LEAPLifts Developer Manual

Spring 2020

Website Setup

This section contains steps to deploy the website on a clean Ubuntu 18.04.3 LTS (x64) server. It was tested on such a server provided by Digital Ocean, so some configuration details may not be identical on a server from a different vendor. Some steps may be different, or may have already been completed on the target deployment server. In these cases, you may skip a step that appears to be completed, or fill in missing steps.

The instructions include steps that create accounts and set passwords. If you choose to use a different account or database name, It is your responsibility to identify configuration points that must be different. Beyond the deployment instructions, the `.env` file and service configuration file will be the most important to review for changes.

For non-Linux based systems, installation steps *will* differ. Review these instructions to understand what is being accomplished. Then, install the appropriate software using the steps specified by the distributors. Lastly, configure these systems to achieve the same end result on your system.

Before starting these steps, you should login to the server over SSH using the `root` account or any account with sudo access.

Update Package Manager and Packages

You must update your package manager to ensure you get the latest packages. Upgrading your packages is recommended, but not required.

```
sudo apt update
sudo apt upgrade -y
```

Install or Verify Git

To verify git is properly installed, check its version. If an error appears, install git.

```
git --version
```

Install Node.js and Yarn

Configure Nodesource

```
cd ~
curl -sL https://deb.nodesource.com/setup_13.x -o nodesource_setup.sh
```

```
sudo bash nodesource_setup.sh
```

Install Node.js

```
sudo apt install -y nodejs
```

Install Yarn

Due to the font size, the text wraps to the next line. Indentation has been used to make it clear where each line ends.

```
curl -sL https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key
     add -
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee
     /etc/apt/sources.list.d/yarn.list
sudo apt update
sudo apt install -y yarn
```

Verify Node.js version is at least `v13.14.0`

```
node -v
```

Verify Yarn version is at least `1.22.4`

```
yarn -v
```

Install PostgreSQL

```
sudo apt install -y postgresql postgresql-contrib
```

Configure PostgreSQL

You will be creating a Postgres user and database, as well as a Linux user with the same name, to store the data.

If you prefer not to follow these exact steps, you must have a username, password, and database that work to access Postgres before continuing.

Switch to Postgres account

```
sudo -i -u postgres
```

Create a new user

The user will be assumed to be called *carpool* for the remainder of this document. If you choose a different username, then the linux user account created in the following steps must have the same changed name.

```
createuser --interactive
```

Create a database

The database name of *carpooldb* may be changed at your preference

```
createdb carpooldb
```

Exit the *postgres* account

```
logout
```

Create a linux user to access the account

You will need to provide a password, other fields can be left blank.
This username must match the Postgres username created in the earlier step.

```
sudo adduser carpool
```

Login to the new account and access the database

If this does not open a Postgres prompt, address the error shown before continuing

```
sudo -i -u carpool
psql -d carpooldb
```

Set a password while in the Postgres prompt

```
\password
```

Exit the Postgres prompt

```
\q
```

Exit the *newly created* linux account

Return to the `root` or sudo-capable account.

```
logout
```

Install Redis

```
sudo apt install -y redis-server
```

Test redis by running `redis-cli`. If you get a command prompt, you're successful. Use `ctrl+d` to exit the prompt.

Configure Redis

Open `/etc/redis/redis.conf` with the text editor of your choice, and scroll down to the General section, looking for the `supervised` configuration:

```
# If you run Redis from upstart or systemd, Redis can interact with
# your supervision tree. Options:
#   supervised no      - no supervision interaction
#   supervised upstart - signal upstart by putting Redis into SIGSTOP
#                        mode
#   supervised systemd - signal systemd by writing READY=1 to
#                        $NOTIFY_SOCKET
#   supervised auto    - detect upstart or systemd method based on
#                        UPSTART_JOB or NOTIFY_SOCKET environment
#                        variables
# Note: these supervision methods only signal "process is ready."
#       They do not enable continuous liveness pings back to your
#       supervisor.
- supervised no
+ supervised systemd
```

On Ubuntu, change *no* to *systemd*. This improves the ability for the system service manager to control redis. *This step is not mandatory to continue setup, but is recommended for best operation of the website.*

While in the configuration file, ensure that the following line is **not** commented out and appears in the file. This prevents redis from being accessed directly from outside the network. It will appear in the Network section of the document.

```
bind 127.0.0.1 ::1
```

Clone the repository

If you are using a different account from your administrator account to run the Carpool project, switch to it. This tutorial document uses the *carpool* account created in the earlier steps. Going forward, the rest of the setup process should take place in that account except where explicitly stated.

After cloning or downloading the folder to the system, change into the directory:

```
cd cs470-leap-carpooling
```

Configure environment variables

Copy the template environment file. The `.env` file is ignored by git, so it won't be committed.
`cp .env_template .env`

You will now need to fill in the environment variable values appropriately. Most values will be filled correctly already, however you will need to configure some:

PGPASSWORD	The password to login to the Postgres account
SESSION_SECRET	A random value to securely sign session values. Use a password generator to set this
GOOGLE_MAPS_BROWSER_KEY GOOGLE_MAPS_SERVER_KEY	Refer to the Google Maps section of this document to generate these keys.
CONTACT_EMAIL	The email shown on the site that the user could contact for more information. For development purposes, this can be any email address.

If you want email notifications or for CAS login to be enabled, refer to the sections in this document to enable those services.

If you are not a *student* at Lafayette college but do have a CAS login, and need to be able to verify the CAS login process works properly during deployment, set `CAS_ROLE_CHECK_DISABLED=true` in the `.env` file to disable this verification step during the CAS process.

Install Project Dependencies

This command will install all the Javascript dependencies required by the project
`yarn`

Migrate The Database

This command will create the database tables
`yarn run migrate:latest`

Verify Server Starts

```
CAS_DISABLED='true' yarn run start
```

If the server exits, read the messages and correct the error. You should be able to browse to *http://[your:ip:here]:8000* and browse the site.

By setting `CAS_DISABLED`, you won't have to configure CAS to test the site. If this is not configured, login will attempt and fail to use CAS.

Install NGINX

This step is best done from an administrator account. If you switched away to clone the project, switch back to an admin account now.

```
sudo apt install -y nginx
```

You can verify that NGINX is running by loading your website at the default port *http://[your:ip:here]*.

Configure HTTPs

If you are using a different operating system, or a server other than NGINX, follow the instructions at certbot.eff.org/lets-encrypt.

Install Certbot

```
sudo apt install -y software-properties-common
sudo add-apt-repository universe
sudo add-apt-repository ppa:certbot/certbot
sudo apt update
sudo apt install -y certbot python3-certbot-nginx
```

Generate Certificates

You will be asked questions, including which domain to generate the certificate for. That domain must be active already to properly generate the certificate.

```
sudo certbot certonly --nginx
```

Configure NGINX

Replace the default NGINX configuration (located at `/etc/nginx/sites-enabled/default`) with the contents of the template provided in the project, in the `nginx` directory.

Replace all four instances of *carpool.cs.lafayette.edu* with the correct domain name for the website, if it's different.

Restart NGINX

```
sudo systemctl restart nginx
```

Setup Carpool Service

To have the project start-up automatically on server boot, enable the service provided in the `service` directory.

Copy the `carpool.service` file to `/etc/systemd/system/carpool.service`.

Register the service and start it up

```
cd /etc/systemd/system
chmod +x carpool.service
systemctl daemon-reload
systemctl enable carpool.service
systemctl start carpool
```

The website should now be running! There may be a delay while it compiles. Use the following commands to manage the service:

Check status and view output logs

```
systemctl status carpool
```

Start, stop, restart service

```
systemctl [start|stop|restart] carpool
```

Email Notifications

About

To facilitate the carpooling process, LEAP Lifts has an opt-out email notification service that sends users the following emails:

- A welcome email after the user completes the onboarding form
- A trip processing email each time the user submits a trip request
- A trip match found email when the matching service has found the user a match for a particular trip

- A trip confirmation email when both members of a match confirm their trip
- A trip reprocessing email when one member of a match cancels the matched request or rejects the match to inform the other member that they no longer have a match and a new one is being found
- A trip reminder email on the day before an upcoming trip

To make use of this service, and to allow users to contact LEAP Lifts for help, a few steps must be taken to set up email sending.

Setup

1. **Set the contact email environment variable**
   If you have not already done so, set the `CONTACT_EMAIL` environment variable to the email address you would like users to contact for help and questions and that you would like emails from LEAP Lifts to be sent from.
2. **Set the SendinBlue environment variables**
   To avoid having to set up a dedicated SMTP server to send email notifications, we set up our SMTP transporter against a well-known SMTP service provider: SendInBlue. We chose SendInBlue because of its high deliverability rate and its free plan, which allows customers to send 9000 emails a month (300 a day) for free. Given the size of Lafayette's student body, SendInBlue's generous free plan would likely be able to completely cover our email sending needs.
   a. Sign up for a free SendInBlue account here.
   b. From the dropdown in the upper right corner, select the SMTP & API option.
   c. Navigate to the SMTP tab, look under the "Your SMTP keys" heading, and copy the value for master password.
   d. Set the `SENDINBLUE_EMAIL` environment variable to the email address you used for your SendInBlue account (which can be the same as the contact email, but doesn't have to be), and set the `SENDINBLUE_PASSWORD` environment variable to the master password you copied above.

More Options

While we determined SendInBlue was a good option for development and testing, and think it is also a viable option for production, you can easily set up the email transporter against another SMTP service provider if you prefer. The Nodemailer module, which we use for email sending, knows the SMTP connection details for a variety of well-known service providers. To switch to another of these service providers, navigate to `src/utils/emails.ts`. Within the declaration of the transporter object, replace 'SendInBlue' with the service name of your provider, as listed here. Then create an account with the service provider and modify the service provider email/password environment variables accordingly.

CAS Login

About

CAS (Central Authentication Service) is a standard protocol which allows websites to authenticate users using an external account system. Lafayette offers a CAS login portal, which makes it easy for users of this site to login using Lafayette credentials.

Setup

Before using CAS on the website, the domain name must be whitelisted. At the time of writing, `carpool.cs.lafayette.edu` had been whitelisted for this purpose. If the domain name has been changed, a request must be sent to the Identity and Access Management department to allow the new site to authenticate over CAS.

Once authorization has been granted to use CAS, the configuration files must be verified to be correct. In the `.env` file, there are two configuration lines referencing CAS (If you don't have these, refer to `.env_template` and copy them over):

CAS_ENDPOINT	Default value: `cas.lafayette.edu/cas` Purpose: Identifies what CAS service to use. This value should not be changed unless Lafayette changes their CAS configuration, or the project is switched to use a different CAS page.
CAS_SERVICE_URL	Default value: `https://carpool.cs.lafayette.edu/sessions/handle-ticket` Purpose: The complete URL to return to after authentication is completed. If the project is deployed to another domain, then the portion `carpool.cs.lafayette.edu` should be switched to that domain. The rest should be left unchanged.

Google Maps API

About

The Google Maps API is used throughout our project to get location and map data. It is used in the settings page and new trip request form in order to autocomplete location data with the full required address field. This API is also used to determine how far apart different locations are and thus determine which if any trip matches can be made. Lastly this API is used to produce

the trip map within the trip details page which shows a visual representation of what the matched trip will entail.

Setup

Two API keys are required, with appropriate authorizations on each. Navigate to console.cloud.google.com to create these authorization keys.

1. Create a project, identifying it as corresponding to this website
2. Enable the appropriate APIs (All within Google Maps)
    a. Directions API
    b. Distance Matrix API
    c. Maps Embed API
    d. Maps JavaScript API
    e. Places API
3. Create two API keys
    a. Server-Side Private Key needs no referrer restrictions, and should have just the following APIs enabled:
        i. Distance Matrix API
        ii. Places API
    b. Client-Side API Key could have an HTTP referrer restriction set to the domain of the website, but it hasn't been tested with such a restriction. It should have the following APIs enabled:
        i. Places API
        ii. Maps JavaScript API
        iii. Directions API
4. Copy these two keys into your `.env` file as `GOOGLE_MAPS_SERVER_KEY` and `GOOGLE_MAPS_BROWSER_KEY` respectively.

Problems

If problems arise, there should be error messages explaining the issue. For the embedded map and autocomplete services, examine the browser console logs for the error messages. For the distance matrix and geocoding services, examine the server logs.

If the logs indicate unauthorized API usage, review that the API is authorized to access the services listed above.

HTTPS

During the setup process, HTTPS is enabled using Certbot and Let's Encrypt. If you need to renew the certificates (which must be done every 90 days, or automated), run the following command:

```
sudo certbot renew
```

Read more about Certbot here: certbot.eff.org

Technologies

Front End

Bootstrap

Bootstrap is used within this software for faster and easier front end development. More specifically, it is used in a variety of features within our project including in our buttons, layouting and overall frontend responsiveness.

JavaScript, CSS, and HTML

A combination of Javascript, CSS and HTML is used for the front end programming. This is the standard suite of software used for any website, since these are the systems that web-browsers understand.

Backend

TypeScript

Our backend code is written in TypeScript files. TypeScript is an extension of Javascript that introduces types to the language (as the name implies). The TypeScript compiler checks that all the types are valid, then converts it to JavaScript.This allows for the use of types for functionality that benefits from types. However, it also allows us to use JavaScript when types are not needed. Much of the code written in the TypeScript files is identical to JavaScript code.

Express

Express is used for the web server framework throughout our project. The web server framework handles the features that are common to nearly any website. This includes handling an incoming request, and running the code we specify based on what URL the user is requesting.

Pug

Pug is used for the frontend rendering of the HTML files within the project. On any given page, there's dynamic information to fill in, like the name of the current user that appears in the

upper-right hand corner of the page. Pug allows us to write a template and fill in the variables easily.

KnexJS and PostgreSQL

Our database system uses PostgreSQL to store records. Rather than interface directly with PostgreSQL, we use a package called Knex.js. Knex builds database queries and helps prevent typos and the like. It also handles preventing SQL injections.

File Structure

The following is a general overview of key parts of the project structure to help developers understand the project organization and determine where new features should be implemented.

`migrations/` Stores migration files, which allow developers to define sets of schema changes to simplify upgrading databases. Migrations should be created using the Knex CLI. The documentation for the CLI is available at knexjs.org.

`nginx/` Stores NGINX default configuration.

`services/` Stores systemd default service configuration.

`public/uploads/` Stores the profile photos users upload on the settings page.

`src/`

- **`jobs/`** Defines and schedules jobs to perform pairing, email sending, and block removals.
- **`middleware/`** Defines functions that need to access or modify the HTTP request and response objects, like methods that perform authentication or add variables needed by the frontend.
- **`models/`** Defines interfaces for various data types in our program, like User, TripRequest, or TripMatch, which allows for easy interaction with the database.
- **`routes/`**
  - **`errors/`** Defines routes for internal and not-found errors.
  - **`settings/`** Defines routes for the settings and onboarding pages.
  - **`trip-requests/`** Defines the route for a user cancelling a trip.
  - **`trips/`** Defines routes for the trip request, trip detail, and dashboard pages.
  - **`unblock-user/`** Defines the route for a user unblocking a blocked user.
- **`utils/`** Stores a variety of miscellaneous utility functions, like those relating to the routing algorithm, creation and sending of emails, geocoding, and location formatting.
- **`validation/`** Defines schemas and constraints for the input fields throughout the website, allowing us to perform both client-side and server-side validation.

`static/` Stores static assets, like photos and icons displayed on the website.

`css/` Defines a small collection of site styles.

`js/` Stores scripts used throughout the site which allow for client-side input validation, autocomplete for certain fields, and more.

`views/` Pug files for miscellaneous site pages like the home, about, and help pages.

`errors/` Pug files for internal and not-found errors.

`sessions/` Pug file for the login page used for development purposes.

`settings/` Pug files for the onboarding and settings pages.

`trips/` Pug files for the trip request, trip detail, and dashboard pages.

Running the Application

Before running the application, ensure that the entire setup process has been completed. When this is the case, the application will be running.

If you make changes and need to restart it:

```
sudo systemd restart carpool
```

If you need to stop or start the application for any reason:

```
sudo systemd stop carpool
sudo systemd start carpool
```

If you want to test the program faster, stop the service with the above command, then use `yarn start` (while in the project directory) to start it directly. This may make debugging easier.

Backups and Restore

All the data (except for cached information, which resides in redis), is stored in the PostgreSQL database. To backup the system, perform a dump of the PostgreSQL database.

```
pg_dump carpooldb > backup_file_name
```

To restore the backup, you must start by creating an empty database (`template0` is a built in template that contains no content).

If your old database was corrupted, delete whatever is left first. Do not run the knex.js migrations, since that may interfere with the backup restoration process, which assumes an empty database.

```
createdb -T template0 carpooldb
psql carpooldb < backup_file_name
```

Documentation for backing up and restoring a PostgreSQL database is located on the PostgreSQL website, and should be considered the definitive guide to backing up and restoring a PostgreSQL database: www.postgresql.org/docs/current/backup.html.

Since there are no other data stores, this process should restore your system to the exact previous state. Since the cache layer hasn't been included, you may notice an increase in Google Maps API calls while the cache layer is re-created.

Database Information

PostgreSQL

We used a relational PostgreSQL database for our project. Our database has 4 tables which keep track of all required data including user information, trip information and trip matching and rejection information. The name of the database can be configured in the .env file. The field corresponding to the database name is named PGDATABASE.

When you start the server with the setup instructions provided in the earlier parts of this document, a script to setup your database will be run. You can run it again at any time (if a new migration is added, for example) by executing the following command while in the project directory:

```
yarn migrate:latest
```

To learn more about migrations, refer to the Knex.js documentation located at knexjs.org.

Database Tables

pair_rejections

This table keeps track of users who have been blocked by each user. Each entry contains the user id of the person who was blocked and the user id of who they were blocked by. Records are added to this table when a user rejections a match with another user. The users a user has blocked are displayed in the blocked users table on the settings page. Records are removed from this table once the unblock button is pressed on the settings page. Records are also removed after a trip has passed if the reason for blocking was not because they specifically didn't want to ride with the other person.

Column Name	Column Type	Column Attributes
id	integer	Primary Key

blocker_id	integer	Non-Null Foreign Key to Users Table
blockee_id	integer	Non-Null Foreign Key to Users Table
expire_after	date	

trip_matches

This table keeps track of all the trip matches that have been made. In addition to containing the user id of the driver and rider it also contains the common travel days that the trip can occur on, if either party of the trip has confirmed and whether or not notification emails have been sent to the users about their trip match. Records are removed from this table if the trip is canceled by either user or if either user rejects the particular trip match. Records are updated whenever either party of the trip confirms the trip.

Column Name	**Column Type**	**Column Attributes**
id	integer	Primary Key
driver_request_id	integer	Non-Null Foreign Key to Trip Requests Table
rider_request_id	integer	Non-Null Foreign Key to Trip Requests Table
first_date	date	Non-Null
last_date	date	Non-Null
rider_confirmed	boolean	Non-Null
driver_confirmed	boolean	Non-Null
first_portion	enum: driver | rider	Non-Null
notification_sent	boolean	Non-Null
created_at	timestamp with timezone	Non-Null

trip_requests

This table contains the information regarding trip requests from users. Each time a new trip request is submitted by the user a new entry is added to this table. This table also keeps track

of information regarding the particular user's desired trip such as the location they will be traveling to and from, the dates they are available, and whether they wish to ride or drive.

Column Name	Column Type	Column Attributes
id	integer	Primary Key
member_id	integer	Non-Null Foreign Key to Users Table
role	enum: driver | rider	Non-Null
location	text	Non-Null
location_description	text	Non-Null
deviation_limit	integer	Non-Null
direction	enum: towards_lafayette | from_lafayette	Non-Null
first_date	date	Non-Null
last_date	date	Non-Null
created_at	timestamp with timezone	Non-Null

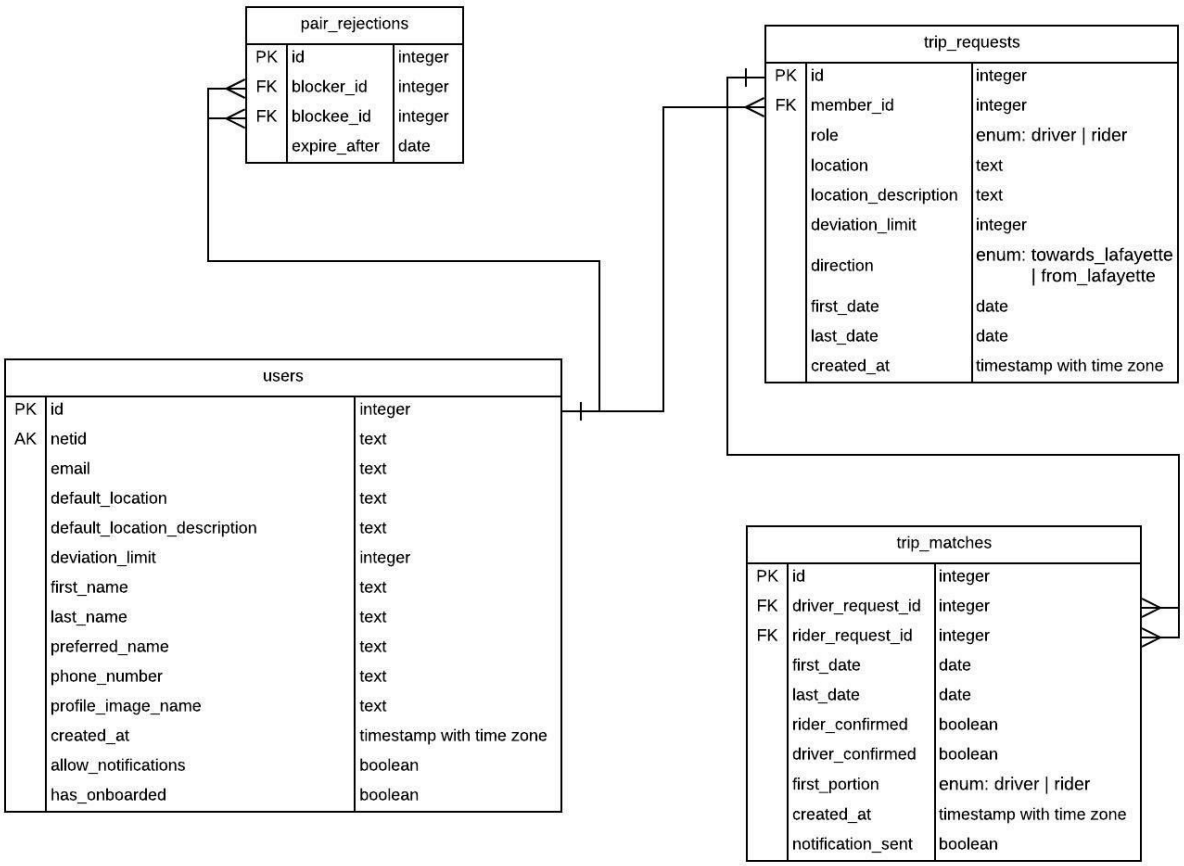
users

This table contains the information corresponding to each user that has used our system. Some of the information such as role, netid and email are populated from CAS. This table also contains information such as phone number, default location, default deviation limit, and profile image which the user can set once they have started using our website.

Column Name	Column Type	Column Attributes
id	integer	Primary Key
netId	text	Non-Null, Unique
email	text	
default_location	text	
default_location_description	text	
deviation_limit	integer	

first_name	text	Non-Null
last_name	text	Non-Null
preferred_name	text	
phone_number	text	
profile_image_name	text	
created_at	timestamp with timezone	Non-Null
allow_notifications	boolean	Non-Null
has_onboarded	boolean	Non-Null

Entity Relationship Diagram



Data Dictionary

Attribute	Type	Description	Relation
id	integer	Primary Key	pair_rejections
blocker_id	integer	Foreign Key to users(id)	pair_rejections
blockee_id	integer	Foreign Key to users(id)	pair_rejections
expire_after	date	The date the block expires if applicable	pair_rejections
id	integer	Primary Key	users
netId	text	CAS provided username	users
email	text	Email address of the user	users
default_location	text	User's default location they will travel to/from if set in the form as a Google Maps place ID	users
default_location _description	text	User's default location they will travel to/from if set in the form as a string	users
deviation_limit	integer	Default limit if set in settings	users
first_name	text	First name from CAS	users
last_name	text	Last name from CAS	users
preferred_name	text	Prefered first name if set	users
phone_number	text	User phone number	users
profile_image _name	text	Name of the file of the user's uploaded profile image	users
created_at	timestamp with timezone	Timestamp of when the user was created	users
allow _notifications	boolean	True if the user wants email notifications false otherwise	users

has_onboarded	boolean	True if the user has completed the onboarding page false otherwise	users
id	integer	Primary Key	trip_requests
member_id	integer	Foreign Key to users(id)	trip_requests
role	enum: driver | rider	The role of the user in a specific trip request	trip_requests
location	text	Google Maps place id of the location the user inputted for trip	trip_requests
location _description	text	String in English of the location the user inputted	trip_requests
deviation_limit	integer	Deviation limit inputted by the user for the trip	trip_requests
direction	enum: towards_lafayette | from_lafayette	Whether the trip is starting or ending at Lafayette College	trip_requests
first_date	date	First date the user is available to make this trip	trip_requests
last_date	date	Last date the user is available to make this trip	trip_requests
created_at	timestamp with timezone	Timestamp of when this trip request was made	trip_requests
id	integer	Primary Key	trip_matches
driver_request_id	integer	Foreign Key to trip_requests(id)	trip_matches
rider_request_id	integer	Foreign Key to trip_requests(id)	trip_matches
first_date	date	First common date on which this trip can be made	trip_matches
last_date	date	Last common date on which this trip can be made	trip_matches
rider_confirmed	boolean	True if the rider confirmed the trip false otherwise	trip_matches
driver_confirmed	boolean	True if the driver confirmed the trip false otherwise	trip_matches

first_portion	enum: driver | rider	Indicates whether the driver or the rider is the one who rides alone for a portion of the trip. This should also match who is going out of their way to carpool.	trip_matches
notification_sent	boolean	True if both users were notified of this trip false otherwise	trip_matches
created_at	timestamp with timezone	Timestamp of when this match was created	trip_matches

Future Modifications and Directions

While LEAP Lifts is fully functional and ready for Lafayette student use as is, there are a variety of features and directions that we think could make LEAP Lifts even more dynamic and useful in the future. We also anticipate that future owners may want to make changes to some of the website content, like the site name, the About page blurb or the FAQ featured on the Help page. Here, we will note how to make those changes, and also explain our vision for how LEAP Lifts could be expanded in the future.

Modifying the Site Name

To allow for easy transfer of site ownership, we defined the name of the site as an environment variable. Anywhere “LEAP Lifts” appears on the website or in emails can be modified simply by modifying the `SITE_NAME` environment variable to equal the owner’s desired website name.

Modifying the About Page

The text featured on the About page can be found and modified in `views/about.pug`.

Modifying the FAQ

The FAQ featured on the Help page can be found and modified in `views/help.pug`. To add a question to this section, copy the following code segment (modify the bold segments, all numbers should be updated with the current question number) and paste it at the same level as the other questions in the accordian structure:

```
.card
  .card-header.p-1#question**Six**
    h2.mb-0
      button.btn.btn-block.shadow-none.text-left.font-weight-bold(
```

```
      data-toggle="collapse"
      data-target="#answerSix"
      aria-expanded="false"
      aria-controls="answerSix"
    )
      i.fas.fa-plus.mr-2
      | Write your question here.
.div.collapse#answerSix(data-parent="#faq" aria-labelledby="questionSix")
  .card-body
    p.mb-0
    | Write your answer here.
```

Future Features

- **Optimize Car Space:** To minimize our carbon footprint, a future iteration of LEAP Lifts could ask drivers to specify how much space they have in their car and then match at most that many riders with a single driver, instead of doing one-to-one matching.
- **More Variables:** To maximize the potential of users accepting their matches, more variables could be added to the trip matching algorithm. For instance, if a driver is looking for riders primarily so that they don't have to drive by themselves for a long trip, riders who are closer to the driver's home address could be prioritized. This is just one example - there are a variety of variables that could be added to the trip request form to increase the compatibility of potential matches.
- **All-in-One:** To streamline the carpooling process, we could incorporate more aspects of trip planning into our website. For instance, we could incorporate sharing the cost of gas and tolls, or add messaging to allow users to contact each other within the context of the website. These aren't necessary features, but they could make the carpooling process more convenient for users.
- **Expand Customer Base:** LEAP Lifts was designed for Lafayette students, but it has the potential to be useful for others, like the families of Lafayette students. Future iterations of the website could add support for non-Lafayette students, although this would require the exploration of alternative methods of authentication.

We hope these suggestions and the information included in this document guide developers as they work to get LEAP Lifts up and running, and look to improve it in the future!