# Design Document

Project: Virtual Exposition
Client: Crossover.com

Online at: http://

## Statement

The major goal of the project is to deliver a web application, allowing companies to book their place in virtual expositions in different exposition events.

## Functional Description

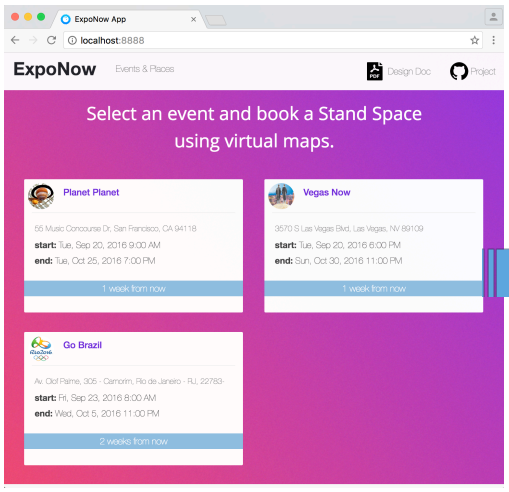
Companies will choose from available events the one they want to take place in, then they will choose their stand within the exposition hall from a map and finally they will receive a report about the users who visited their stand on the event after it is over.

1. The home screen displays a map with different event places highlighted on it.
2. Selecting an event on the map displays the event details (name, location, event dates) right below the map and the “Book your place” button become active.
3. Clicking “Book your place” will take the user to the exposition hall map, it is a virtual map for the exposition hall with different stands, which he can navigate through it and book his stand.
4. Booked stands is highlighted as booked, the logo of the booking company will be displayed on top of the stand, below it the marketing documents (could be downloaded) and the contact details.
5. Free stands is highlighted as free, and on top of it the price.
6. The user can select any empty stand to book; clicking on an empty stand shows a popup with details of the stand, a real image of it and a “Reserve” button.
7. Clicking on reserve takes the user to the registration page where he supposed to provide: contact details, upload marketing documents, company admin and company logo.
8. Clicking on “Confirm Reservation” reserves the stand for the user, takes him to the exposition hall screen viewing the booked stand with the user’s company details on it.
9. Finally the company admin receives a report by mail about the users of the stand after the event is over

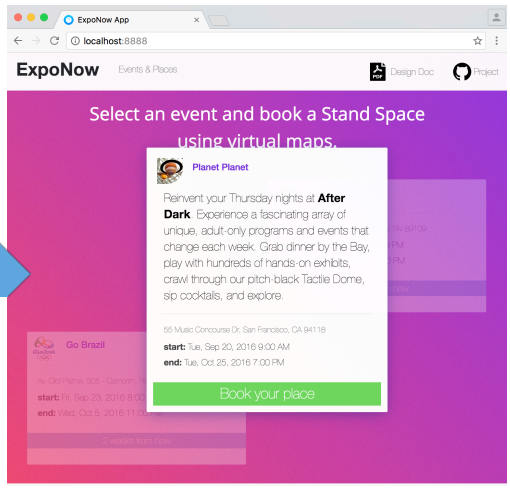
# Wireframes

Layout scaled for screens with width up to 1200px:

HOME PAGE



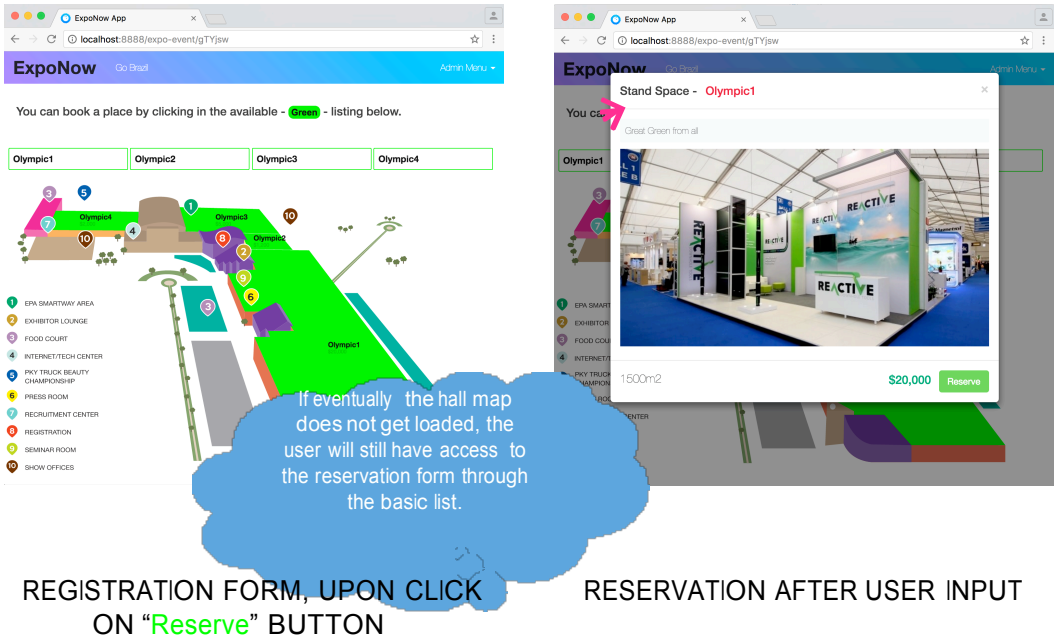

HOME PAGE WITH AN EVENT SELECTED




Important: The home page will show only events with starting date/time greater than now.

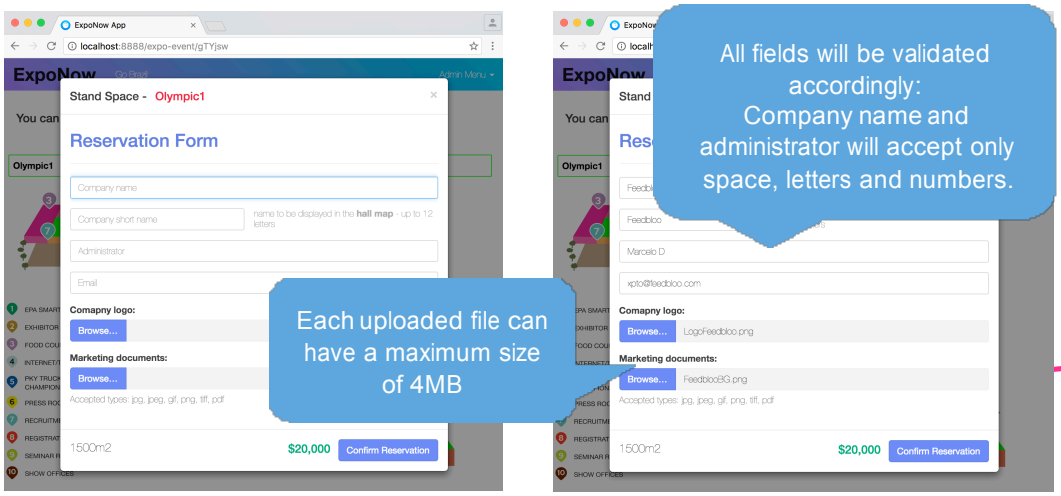
HALL MAP FOR A EVENT SELECTED FROM HOME PAGE, AFTER CLICK ON “Book your place”

DETAILS FOR SELECTED STAND SPACE, IN THIS CASE THE “Oplympic1”

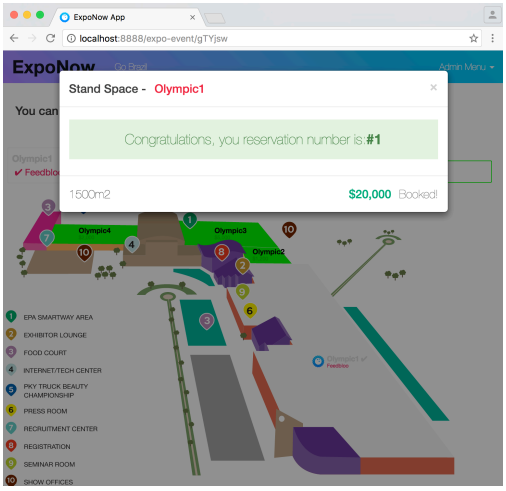



REGISTRATION FORM, UPON CLICK ON “Reserve” BUTTON

RESERVATION AFTER USER INPUT

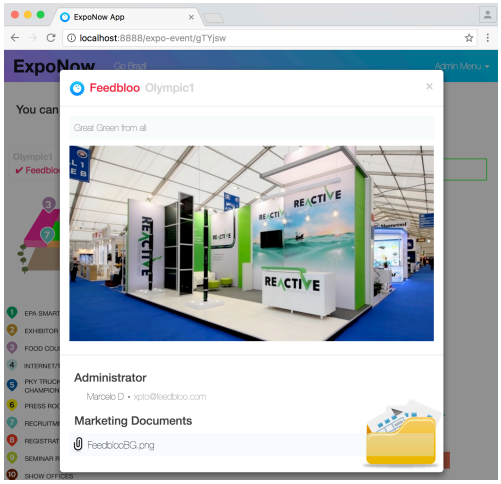


## RESERVATION MESSAGE, UPON CLICK ON "Confirm Reservation" BUTTON

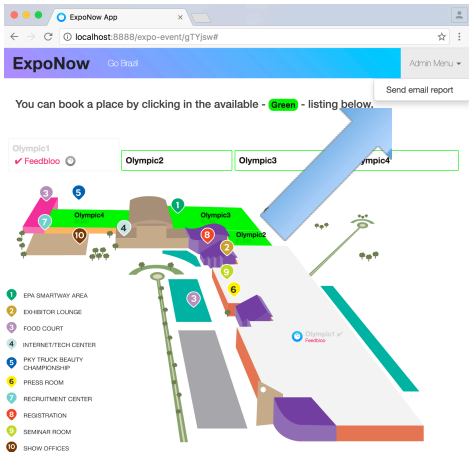



## HALL MAP SHOWING STAND SPACE RESERVED WITH RESPECTIVE COMPANY LOGO AND NAME

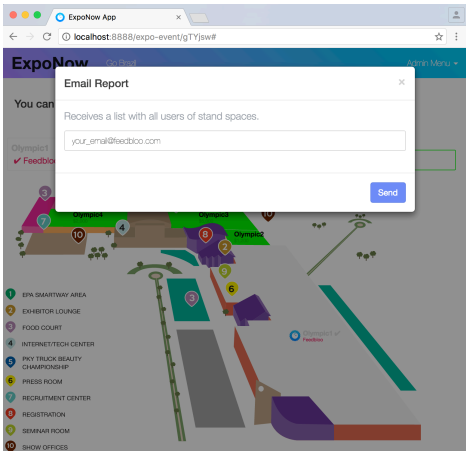

## DETAILS FOR A STAND SPACE WHICH HAS BEEN ALREADY BOOKED



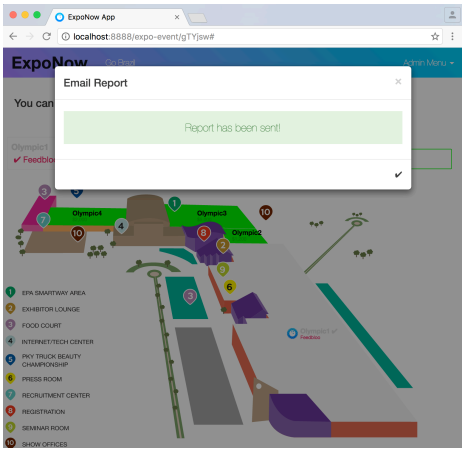
ADMIN ACCESS THE OPTION TO SEND EMAIL REPORT, IN THE HALL MAP OF THE EVENT




IF THE EVENT IS FINISHED OR HAVE AT LEAST ONE STAND SPACE RESERVED, ADMIN INFORM HIS EMAIL



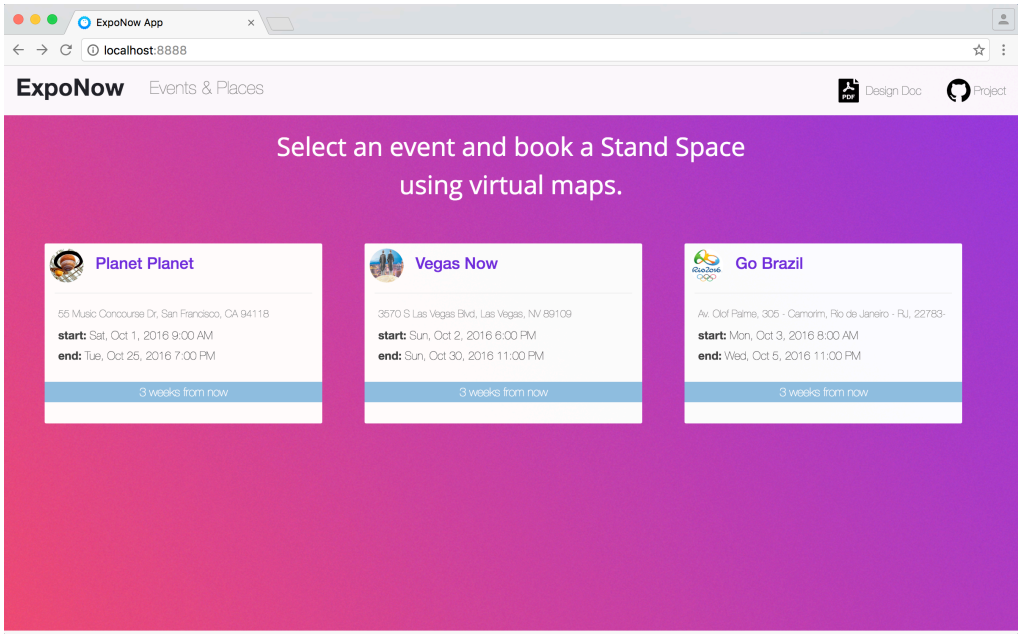

IF EMAIL HAS BEEN SENT, SHOW POSITIVE MESSAGE



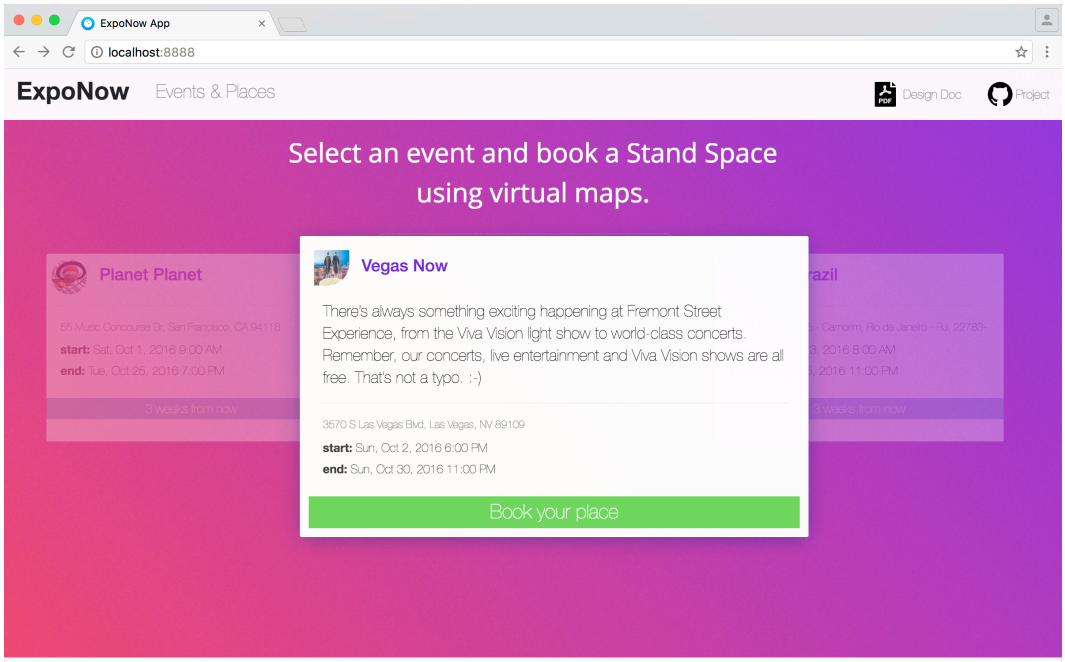

In this project we are using SendGrid API to send the Email Report notifications.

Layout scaled for screens with width greater than 1200px:

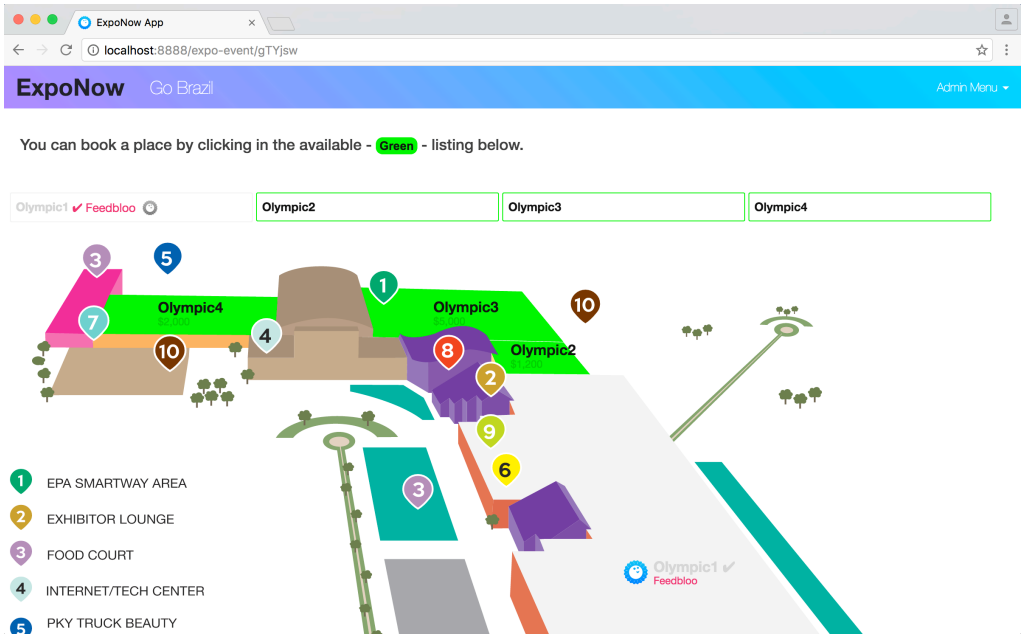
HOME PAGE – EVENTS WILL BE ORGANIZED IN TWO COLUMNS



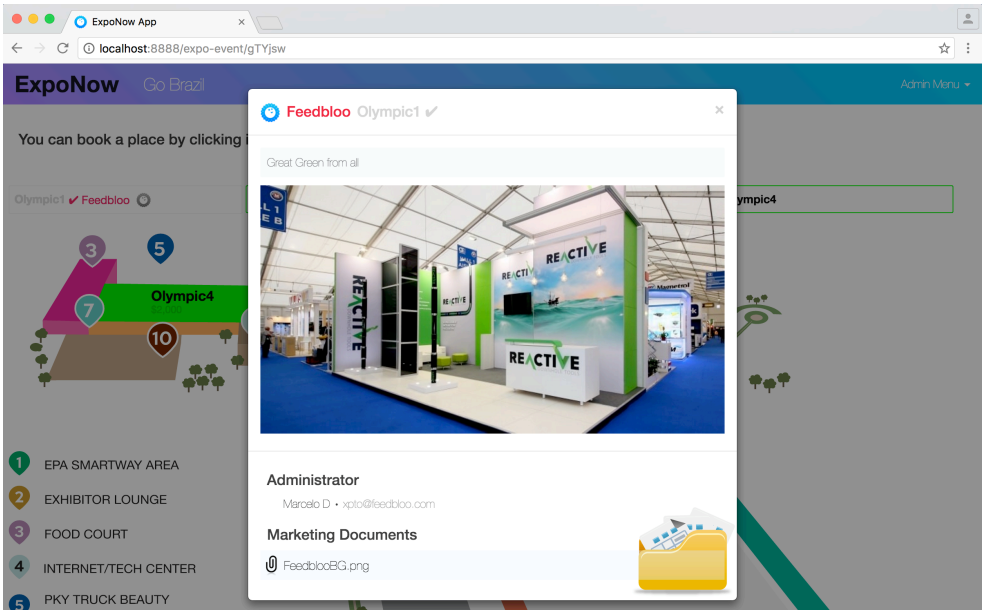

HOME PAGE WITH AN EVENT SELECTED



## HALL MAP



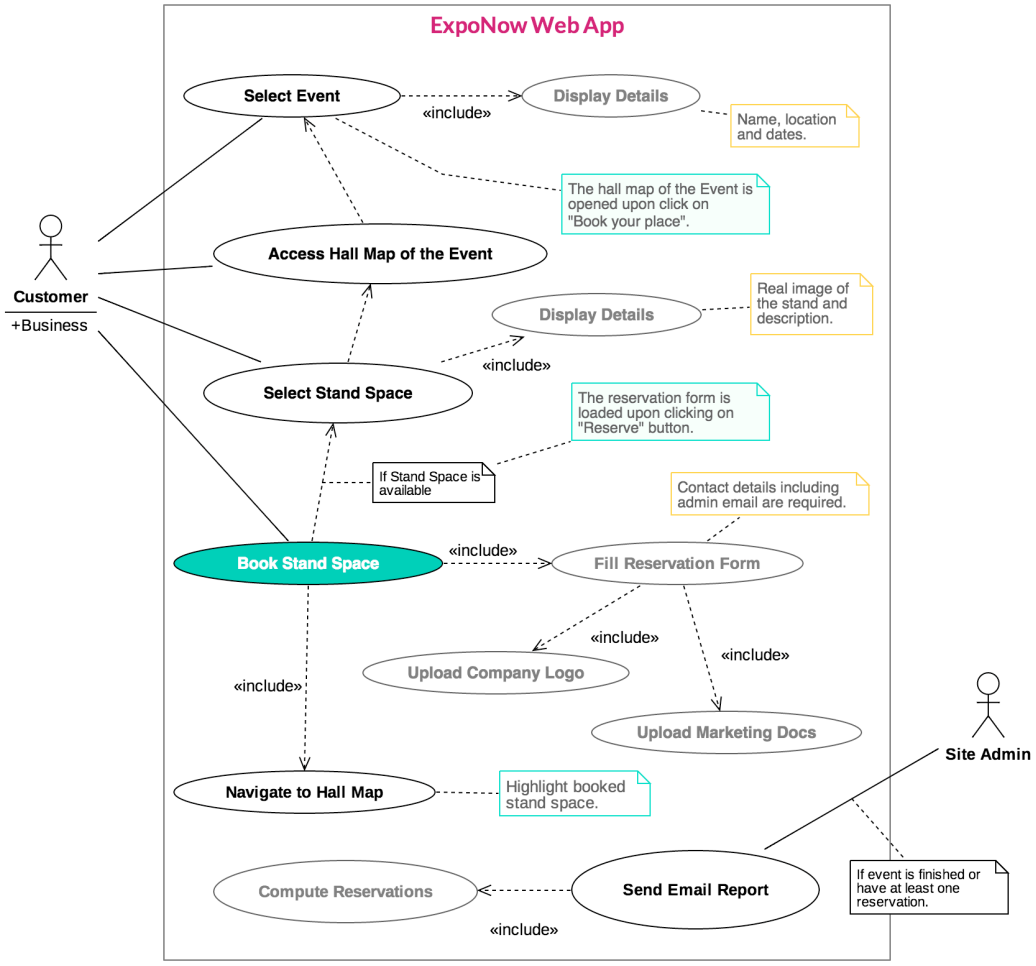

## DETAILS FOR A STAND SPACE WHICH HAS BEEN ALREADY BOOKED



## How the Application Works

The Use Case UML Diagram below can give you a glimpse of how the app works.

This diagram shows briefly all currently available features in the application, and dependencies between them.




## Configuring Environment

The application need PHP in order to run, if installing PHP may sound challenging you could also try Laravel Homestead.

Important note about Apache web server, make sure to have the correct directory permission for the application directory in the file apache2.conf, which should look like this:

```
</VirtualHost>Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride all
    Require all granted
</Directory>
```

Remember also that the Virtual Host configuration, which should be in a file such as /etc/apache2/sites-enabled/000-default.conf, should point the root directory to the public/

```
<VirtualHost *:80>
    ServerAdmin your-email@yourdomain.com
    DocumentRoot /var/www/html/Virtual Exposition/public
    …
</VirtualHost>
```

Considering that you already have a stable version MySQL database and that your PHP version is >= 5.5.9, with OpenSSL PHP Extension, PDO PHP Extension Mbstring PHP Extension and Tokenizer PHP Extension, then:

1. Copy the application files, which you should have already received together with this Design document, into a directory for instance named exposition/, and create a Virtual Host on Apache webserver pointing it to the directory public/ that exists in the set of directories of the application. Publishing a Virtual Host may vary, but following this link from laravel-recipes you should be able to do it.

2. Once the Laravel application has been installed, we can then create our schema in the MySQL database with the commands from the file doc/database/exponow.sql.

   The commands from the file above will create a schema called Virtual Exposition with the records related to three exposition events.

3. Now that the database schema for the application is ready, we need to configure our application to operate with it:

   a. Open the file .env find the following parameters (DB_DATABASE, DB_USERNAME, DB_PASSWORD) and change them accordingly with your database credentials:

      DB_DATABASE=Virtual Exposition
      DB_USERNAME=< your database username >
      DB_PASSWORD=< your username password >
      Important: the user informed above must have privileges to create/drop tables in the schema of our application exposition. During the development we have used the default MySQL user root.

b. Open the file config/database.php, find the array key 'default' and make sure it's configured as 'default' => env('DB_CONNECTION', 'sqlite').

   This will specify to Laravel that the default database connection parameters defined in the file .env should be used.

4. The steps above should be enough to have the application running. But since we are using Laravel as framework, we could also create our database tables dynamically as we code/test our application. Laravel provides a really powerful console utility called artisan, which gives us an interactive way to create our database as we build our application using MVC architecture.

   Thus when we reach the point to ship our app to the production environment we would be using an external tool like MySQL Workbench to Generate an E-R Diagram of the tables by reverse-engineering our schema exposition.

## Building the Application

### Database

If you want have a glimpse of the process of building the application you can read this section, otherwise just jump to the section bellow Application Maintenance.

As mentioned above we are using Laravel's artisan utility to create the PHP stub files responsible to either generate/delete our database tables.

One of the benefits of using artisan is that we will have an automated way to perform migrations in the database during the deployment of new versions of our application. For more details see Migrations.

To do that we go back to the Terminal window, which should be currently under the directory exposition/, as done in the step 1 of Configuring Environment, and execute a command like this below:

```
php artisan make:migration create_expo-event_table --create=expo_event

php artisan make:migration create_expo-stand_table --create=expo_stand

php artisan make:migration create_booking_table --create=booking
```

The commands above will generate the stub file such as 2016_08_20_210129_create_expo-event_table.php, which must be modified to have the column names of the table expo_event.

We have named our table as expo_event, instead of event, since more likely we would face name collision along the way in the development cycle, considering that frameworks, such as Laravel, often use the word event to name essential components. Beside that we can easily track in the future all snippets that may refer the table expo_event.

Within this migration file we could also define/trigger further routines associated with a specific application version deployment.

IMPORTANT: Laravel's migration API allows us to undo/rollback a migration through the command php artisan migrate:rollback, which at some point will invoke the method CreateExpoEventTable::down() of the respective Migration class created in the stub PHP file. But if the undo routine requires to drop a column from the database, Laravel will need a specific package to handle the operation. To install this specific package, which is called Doctrine/DBAL, while being in our root application directory exposition/ we can run the command below via Terminal.

```
composer require doctrine/dbal
```

## Additional required modules

Being in the root application directory exposition/ we can run the following commands via Terminal.

For markup elements such as form inputs.

```
composer require laravelcollective/html
```

Update composer with command:

```
composer update
```

To complete the installation for the lib above, go to config/app.php add this lines:

in providers group:

```
Collective\Html\HtmlServiceProvider::class,
```

in aliases group:

```
'Form' => Collective\Html\FormFacade::class,
'Html' => Collective\Html\HtmlFacade::class,
```

# Application Maintenance

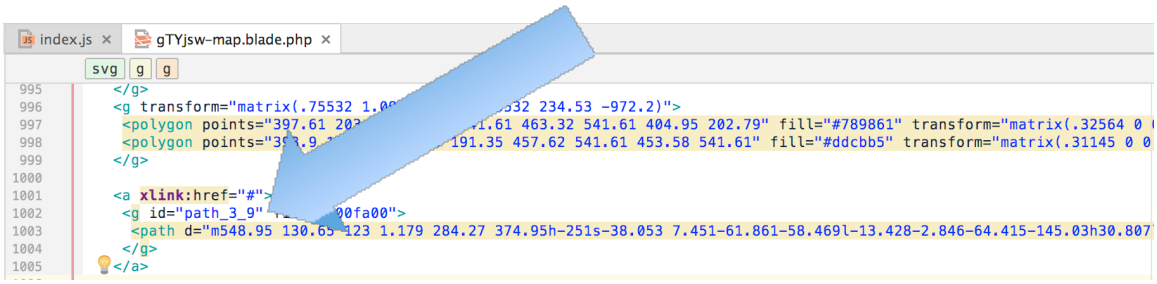
## Directories

We won't comment what are the typical directories of a Laravel based application, if needed you can access Laravel's website and get rich of it, but is important to remember that since our application will be handle file uploading under the directory public/user-assets, no matter what user you configure in your web server (Apache in our case), such user must have privileges to create/modify files under this directory.

## SVG Files

When the user navigates in the application to the exposition hall map, the virtual map loaded in the page is a SVG file. In order to enable the JavaScript that runs in the page to read/access correctly each stand space defined in the SVG file, these stand spaces must be already identified with the respective Ids of the exposition event and the stand space, thus the JavaScript can find the selectable area of the map and process further transformations, such as highlight what has been already reserved and for who.

The pattern for the Id that must be assigned to the <g> element in the SVG markup that represents the stand space is path_[id of the event]_[id of the stand]. In the example below the referred stand space is defined for the exposition event with Id 3 and stand Id 9, the Id of the event comes from the database table `expo_event` and the Id of the stand comes from the table `expo_stand`:

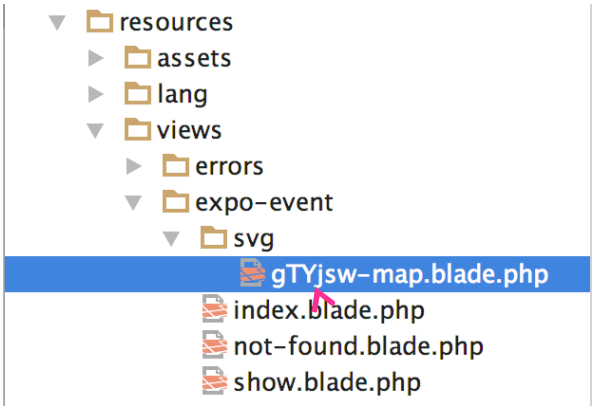



It also required wrapping each <g> element that represents a stand space within an <a> element, therefore elements that specify the stand space will be clickable. If you notice the example above, you can see that the path defining that stand space exists within the identified <g> element, this structure should be followed while creating the SVG file.

Now that the element that represents the stand space is properly identified, the JavaScript that runs in the page will process the list of stands sent by server-side e render the setup of the SVG file.

We recommended that when exporting SVG file in the software you are using, you chose the optimized version, i.e. to not export metadata, since they won't be useful in our application.

The SVG files must be saved within our application in the directory: `resources/views/expo-event/svg/`. The name of the file must be structured as follows: [event hash code]-map.blade.php. The event hash code comes from the database table `expo_event` column `code`. This code should be a 6 char length e must be unique. Se the example below that shows the file and respective database record of it

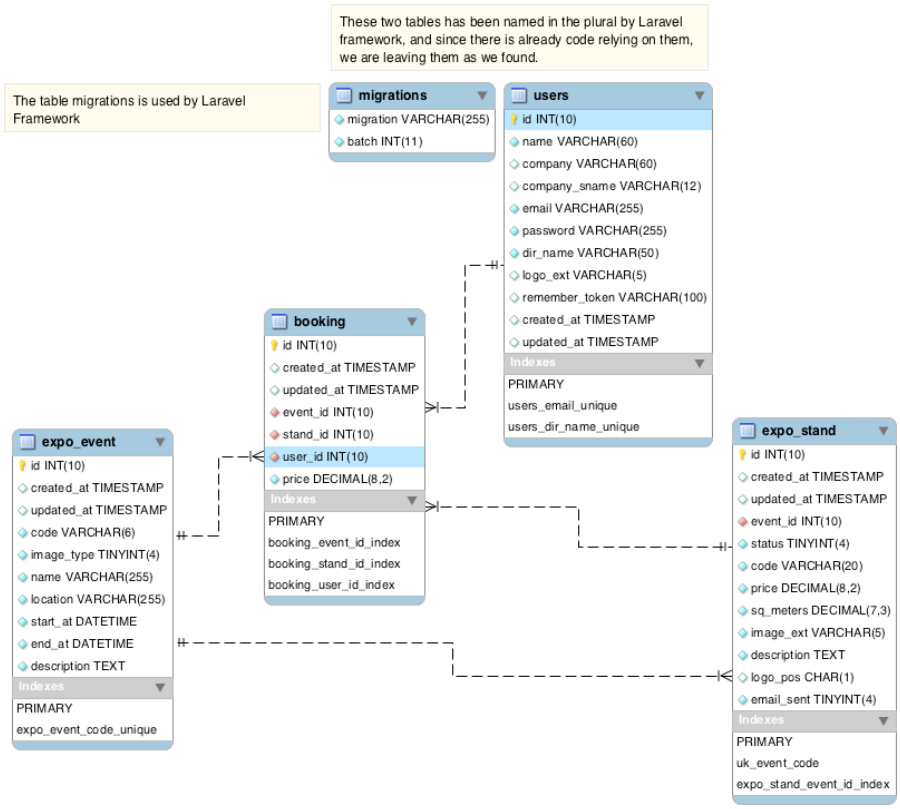



| id | created_at | updated_at | code | image_type | name | location | start_at |
|---|---|---|---|---|---|---|---|
| 1 | 2016-08-28 22:53:57 | 2016-08-28 22:53:57 | xpzJsw | 0 | Planet Planet | 55 Music Concourse Dr, San Francisco, CA 94118 | 2016-09-01 0 |
| 2 | 2016-08-28 22:53:57 | 2016-08-28 22:53:57 | Jhsu8s | 0 | Vegas Now | 3570 S Las Vegas Blvd, Las Vegas, NV 89109 | 2016-09-02 1 |
| 3 | 2016-08-28 22:53:57 | 2016-08-28 22:53:57 | gTYjsw | 0 | Go Brazil | Av. Olof Palme, 305 - Camorim, Rio de Janeiro -... | 2016-09-03 0 |

Depending how the hall map is designed for the SVG file, sometimes will be necessary make an adjustment where the logo of the customer should be load close to the stand name, see in the example below that the stand space Olympic2 will not have available space to accommodate the logo on left, only on the right side. This little adjustment can be defined in the record of the stand space in the database table `expo_stand` column `logo_pos`. Use *t* for *top*, *l* for *left*, *b* for *bottom* and *r* for *right*.

# Diagrams

## Entity-Relationship Model



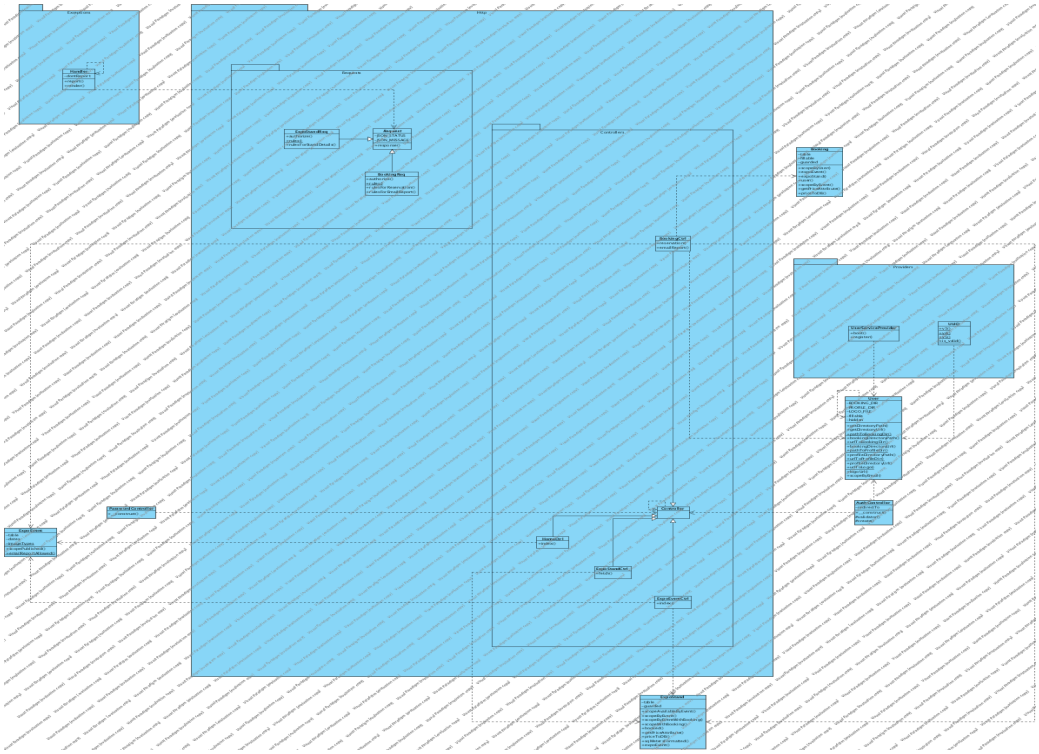

Since the Virtual Exposition application is intended only for demonstration, at least for now, no strong name pattern has bee utilized to define the name of tables and columns. For a more robust application we could use a strategy of meta-names for instance:

- Tables could be named as A001, A002, B001, etc. where the letter specifies the logical module in which the table exists, and the number would be a regular auto-increment number.
- Columns could be named as A001ID, A001IN_001, A001CH_001, A001VC_002, A001VC_002_enUS, etc. In this manner the column name would have a self-descriptive name containing the type, sequence, locale, etc. about the data that is housing.

## Class Diagram

The diagram below is contains only the classes that have been created/managed for the application. The Laravel framework utilizes several other classes to enable our application to run, but since these classes have not been manipulated during the development they are not included here.

The image below has been generated by Visual Paradigm modeling tool and is only a glimpse. To see the same image with better resolution, please find the file *ClassDiagram_HD.png* that is placed in the directory doc/uml-diagrams/ under the root directory of the application.



## Code Documentation

All PHP files are documented using PHPDoc notation.

# Final considerations

## Development Environment

### Back-end

- System: Mac OS X Ysemite10.10
- Web Server: Apache 2.0;
- IDE: PhpStorm 2016.2
- Language: PHP 7.0
- Database: MySQL 5.7.13
- Dependency package manager for PHP: Composer 1.2.0
- PHP Framework: Laravel 5.2
- UML Diagrams: StarUML 2.7.0
- Database E-R Diagram: MySQL Workbench
- Version Control System: Git 2.9.2

### Front-end

- HTML/JavaScript/CSS
- JS Framework: Backbone 1.3.3
- CSS Framework: Bootstrap 3.3.7