of South Carolina

COOK-IT!: A WEB APPLICATION FOR EASY MEAL PLANNING

By

Carol Juneau
Garrison Davis
Matthew Lewis
Isaac Luther
Eiman Najjar

Submitted in Partial Fulfillment
of the Requirements for
Graduation with Honors from the
South Carolina Honors College

May 2022

Approved:

Dr. Jose M. Vidal
Director of Thesis

N/A
Second Reader

Steve Lynn, Dean
For South Carolina Honors College

Thesis Summary

Cook-it! is a web application for easy meal planning. Our team designed, developed, and deployed this application as part of the Computer Science Capstone project at the University of South Carolina. Users of *Cook-it!* are able to utilize many useful features of the application, such as communicating in a forum and adding recipes to their meal plan. *Cook-it!* has been developed by our team over the course of two semesters, involving several version releases and thorough testing.

The deployed website can be found here: https://secret-fortress-80929.herokuapp.com/

A copy of the team's shared GitHub repository can be found here: https://github.com/caroljuneau/Copy-Of-Cook-it/

Table of Contents

Introduction

Abstract

Cook-it! is a web application for meal planning based on the Django framework and deployed on the Heroku platform. This application has an intuitive interface to make it easy to use. The project has been developed over two semesters, roughly separated into a planning phase and an implementation phase. *Cook-it!* incorporates a robust feature set and an attractive design. Its core purpose is to make it easy for users to plan meals, interact with other users, and keep track of user information such as a grocery list.

Development Process

This project was completed over two semesters as part of the CSCE 490 and 492 courses at the University of South Carolina. The course director was Dr. Jose M. Vidal. Our team met weekly, either in-person or over Microsoft Teams, to meet deadlines for the course milestones. These milestones may be broken up into two phases: planning and implementation. Some of the major milestones are outlined as follows.

Planning Phase

First, our team decided on an idea for the app: a web application for meal planning. As background, we wrote distinct personas for potential users of the app, as well as user stories describing features they would use. Ideas for the design, including major screens of the app, were sketched out using a chosen color scheme.

We made a thorough list of requirements of the app as if we were developing a fully featured app for a client. We researched several technologies and platforms we would use to create the app, and we practiced using GitHub to manage source control.

An architecture document was written to describe a big-picture overview of how our app's code would be organized. Another document was written to describe potential ethical, legal, and security concerns of the app.

To wrap up the first semester, we released a v0.1 "proof of concept" of the app, which was deployed to the web with a few small features implemented.

Implementation Phase

The second semester of the course involved several more releases of the app, along with careful testing. A v0.5 beta release was deployed, which implemented most of the main features of the app with no major crashes or bugs. After more work, we deployed v0.9, the release candidate in which all planned features of the app were implemented with no major crashes or bugs.

For quality assurance, another team in the course evaluated our app and let us know of any issues to fix. We also tested our app thoroughly with unit and behavioral tests.

Our team filmed a video presentation of the app, which is located on the website's home page. The home page shows off the app to the user, explaining how and why to use the app, screenshots of the app, and who created the app.

Finally, we deployed the v1.0 release, in which we finalized design, made sure all features were implemented, and fixed all known bugs.

Planning

Technologies

Back-end

GitHub [1] is an online platform for collaborative software development and version control using Git. Our GitHub repository contains all the code to run our app locally; this code has also been deployed to the web.

Python [2] is a popular high-level programming language. Python is versatile and easy to use for all levels of experience with programming. Since we worked on a team with multiple people writing code, we used the PEP 8 Style Guide [3] to keep code clean, uniform, and well-formatted.

Django [4] was used as our backend web application framework. It is a high-level framework written in Python. We chose Django because it is free and open source, easy to use, secure, and works with most major databases.

PostgreSQL [5] was used as our database system. Also known as Postgres, it is a free and open-source object-relational database management system that provides features on top of the SQL language. Postgres is a reliable and extensible system, and it can connect with many languages including Python which is widely used in this project.

Heroku [6] is a cloud platform that natively uses Postgres as its database system. Heroku is free for non-commercial apps, and it is easy to deploy web apps using Git.

The Spoonacular Recipe API [7] is an API with over 360,000 recipes in an open-source database. Its search feature allows one to search for recipes using natural language queries, as well as considering parameters such as diet or food intolerances. The API has a free plan that allows up to one request per second.

Selenium WebDriver [8] is a tool we used for behavioral testing that automatically drives a browser and performs actions as if it were a user. For example, a test can automatically open up a

browser to a local copy of *Cook-it!*, and create an account by entering mock data and submitting it.

Front-end

HTML and CSS [9] are languages used to build the layout and style of web pages. We manually wrote these files for our site's front-end.

ReportLab [10] is a free and open-source toolkit for generating PDF documents. We used ReportLab for generating a PDF of a user's weekly meal plan.

Bootstrap [11] is a free and open-source library for frontend development. We used Bootstrap to apply styles to elements of our site including buttons and the navigation bar.

Requirements

Before working on any code, our team decided upon a list of features that should be implemented for a meal-planning web application. Some additional features that could potentially be implemented but were outside of our project's scope are listed in the Future Work section.

User Accounts

Users should be able to create an account, log in to the website, and be able to edit their profile. A user's profile should include a section for allergies or intolerances.

Recipes

Users should be able to add their own recipes and search for new recipes. These recipes can be added to a user's meal plan. A user should be able to edit or delete the recipes as needed.

Forum

Users should be able to interact with other *Cook-it!* users in a public forum. Users should be able to write and delete their own posts, as well as make comments on other users' posts.

Calendar

Users should have access to a calendar to keep track of a personal meal plan. Users can add or remove recipes on different days.

Grocery and Pantry

Users should be able to create a grocery list, containing ingredients they wish to purchase. Users should be able to create a pantry list, containing ingredients they currently have along with their expiration dates.

Design

A set of design guidelines was also decided on by the team ahead of coding the application.

Navigation

Each core feature of the site described above should have its own page on the website. Each of these pages should be easily accessible via a navigation bar at the top of every screen of the app. The navigation bar should also indicate which page the user is currently on.

Color Scheme

The app should have the same color scheme all throughout. Variations of our color scheme may be used, such as using brighter or darker versions of a color when a user hovers over a button.

In our sketches we used a color scheme with red and green colors [Figure 1]. However, once we started coding some features of the app and saw how it looked, we agreed that the color scheme made the app look unprofessional and we decided to switch to another color scheme.

Figure 1. Old Color Scheme

Our new color scheme used shades of red and blue colors [Figure 2]. Elements of the website should only use these colors or brighter/darker variations of these colors.

Figure 2. New Color Scheme

Fonts

The website's logo features the title *Cook-it!* in a dark red color, using a bold serif font called Rockwell [Figure 3]. The rest of the website uses Franklin Gothic, which is a sans-serif font.

Figure 3. *Cook-it!* Logo

Pages

The application should be divided into different pages for each core feature of the app.

The recipes page should display a list of the user's recipes and buttons for adding or creating new recipes.

On the forum, posts should be displayed in reverse chronological order, with newest posts at the top and oldest posts at the bottom. A user should be able to click on a post to see its complete content and any comments from other users.

Since a user's grocery list and pantry are both lists of ingredients, the layout of those pages should be similar to each other. The pages should show a bulleted list of ingredients, each one list item having buttons to edit or delete the item.

The calendar page should allow the user to look at a monthly, weekly, or daily view. Each day on the calendar should display the user's scheduled recipes. Since boxes in a calendar typically have a fixed size, recipe titles should be cut off with ellipses when they are too long to fit.

Implementation

While our team's first semester focused on planning, the second semester focused on implementing the features and design of the app.

Features

User Accounts

Accounts in *Cook-it!* are managed by Django's user authentication system [11]. This system conveniently handles many functions related to user accounts, including authentication and authorization, permissions, cookie-based sessions, and password strength checking. While the forum page is viewable to any user, most of the application's pages require a user to be logged in.

Forum

Users can add to the forum by creating forum posts or image posts. A forum post has a title, introduction, body, and (optionally) an image. Image posts are simpler than forum posts; they are only an image submitted by the user. Posts on the forum show the username of the user who created them. Users can "follow" other users, meaning that they can have a feed that only shows forum and image posts from users they have chosen. Users can also comment on their own or other users' forum posts, allowing for two-way communication.

Recipes

Users can either create their own recipes by typing information into a form, or they can search recipes via the Spoonacular API. The user enters a keyword, which is queried in the Spoonacular recipe database, and the top five recipe results will be displayed.

The main recipes page shows a list of the user's added and created recipes. Each recipe has buttons to schedule, edit, or delete the recipe. "Schedule" will prompt the user to choose a day for the recipe to be added to their meal plan. "Edit" allows the user to make any changes to their copy of the recipe.

Calendar

The calendar page uses an HTML table to display a month, week, or day. Each day is checked against the dates from a user's list of scheduled recipes and will show all recipes the user has scheduled for that day. Recipes can be deleted with the "x" button or edited by clicking on the recipe title.

The user can switch between month, week, or day views, and there are "previous" and "next" buttons to navigate between neighboring months, weeks, or days.

In the week view, there is an option for the user to generate a PDF of the user's meal plan for that week. This PDF is generated using ReportLab, and it constructs a table with the days of the week in one column and their associated recipes in the second column.

Grocery and Pantry

The user can add ingredients, which are saved as "foodingredient" objects which can be used in the grocery or pantry list. When typing in ingredients to add to a list, there will be a dropdown list to autocomplete the item name if it has been added previously. You can also choose the quantity of an ingredient; the default quantity value is 1.

The grocery page allows the user to send the entire list to the pantry; this feature is useful if a user has gone to the grocery store and purchased all items on the list. The pantry page allows the user to add in an expiration date for each ingredient, allowing the user to keep track of fresh or old items in their pantry.

Architecture

A database schema for the application is shown in Figure 4. It shows all the relevant objects stored in the Postgres database. Some objects in the database have been excluded from the diagram because they are only used in the Django backend.

Most objects are involved in a many-to-one relationship with the user, in which a user is associated with zero or more of another object. For example, one user will have zero to many recipes. This many-to-one relationship is reflected by a line connecting the two objects, where the many-object has a crow's foot, and the one-object has a short vertical line.

Figure 4. Database Schema

Application Interface

The *Cook-it!* navigation bar allows for easy navigation between different pages. It uses a tabbed interface which emphasizes the current page and separates user authentication pages to the right side of the screen. When a user is logged in, they have access to all the main pages of the application [Figure 5]. When a user is logged out, tabs for the Recipes, Calendar, Groceries, and Pantry pages are hidden, since a user must have an account to use those features [Figure 6].

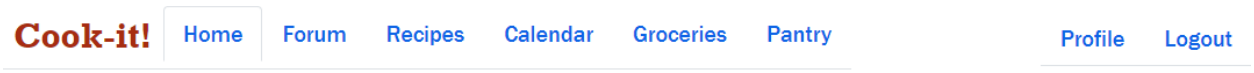

Figure 5. Navigation Bar for Logged In User

Figure 6. Navigation Bar for Logged Out User

When the user performs certain actions, such as creating an account or adding a recipe, a one-time notification message will appear at the top of the page, just underneath the navigation bar [Figure 7]. These temporary messages are managed by Django's messaging framework.

Figure 7. Example Django Success Message

Note that most pages, with the exception of the Calendar page, feature a large image on the right side of the screen. This is to fill up white space and make the page more visually appealing to the user. When the width of the user's screen is small, such as when using a mobile device, the image will be dropped to the bottom of the scrollable page.

A screenshot of the Recipes page is shown in Figure 8. Each recipe is contained in its own box displaying the title, cook time, and buttons to schedule, edit, or delete the recipe. The top of the page shows the two options to search recipes (via the API) or create a new recipe.

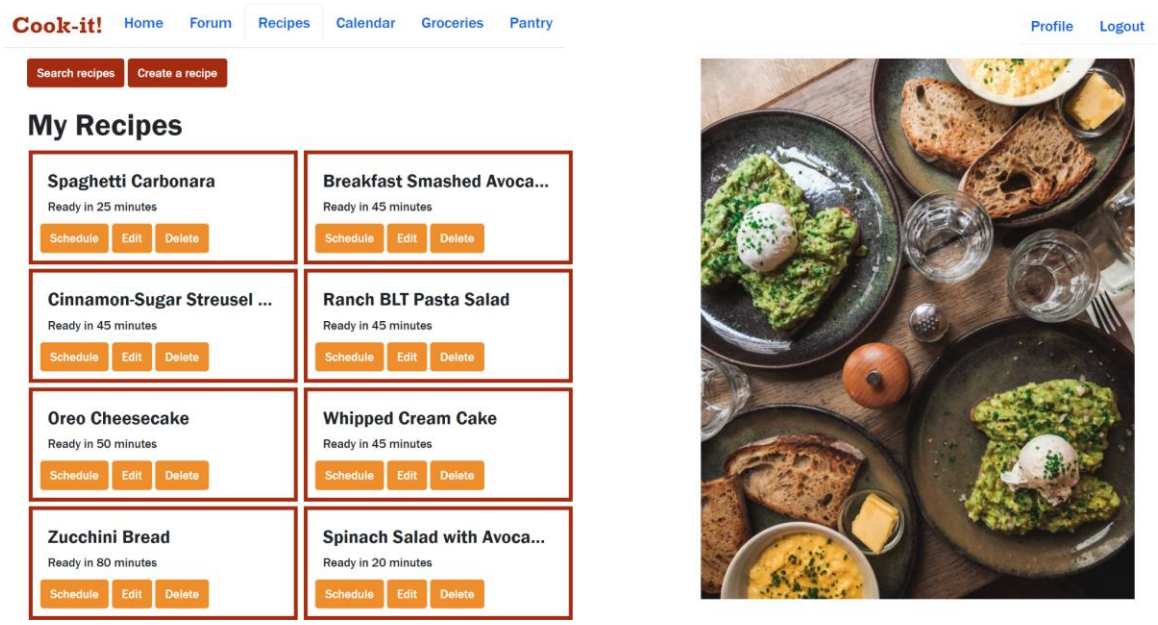

Figure 8. Recipes Page

The Groceries page is shown in Figure 9. The user can add an ingredient by typing it into the item name field and choosing a quantity. Each item in the list has buttons to edit, delete, or send that ingredient to the pantry. The Pantry page looks similar, but items do not have a "send to pantry" button, and the items show an expiration date along with the quantity.

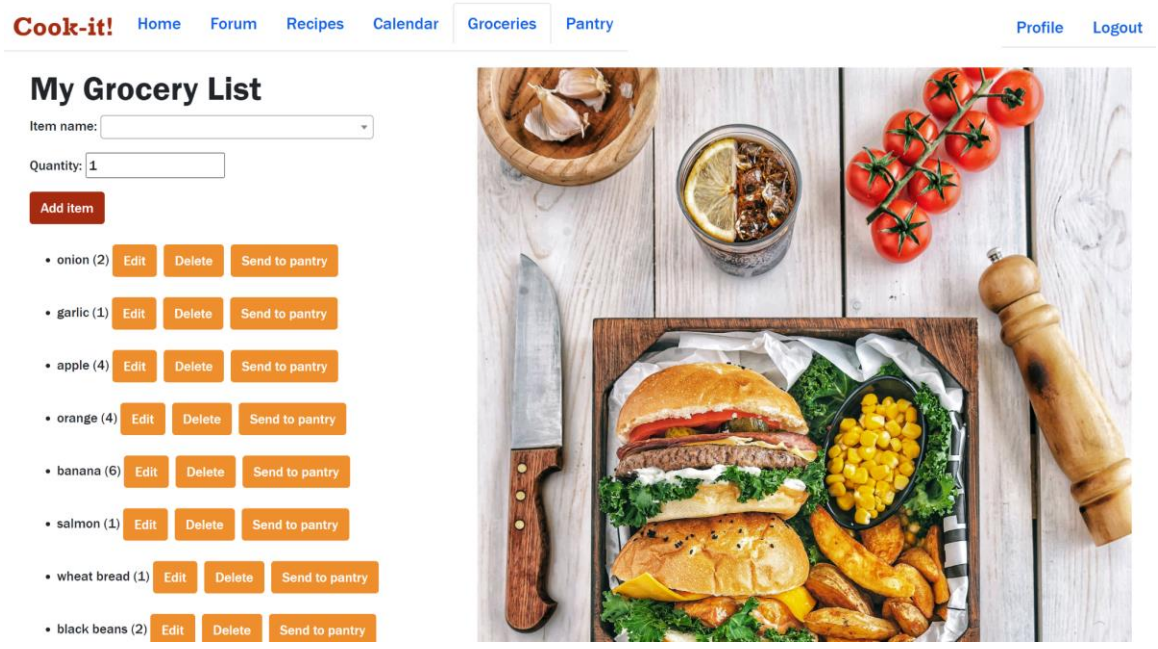

Figure 9. Groceries Page

The Forum page is shown in Figure 10. There are buttons to add a new post, and to see a feed with only posts from people the user follows. Forum posts are located on the left side of the screen, separated by horizontal bars. When hovered over, the background color of the post will change. The right side of the screen has an image for filling up white space, and underneath shows all the image posts in individual boxes.

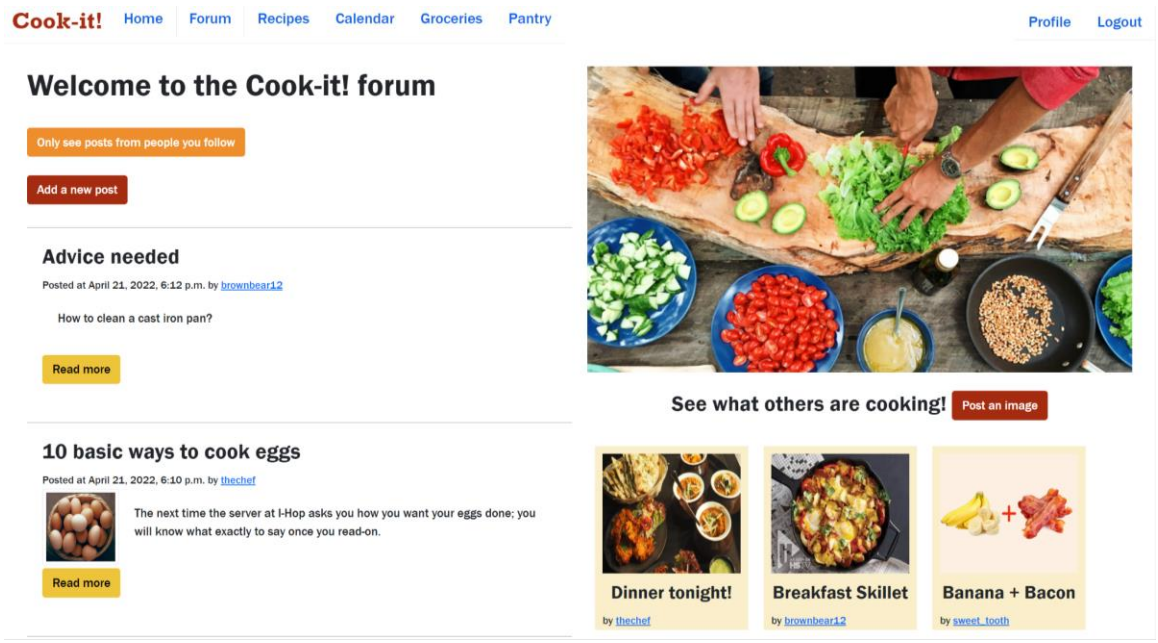

Figure 10. Forum Page

A month's view of the Calendar page is shown in Figure 11. The user can see each day's scheduled recipes and change views using the buttons at the top.

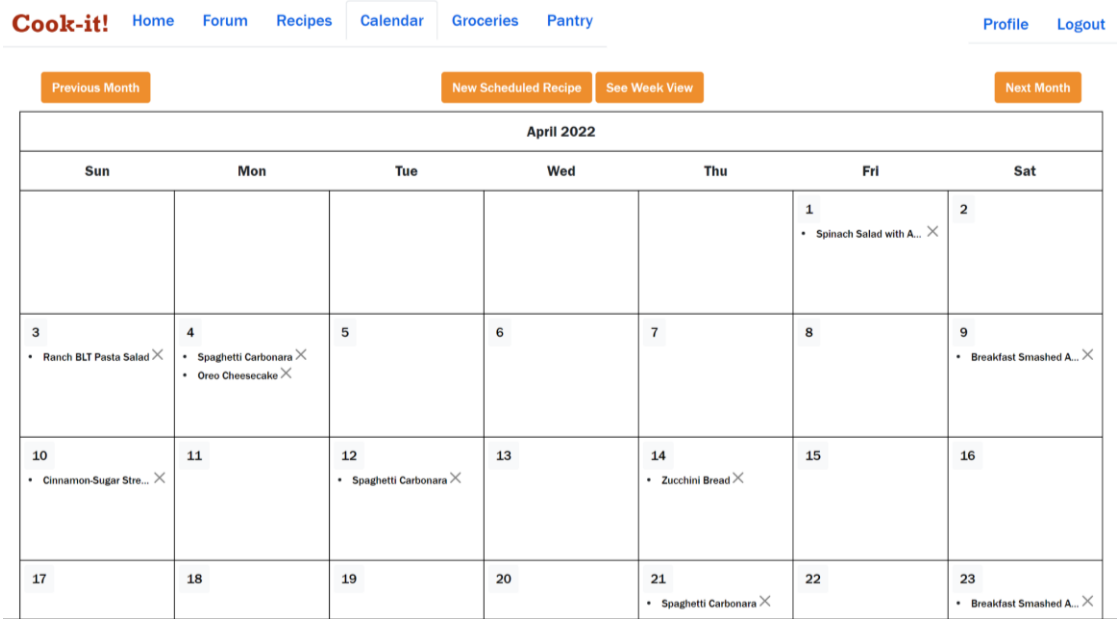

Figure 11. Calendar Page

Testing

Tests for the *Cook-it!* application were written in Python using the Selenium WebDriver. The web driver simulates opening a Google Chrome browser and performs behavioral tests automatically. The driver navigates the page by selecting HTML elements and performs actions such as entering data and clicking.

These tests cover most of the core functionality of the app. The start of each test will create and/or log in to a testing account. Then, certain functions are performed such as creating a recipe or adding an ingredient to the pantry. These are automatically performed, but the test will wait a few seconds in between tasks to allow the person running the test to see what it is doing. When finished testing, it will delete objects created during testing and leave the browser open for the human to perform any manual testing.

Discussion

Technical Obstacles

Our team originally planned to use Elastic Beanstalk, an AWS service for deployment [13]. However, we encountered issues that made the web app difficult to deploy. Additionally, there were problems using AWS S3 storage where the account was incorrectly charged money when using the free-tier level of resources, and we had difficulty reaching support.

As a result, the team decided to switch to Heroku to deploy our application. Unfortunately, the Heroku free plan causes the web app to "sleep" after 30 minutes of inactivity, resulting in user-added images being wiped out of storage.

Future Work

Cook-it! was built as an educational project rather than an app that is ready to be released to the public. With a few additions and changes, it could potentially make a successful commercial application.

One of our team's limitations was to only use free third-party services for deployment, storage, and the recipe API, making some of the application's features slow or limited. If paid services were used instead, it could speed up the app, allow for more storage of user data, and allow for more users using the app simultaneously.

For a successful meal-planning app, more features could be implemented. In the planning phase of the project, our team thought of extra features that could be useful to users, including suggested recipes, rating recipes, scanning barcodes to add ingredients to pantry, and auto-ordering groceries.

Another direction to take the application would be to make it into an Android and/or iOS app. While our web app can run on mobile browsers, it was designed for a desktop browser and may not look as intended on mobile devices. Many people today prefer to use apps on their phones rather than their computers, so developing a mobile application could facilitate a wider user base.

Conclusion

The goal of *Cook-it!* is to provide a platform for users to easily manage meal planning. Development of the application involved the use of several different technologies, planning of requirements and design, and testing and deployment of the app. The project involved both front-end and back-end development, successfully incorporating them into one web application for the user.

References

[1] “GitHub Features: The Right Tools for the Job.” GitHub. Accessed May 2, 2022. https://github.com/features.

[2] “Welcome to Python.org.” Python. Accessed May 2, 2022. https://www.python.org/about/.

[3] “PEP 8 – Style Guide for Python Code.” Python. Accessed May 2, 2022. https://peps.python.org/pep-0008/.

[4] “Django Overview.” Django. Accessed May 2, 2022. https://www.djangoproject.com/start/overview/.

[5] “About PostGreSQL.” PostgreSQL. Accessed May 2, 2022. https://www.postgresql.org/about/.

[6] “What Is Heroku?” Heroku. Accessed May 2, 2022. https://www.heroku.com/what.

[7] “Recipe API.” Spoonacular Recipe and Food API. Accessed May 2, 2022. https://spoonacular.com/api/docs/recipes-api.

[8] “Selenium WebDriver.” Selenium. Accessed May 2, 2022. https://www.selenium.dev/documentation/webdriver/.

[9] “HTML & CSS.” W3C. Accessed May 2, 2022. https://www.w3.org/standards/webdesign/htmlcss.

[10] “ReportLab Open Source.” ReportLab. Accessed May 2, 2022. https://www.reportlab.com/opensource/.

[11] “User Authentication in Django.” Django. Accessed May 2, 2022. https://docs.djangoproject.com/en/4.0/topics/auth/.

[12] “Bootstrap.” Bootstrap. Accessed May 2, 2022. https://getbootstrap.com/.

[13] “AWS Elastic Beanstalk – Deploy Web Applications.” Amazon. Accessed May 2, 2022. https://aws.amazon.com/elasticbeanstalk/.