

T2A2 API Webserver Project

The directory for this application can be found here:

[Link to T2A2 API Webserver Project directory.](#)

Identification of the problem you are trying to solve by building this particular app.

- Roaster Connect exists to connect local coffee roasters and customers looking for single origin coffee beans that are roasted locally. It is an easy to use platform that allows roasters to post what beans they have available, including the variety and flavour notes, while also allowing customers to order specific beans in whatever quantity, grind, and roast they'd like. The roasters then fulfil this order and mail it to the user through a third party service.

Why is it a problem that needs solving?

- While there are many local coffee roasters, it is sometimes difficult to tell what coffee beans they have available on any given day, especially if the roaster doesn't have a strong online presence. Additionally, sometimes the same variety grown in a different area can have a slightly different flavour profile. This app allows for clear, standardised criteria from roasters without overloading the customer with information.

Why have you chosen this database system. What are the drawbacks compared to others?

- I've chosen PostgreSQL as it's a well known database management system that supports a large amount of SQL syntax and allows unstructured data handling, allowing PostgreSQL to work with multiple languages (Derivaux, 2019). As this app is written using Python and utilises JSON for returning information; having a DBMS that supports multiple languages was necessary (Derivaux, 2019). Additionally as this app utilised multiple tables and queries, having a DBMS that supports common table expressions and big queries, as PostgreSQL does, was needed (Derivaux, 2019).
- The drawbacks of PostgreSQL are that it does not provide data tooling for data compression and it doesn't store data in columnar tables (Derivaux, 2019). Additionally, it doesn't support machine learning; though none of these drawbacks were relevant for Roaster Connect at this time (Derivaux, 2019).

Identify and discuss the key functionalities and benefits of an ORM

- ORM means object-relational-mapping, which is a technique for creating a layer between the language being used and database; this allows programmers to write in their

language of choice instead of SQL, which can be very time consuming (Liang, 2021). Additionally ORMs standardise interfaces, reducing boilerplate and making the development process quicker (Liang, 2021). ORMs translate data and create a structured map, the mapping explains how objects are related to different tables and allows ORMs to convert data between tables and generate the SQL code for relational databases to change in response to the changes made in the application (Liang, 2021). Once the mapping is created, the ORM mapping will manage the data and the developer will not need to write additional low-level code (Liang, 2021).

- The benefits of using an ORM are that it minimised the amount of SQL knowledge needed to work on databases and increased the speed of development for the application, as well as can keep track of database changes making it easier to fix errors and update the application in the future (Liang, 2021). It also allows code to be reused and allows the use of a class library, which also assists in speeding up development (Liang, 2021).

Document all endpoints for your API

- Address routes: address/
 - Methods: GET
 - Authentication: JWT token (roaster only for user's safety)
 - Required data: none
 - Expected response: Will list all addresses currently in the database
 - Error handling: error handling for not having roaster permissions

address/add

- Methods: post
- Authentication: JWT token (roaster and user)
- Required data: address fields to be appropriately filled out
- Expected response: Will add a new address to the database

Error handling: error handling for incorrect fields being filled out

- Bean routes: bean/
 - Methods: GET
 - Authentication: none
 - Required data: none
 - Expected response: Will list all bean varieties currently available

bean/search

- Methods: GET
- Authentication: none
- Required data: a search parameter in the address
- Expected response: Will list all bean varieties based on search parameter
- Error handling: error handling for not no beans matching search parameters

bean/[int:id](#)

- Methods: GET
- Authentication: none
- Required data: a specific bean id
- Expected response: Will list the bean connected to that specific id

bean/add

- Methods: POST
- Authentication: JWT token (roaster)
- Required data: All bean fields to be filled in
- Expected response: Will add a new bean variety to the database if posted with roaster authentication
- Error handling: error handling for not having roaster permissions

bean/update/[int:id](#)

- Methods: PUT
- Authentication: JWT token (roaster)
- Required data: a specific bean id in the address, bean fields filled out in body
- Expected response: Will update the fields of a specific bean variety if updated from a roaster
- Error handling: error handling for not having roaster permissions and if bean variety is not in database

bean/delete/[int:id](#)

- Methods: DELETE
- Authentication: JWT token (roaster)
- Required data: a specific bean id in the address
- Expected response: Will delete a specific bean variety if updated from a roaster
- Error handling: error handling for not having roaster permissions and if bean variety is not in database

- Order routes: order/

- Methods: GET
- Authentication: JWT token (roaster)
- Required data: none
- Expected response: Will display a list of all current orders to roasters
- Error handling: error handling for not having roaster permissions

order/[int:id](#)

- Methods: POST
- Authentication: JWT token (user)
- Required data: specific bean id in the address, order fields filled in in the body
- Expected response: Will allow users to place an order for a specific bean variety

- Error handling: error handling for not having user permissions and if that bean variety doesn't exist in the database

order/delete/[int:id](#)

- Methods: DELETE
- Authentication: JWT token (roaster)
- Required data: specific bean id in the address
- Expected response: Will allow roasters to delete orders that have been fulfilled
- Error handling: error handling for not having roaster permissions and if that order doesn't exist in the database

- Roaster routes: roaster/register

- Methods: POST
- Authentication: none
- Required data: roaster fields filled in in the body
- Expected response: Will allow a roaster to create an account
- Error handling: error handling for if the email or username is already in use

roaster/login

- Methods: POST
- Authentication: none
- Required data: roaster username and password in the body
- Expected response: Will allow a roaster to login and get an authentication token
- Error handling: error handling for if the username or password is incorrect

- User routes: user/register

- Methods: POST
- Authentication: none
- Required data: user fields filled in in the body
- Expected response: Will allow a user to create an account
- Error handling: error handling for if the email or username is already in use

user/login

- Methods: POST
- Authentication: none
- Required data: user username and password in the body
- Expected response: Will allow a user to login and receive a JWT token
- Error handling: error handling for if the username or password is incorrect

An ERD for your app

 The ERD for T2A2

Detail any third party services that your app will use

- Once implemented and live, Roaster Connect will make use of various third party services such as PayPal/Visa/Mastercard: secure online payments Aramex and/or other delivery services: quick and efficient delivery of coffee beans to customers Github: a cloud based service that is used to store, track, and manage changes to code.
- Modules and libraries: Bcrypt: Password hashing function Click: Python package used for command line interface Flask: a web framework written in Python JWT-Extended: adds support for using JSON web tokens to Flask Marshmallow: an ORM/ODM agnostic library for converting complex data types to and from Python Itsdangerous: a Python library for passing data to/from untrusted environments Jinja: a web template engine for Python MarkupSafe: implements a text object that escapes characters, this is safe to use in HTML and XML Psycopg: PostgreSQL database adapter for Python PyJWT: Python library that allows the developer to encode and decode JSON web tokens Python-dotenv: a Python module that reads key value pairs from .env files and sets them as environmental variables Six: a Python library that provides utility functions for smoothing over differences between python versions SQLAlchemy: a Python SQL toolkit and object relational mapper Werkzeug: a BSD-licences utility package for Python essential for web server gateway interface Zipp: a ZipFile subclass that ensures that implied directories are always included in the namelist
- Tech stack: Once this app is fully implemented it will include: Front end: CSS SCSS/SASS Javascript HTML Heroku Backend: Flask Python Postman PostgreSQL

Describe your projects models in terms of the relationships they have with each other


- Models:
 - Addresses: This model is for the addresses of the users and roasters.
 - One Address has one User - or -
 - One Address has one Roaster
 - Beans: This model is for the bean varieties used in the orders and posted by roasters.
 - One Bean variety can be in one or more Orders
 - Many Bean varieties can be added by many Roasters
 - Orders: This model represents the coffee bean orders placed by users and fulfilled by roasters.
 - Roasters fulfil Orders.
 - Users can place zero or more Orders.
 - One Bean variety can be in one or more Orders.

- Roasters: This model represents the roasters posting the bean variety options and fulfilling orders placed by users.
 - One Roaster has one Address.
 - Many Roasters can post many Bean varieties.
 - One roaster can fulfil none or many Orders depending on the Beans ordered.
- Users: Users place orders of coffee bean varieties to roasters.
 - One User has one Address.
 - One User can place zero or more Orders.

Discuss the database relations to be implemented in your application

- Roaster Connect was created to connect customers (users) and roasters through ordering and supplying single origin coffee bean varieties. Roasters have the ability to post available bean varieties for users to browse and order. Both users and roasters must have an account to be able to use any app features outside of viewing available bean varieties.
- In order for both users and roasters to access areas that are specific to them, they need to have authentication tokens specific to users or roasters. The roasters need this in order to edit, delete, or post new bean varieties; as well as to view and delete current orders and view user addresses. Users need to have authentication tokens in order to place orders. Users cannot edit, delete, or post new beans nor can they view or interact with current orders. Roasters cannot place orders, though they are welcome to create user accounts if they wish to do so.
- All of this information is stored in a database of multiple tables that relate to each other in the manner discussed and displayed in the ERD shown above.

Describe the way tasks are allocated and tracked in your project

- The trello board for this project can be found here: [Link to T2A2 API Webserver Project trello board](#)  The Trello board for T2A2
- Tasks were allocated and tracked using a Trello board. Tasks were broken down into manageable segments with priority given to coding the app and documentation left until later as there were likely to be multiple edits to the code. All tasks were given appropriate due dates in order to allow plenty of time to complete them. While there have been some roadblocks, mainly the head developer of the project contracting COVID, the first iteration of the project was able to be completed with a small extension.
- Future iterations will include more frontend development and cleaner interface for customers and roasters. Additionally, a payment and delivery system will be implemented.

Reference list

Derivaux, S., 2019. PostgreSQL for data science : pro and cons | Data into results. [online] Data into Results. Available at: <https://dataintoreresults.com/post/postgresql-for-data-science-pro-and-cons/> [Accessed 26 September 2022].

Liang, M., 2021. Understanding Object-Relational Mapping: Pros, Cons, and Types. [online] AltexSoft. Available at: <https://www.altexsoft.com/blog/object-relational-mapping/> [Accessed 27 September 2022].