## Implementation details

Now, the project needs to be opened by the file *CarTrawler.xcworkspace*. More details on the README file and in the sections bellow.

The presented challenge was to make small changes on the Generic project (https://github.com/CartrawlerGit/Generic). To achieve it, I forked to project and made the changes based on 4 “categories”: Structure, Project, Class Responsibility and Tests.

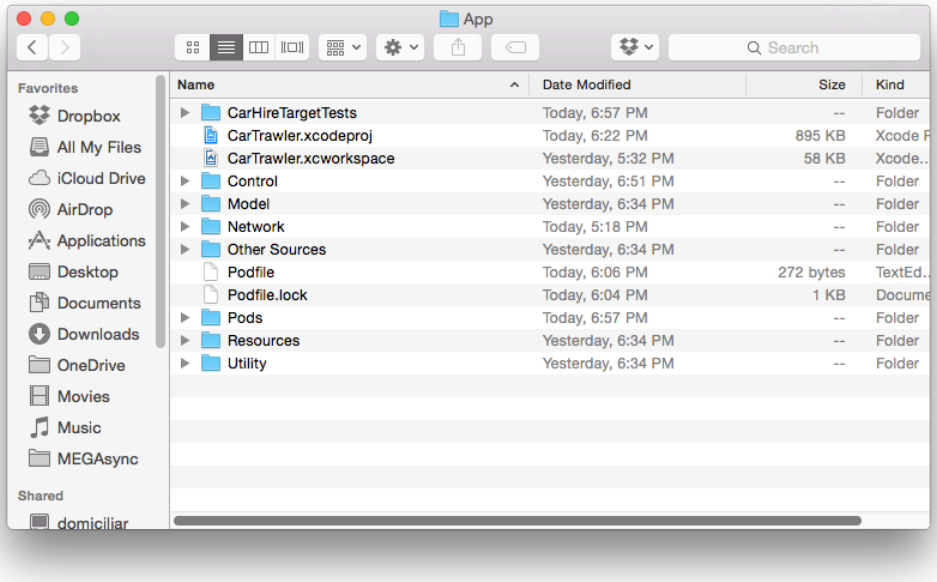
This document is here to present the changes and explain what was made and why. I made the changes on specific classes, but the concept can be extrapolated to the entire project.

P.s.: The evolution of the changes can be seen on the commit history. The screenshots used on this document, for a better visualization, can be found on the email attachment.

## Structure

1 – The entire project was mapped in the folder structure **exactly** as the group structure inside *Project Navigator* of Xcode. To do it, I used the gem synx (https://github.com/venmo/synx). With this structure is easier to identify a file or search for it.

The custom script in the *Build Phases* of the CarHireTarget had to be edited, once that the CarTrawler-Info.plist and Settings.Bundle files were moved to the Resources folder.



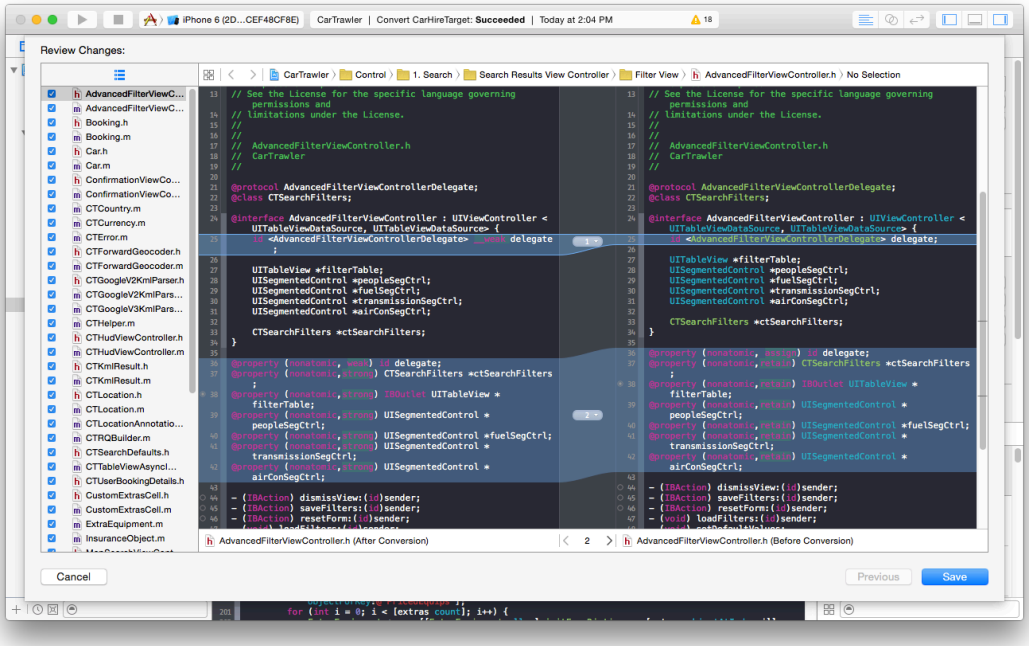

Screen 1 - New Folder structure

## Project

I saw that the project had an “old” structure. Most of the classes did not use ARC, the third party libraries was included as files in the main target, the images was include as general files and new flags of the Build Settings and Info plist was not being used.

1 – All the project was converted to ARC. To achieve it:

1.1 - I used the convert tool of Xcode (Edit -> Convert -> To Objective-C Arc…) All the classes and properties that couldn’t be converted by the tool was converted by code.




Screen 2 - Edit tool to convert to ARC

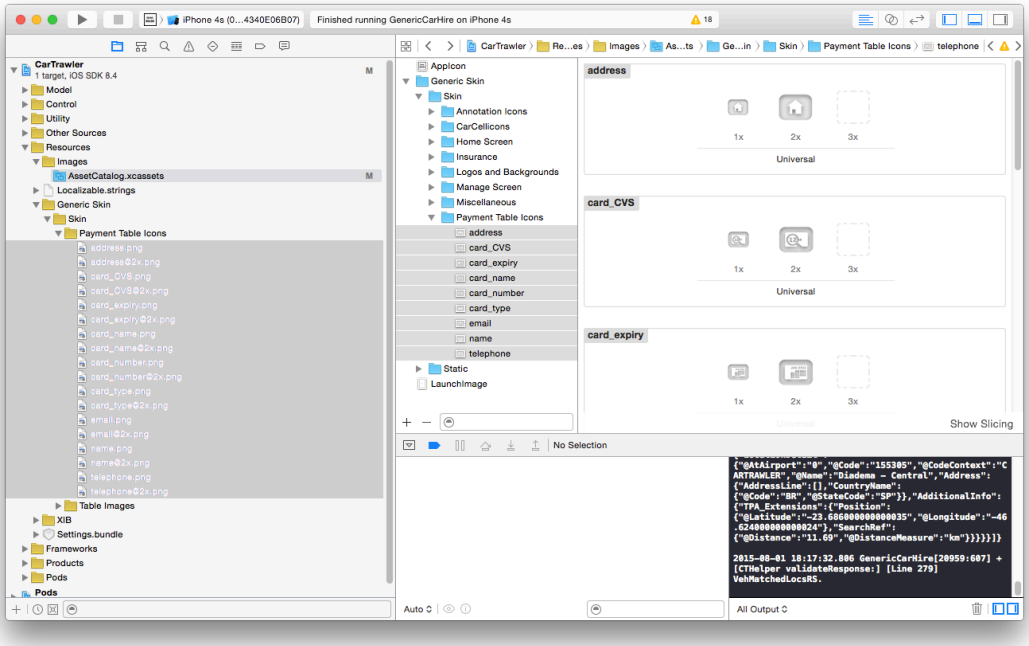
1.2 – After the execution of the tool, all the classes and properties that couldn’t be converted by the tool was converted by code. It included remove the iVar of the ViewControllers and convert the IBOutlets from retain to weak.

```
@interface CustomCarDisplayCell : UITableViewCell

@property (nonatomic, weak) IBOutlet UILabel *additionalExtrasLabel;
@property (nonatomic, weak) IBOutlet UILabel *numberOfPeopleLabel;
@property (nonatomic, weak) IBOutlet UILabel *currencyLabelBG;
@property (nonatomic, weak) IBOutlet UILabel *currencyLabel;
@property (nonatomic, weak) IBOutlet UIImageView *vendorImageView;
@property (nonatomic, weak) IBOutlet UILabel *totalLabel;
@property (nonatomic, weak) IBOutlet UILabel *numberOfDoorsLabel;
@property (nonatomic, weak) IBOutlet UIImageView *acImageView;
@property (nonatomic, weak) IBOutlet UIImageView *fuelTypeImageView;
@property (nonatomic, weak) IBOutlet UIImageView *transmissionType;
@property (nonatomic, weak) IBOutlet UILabel *fuelLabel;
@property (nonatomic, weak) IBOutlet UILabel *baggageLabel;
@property (nonatomic, weak) IBOutlet UIButton *moreInfoBtn;
@property (nonatomic, weak) IBOutlet UILabel *infoLabel;
@property (nonatomic, weak) IBOutlet UILabel *carMakeModelLabel;
@property (nonatomic, weak) IBOutlet UIImageView *carImageView;

@end
```

Screen 3 - Weak IBOutlet

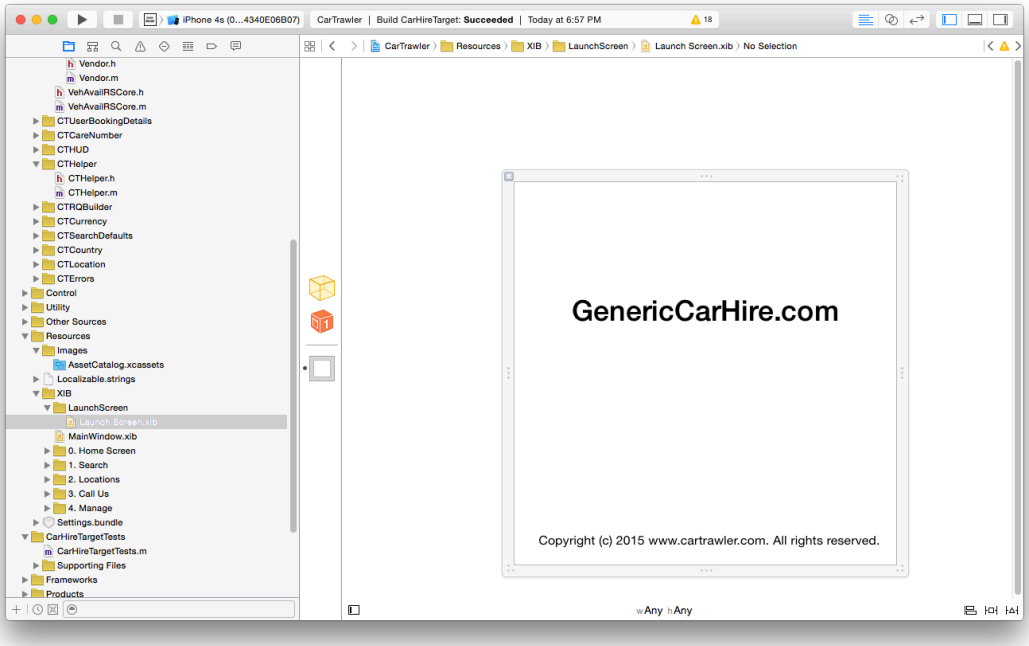
1.3 – The dealloc methods and nil operation on the viewDidUnload were removed.

2 – The images were moved to an *Asset Catalog*. It helps to see the image assets, it's resolutions and colors. Beside it, the system can run compressor images and, in the new Xcode, enables the app slice by default. With it, the images were removed from the project groups structure and centered on the Resources folder.



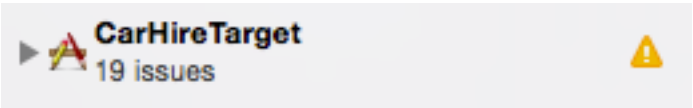

Screen 4 - Images on an AssetCatalog

3 – Once that the Asset Catalog is being used, the Launcher Images were moved to it and I created an XIB to be a launcher screen for iOS 8 and latter. This XIB has a basic of Auto Layout and UI Constraints.

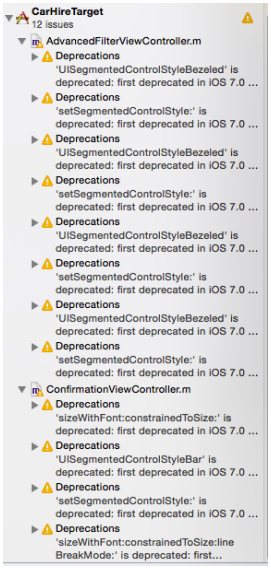



Screen 5 - Launch Screen

4 – All the warnings related to code were solved. Only the warnings related to use of iOS 7 obsoleted code remains.

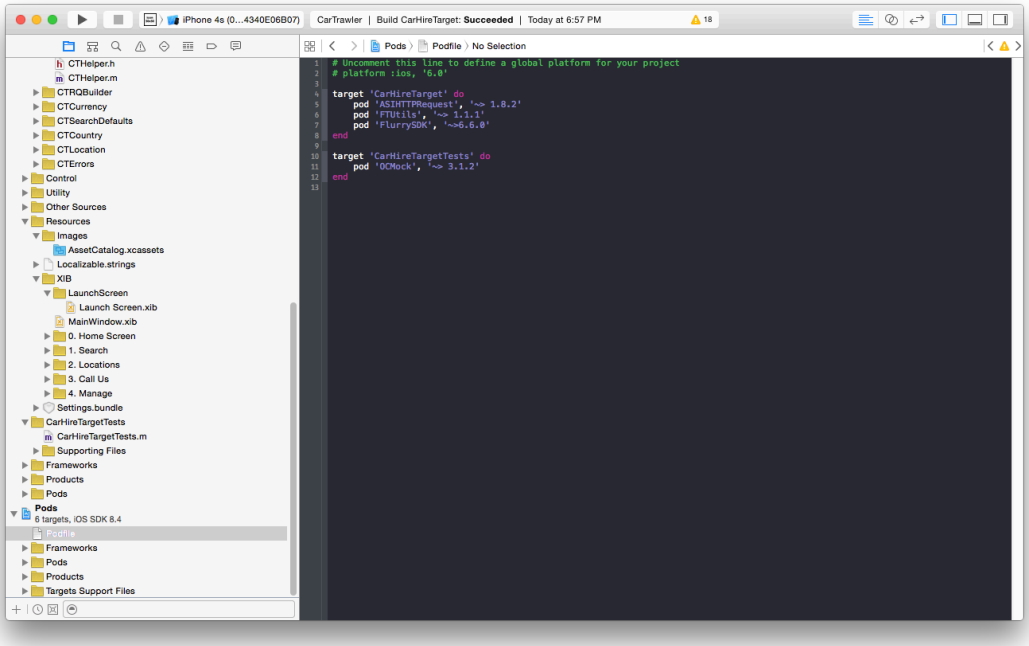



Screen 6 - Warnings before




Screen 7 - Warnings After

5 – The third party libraries were moved to an external project and converted to libraries to be compiled before be add to the target. It helps to manage those libraries and it's versions. To do it, I used the Cocoapods (https://cocoapods.org).




Screen 8 - Using Cocoapods to manage libraries

6 – Some code changes was made to improve execution on iOS 8.

**Custom iOS Target Properties**

| Key | Type | Value |
|---|---|---|
| Bundle name | String | ${PRODUCT_NAME} |
| Launch screen interface file base name | String | Launch Screen |
| ▶ CFBundleIcons~ipad | Dictionary | (0 items) |
| Application does not run in background | Boolean | YES |
| Localization native development region | String | English |
| Bundle version | String | 567 |
| Status bar style | String | Gray style (default) |
| Main nib file base name | String | MainWindow |
| Bundle OS Type code | String | APPL |
| Status bar is initially hidden | Boolean | NO |
| Bundle versions string, short | String | 2.0.1 |
| InfoDictionary version | String | 6.0 |
| Executable file | String | ${EXECUTABLE_NAME} |
| View controller-based status bar app... | Boolean | NO |
| Bundle identifier | String | com.genericcarhire.phone |
| Icon already includes gloss effects | Boolean | YES |
| Bundle creator OS Type code | String | ???? |
| ▶ Icon files (iOS 5) | Dictionary | (0 items) |
| Application requires iPhone environment | Boolean | NO |
| Bundle display name | String | ${PRODUCT_NAME} |
| ▶ Supported interface orientations | Array | (1 item) |
| NSLocationWhenInUseUsageD... | String | We need your location to search f |

Screen 9 - iOS 8 specific code

## Class Responsibility

The code changes were centered on giving a better “responsibility structure” for the classes. It improves the testability, reuse and creates clean view controllers. The implementation was focused on a few classes with the intension to show the logic implemented, and can be replicated to the entire project.

The changes can be found on the files: ManagerViewController, Fee, Booking, CTCountry, CTHelper and the new classes for network request and response validation.

### 1 – Network

The network operations were being made the controllers. With that structure, is hard to test the specific endpoints and the return validation. To fix it, I create 2 groups: Requester and ResponseValidator.

The requesters are responsible to convert the Objective-C objects into HTTP parameters. So, the endpoints can be called from any class and be tested individually.

1.1 – CTRequester

The requester class is responsible to understand the HTTP lib, format and start the requests. On this project, I made the POST operation as an example of use.

This requester implementation allows me to, on test targets, apply a swizzle on the methods and create a fake API. So, the unit tests and acceptance tests do not depend on network quality and do not create fake data on the server.

```
@implementation CTRequester

+ (ASIHTTPRequest *)postRequestWitUrl:(NSURL *)url data:(NSData *)data andDelegate:(id<ASIHTTPRequestDelegate>)delegate
{
    ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
    [request setDelegate:delegate];
    [request appendPostData:data];
    [request setRequestMethod:@"POST"];
    [request setShouldStreamPostDataFromDisk:YES];
    [request setAllowCompressedResponse:YES];

    [request startAsynchronous];

    return request;
}

@end
```

Screen 10 - CTRequester building the POST operation

## 1.2 – CTBookingRequester

An example of specialization of the Requester is the CTBookingRequester. This class receives the properties of a booking and is able to config a request and executes it. On the example, the method receives an booking email, booking id, a handler for the request and is able to dispatch it.

```
@implementation CTBookingRequester

+ (ASIHTTPRequest *) requestBookingWithEmail:(NSString *)bookingEmail bookingId:(NSString *)bookingId andDelegate:(id<
    ASIHTTPRequestDelegate>)delegate
{
    NSString *jsonString = [NSString stringWithFormat:@"{%@%@}", [CTRQBuilder buildHeader:kGetExistingBookingHeader],
        [CTRQBuilder OTA_VehRetResRQ:bookingEmail bookingRefID:bookingId]];

    if (kShowResponse) {
        DLog(@"Request is \n\n%@\n\n", jsonString);
    }

    NSData *requestData = [NSData dataWithBytes: [jsonString UTF8String] length: [jsonString length]];
    NSURL *url = [NSURL URLWithString:[NSString stringWithFormat:@"%@%@", kCTTestAPI, KOTA_VehRetResRQ]];

    return [super postRequestWitUrl:url data:requestData andDelegate:delegate];
}

@end
```

Screen 11 – CTBookingRequester

## 1.3 – CTResponseValidator

The CTResponseValidator is a protocol that creates a generic HTTP response handler. With it, the CTHelper class can delegate the parse of response objects to specific implementations.

```
@protocol CTResponseValidator <NSObject>

+ (id)validateResponseObject:(id)response;

@end
```

Screen 12 - CTResponseValidator protocol

## 1.4 – CTError+ResponseValidator

Once that all the HTTP responses can be handled by a CTResponseValidator, I created a category for CTError that is capable of parse the response and build the errors array.

This specialist class makes the code testable and readable.

```objc
#import "CTError+ResponseValidator.h"

@implementation CTError (ResponseValidator)

+ (id)validateResponseObject:(id)response
{
    if ([[response objectForKey:@"Errors"] isKindOfClass:[NSArray class]]) {
        NSArray *errors = (NSArray *)[response objectForKey:@"Errors"];
        return  [self handleMultipleErrors:errors];
    }

    id errors = [[response objectForKey:@"Errors"] objectForKey:@"Error"];

    if ([errors isKindOfClass:[NSDictionary class]]) {
        return [self handleSingleError:errors];
    }

    if ([errors isKindOfClass:[NSArray class]]) {
        return [self handleMultipleErrors:errors];
    }

    return nil;
}

+ (NSArray *)handleSingleError:(NSDictionary *)response
{
    DLog(@"There is only one error");
    return @[[[CTError alloc] initFromErrorRS:[response objectForKey:@"Error"]]];
}

+ (NSArray *)handleMultipleErrors:(NSArray *)errors
{
    DLog(@"There is more than one error");
    NSMutableArray *actualErrors = [[NSMutableArray alloc] initWithCapacity:errors.count];
    for (NSDictionary *dict in errors) {
        [actualErrors addObject:[[CTError alloc] initFromErrorRS:dict]];
    }
    return [NSArray arrayWithArray:actualErrors];
}

@end
```

Screen 13 - CTError category to handle HTTP responses

## 1.5 – CTBookingResponseValidator

Another example of use of the ResponseValidator is the CTBookingResponseValidator. This class has the implementation of the CTResponseValidator protocol and a implementation for the complete response (error and success).

```objc
@implementation CTBookingResponseValidator

+ (id)validateResponseObject:(id)response
{
    if ([[[response objectForKey:@"VehRetResRSCore"] objectForKey:@"VehReservation"] objectForKey:@"@Status"]) {
        NSString *statusStr = [[[response objectForKey:@"VehRetResRSCore"] objectForKey:@"VehReservation"] objectForKey:@"@Status"];

        if ([statusStr isEqualToString:@"Confirmed"]) {
            Booking *b = [[Booking alloc] initFromRetrievedBookingDictionary:[[response objectForKey:@"VehRetResRSCore"]
                objectForKey:@"VehReservation"]];
            return b;
        }
    }

    return nil;
}

+ (void)validateCompleteResponse:(id)response withSuccess:(CTBookingResponseValidatorSuccess)sucess andError:
    (CTBookingResponseValidatorError)error
{
    NSArray *errorObject = [CTError validateResponseObject:response];
    if (errorObject) {
        if (error) {
            error(errorObject);
        }
        return;
    }

    Booking *booking = [self validateResponseObject:response];

    if (sucess) {
        sucess(booking);
    }
}

@end
```

Screen 14 – CTBookingResponseValidator

```objc
- (void) getBookingDetails:(NSString *)bookingEmail bookingID:(NSString *)bookingID {
    self.hud = [[CTHudViewController alloc] initWithTitle:@"Searching"];
    [self.hud show];

    [CTBookingRequester requestBookingWithEmail:bookingEmail bookingId:bookingID andDelegate:self];
}

- (void) requestFinished:(ASIHTTPRequest *)request {
    NSString *responseString = [request responseString];

    if (kShowResponse) {
        DLog(@"Response is \n\n%@\n\n", responseString);
    }

    id response = [responseString JSONValue];

    [self.hud hide];
    //[hud autorelease];
    self.hud = nil;

    [CTBookingResponseValidator validateCompleteResponse:response withSuccess:^(Booking *booking) {
        if (booking) {
            [self saveCustomObject:booking];
            [self.arrayOfBookings addObject:booking];
            DLog(@"Break");
            [self setupPage];
            [self hideGetBookingInfoView];
        } else {
            UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Warning" message:@"Your booking has either been cancelled or is currently unconfirmed." delegate:self
                cancelButtonTitle:nil otherButtonTitles:@"OK", nil];
            [alert show];
        }
    } andError:^(NSArray *errors) {
        for (CTError *er in errors)
        {
            if ([er.errorShortTxt isEqualToString:@"No matching bookings found"]) {
                UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"No matching bookings found" message:@"Check your Booking ID and Email Address and try again." delegate:self
                    cancelButtonTitle:nil otherButtonTitles:@"OK", nil];
                [alert show];
                //[alert release];
            } else {
                UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Warning" message:er.errorShortTxt delegate:self cancelButtonTitle:nil otherButtonTitles:@"OK", nil];
                [alert show];
                //[alert release];
            }
        }
    }];
}
```

Screen 15 - CTBookingRequest and CTBookingResponseValidator working together

## 2 - Model

For the models, my mind is to create immutable models and split the responsibility in specific categories. It helps to understand the different ways to interact and handle to models.

One practical example of this is creating a model that can be initialized from multiple APIs (Foursquare, Google Places and Apple Maps, for example). The model information are encapsulated on the class and the parse of each specific API is reserved to categories.

In this project I applied this logic on the models Booking, Fee e CTCountry. But, in this document I'll only explain the CTCountry implementation.

## 2.1 – CTCountry

As explained above, the model class CTCountry is an immutable object with, only, the public initializers. The initializers are based on the context of the model and what ways this model can be considered valid.

```
@interface CTCountry : NSObject

@property (nonatomic, copy, readonly) NSString *currencyName;
@property (nonatomic, copy, readonly) NSString *currencyCode;
@property (nonatomic, copy, readonly) NSString *currencySymbol;
@property (nonatomic, copy, readonly) NSString *isoCountryName;
@property (nonatomic, copy, readonly) NSString *isoCountryCode;
@property (nonatomic, copy, readonly) NSString *isoDialingCode;

- (instancetype) initWithIsoCountryName:(NSString *)isoCountryName isoCountryCode:(NSString *)isoCountryCode andIsoDialingCode:
    (NSString *)isoDailingCode;

- (instancetype) initWithCurrencyName:(NSString *)currencyName currencyCode:(NSString *)currencyCode andCurrencySymbol:(NSString *)
    currencySymbol;

- (instancetype) initWithCurrencyName:(NSString *)currencyName currencyCode:(NSString *)currencyCode currencySymbol:(NSString *)
    currencySymbol isoCountryName:(NSString *)isoCountryName isoCountryCode:(NSString *)isoCountryCode andIsoDialingCode:(NSString *)
    isoDailingCode;

@end
```

Screen 16 – CTCountry

## 2.2 – CTCountry+NSArray

The first category for this model is the initializer based on the CarTrawler API. In the API response, the country has to be created based in an array of strings. This code in a category makes this rule clear and easy to maintain.

```
@implementation CTCountry (NSArray)

+ (instancetype) countryFromArray:(NSMutableArray *)csvRow
{
    NSString *isoCountryName = [csvRow objectAtIndex:0];
    NSString *isoCountryCode = [csvRow objectAtIndex:1];
    NSString *isoDialingCode = [csvRow objectAtIndex:2];

    CTCountry *ctCountry = [[CTCountry alloc] initWithIsoCountryName:isoCountryName isoCountryCode:isoCountryCode andIsoDialingCode:
        isoDialingCode];
    return ctCountry;
}

@end
```

Screen 17 - CTCountry+NSArray

## 2.3 – CTCountry+Coding

The second responsibility encapsulated on a category was the implementation of the NSCoding protocol. Once that this object is saved on the NSUserDefaults, the encoding and decoding of properties are easily explained on this category.

```objc
#import "CTCountry+Coding.h"

@implementation CTCountry (Coding)

- (void)encodeWithCoder:(NSCoder *)aCoder
{
    [aCoder encodeObject:self.isoCountryName forKey:@"ctCountry.isoCountryName"];
    [aCoder encodeObject:self.isoCountryCode forKey:@"ctCountry.isoCountryCode"];
    [aCoder encodeObject:self.isoDialingCode forKey:@"ctCountry.isoDialingCode"];
    [aCoder encodeObject:self.currencyCode forKey:@"ctCountry.currencyCode"];
    [aCoder encodeObject:self.currencySymbol forKey:@"ctCountry.currencySymbol"];
    [aCoder encodeObject:self.currencyName forKey:@"ctCountry.currencyName"];
}
- (id)initWithCoder:(NSCoder *)aDecoder
{
    NSString *isoCountryName = [aDecoder decodeObjectForKey:@"ctCountry.isoCountryName"];
    NSString *isoCountryCode = [aDecoder decodeObjectForKey:@"ctCountry.isoCountryCode"];
    NSString *isoDialingCode = [aDecoder decodeObjectForKey:@"ctCountry.isoDialingCode"];
    NSString *currencyCode = [aDecoder decodeObjectForKey:@"ctCountry.currencyCode"];
    NSString *currencySymbol = [aDecoder decodeObjectForKey:@"ctCountry.currencySymbol"];
    NSString *currencyName = [aDecoder decodeObjectForKey:@"ctCountry.currencyName"];

    return [self initWithCurrencyName:currencyName currencyCode:currencyCode currencySymbol:currencySymbol isoCountryName:
        isoCountryName isoCountryCode:isoCountryCode andIsoDialingCode:isoDialingCode];
}

@end
```

Screen 18 - CTCountry+Coding

## 2.4 – CTCountry+Factory

Once that this model is immutable we need a way to “update” the values of an instance. This category is responsible to make public the proper ways that this model can be changed and still be a valid struct. And, to finish the pattern, it ensures that the model is immutable by creating a new instance with the changed values.

```objc
@implementation CTCountry (Factory)

- (instancetype) countryWithCurrencyCode:(NSString *)currencyCode
{
    return [self countryWithCurrencyCode:currencyCode andCurrencySymbol:self.currencySymbol];
}
- (instancetype) countryWithCurrencyCode:(NSString *)currencyCode andCurrencySymbol:(NSString *)currencySymbol
{
    CTCountry *newCountry = [[CTCountry alloc] initWithCurrencyName:self.currencyName currencyCode:currencyCode currencySymbol:
        currencySymbol isoCountryName:self.isoCountryName isoCountryCode:self.isoCountryCode andIsoDialingCode:self.isoDialingCode];
    return newCountry;
}
- (instancetype) countryWithIsoCountryName:(NSString *)isoCountryName andIsoCountryCode:(NSString *)isoCountryCode
{
    return [self countryWithIsoCountryName:isoCountryName isoCountryCode:isoCountryCode andIsoDailingCode:self.isoDialingCode];
}
- (instancetype) countryWithIsoCountryName:(NSString *)isoCountryName isoCountryCode:(NSString *)isoCountryCode andIsoDailingCode:
    (NSString *)isoDailingCode
{
    CTCountry *newCountry = [[CTCountry alloc] initWithCurrencyName:self.currencyName currencyCode:self.currencyCode currencySymbol:
        self.currencySymbol isoCountryName:isoCountryName isoCountryCode:isoCountryCode andIsoDialingCode:isoDailingCode];
    return newCountry;
}

@end
```

Screen 19 - CTCountry+Factory

P.s.: The same thought was applied for the Booking and Fee models.

## 3 – Controller and CTHelper

Those changes on network classes and models made significant changes on helpers and controllers.

The controllers became more clear and specific, and so, more testable and easier to maintain.

The CTHelper class lost responsibility and had the same improvements of the controllers.

## Tests

To finish the small changes, I created a test target and implemented a simple test that checks if the Fee initializer is parsing the correct information and if the private method is being called.

```
@interface CarHireTargetTests : XCTestCase

@end

@interface Fee (Private)

+ (NSString *)parsePurpose:(NSString *)purpose;

@end

@implementation CarHireTargetTests

- (void)testExample {
    NSDictionary *feeDictionary = @{@"@Amount":@"23.45", @"@CurrencyCode":@"BRL", @"@Purpose":@"23"};

    id feeMock = OCMClassMock([Fee class]);
    OCMExpect([feeMock parsePurpose:[OCMArg any]]).andForwardToRealObject;

    Fee *mockedFee = [Fee feeWithDictionary:feeDictionary];

    OCMVerifyAll(feeMock);

    XCTAssertEqualObjects(mockedFee.feeAmount, @"23.45");
    XCTAssertEqualObjects(mockedFee.feeCurrencyCode, @"BRL");
    XCTAssertEqualObjects(mockedFee.feePurpose, @"23");
    XCTAssertEqualObjects(mockedFee.feePurposeDescription, @"Fee to pay on arrival.");
}

- (void)testPerformanceExample {                    Time: 0.000 sec (inf% worse, 41% STDEV)
    // This is an example of a performance test case.
    [self measureBlock:^{
        [self testExample];
    }];
}

@end
```

Screen 20 - Test of the Fee model

## EXTRA

There was a few patterns and changes that I did not do because those would be really huge changes:

- Remove the setFrame operations from the code and change it for Auto Layout constraints
- Improve UITableView implementations and remove few graphic elements from controller Xib and put it on UITableViewCell specifics Xibs
- Remove the UITableViewDelegate and UITableViewDataSource code to specific classes. So, I would be able to use a single UITableViewController class for every UITableView on the project.
- Split the UIViewController on more sub controllers.
- Interface tests
- Put the CTTableViewAsyncImageView class and all the related classes into a external project and link by cocoapods. This code can be used in other projects and should be on an external lib.

- Change the CTTableViewAsyncImageView from a UIView implementation to a UIImageView category.