RENTALCAR
Technical Documentation

Software Engineering 2016

Besart Vishesella

Ysée Monnier

http://projects.yseemonnier.com/rentalcar/

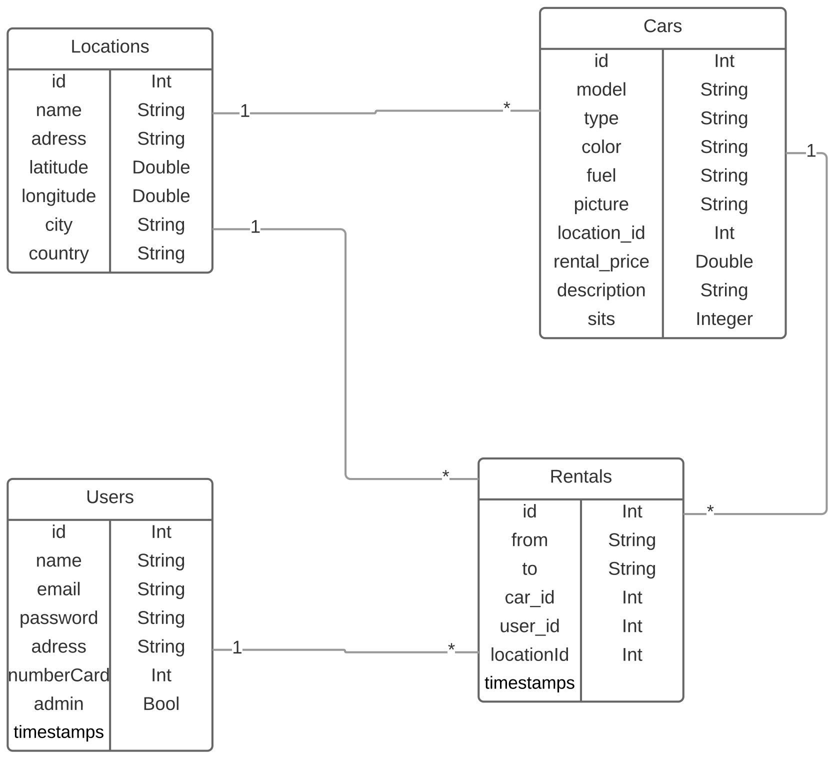
Summary

Website

Components and Libraries

- Laravel framework: Php framework - Backend part of the website.
    - Authentification
    - Routing system (API)
    - MVC pattern
    - Eloquent ORM (Database)
    - Migration and seeder
- AngularJS/JQuery/GoogleMap:
    - DOM Manipulation
    - Map Manipulation
    - Create Event/Catch Event (JS)
    - Ajax (POST, GET) to web service
- Composer: dependencies manager

RentalCar use MySQL Database, below the database relationship schema:



How works the website

Laravel framework follow the Model-View-Controller Pattern. Below a schema which shows how work MVC pattern.

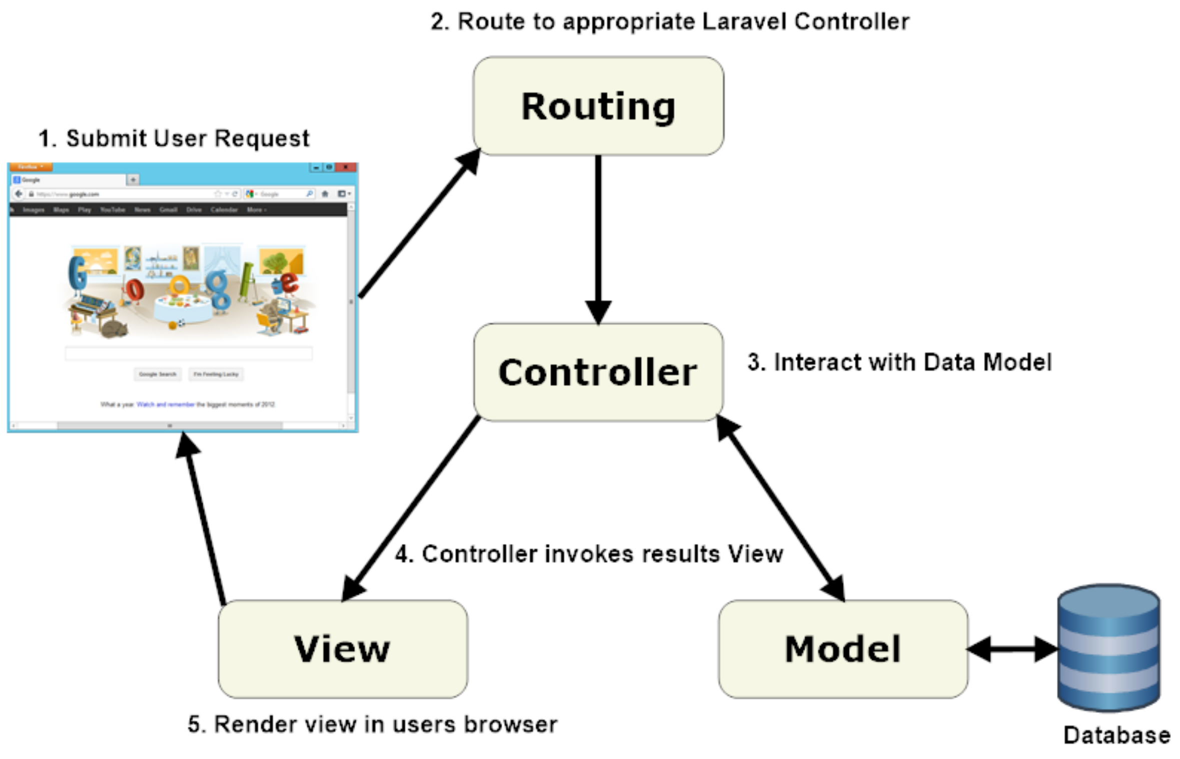

2. Route to appropriate Laravel Controller
Routing
1. Submit User Request
Controller
3. Interact with Data Model
4. Controller invokes results View
View
Model
5. Render view in users browser
Database

Image from laravelbook.com

At the beginning, we declare a route (app/Http/route.php):

Route::get('/users', 'FooController@FooMethod');

When the user calls '/user' url, the method **FooMethod** of **FooController** (/app/Http/Controller) will be called. The method's controller interface with Data Model (Eloquent) and render view in user browser. It is very simple.

Application Directories Structure

The `app` directory contains the core code of application.

The `bootstrap` directory contains a few files that bootstrap the framework and configure autoloading, as well as a `cache` directory that contains a few framework generated files for bootstrap performance optimisation.

The `config` directory, as the name implies, contains all of your application's configuration files.

The `database` directory contains your database migration and seeds.

The `public` directory contains the front controller and your assets (images, JavaScript, CSS, etc.).

The `resources` directory contains views, raw assets (LESS, SASS, CoffeeScript), and localisation files.

The `storage` directory contains compiled Blade templates, file based sessions, file caches, and other files generated by the framework. This directory is segregated into `app`, `framework`, and `logs` directories. The `app` directory may be used to store any files utilised by the application. The `framework` directory is used to store framework generated files and caches. Finally, the `logs` directory contains application's log files.

The `tests` directory contains automated tests. An example PHPUnit is provided out of the box.

The `vendor` directory contains the Composer dependencies.

Application Routing Structure

Authentification

GET /login

return login page view

POST /login [test]

parameter: **email -** string

parameter: **password -** string

GET /register

return register page view

POST /register [create user]

parameter: **name -** string

parameter: **email -** string

parameter: **password1 -** string

parameter: **password2 -** string

GET /logout [**disable token**]

return home page view

Divers

GET / [return home view]

GET /home [return home view]

GET/about [return about view]

<u>Find</u>

GET /find [return find view]

POST /find

parameter: **start -** date

parameter: **end -** date

parameter: **city -** string

parameter: **place -** integer

Success: return fetch available cars

Failed: return error

GET /find/book/car/{**id**}/{**from**}/{**to**} [create rental]

id: car id

from: date from

to: date to

<u>User</u>

GET /booking [return booking user]

Administration

GET /admin

return admin home page

GET /admin/user/list

return list of all users

GET /admin/cars

return list of all cars

GET /admin/car/add

return add car form view

POST /admin/car/add [create car]

parameter: **model** - string

parameter: **type** - string

parameter: **description** - string

parameter: **color** - string

parameter: **fuel** - string

parameter: **rental_price** - double

parameter: **sits** - integer

parameter: **location** - integer

parameter: **picture** - file

GET /admin/car/{**id**}/delete [delete {**id**} car]

GET /admin/car/{**id**}

return detail car

GET /admin/locations

return list of all locations

GET /admin/location/add

return add location form

POST /admin/location/add [create location]

parameter: **name** - string

parameter: **address** - string

parameter: **city** - string

parameter: **country** - string

parameter: **latitude** - double

parameter: **longitude** - double

GET /admin/location/{**id**}/delete [delete {**id**} location]

API Routing Structure

API for mobile side

All request return JSON response with statut code and data message

HEADER:

    token: “aXPZmS6L8ZG3mX8e6hlf79jXs88bp2cC” (example)
    statut code: 200/400 …
 {
    data : “message”
 }

<u>Authentification</u>

GET /api/login [check information]

- parameter: **email** - string
- parameter: **password** - string
- Success - return response 200 + user information + token
- Failed - return response 400

POST /api/register [create user]

- parameter: **name** - string
- parameter: **email** - string
- parameter: **password** - string
- parameter: **admin** - bool
- parameter: **address** - string
- **Success** - return response 200 + user information + token
- **Failed** - return response 400

<u>Util</u>

GET /api/booking

return booking of user

GET /api/find

parameter: **start** - date

parameter: **end** - date

parameter: **city** - string

parameter: **place -** integer

Success - return response 200 + fetch available cars

Failed - return response 400

GET /api/findCities

return all cities

GET /api/findPlaces/{city}

return all place of {city}

POST /api/booked [create rental]

parameter: **from** - date

parameter: **to** - date

parameter: **user_id** - integer

parameter: **car_id -** integer

Success - return response 200 +message

Failed - return response 400 + error message

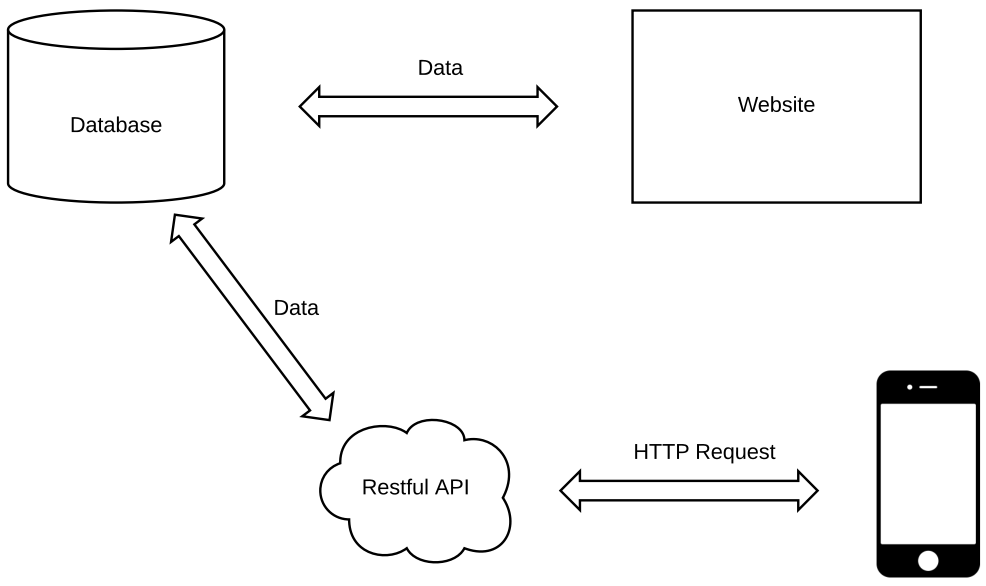
iOS Application

The iOS application communicate with Restful API.

Components and Libraries

- Xcode 7.3
- Deployment iOS 9.3
- UIKit
- Alamofire library: make HTTP request (POST, GET) - https://github.com/Alamofire/Alamofire
- EPCalendarPicker(contributor): Calendar Picker - https://github.com/ipraba/EPCalendarPicker
- SwiftyJSON: JSON Parser - https://github.com/SwiftyJSON/SwiftyJSON

General Structure



Design

See DesignIOS attachment.

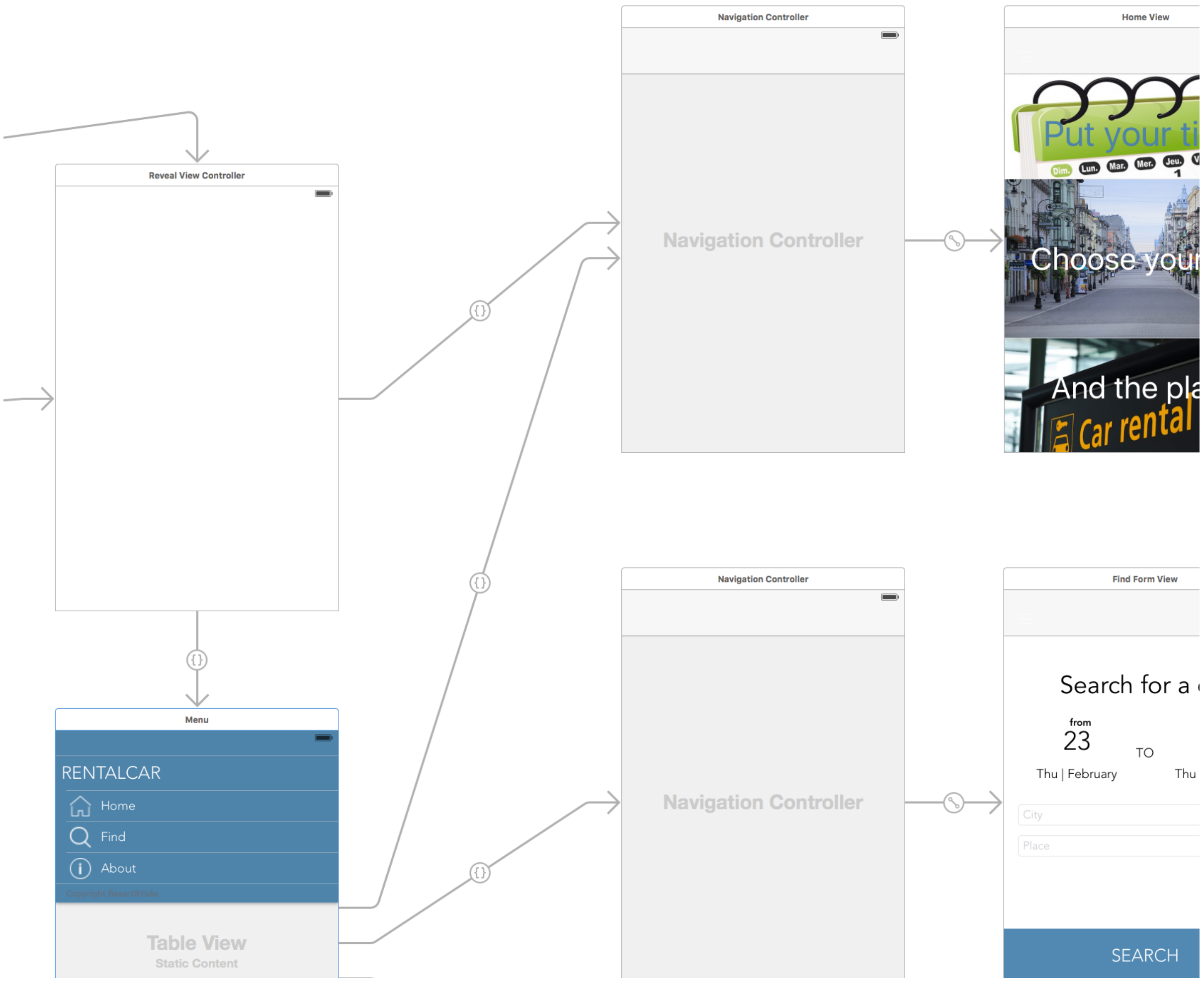
Application Directories Structure

The `model` directory contains model files which create when receive or send data.
The `user` directory contains the core of user part: Login, Register et UserPopover controllers.
The `home` directory contains the core of home part: Home and About controllers.
The `find` directory contains the core of find part: Pick date, city, places, to make a search, to see list of available cars, details and booked a car.
The `booking` directory contains the core of booking part: see booking user.

How work RentalCar Application

View

All views are created from the Main.Storyboard file:



Each views are connected to a ControllerFile (UIViewController) or ViewFile (UIView). With this connection, we can all UIView element from ControllerFile.

So in the ControllerFile, we can for example, initialize label or button, create handler for button, create animation, etc…

To communicate between views, we use a specific protocol called ‘Segue’. This segue allows to go to the next view with transition and we can put data.

Request

All Http request are performed via Alamofire library. This very useful, because these request are executed asynchronously. We can get information about downloading, put header, filter depending statut code, choose response(string, json, data) etc…

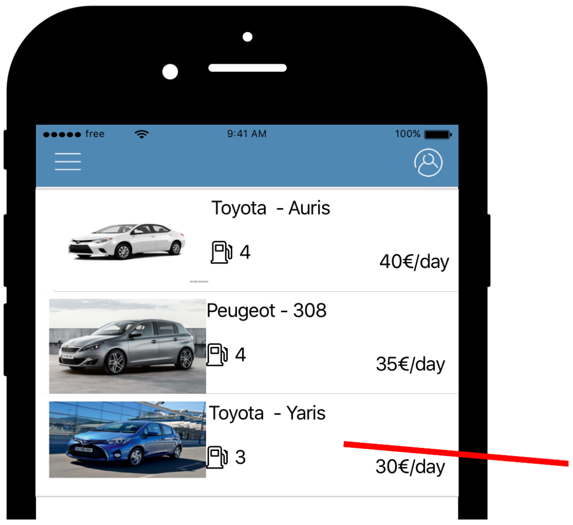
Exemple:

```
Alamofire.request(.GET, “https://myWebService.com”, parameters:
[“foo”: “bar”]).responseJSON { response in
        print(response.request)
        print(response.data)
        print(response.result)
}
```

UITableView

Good feature in iOS UITableView. Certain UI Component use a delegate, a delegate is a group of functions which communicate with the UI Component.

For exemple UITableView (List): Number of row, when you tap on a cell, etc…



```
 override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return (self.cars?.count)!
    }
    override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath)
-> UITableViewCell {
        let cell: CarCellViewController = tableView.dequeueReusableCellWithIdentifier("CarCell") as!
CarCellViewController
            return cell
    }
    override func tableView(tableView: UITableView, didSelectRowAtIndexPath indexPath:
NSIndexPath) {
        print("select row \(indexPath.row)")
        self.index = indexPath.row
        self.performSegueWithIdentifier("DetailSegue", sender: self)
    }
```