CS 165 Project Documentation: **Iskolivery** - a crowdsourced school supplies courier service (atbp.)

Implementation by: **Marc Teves**
with some ideas from CS 191 groupmates:
Aleksei Fernandez
Bridgette Legaspi

15 December 2018

1 Readme

IMPORTANT: Before testing, make sure to read `lighttpd.conf` first. The absolute path must be known to the web server daemon.

Tested on:

- lighttpd 1.4.51
- mysql 5.7.24
- php 7.2.10

Just run `setup.sh` to begin the application and open `localhost:3000` on your favorite web browser that isn't Internet Explorer 9 or earlier. Make sure that your mysql password is unset, or edit the script to include your login credentials.

Existing users:

- mary@upd.edu.ph aaa
- maria@upd.edu.ph pass
- madeline@upd.edu.ph pass

Demo link is at `https://youtu.be/DVJ7wJz0i8s`

2 Project Description

2.1 Background

Iskolivery is an application that matches *requesters* and *request fulfillers* (hereafter known as fulfillers) using location data and user-provided schedules. Requesters create and post requests, which are a list of tasks. Tasks are completed by fulfillers, and different fulfillers can complete separate tasks from a single request. Each user can act as a requester and a fulfiller, depending on what is convenient for them at the time.

2.2 Scope

The app's user registration aims to be focused on University of the Philippines Diliman students for its credentials and user identity verification.

2.3 Objectives

- User registration
  - Registering an account
  - Logging into an account
- Making requests
  - Posting new requests
  - Editing task list (as user is posting request, not after)
  - Deleting posted requests
  - Viewing posted requests by the user
- Fulfilling requests
  - Accepting posted requests
  - Cancelling accepted requests
  - Viewing available requests posted by other users
  - Viewing accepted requests

The user interface will be a web browser connected to a database.

3 Technical Specifications

The project uses MySQL as a DBMS. Access is needed to the host machine running the database in order to perform the database setup specified in `database_setup.sql`.

There will be a lot of queries that find the top X closest entities to another entity. MySQL's geometry and support is used for optimizing these queries and making them possible.

Location data is stored in the EPSG 4326 spatial reference system. Given the small scales involved (on or around UP Campus), great circle distances were not used, instead flat plane distances were used.

The data is served by `lighttpd`, a web server daemon, using PHP-cgi to define what actions the server takes and to define most of the application logic. Only a small part of the application logic is included in the database setup and SQL queries.

4 ER Diagram + Schema

A detailed description of the schema is available at `database_setup.sql`. Tables Report and Rating are not used.

4.1 Entities

- User (id INTEGER, name STRING, upmail STRING, password STRING, rating FLOAT, image_url STRING, location_id INTEGER)
- Request (id INTEGER, posted_by INTEGER, info STRING, status_code INTEGER, deadline DATE)
- Task (id INTEGER, info STRING, status_code INTEGER, bounty FLOAT, task_in INTEGER, location_id INTEGER)
- Report (id INTEGER, info STRING, action_taken INTEGER, task_id INTEGER, filed_by INTEGER, filed_against INTEGER)
- Location (id INTEGER, point POINT, loc_shortname STRING)

4.2 Relationships

- Rating (request_id INTEGER, task_id INTEGER, requester_rating FLOAT, fulfiller_rating FLOAT)

5 Functionalities

These are the complete data manipulation functionalities included in the application. Queries involved are shown and the files that enact them are listed. A ? marks data that is supplied either by user-input or retrieved beforehand.

5.1 Create

5.1.1 Creating a New User

Before a user can use the application, they must have an account first. The user can create an account at the homepage, after which they can use it to log-in. Relevant queries for inserting a new account are included in `user_add_edit.php`. The password is hashed first before being included in the database.

```
INSERT INTO User (name, upmail, password)
VALUES (?, ?, ?)
```

At the start, a user does not have a location yet, and thus cannot view available tasks. After logging in, a user is immediately made to select a location.

5.1.2 Creating a New Request

A new Request can be made by selecting the Add Request button in the homepage after logging in. Currently, the deadline cannot be set to any other day than December 24th as a user interface for submitting a proper DATE string was not made. Query can be found in `add_request.php`

```
INSERT INTO Request (posted_by, info, deadline)
VALUES (?, ?, ?)
```

5.1.3 Creating a New Task

A new task can be made by selecting the Add New Task to Request button in the page after selecting a Request for viewing. The new Task is automatically attached to the Request being viewed. Query can be found in `add_task.php`.

```
INSERT INTO Task (info, bounty, task_in, location_id)
VALUES (?, ?, ?, ?)
```

5.2 Read

5.2.1 Viewing Closest Tasks

On the user's homepage after logging in, a list of the 5 nearest tasks, ordered from closest to farthest, can be seen on the rightmost table. The query defined in `user_info.php` uses the view NearbyTask.

```
CREATE VIEW NearbyTasks (id, last_assigned, poster, name, info, bounty,
        status_code, deadline, source, target, point) AS
SELECT Task.id, Task.last_assigned, Request.posted_by, User.name, Task.info,
        Task.bounty, Task.status_code, Request.deadline, A.short_name,
        B.short_name, A.point
FROM Request
JOIN Task ON Request.id = Task.task_in
JOIN Location AS A ON Task.location_id = A.id
JOIN User ON Request.posted_by = User.id
JOIN Location AS B on User.location_id = B.id
WHERE Request.status_code > -1;

SELECT NearbyTasks.*, ST_Distance(userloc, point) as distance
FROM NearbyTasks,
(SELECT point AS userloc FROM User
JOIN Location ON Location.id=User.location_id WHERE User.id=?) AS
UserLoc
WHERE poster <> ? AND (last_assigned <> ? OR last_assigned IS NULL)
ORDER BY distance LIMIT 5;
```

A limit of 5 is defined in the PHP code, but this is just an arbitrarily chosen number to demonstrate that the app only chooses nearby requests.

5.2.2 Viewing Accepted Tasks

On the user's homepage after logging in, a list of all the user's accepted tasks can be seen on the top leftmost table. The query defined in `user_info.php` uses the view NearbyTask.

```
SELECT * FROM NearbyTasks WHERE last_assigned = ?';
```

5.2.3 Viewing Posted Requests

On the user's homepage after logging in, a list of all the user's posted requests can be seen on the bottom leftmost table. The query is defined in `user_info.php`.

```
SELECT Request.id, Request.info, Status.info AS status,
        Request.deadline, Location.short_name AS location
FROM Request
JOIN Status ON Request.status_code=Status.id
JOIN User ON Request.posted_by=User.id
JOIN Location ON User.location_id=Location.id
WHERE Request.posted_by = ?;
```

5.3 Update

5.3.1 Accepting Tasks

A Fulfiller (User) can accept a task. Their ID is recorded in *last_assigned* to record that they have accepted it. The status of the task will be changed from 0 (pending, not accepted) to 1 (pending, accepted). A transaction to achieve this is defined in `accept_task.php`

```
START TRANSACTION;
        UPDATE Task
        SET Task.last_assigned=?
        WHERE Task.id=?;
        UPDATE Task
        SET Task.status_code=1
        WHERE Task.id=?;
COMMIT;
```

5.3.2 Unaccepting Tasks

A Fulfiller (User) can unaccept a task. Their ID is removed from *last_assigned* to record that they have unaccepted it. The status of the task will be changed from 1 (pending, accepted) to 0 (pending, not accepted). A transaction to achieve this is defined in `unaccept_task.php`

```
START TRANSACTION;
        UPDATE Task
        SET Task.last_assigned=NULL
        WHERE Task.id=?;
        UPDATE Task
        SET Task.status_code=0
        WHERE Task.id=?;
COMMIT;
```

5.3.3 Fulfilling Tasks

A requester can mark a task fulfilled by the user that accepted it. The status of the task will be changed from 0 (pending, accepted) to 2 (complete). The transaction can be found in `fulfill_task.php`. A check for whether or not this is the last completed transaction is done so as to mark the parent Request fulfilled.

```
START TRANSACTION;
        UPDATE Task
        SET status_code=2
```

```
        WHERE Task.id=?;
        IF (SELECT COUNT(*) AS count FROM
                Task JOIN Request ON Task.task_in=Request.id
                WHERE Task.status_code<2
                AND Request.id IN(SELECT task_in FROM Task WHERE Task.id=?)) == 0
        THEN
                UPDATE Request
                SET status_code=2
                WHERE Request.id
                IN(SELECT task_in FROM Task WHERE Task.id=?);
        END IF;
COMMIT;
```

5.4 Delete

A future plan for the project is to prevent users from deleting anything at all and only allow for disabling, but for now deleting is allowed to satisfy project requirements.

5.4.1 Delete Posted Request

The user can delete a Request. Requests automatically delete all Tasks attached to it. The query is defined in `delete_request.php`

```
                # CREATE TABLE Task ()
                # task_in INT NOT NULL REFERENCES Request ON DELETE CASCADE,
        DELETE FROM Request
        WHERE Request.id=?;
```

5.4.2 Delete Posted Request

The user can delete a Task. The query is defined in `delete_task.php`

```
        DELETE FROM Task
        WHERE Task.id=?;
```

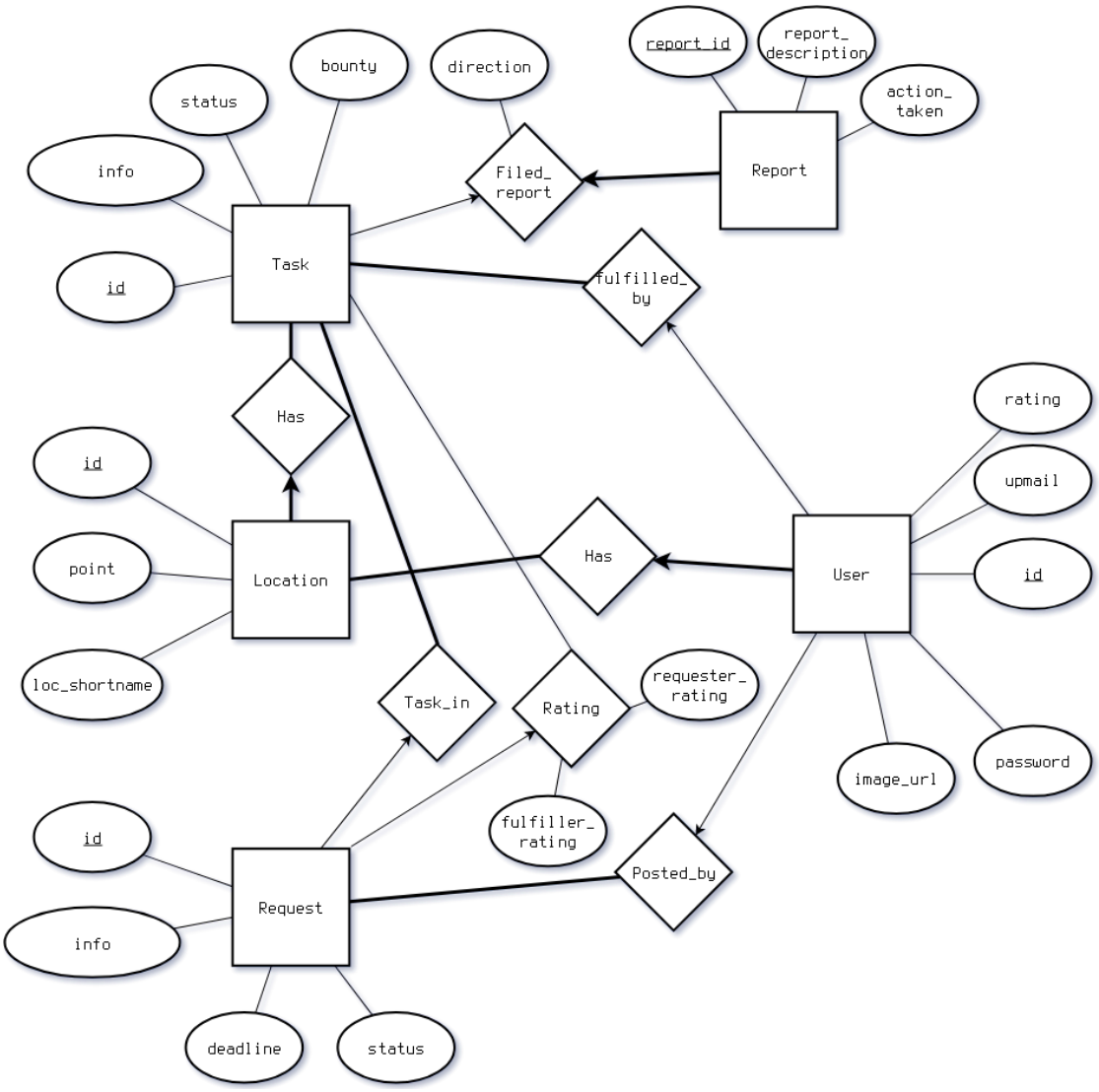

Figure 1: Complete ER diagram, detailing all the relationships.