**DEVELOPER MANUAL**

This document serves as a guide for developers interested in continuing the work done by our team in developing this application. If you wish to contribute, you can subscribe to our Github repository and make pull requests. The document starts with an overview of our application, the Python libraries used, setting up the environment and being able to make changes to the queries.

**Overview**

The application was built using the Python Flask framework, MongoDB, Flask-PyMongo and Mongo Engine. Flask is a micro framework for creating web applications. Although Flask is not a MVC framework, we have created a model similar to that of an MVC framework by combining models for our database collections with the routing and templating capabilities of Flask. More on Flask here: <https://flask.palletsprojects.com/en/1.1.x/>

MongoDB Atlas is the database of choice for storing our application data. MongoDB Atlas is a fully-managed cloud database which provides a means for deploying and managing the clusters used. It is also a good choice for collaboration projects such as ours. More about MongoDB Atlas here: [https://docs.atlas.mongodb.com/#](https://docs.atlas.mongodb.com/%23)

Flask-PyMongo is a library used in our application for querying the MongoDB document store. Flask-PyMongo enables you to directly query your database using the MongoDB API. We use this in some queries of our application where MongoEngine’s querying capabilities are not sufficient. More on Flask-PyMongo here: <https://flask-pymongo.readthedocs.io/en/latest/>

MongoEngine is a Document-Object Mapper, similar to an ORM that helps map the different entities/Objects in our application such as Driver, Route and Assignment to collections in MongoDB. More about MongoEngine here: <http://docs.mongoengine.org/tutorial.html>. Atlas is a fully managed cloud database system

MongoEngine and PyMongo are used to perform the CRUD operations in our application.

**Setting up the Environment**

Note: You can skip step 1 if you have the files already stored locally.

To begin with, start by cloning the Git repository to your local machine from the following link:

<https://github.com/michaelcwatts123/DB_FinalProject.git>

You can simply run the command:

git clone <https://github.com/michaelcwatts123/DB_FinalProject.git>

Note: cloning the entire repository brings the application as well as the sample data and documentation associated with it. The application resides in the BusNetwork folder.

Once done cloning, set up a virtual environment in Python for the application which enables you to manage separate package installations for the ones used in this project that are not part of the standard library. Activate the virtual environment.

Then install all the application dependencies from the requirements.txt file and set the Flask environment variable by running the following commands:

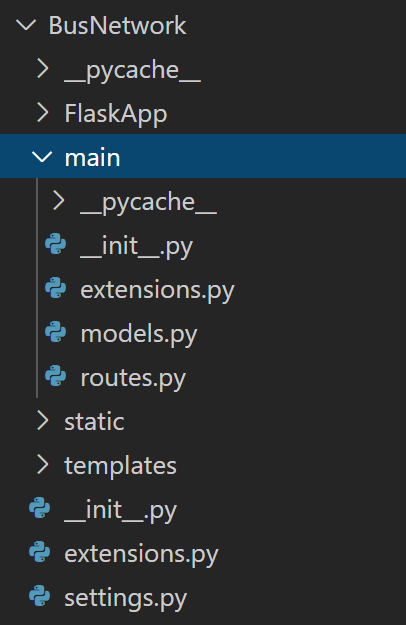
UNIX Bash: *$ export FLASK\_APP=BusNetwork*

Windows CMD: *> set FLASK\_APP=BusNetwork*

The final step is to start the Flask app with the following command: flask run

This will launch the app on the builtin server in your local machine. You can access the application UI at the address in the terminal or command prompt. Find out more on the UI in the user documentation provided for this application.

**Folder structure:**



The main folder contains the routes.py and the models.py files. The routes file holds great significance in our application. It has all the functions that contain:

* Queries for the database,
* Data parser for the input files
* Validation logic for the objects in our application i.e drivers, routes and assignments
* URL paths for all the function calls and
* Routing of rendered templates with the specific data for the URL paths
* Routes for 404 and internal server errors

The models file contains the classes for all the objects (i.e driver, route and assignment) in our application which are also the collections in the database. We have a model or a class for each collection in the database. The data types for each one of our fields/attributes are set in this file.

The static folder contains the javascript and css for the UI. The templates folder holds all the HTML files for the application.

**Steps for creating MongoDB Atlas Cluster**

* Start by registering and logging in to your account at the Atlas portal here: <https://www.mongodb.com/cloud/atlas/register>
* Once logged in, click on the Build a Cluster button, choose the cluster tier of choice based on the project requirements. You can begin with the free cluster for test purposes.
* Next, choose the cloud provider of choice between AWS, Azure and GCP. Then select the cloud provider region.
* Enter a name for your cluster in the following step and click on the Create Cluster button
* Next up create a database user for the database with appropriate rights in order to be able to perform CRUD operations from our application. Make a note of this username and password.
* The final step is to import the connection string to the Flask application. You can do this by clicking on the Connect button in your cluster dashboard, choose Python and the version number in the pop up. Then make a note of the connection string with <username> and <password > replaced with the values from the previous step.

The init file in the root directory of the app is where we instantiate an app object and return it. In the \_\_init\_\_.py file we add the MongoDb cluster information from the previous step, so that PyMongo and MongoEngine can use it to make connections with our document store and run queries against it. Please add your cluster information to app.config["MONGO\_URI"] before you start the application. The mongo object returned can now be used in all of our routes.

**The Queries**

One of the primary functionalities of our application is to give users the ability to run queries against MongoDb. Both Flask-PyMongo and MongoEngine are used in our application to accomplish this. The 5 queries for this project are in the routes.py file with a route function for each query.

For queries using the MongoEngine Document\_Object Mapper, we use the objects attribute of the document classes in the Models file. This method is a QuerySetManager and returns a Queryset object that contains the results of the query.

Here’s the syntax for MongoEngine queries: <classname>.objects(<attribute conditions for the query>)

Examples from our application of a MongoEngine query:

Driver.objects(Q(firstName=driver\_fn) & Q(lastName=driver\_ln))

The above query searches the Driver collection in the database where firstName and lastName matches the user input in the form of Query1 tab

For queries using Flask-PyMongo, use the Flask-PyMongo objects which provides direct access to the database cluster with the db attribute. First we add Flask-PyMongo to our code by passing Flask application to PyMongo() method and storing that in a variable:

Example: mongo = PyMongo(app)

Then we can simply use that variable to access all the collections in our cluster and query against it

Ex:

mongo.db.routes.find({"routeNumber":{"$in":x}}, {'\_id': False,'\_cls': False,'driver\_id':False})