Trip Server

Frank Dean

This User Guide is for using Trip, version 2.2.1, 4 July 2023 in a browser. Trip is an application supporting trip recording and itinerary planning.

Table of Contents

1 Overview

This document describes using `trip-server` — Trip Recording and Itinerary Planning, version 2.2.1, 4 July 2023.

2 Invoking trip-server

2.1 Synopsis

trip-server [-h | –help] [-v | –version] [-s | –listen { address }] [-p | –port { port }] [-r | –root { directory }] [-e | –expire-sessions] [-c | –config-file { filename }] [-u | –upgrade] [-V | –verbose]

2.2 Description

`trip-server` is an application supporting trip recording and itinerary planning.

The intended use is for a hiker, mountain-biker or other adventurer, to be able to publish and share their planned itinerary, then subsequently log their positions at intervals, allowing someone else the ability to monitor their progress.

In the event of contact being lost, the plans and tracking information can be passed to rescue services etc., to assist with locating the missing adventurer.

2.3 Options

-h, –help Show help, then exit.

-v, –version
: Show version information, then exit.

-s, –listen=ADDRESS
: Listen address, e.g. 0.0.0.0

-p, –port=PORT
: Port number, e.g. 8080.

-r, –root=DIRECTORY
: Document root directory.

-e, –expire-sessions
: Expires any active user web sessions.

-c, –config-file=FILENAME
: Configuration file name.

-u, –upgrade
: Upgrade the database.

-V, –verbose
: Verbose output

3 Upgrading from Trip Server v1

3.1 Database URI

There are some differences between what the Node.js pgmodule (`https://github.com/brianc/node-postgres`) will accept and those defined as the PostgreSQL libpq Connection Strings. (`https://www.postgresql.org/docs/current/libpq-connect.html#LIBPQ-CONNSTRING`)

The former allows a `socket` scheme and a `database` parameter. The latter requires `postgresql` and `dbname` to be use respectively. It should be possible to form the connect string as a valid `libpq` URI without any query parameters, which the Node.js `pg` module will also work with, including connecting via a socket. This allows both versions to run using the same application configuration file.

E.g.

Connect via a socket, using the `peer` method:

```
uri: "postgresql:///%2Fvar%2Frun%2Fpostgresql/trip"
```

3.2 Database Changes

The `pgcrypto` extension is needed for user password validation. Run the following command in the Trip database:

```
$ psql trip
CREATE EXTENSION pgcrypto;
```

This is also executed when running `trip-server` with the `--upgrade` option.

3.2.1 Session Table

There is a new table to handle user sessions. It is created after running `trip-server` with the `--upgrade` option.

3.2.2 Passwords

The Blowfish algorithm appears to have been changed in PostgreSQL 13. The following SQL will update old style encrypted passwords to work in PostgreSQL 13.

```
UPDATE usertable SET password = '$2a' || SUBSTRING(password, 4, 57)
WHERE password NOT LIKE '$2a%';
```

This is also executed when running `trip-server` with the `--upgrade` option.

4 PostgreSQL Setup

The following packages are required to run the application with PostgreSQL (`https://www.postgresql.org`):

- postgis

On a Debian distribution (`https://www.debian.org`), install the `postgis` and `postgresql` packages.

4.1 Creating a Database User

Create a user for the application, with minimal rights, entering the password to be used by the `trip` system user, when prompted by the `createuser` utility:

```
$ sudo -v
$ sudo -u postgres createuser -P trip
```

4.2 Creating the trip Database

Create a database for use by the application:

```
$ sudo -u postgres createdb trip --owner=trip --locale=en_GB.UTF-8
```

If the default locale for the database does not match the running user's locale, e.g. `en_GB.UTF-8`, specify the locale when creating it. E.g.:

```
$ sudo -u postgres createdb trip --owner=trip --locale=en_GB.UTF-8 \
  --template=template0
```

4.2.1 Database User Permissions

Allow the `trip` user access to the database by editing the PostgreSQL database configuration:

```
$ sudoedit /etc/postgresql/13/main/pg_hba.conf
```

Insert a line, usually as the first non-comment line, to `pg_hba.conf`, with:

- TYPE = local
- DATABASE = trip
- USER = trip
- ADDRESS =
- METHOD = md5

e.g

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   trip          trip                                    md5
```

Restart PostgreSQL:

```
$ sudo systemctl restart postgresql
```

Test connecting to the database as the `trip` user and create the `pgcrypto` extension.

```
$ psql -U trip -d trip
Password for user trip:
psql (13.5 (Debian 13.5-0+deb11u1))
```

```
Type "help" for help.
```

Whilst connected to the database, create the `pgcrypto` module:

```
trip=> CREATE EXTENSION pgcrypto;
```

4.3 Creating Tables and Roles

As a Unix user who is also a postgresql superuser:

```
$ cd ./provisioning/schema
$ psql trip <10_trip_role.sql
$ psql trip <20_schema.sql
$ psql trip <30_permissions.sql
```

Optionally, populate the database with data that can be used to perform end-to-end tests. Do not insert the test data into a production database as it contains default application admin user credentials.

```
$ psql trip <90_test-data.sql
```

4.3.1 Lookup Tables

The following tables are used to define lookup values for select boxes in the web application:

`waypoint_symbol`
: Key-value pairs describing waypoint symbols. The `key` is written to waypoint entries when downloading GPX files.

`track_color`
: Key-value pairs together with an HTML colour code. The 'key' is written to track entries when downloading GPX files and the HTML colour code is used to render the tracks on the itinerary map page.

`georef_format`
: Key-value pairs define how to format output of latitude and longitude values on the itinerary waypoint edit page. Format parameters are defined using the `{%}` symbol and have the following meanings:

- %d - degrees
- %m - minutes
- %s - seconds
- %D - zero prefix single digit degree values
- %M - zero prefix single digit minute values
- %S - zero prefix single digit second values
- %c - output the cardinal value, S, E, W or N
- %i - output a minus sign for W and S
- %p - output a minus sing for W and S and a plus sign for E and N

E.g. a format string of %d°%M'%S"%c would result in a lat/long value of '`1.5,-2.5`' being displayed as 1°30'00"N 2°30'00"W

Scripts to create default values for these lookup tables are in the `./provisioning/schema` folder;

- `60_waypoint_symbols.sql`

- 40_path_colors.sql
- 50_georef_formats.sql

The default waypoint symbols and track colours are generally appropriate for Garmin devices. In fact, the colours are the only ones allowed by the Garmin Extensions XSD (http://www8.garmin.com/xmlschemas/GpxExtensionsv3.xsd).

4.3.2 Indexes for Query Performance

The location table has an index that is clustered on the time column to improve the query performance of date range queries. If the table becomes large and performance degrades, run the psql cluster command from time-to-time to re-cluster it. Note an exclusive lock is placed on the table for the duration of the cluster command execution.

See http://dba.stackexchange.com/questions/39589/optimizing-queries-on-a-range-of-timestar for more information.

4.4 Creating an Initial Admin User

An initial admin user needs to be created in the database. Thereafter, that user maintains other users using the web application. Creating the initial admin user fundamentally consists of making entries in the usertable, role and user_role tables.

Firstly, create the entries in the role table by running the following script using psql:

```
INSERT INTO role (name) VALUES ('Admin'), ('User');
```

An initial admin user can be created similarly to the following, replacing each value appropriately:

```
INSERT INTO usertable (firstname, lastname, email, uuid, password, nickname)
VALUES ('admin', '', 'admin@trip.test', gen_random_uuid(),
crypt('SECRET', gen_salt('bf')), 'admin');

INSERT INTO user_role (user_id, role_id)
VALUES (
(SELECT id FROM usertable u WHERE u.nickname='admin'),
(SELECT id FROM role WHERE name='Admin')
);
```

5 Configuration

Please refer to the comments in the `trip-server-dist.yaml` file for information on the application's configuration parameters.

By default the application looks for its configuration file under the installation prefix (usually `/usr/local`) named `./etc/trip-server.yaml`. Alternatively, it can be overridden with the `--config-file` command line option.

The configuration file should be readable by the user running `trip-server`, e.g. `trip`, but not world-readable.

On a Debian system (`https://www.debian.org`), you can create a system user to run the application as follows:

```
sudo adduser trip --system --group --home /nonexistent --no-create-home
```

On other systems, you will probably need to use the less friendly `useradd` command.

Run `trip-server` using the `trip` user and set the file permissions as follows:

```
$ sudo chmod 0640 /usr/local/etc/trip-server.yaml
$ sudo chown trip:trip /usr/local/etc/trip-server.yaml
```

Refer to the Vagrant configurations in the source distribution of trip-server, under the `./provisioning` directories for information relating to configuring and running the application as a daemon under `systemd`.

5.1 Tile Server Configuration

Most if not all tile server providers have policies that you must comply with and there may be sanctions if you fail to do so. E.g. If you are using the OpenStreetMap (`https://www.openstreetmap.org/`) tile server, read and comply with their Tile Usage Policy (`https://operations.osmfoundation.org/policies/tiles/`). Please ensure you configure the following entries correctly for the appropriate element of the `tile.providers` section(s) of `config.yaml`.

`userAgentInfo`
: This is the e-mail address at which the system administrators can contact you

`refererInfo`
: A link to a public website with information about your application's deployment

Note these entries are sent in the HTTP header of each tile request and will therefore end up in system logs etc. Currently the tile requests are sent over HTTP, therefore you should not mind this data being exposed.

The `tile.providers[x].mapLayer` entries provide the ability to display tile map attributions most if not all tile providers require you to display.

The `mapLayer.name` attribute will be displayed when the map layers icon is activated. Only `xyz` map types are supported, so the `mapLayer.type` attribute should always be `xyz`.

Map attributions are displayed on the map using the `mapLayer.tileAttributions` section of the `tile.providers` attribute, which allows attributions to be rendered with appropriate HTML links. The `tileAttributions` are an array of items that have either text, text and link or just link attributes. If the entry contains just text, the text will be displayed

in the map attribution. If a link is included, the text will be wrapped in HTML link tags and included in the map attribution. The entries are displayed in the sequence they have been defined.

For development, if no map tile provider is configured, dummy tiles are displayed showing the x, y & z coordinates of the tile. Alternatively, you can use Docker (`https://www.docker.com`) to run a small tile server. See the section on Docker in this application's README in the source distribution and the various `docker-compose-map-*.yaml` configuration files for hints.

5.2 Elevation Data

The Consortium for Spatial Information (`http://srtm.csi.cgiar.org/`) (CGIAR CSI) make Digital Elevation Model (`https://en.wikipedia.org/wiki/Digital_elevation_model`) data covering about 80% of the globe, available for download. It has been sourced and enhanced from data gathered by the NASA Shuttle Radar Topographic Mission (SRTM).

From the main page of the CGIAR CSI website, follow the link to `SRTM Data` to download `zip` files that contain `tiff` files with 5m x 5m elevation data.

Extract the `tiff` files to a folder, e.g. `/var/local/elevation-data` and configure an `elevation` section in `config.yaml`, e.g.

```
elevation:
  tileCacheMs: 60000
  datasetDir: /var/local/elevation-data/
```

When the Trip Server application is started, it reads all the `tiff` files in the folder specified by the `elevation.datasetDir` parameter and creates an in memory index containing the area covered by each tile. When elevation data is required for a specific location, the relevant tile is loaded, the response provided, and the tile retained in memory for the number of milliseconds specified by the `elevation.tileCacheMs` parameter.

The `tiff` files take up a lot of space. Where space is at a premium, consider storing them in a compressed file system, e.g. on Linux use Squashfs. (`http://squashfs.sourceforge.net`)

e.g.

1. Download files to `~/downloads/srtm`

   ```
   $ mkdir -p ~/downloads/srtm
   $ cd ~/downloads/srtm
   $ wget http://srtm.csi.cgiar.org/wp-content/uploads/files/srtm_5x5/tiff/srtm_72_22
   ```

2. Extract the tiff files to `~/tmp/tiff`

   ```
   $ mkdir -p ~/tmp/tiff
   $ cd ~/tmp/tiff
   $ find ~/downlods/srtm -name '*.zip' -exec unzip -n '{}' '*.tif' \;
   ```

3. Create a Squashfs compressed file containing the `tiff` images

   ```
   $ mksquashfs ~/tmp/tiff /var/local/elevation-data.squashfs -no-recovery
   ```

   The `-no-recovery` option is to stop Squashfs leaving a recovery file behind in the destination folder. However, it does mean that should the operation fail, there is no

recovery information to unwind the command. This is probably more of a potential problem when appending to an existing Squashfs file.

4. Optionally, delete or archive the downloaded `zip` files to free up space.
5. Download more files, extract them and squash them using the above steps. Repeating the `mksquashfs` command as above will append to an existing Squashfs file.
6. You can list the contents of the Squashfs file with:

```
$ unsquashfs -i -ll /var/local/elevation-data.squashfs
```

7. Test mounting the Squashfs file

```
$ mkdir -p /var/local/elevation-data/
$ sudo mount -t squashfs /var/local/elevation-data.squashfs /var/local/elevation-da
$ ls /var/local/elevation-data/
$ sudo umount /var/local/elevation-data
```

8. Add an entry to `/etc/fstab` to mount the Squashfs file on boot:

```
$ echo '/var/local/elevation-data.squashfs \
/var/local/elevation-data squashfs ro,defaults 0 0' \
| sudo tee -a /etc/fstab
```

9. Mount using the `/etc/fstab` entry:

```
$ sudo mount /var/local/elevation-data
$ ls /var/local/elevation-data
$ sudo umount /var/local/elevation-data
```

10. If need be in the future, you can extract the files from the Squashfs file with:

```
$ unsquashfs -i /var/local/elevation-data.squashfs
```

Which will extract all the files to a sub-folder of the current working folder named `squashfs-root`.

Use the `-f` parameter if the `squashfs-root` folder already exists.

11. To extract select files, create another file containing the names of the files to be extracted, prefixed by a forward-slash. e.g. `/srtm_11_03.tiff`.
12. $ unsquashfs -i -e list-of-files.txt /var/local/elevation-data.squashfs
13. See SquashFS HOWTO (`http://tldp.org/HOWTO/SquashFS-HOWTO/creatingandusing.html`) for more information

6 Internationalisation and Localisation

A quick way to test localisation is to temporarily set the environment variable `LC_ALL` when running the server, e.g.

```
$ LC_ALL=es_ES.UTF-8 ./src/trip-server
```

To update the PO files for translation:

```
$ cd po
$ make update-po
```

After updating the translations, run the same command to re-create the binary files, then install them in the correct location with:

```
$ make
$ sudo make install
```

7 Proxy Web Servers

The section describes deploying Trip with either the Apache or Nginx web servers.

7.1 Nginx Web Server

Setting up Nginx isn't documented here but can readily be determined by looking at the Vagrant setup scripts under the `./provisioning/` folder in the source distribution, or by deploying using Vagrant and examining the working Vagrant installation.

7.2 Apache

Optionally, the application can be run behind an Apache (`http://httpd.apache.org/`) server, proxying requests to the application.

This has the benefit of allowing the application to co-exist with other applications on the same server instance all running on the standard port 80. Security of the server can also be enhanced by installing and configuring the mod-security Apache module (`https://modsecurity.org/`).

7.2.1 Reverse Proxy Configuration

Configure Apache 2 to enable the `mod_proxy` and `proxy_wstunnels` modules. On Debian this can be done with:

```
$ sudo a2enmod proxy
$ sudo a2enmod proxy_wstunnel
$ sudo a2enmod rewrite
```

The application should be run over HTTPS to keep the login credentials secure, otherwise others can see and re-use those credentials.

Modify the server configuration to implement the following Apache rewrite rules. Note that the default `socket.io` path is prefixed with `wstrack\` so that multiple applications using websockets can be run on the same Apache server using different prefixes. (TRIP uses websockets to provide updates to the tracking map.) The TRIP web client app will prefix the path when it is not calling a localhost URL. These rules need to be in a `<VirtualHost _default_:443/>` or `<Directory/>` section of the `mod_ssl` configuration file.

```
RewriteEngine on
RewriteCond %{REQUEST_URI}  ^/wstrack/socket.io  [NC]
RewriteCond %{QUERY_STRING} transport=websocket    [NC]
RewriteRule /wstrack/(.*)           ws://localhost:8080/$1 [P,L]
```

Add the following to `trip.conf` outside the `<directory\>` directive:

```
<IfModule mod_proxy.c>
  ProxyPass /wstrack/socket.io/ http://localhost:8080/socket.io/
  ProxyPassReverse /wstrack/socket.io/ http://localhost:8080/socket.io/

  ProxyPass /trip/rest http://localhost:8080
  ProxyPassReverse /trip/rest http://localhost:8080
</IfModule>
```

7.2.2 Redirecting to HTTPS

It is useful to ensure all users use HTTPS by providing a redirect rule to redirect any HTTP requests to use HTTPS. However, some logging clients do not support HTTP, so it may be preferable to exclude the logging patterns from redirection. Generally, the logging URLs will be of the form `http://${HOST}:${PORT}/trip/rest/log_point`.

This rule will redirect URLs excepting those like `/trip/rest/` which can then be used by tracker clients that do not support HTTPS or redirections, to log locations without being redirected.

This rule needs to be in the `<VirtualHost *:80/>` section of the HTTP server.

```
RedirectMatch ^/trip/app/(.*)$ https://${MY_HOST}/trip/app/$1
```

7.2.3 Redirecting Traccar Client URLs

The Traccar Client app (`https://www.traccar.org/client/`) does not provide a facility to define a URL prefix. All calls are to the server root.

A workaround is to configure the Apache server to redirect both HTTP and HTTPS requests that match the pattern of Traccar Client logging requests to the `/trip/rest/log_point` URL prefix.

To support using the Traccar Client, enter the following in the Apache `<VirtualHost/>` sections:

```
# Redirect for Traccar Client
<IfModule mod_rewrite.c>
        RewriteEngine On
        RewriteCond "%{QUERY_STRING}" "^id=[\da-f]{8}-[\da-f]{4}-[\da-f]{4}-[\da-f]{4}
        RewriteRule ^/ /trip/rest/log_point [PT,QSA]
</IfModule>
```

8 Testing and Developing Trip

8.1 Vagrant

Vagrant (`https://www.vagrantup.com`) provides a simple and powerful development environment.

8.1.1 Quick Start Using Vagrant

This option provides a working example of the application running in a VirtualBox (`https://www.virtualbox.org`) virtual machine (VM) for those [operating systems supported by Vagrant. (`https://www.vagrantup.com/downloads.html`) This also provides a complete example of running the application behind the Nginx (`https://nginx.org`) ("engine x") HTTP reverse proxy server. It is suitable for development or demonstration, but not as a production system.

Note: Installing all the required software, including the Vagrant box involves downloading approximately 600MB of data. Perhaps more of an "easy-start" rather than a "quick-start".

1. Download and install VirtualBox
2. Download and install Vagrant
3. Clone this repository to a suitable location on the machine you are going to use to host the application and VM:

   ```
   $ cd ~/projects
   $ git clone git://www.fdsd.co.uk/trip-server-2.git
   ```

4. Start the Vagrant VM

   ```
   $ cd ~/projects/trip-server
   $ vagrant up debian
   ```

   The first time this is run, it will download a Vagrant box (`https://www.vagrantup.com/docs/boxes.html`) containing a Debian Linux distribution (`https://www.debian.org`), then install the required Debian packages, modify the default configuration and start the TRIP server.
5. Use your browser to navigate to `http://localhost:8080/` on the host machine and login providing the following credentials:
   - user@trip.test rasHuthlutcew7
   - admin@trip.test 7TwilfOrucFeug
6. When finished, halt the server with:

   ```
   $ vagrant halt debian
   ```

Vagrant shares the source folder with the VM so that you can modify the source files on the host server and immediately impact the deployed application. This gives you a complete working development environment.

Should you need it, e.g. for running a GUI in Vagrant, the `vagrant` user's default password is usually `vagrant`.

Rendering of map tiles is disabled by default, in order to respect OpenStreetMap's Tile Usage Policy. (`https://operations.osmfoundation.org/policies/tiles/`) You will

need to follow the instructions below, in the See Section 5.1 [Tile Server Configuration], page 7, section, before map tiles are rendered.

If you forget the admin user (`admin@trip.test`) password, login into the VM and modify the database entry in the PostgreSQL database. Replace `SECRET` with your desired password.

```
$ cd ~/projects/trip-server
$ vagrant ssh
$ psql trip
trip=# UPDATE usertable SET password=crypt('SECRET', gen_salt('bf')) WHERE nickname='a
trip=# \q
```

You can configure the time zone and locale settings by running the following commands on the guest VM and following the prompts:

```
$ sudo dpkg-reconfigure tzdata
$ sudo dpkg-reconfigure locales
```

Optionally, apply the latest Debian updates with:

```
$ sudo apt-get upgrade
```

View the `Vagrantfile` configuration file in the root of the `trip-server` folder for some examples you can modify. E.g. you can enable the `config.vm.network "public_network"` option to make the VM accessible from the public network. This would allow you, for example, to test location updates, using a GPS enabled device sharing the same private LAN as the host VM. Note the warnings in the Vagrant documentation for this setting (`https://www.vagrantup.com/docs/networking/public_network.html`), as for convenience, **the VM is insecure by default and design**.

8.1.2 Trouble-shooting

8.1.2.1 Guest additions on this VM do not match the installed version

`Guest additions on this VM do not match the installed version of VirtualBox!`

This means the installed box needs updating or a different version of VirtualBox needs to be used.

There is a script in `./provisioning/init.sh` that automates the following manual process:

In the past, it was sufficient simply to install the vagrant-vbguest package. (`https://github.com/dotless-de/vagrant-vbguest`)

```
$ vagrant plugin install vagrant-vbguest
```

However, if that fails:

1. Check the vagrant-vbguest plugin status:
   ```
   $ vagrant vbguest --status
   ```
2. If the guest version does not match the host, do:
   ```
   $ vagrant vbguest --do install [ $TARGET ]
   ```
3. This may fail. Halt and restart Vagrant:
   ```
   $ vagrant halt
   ```

4. Restart Vagrant and check the status again:

```
$ vagrant up
$ vagrant vbguest --status
```

The vbguest plugin host and guest versions should now match.

For further information, see the "Existing VM".

8.1.3 Vagrant Errors

8.1.3.1 incompatible character encodings

```
incompatible character encodings: UTF-8 and ASCII-8BIT (Encoding::CompatibilityError)
```

If you receive this error when running `vagrant up`, even after `vagrant destroy`, use VirtualBox to see if the VM still exists. If so, delete it from within VirtualBox.

```
$ VBoxManage list vms
```

This can occur after deleting the project's `.vagrant` sub-folder (e.g. through `git clean`), whilst there is still an activate Vagrant instance. Web searches suggest there are other scenarios that result in a similarly confusing error message.

8.2 Docker

As a convenience, Docker (`https://www.docker.com`) is installed in the Debian (`https://www.debian.org`) flavour of Vagrant. See the Section 8.1 [Vagrant], page 13, section for more information on using Vagrant for testing this application.

A minimum of two Docker containers are required to run the application. One for the PostgreSQL database (`https://www.postgresql.org`) and another for the application. A third container can be run to provide a map tile server.

8.2.1 Building the Docker Containers

This section describes building the docker containers from this project's source.

To build the database container:

```
$ cd ./trip-server-2
$ docker build -f Dockerfile-postgis -t fdean/trip-database:latest .
```

The application container is built from a source tarball which needs to be in the root directory of the project and match the version number in 'configure.ac'.

```
$ cd ./trip-server-2
$ make dist
$ docker build -t fdean/trip-server-2:latest .
```

8.2.2 Running the Application with docker-compose

To run the application with the database:

```
$ docker compose up -d
$ docker compose logs --follow
```

Stop the container with (and wipe the database):

```
$ docker-compose down --volumes
```

Leave off the `--volumes` switch if you wish to keep the database.

8.2.2.1 OpenStreetMap Tile Server

Two other `docker-compose` files can be used to create a map tile server using the Docker container for OpenStreetMap tile server (`https://github.com/Overv/openstreetmap-tile-server`)

One imports the OpenStreetMap (`https://www.openstreetmap.org/`) data. The other runs the tile server after the data has been imported.

It is unlikely there would be sufficient memory or disk space in the Vagrant VM to run the Tile Server. It would be best to use a dedicated server for this process.

1. The import process temporarily requires a lot of memory. You may need to create and mount a swap file:

```
$ sudo dd if=/dev/zero of=/swap bs=1G count=2
$ sudo chmod 0600 /swap
$ sudo mkswap /swap
$ sudo swapon /swap
```

2. Create Docker volumes for storing the database and caching map tiles:

```
$ docker volume create osm-data
$ docker volume create osm-tiles
```

3. It is recommended to use a small region to minimise resource usage.

   Import OSM data for Monaco:

```
$ docker compose -f docker-compose-map-import.yml up
```

   The process must complete without error. Check the final output carefully. If it fails, it is necessary to delete and re-create the Docker `osm-data` volume.

```
$ docker volume rm osm-data
$ docker volume create osm-data
```

4. Once the import completes successfully, run the map tile server:

```
$ docker compose -f docker-compose-map-server.yml up -d
```

   To stop the tile server:

```
$ docker compose -f docker-compose-map-server.yml down
```

8.3 Testing with Curl

The application can be tested outside a browser by using the `curl` command line utility.

1. Login and get a valid session ID, using `curl`:

```
$ curl -i -X POST -d email='user@trip.test' \
-d password='rasHuthlutcew7' \
http://localhost:8080/trip/app/login | grep SESSION_ID
```

2. Fetching a single page with `curl` using the session ID obtained from the previous command:

```
$ curl -H 'Cookie: TRIP_SESSION_ID=b3571314-d5c4-4690-8164-8384fd748faa' \
'http://localhost:8080/trip/app/tracks'
```

3. Submitting a file for file upload:

```
$ curl -i -H 'Content-Type: multipart/form-data' \
```

```
-H 'Cookie: TRIP_SESSION_ID=b3571314-d5c4-4690-8164-8384fd748faa' \
'localhost:8080/trip/app/itinerary/upload' \
--form 'file=@/path/to/file.gpx' -F 'id=2450' -F 'action=upload'
```

8.4 Load Testing

Load testing can be done with `curl` and `ab` (Apache Bench, usually installed with Apache 2).

1. Login and get a valid session ID as described above using `curl`. See Section 8.3 [Testing with Curl], page 16.
2. Test fetching a single page with `ab` using the session ID obtained from the previous command:

   ```
   $ ab -v 3 \
   -C 'TRIP_SESSION_ID=b3571314-d5c4-4690-8164-8384fd748faa' \
   http://localhost:8080/trip/app/tracks
   ```
3. Make sure the response gives a valid response, proving the session ID is valid and working.
4. Add the options `-n 1000 -c 50` to perform 1,000 page fetches, with a maximum of 50 concurrent requests. Add the `-k` option if you want to test `keep-alive` requests. Make sure to configure the build with the `--enable-keep-alive` option.

If using `keep-alive`, it's best to limit the number of concurrent requests to no more than the number of workers the application has been configured to start, otherwise some requests will fail. As far as I understand, there is no real benefit using `keep-alive` these days, so by default it is disabled in the build.

9 Miscellaneous

The following sections mostly relate to information around system maintenance and application development.

9.1 Useful Queries

9.1.1 Map Tiles

Monthly cumulative totals of map tile usage for the last year:

```
SELECT year, month, max(count) AS cumulative_total FROM (
        SELECT time, extract(year from time) AS year,
        extract(month from time) AS month,
        extract(day from time) AS day,
        count FROM tile_metric ORDER BY time DESC) AS q
GROUP BY q.year, q.month ORDER BY q.year desc, q.month DESC LIMIT 12;
```

Count of expired tiles:

```
SELECT count(*) FROM tile WHERE expires < now();
```

Count of unexpired tiles;

```
SELECT count(*) FROM tile WHERE expires >= now();
```

Count of expired tiles older than 90 days:

```
SELECT count(*) FROM tile WHERE expires < now() AND
updated < now()::timestamp::date - INTERVAL '90 days';
```

Delete expired tiles older than 90 days:

```
DELETE FROM tile WHERE expires < now() AND
updated < now()::timestamp::date - INTERVAL '90 days';
```

Delete all expired tiles:

```
DELETE FROM tile WHERE expires < now();
```

9.1.1.1 Freeing System Disk Space

This section describes freeing up system disk space after deleted tiles (or other records).

To see how much space is begin used by the whole database:

```
SELECT pg_size_pretty(pg_database_size('trip'));
```

To see how much space is being used the the tiles table:

```
SELECT pg_size_pretty(pg_table_size('tile'));
```

Normally, a PostgreSQL installation will be configured to run the VACUUM command (`https://www.postgresql.org/docs/9.4/static/sql-vacuum.html`) automatically from time-to-time. This allows deleted records to be re-used, but does not generally free up the system disk space being used by the deleted records. To do that, the `VACUUM` command needs to be run with the `FULL` option.

Note that `VACUUM FULL` requires an exclusive lock on the table it is working on so cannot be run in parallel with other database operations using the table.

See the Recovering Disk Space (https://www.postgresql.org/docs/9.4/static/routine-vacuuming.html#VACUUM-FOR-SPACE-RECOVERY) section of the PostgreSQL documentation (https://www.postgresql.org/docs/9.4/static/index.html) for more information.

To free up the system disk space used by the tiles table, in `plsql` run:

```
VACUUM FULL tile;
```

or

```
VACUUM (FULL, VERBOSE) tile;
```

To free up the system disk space used by all tables:

```
VACUUM FULL;
```

or

```
VACUUM (FULL, VERBOSE);
```

9.1.2 Useful Queries for Testing

Copy location records for user with id 1 to user with id 2

```
INSERT INTO location (user_id, location, "time", hdop, altitude, speed, bearing)
SELECT 2, location, "time", hdop, altitude, speed, bearing FROM location WHERE user_id
```

Moved yesterday's test location data forward by 1 day:

```
UPDATE location SET time = time + INTERVAL '1 day'
WHERE user_id='1' AND time >= now()::timestamp::date - INTERVAL '1 day'
AND time <= now()::timestamp::date;
```

9.1.2.1 Copying Data

```
CREATE TABLE temp_location (LIKE location);

INSERT INTO temp_location SELECT * FROM location q
WHERE user_id=29 AND time >= '2015-12-14' AND
time <= '2015-12-14T23:59:59'

UPDATE temp_location SET user_id=3, id=NEXTVAL('location_seq'::regclass);

INSERT INTO location SELECT * FROM temp_location;
```

10 Database Backup

Backup just the schema, no data:

```
$ pg_dump --schema-only --no-owner --no-privileges trip > schema.sql
```

Backup just the data, keeping the invariably large `tile` table separate:

```
$ pg_dump --data-only --no-owner --no-privileges --exclude-table=tile
trip \
> test-data.sql
$ pg_dump --data-only --no-owner --no-privileges --table=tile trip \
> tiles.sql
```

Backup schema, data and privileges, including commands to recreate tables, excluding the tile data:

```
$ pg_dump --clean --if-exists --no-owner --exclude-table-data=tile trip
\
> test-schema-data.sql
```

The above backup is suitable for every-day backup. If you intend to restore from the backup as part of your development and test cycle, remove the tile table data exclusion so that the cache is not lost.

Index