Developer Documentation

What is 1-Stop Car Shop?

Welcome to 1-Stop Car Shop! This free software combines “all present market listings” (simulated data) for new and used cars onto one website (by means of a large, constantly updating database of cars) with recently sold car prices included (if applicable) and provides a way to have your car’s price estimated instantly (with the help of MarketCheck’s Price Predictor API). Save time by using the functionalities of our web service, rather than finding each function of the service on a different site!

This web service was created from scratch mainly as a team-based class project for CS3300 at the University of Colorado Colorado Springs by Computer Science students John Camargo, Jackson Seales, Kylie Sanchez, and Lucas Estevez

Built With

- Python (data creation, Flask for backend)
- HTML and CSS with JS
- SQLite 3
- MarketCheck’s Price Predictor API
- Render (Web Hosting)

Set-up and Execution

Prerequisites

- GitHub account
- Flask version 3.0.3
- requests version 2.25.1
- bcrypt version 4.2.1
- flask_cors version 5.0.0
- flask-jwt-extended version 4.7.0
- email-validator version 2.2.0

Architecture

/root
| -- /Code
| -- Python/.py files (not including app.py)
| -- /User Data
| -- Files such as saved_cars.csv, and user database
| -- /Documentation

| -- Data
| -- Devs and Users
| -- Images
| -- Legal
| -- /static <-- Flask is designed to serve static files only from the static directory by default
| -- Car Data
| -- CSS_Files
| -- Images
| -- PapaParse (in this case used for .csv parsing)
| -- /templates <-- will keep all the web pages/html files
| -- base.html (optional if using Jinja)
| -- index.html
| -- Directories for different landing pages
| -- /Testing <-- includes test case files and other files for testing deployments
| -- .gitattributes
| -- render.yaml <-- render template file; go to Tools -> Render for more information
| -- requirements.txt <-- system dependencies to be installed on deployment (Prerequisites)
| -- app.py <-- creates backend (server) for service; includes routes and price predictor/user logic; **vital code**
| -- start_app.sh <-- runs app.py (which builds the backend)
| -- LICENSE/README

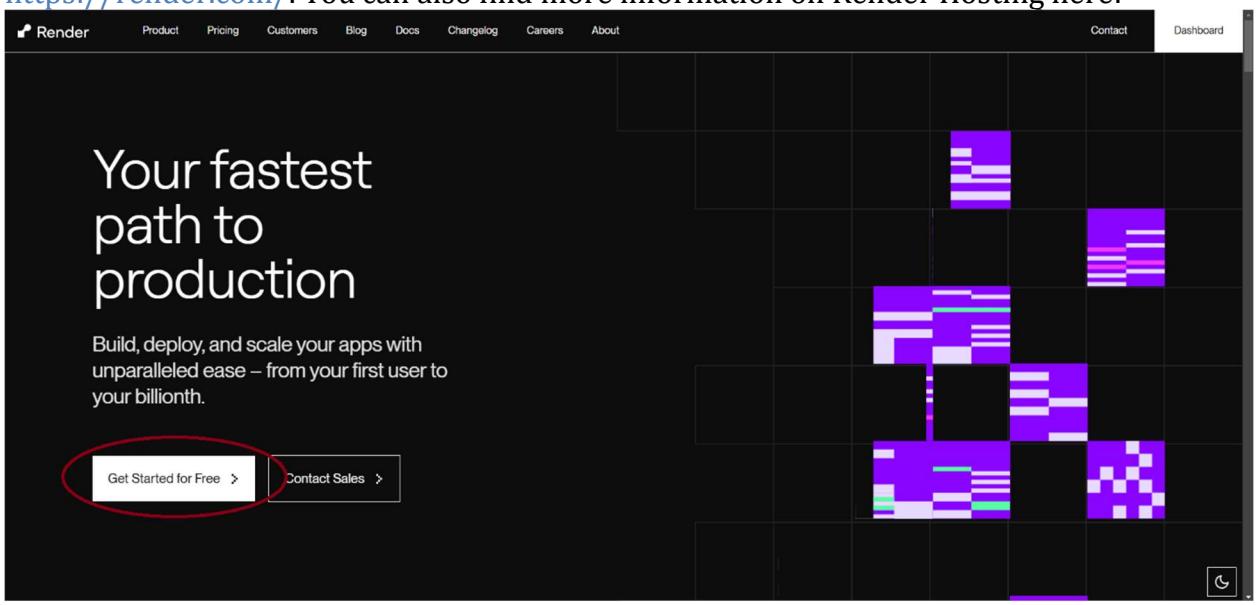
Installation and Hosting/Execution

1. Create an account with MarketCheck and get a free Car Price Predictor API key at https://www.marketcheck.com/apis/

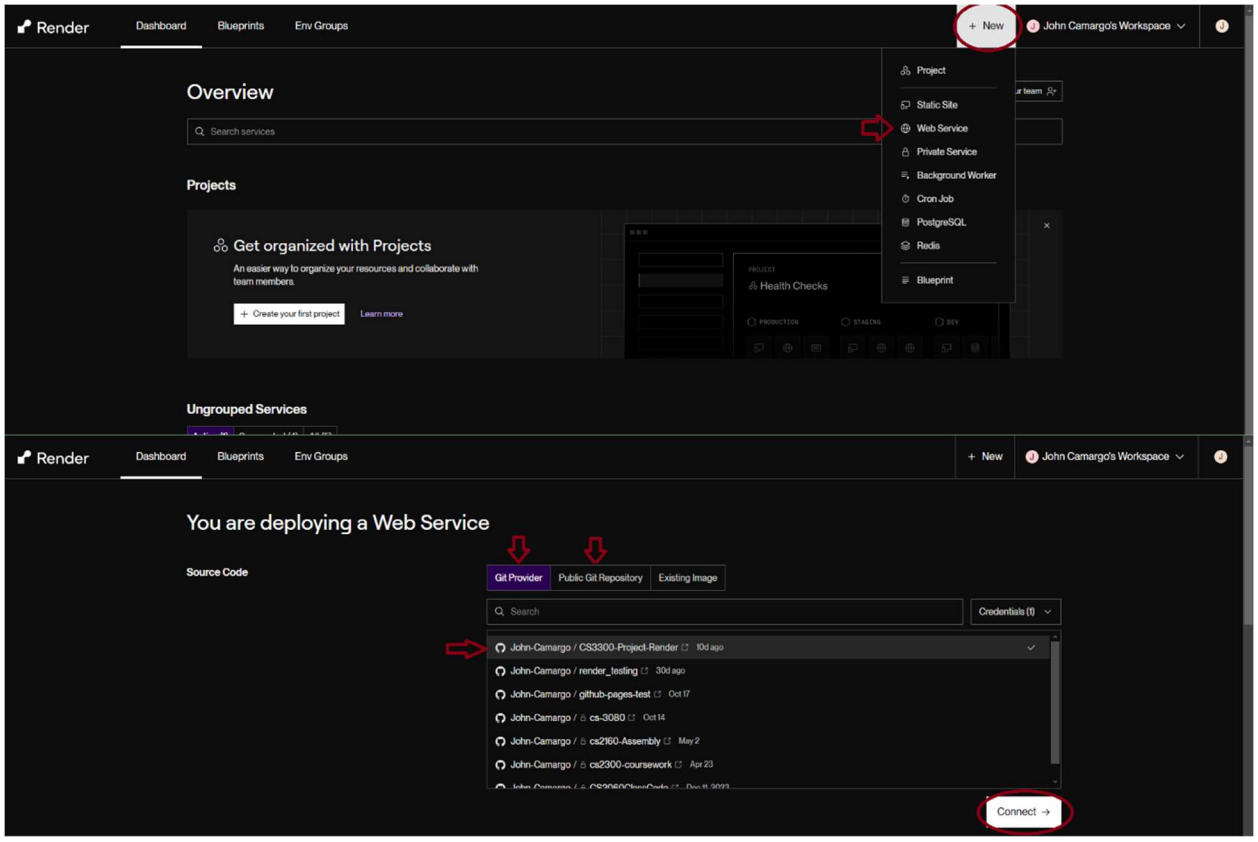
2. Clone the repo (this step is optional as you may also just provide a URL to the project repo instead); more info can be found at https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository:

   git clone https://github.com/LucasJEstevez/CS3300-Semester-Project

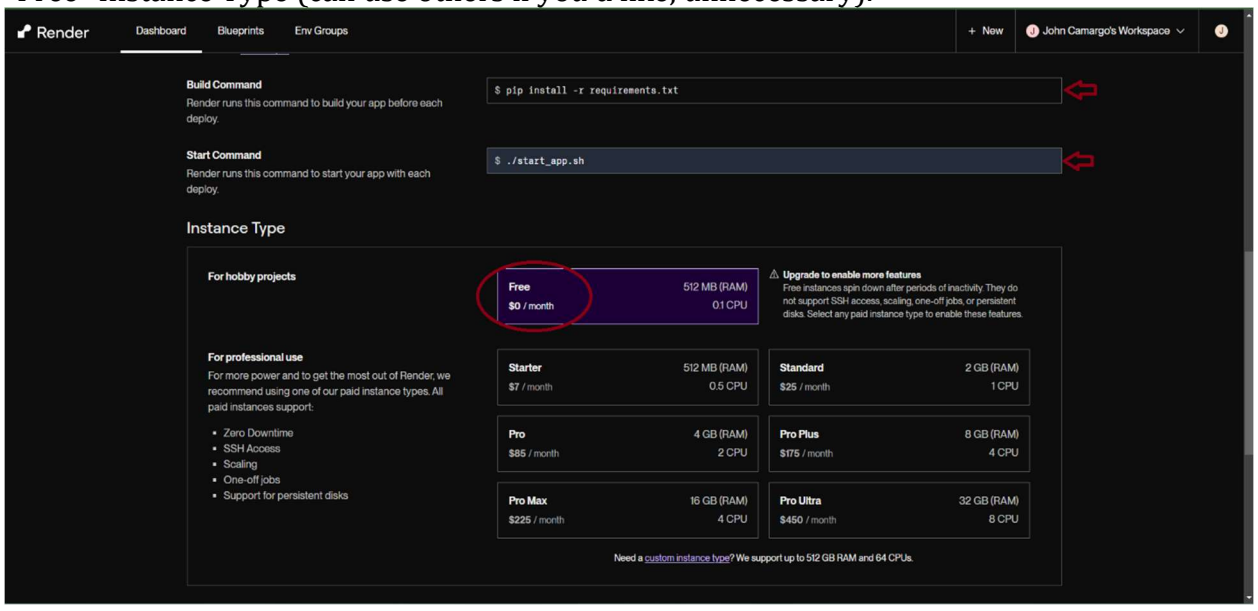
3. Create a Render account (Get Started for Free) and link to your Github account at https://render.com/. You can also find more information on Render Hosting here!



4. Once you have an account with Render which is attached to your GitHub account, in the Dashboard, click the "+ New" tab next to your current workspace in the top-left of the page. Select Web Service. You should see the GitHub repository you just cloned as one of the first options under "Git Provider". Alternatively, if you choose not to clone the repo, you can paste the link to this GitHub repository directly into the "Public Git Repository" section.

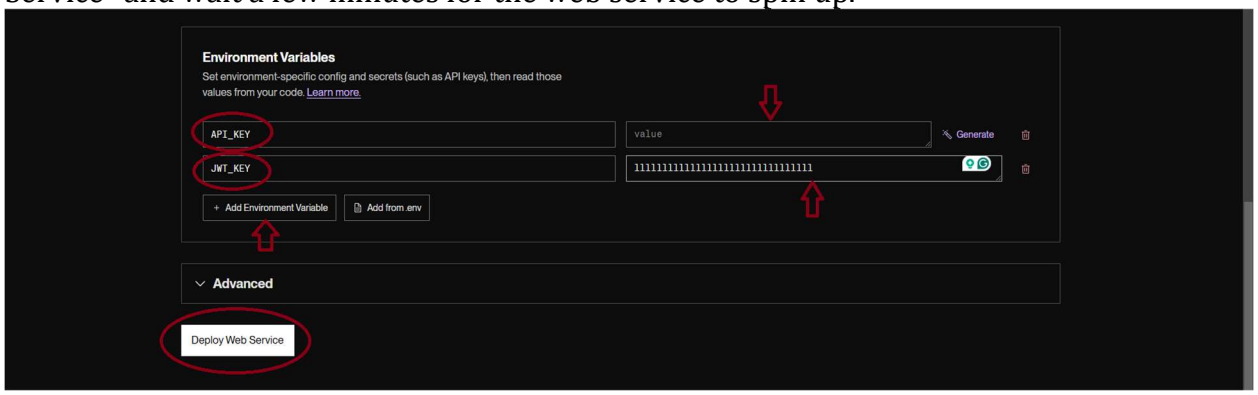


5. Next, after naming the project, you can leave most sections blank/with their default values, but scroll to Build Command and Start Command, and input: pip install -r requirements.txt and ./start_app.sh respectively. Also ensure you are using the "Free" Instance Type (can use others if you'd like, unnecessary).

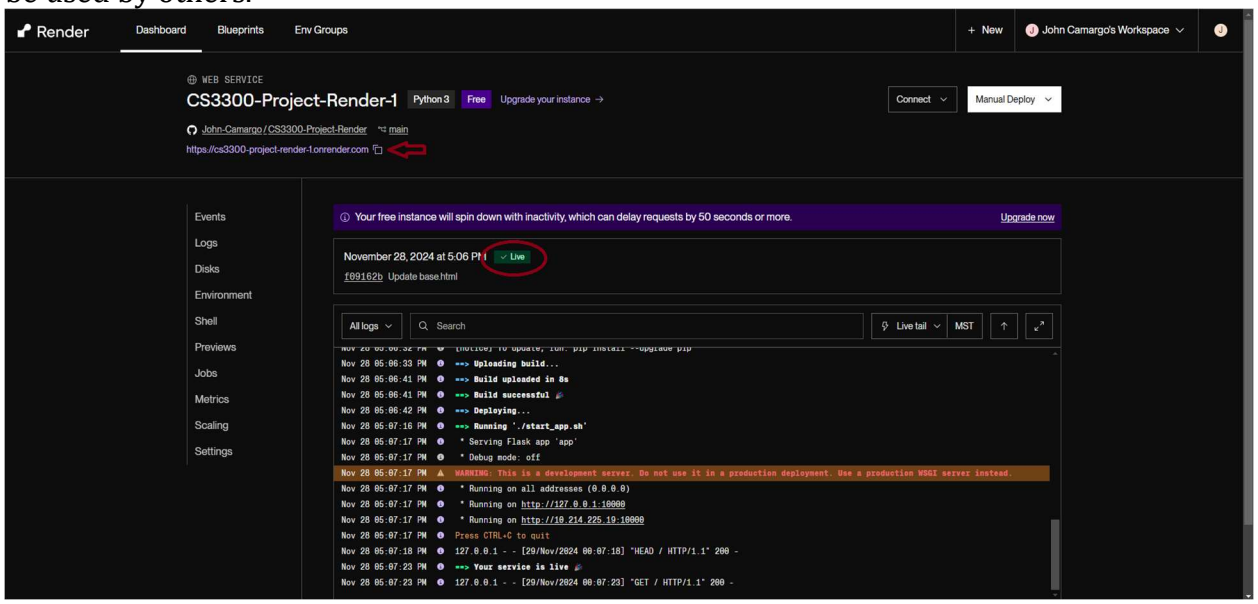


6. Finally, before deploying the web service, scroll to Environment Variables and enter API_KEY as an environment variable (inputting the key string given to you by

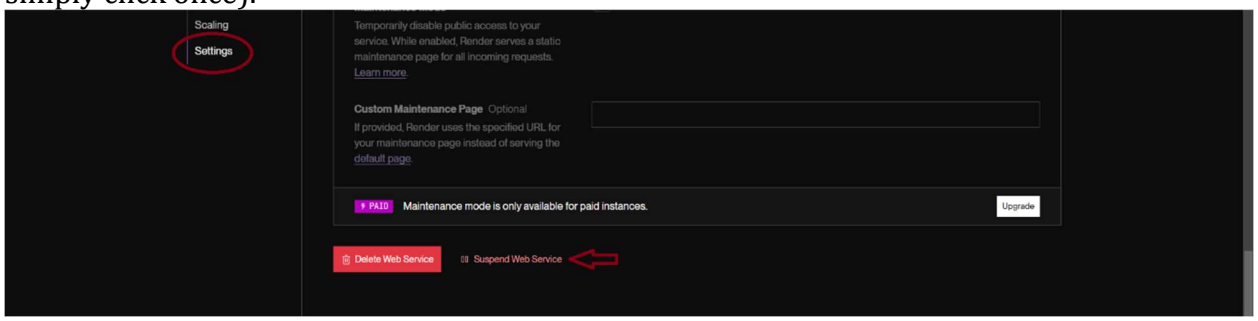
MarketCheck to the “value” space) and JWT_KEY as an environment variable (can be a randomly generated string of numbers and letters; input the key string to the “value” space); hit “+ Add Environment Variable”. Set up complete! Hit “Deploy Web Service” and wait a few minutes for the web service to spin up.



7. To go to the website for the service, follow the link provided by Render towards the top of the page. As long as the service is not suspended, this link is sharable and can be used by others.



8. To suspend the service, navigate to the Settings section on the left-hand side of the Render menu, and scroll until you see “Suspend Web Service” in red text (which you simply click once).



From there, feel free to add to, delete, and create any files in the repository (though some files like render.yaml and index.html are necessary for Render hosting)!

If you are a developer who directly uses the project repository, you can add, delete, and modify the repo as you need (may need to create a pull request first).

Features

Functional Requirements

- Ability to estimate a car's value based on make, model, miles, year, or VIN
- Must not save user vehicle information when getting prices checked (per API licensing), rather be outputted on the webpage itself
- Users able to create an account on the website (but optional)
- After account creation, the account holder shall be able to save/add cars to an expandable list
- When browsing available vehicles, the user should be able to filter cars by certain criteria (make, miles, year, etc.) to look at individual listings
- Vehicle data must only be of .csv format (processed with PapaParse), system programs (such as data creation and back end) as .py (or Python-based); user data (minimally passwords) to be hashed with BCrypt

Non-Functional Requirements

- Easy-to-use website interface
- Quick price estimation feedback (<5 seconds)
- Zero access to API key
- The user's login and save data must be private and secure

Tools

Render

We used Render to host our Web Service (for free), it was easy to use after understanding the basic blueprint for a Render web service.

Render help and documentation, along with specifics/limits of what you can access with the free tier: About: https://render.com/about; Documentation: https://render.com/docs/free

Flask

We use Flask (with Python) as our backend, creating our dynamic web pages and landing pages. Other than that, Flask is mainly used for API and user data/login requests and some testing aspects.

Flask documentation: https://flask.palletsprojects.com/en/stable/

Papa Parse

Our software uses an external resource called Papa Parse. This is a CSV parser using JavaScript that we implemented to parse through our car data and display it to our various webpages. As we parse through the CSV, each row is appended to an HTML class. Due to the large size of our data, we used the step option provided by Papa Parse which streams the input. This can result in slower load times depending on the file's relative location but felt necessary for the size of our files due to limited browser memory.

Papa Parse GitHub: https://github.com/mholt/PapaParse

SQLite3

Our software uses an external database tool called SQLite3, a software library that provides a lightweight database engine which allows us to store our user-login information.

SQLite documentation: https://www.sqlite.org/docs.html

BCrypt

In order to store our user data securely, we use BCrypt to hash our users' passwords before storing them in our database so we do not keep plaintext passwords stored.

BCrypt Documentation: https://pypi.org/project/bcrypt/

MarketCheck API

Used to obtain instant vehicle price estimation after user inputs vehicle data into website.

MarketCheck API Documentation: https://www.marketcheck.com/apis/cars/ and https://apidocs.marketcheck.com/

Email-Validator

To help us validate user-inputted emails (when creating an account), we used the Email-Validator Python library.

Email-Validator Documentation: https://pypi.org/project/email-validator/