# Technical Documentation

## Notes

Please note that while this has been tested to work, occasionally the files like to act up, if you ever have issues, attempt to run the code by using “run and debug”, as that I’ve noticed seems to fix it a bit.

## Authentication Routes

For every page of this website (with the exception of signup and login, of course), users must be logged in (@login_required) to access them.

*/riderSignup, /driverSignup*

In these routes we allow the users to sign up, specifying whether we’re expecting a rider or a driver. To successfully sign up, we’re expecting a username, password and email from our riders, and additional information about the driver’s car on a driver signup.

Methods: GET, POST

Response: The response here would include a response of 302, on a successful signup or if the user is already logged in and it redirects them to the welcome page and a 200 on a GET request when the user is not logged or if anything else happens (such as a taken username). Our response’s type is once again an HTML page and the content body would include everything rendered from the signup.html template.

*/login*

In this route we allow our users to login, there will be no difference in login depending on if the user is a rider or a driver.

Methods: GET, POST

Response: The response here would include a response of 302, on a successful login or if the user is already logged in and it redirects them to the welcome page and a 200 on a GET request when the user is not logged or if anything else happens (such as invalid login credentials). Our response’s type is once again an HTML page and the content body would include everything rendered from the login.html template. In this route, we also need the username, and password parameters to be able to successfully log the user in.

Note that there is a hardcoded admin login as well (username: admin, password: admin123) which will simply render the admin's HTML page as opposed to redirecting you.

*/logout*
In this route we allow our users to logout
Methods: GET
Response: The response here would include a response of 302, on a successful logout and we would redirect the user to the home page (which will in turn take them back to the login page)

## Profile Routes

This will include every route from the profile, such as the profile itself, but also the routes that allow the user to edit or delete their driver info.

*/profile*
In this route we display our driver's car information such as the car color, car type, and their license plate number.
Methods: GET
Response: Our Status Codes would be a 200 on a success and a 302 if it's not a success as it will redirect to the error page. The HTML content would be rendered from the profile.html' template, which would require the following variables to be properly filled out:
a car object with details of the driver's car, such as carType, carColor and licensePlate
We redirect to this route after a driver signs up to IT Girl Transport so that they can confirm their car information and change it, as well as giving them the option to remove it entirely.

*/deleteDriverInfo*
In this route we allow our drivers to delete the car information they originally added. This will remove the entire entry from our mongoDB collection.
Methods: GET
Response: There would be no HTML response as this simply redirects you back to your profile page. Given this, this also means that our status code on a success would be 302.

*/editDriver*

In this route we allow our users to edit their car information. This means we can either change information that's already there, or add information if there is no car information already there (for example if the driver recently deleted it). The information the user would need to input includes:

car type
car color
license plate

Methods: GET, POST
Response: The content would be rendered from the editDriver.html template. The status codes would be a 302 on a successful addition and a 200 on the GET when the user is logged in.

## Ride Routes

This will include everything from requesting a ride as a rider, to accepting rides as a driver, displaying the map and the invoices. Most routes in this category are identified by the ride's id.

*/requestRide*
In this route we allow our *riders* to request a ride. This means they can select when they want to be picked up (time), as well as their ideal car for the pickup, where they are and where they would like to go. The information the user would need to input includes:

car type,
pickup location (longitude and latitude), destination location (longitude and latitude)
time
and their payment details (card expiration date, cvv, card number and card holder name)

Methods: GET, POST
Response: The content would be rendered from the requestRide.html template. The status codes would be a 302 on a successful request and a 200 on the GET when the user is logged in.

*/waitForDriver*
In this route we allow our *riders* to wait for their ride to be picked up, as well as the option to cancel their ride if they decide it's taking too long or if they no longer want to be driven somewhere.
Methods: GET

*/pickRide*

In this route we allow our *driver* to select a ride that they wish to pick up. This page should be more or less empty until a user logs in to request a ride, unless a ride was requested before a driver logged in and it has yet to be picked up.

Methods: GET

*/selectRide/<carType>/<riderId>*

In this route, we do the actual selection of the ride, and change all of the necessary information in our DBs, lists and dictionaries to make sure everything is up to date and continuing to function properly.

Methods: GET

*/ride/<rideId>*

This is where we can see the ride's actual details and the pointers of where the rider wants to go versus where they're starting from.

Methods: GET

*/ride/<rideId>/chat*

This is the route that allows our rider and driver to chat with one another.

Methods: GET, POST

*/ride/<rideId>/arrived*

This route allows the driver/rider to determine if they have arrived at their destination and to end the ride. This route will also then redirect both the driver and the rider to their respective invoice pages.

Methods: GET

*/ride/<rideId>/invoice*

This route displays the invoice of a given ride, and the relevant information such as how much money was made and other relevant information about the ride.

Methods: GET

## Environment File Configuration

Now, with most of these examples, you're encouraged to change them, but here's a guide through what everything in the environment file does and the format of filling them out the way I do.

*MONGO_URI:*

Description: MongoDB connection URI.

Example: mongodb://localhost:27017

*SECRET_KEY:*

Description: Secret key for securing the application.

Example: asdfghjklasdfghjklasdfghjklasdnjkl

*SQLITE_URI:*

Description: SQLite connection URI.

Example: ITGirlTransport.db

*MAIL_SERVER:*

Description: SMTP server for sending emails.

Example: sandbox.smtp.mailtrap.io

*MAIL_PORT:*

Description: Port for the email server.

Example: 2525

*MAIL_USERNAME:*

Description: Username for email server authentication.

Example: 07063aa155554f

*MAIL_PASSWORD:*

Description: Password for email server authentication.

Example: 211cf5400a14ba

*MAIL_USE_TLS:*

Description: Use TLS for email communication.

Example: True

*MAIL_USE_SSL:*

Description: Use SSL for email communication.

Example: False