Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name: **Octavian Morcov**
Student ID: **X13101242**
Supervisor: **Muhammad Iqbal**

SECTION 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarised overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature:_____
Date:_____

NB. If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the College's Disciplinary Committee. Should the Committee be satisfied that plagiarism has occurred this is likely to lead to your failing the module and possibly to your being suspended or expelled from college.

Complete the sections above and attach it to the front of one of the copies of your assignment,

What constitutes plagiarism or cheating?

The following is extracted from the college's formal statement on plagiarism as quoted in the Student Handbooks. References to "assignments" should be taken to include any piece of work submitted for assessment.

Paraphrasing refers to taking the ideas, words or work of another, putting it into your own words and crediting the source. This is acceptable academic practice provided you ensure that credit is given to the author. Plagiarism refers to copying the ideas and work of another and misrepresenting it as your own. This is completely unacceptable and is prohibited in all academic institutions. It is a serious offence and may result in a fail grade and/or disciplinary action. All sources that you use in your writing must be acknowledged and included in the reference or bibliography section. If a particular piece of writing proves difficult to paraphrase, or you want to include it in its original form, it must be enclosed in quotation marks

and credit given to the author.

When referring to the work of another author within the text of your project you must give the author's surname and the date the work was published. Full details for each source must then be given in the bibliography at the end of the project

Penalties for Plagiarism

If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the college's Disciplinary Committee. Where the Disciplinary Committee makes a finding that there has been plagiarism, the Disciplinary Committee may recommend

- that a student's marks shall be reduced
- that the student be deemed not to have passed the assignment

- that other forms of assessment undertaken in that academic year by the same student be declared void
- that other examinations sat by the same student at the same sitting be declared void

Further penalties are also possible including

- suspending a student college for a specified time,
- expelling a student from college,
- prohibiting a student from sitting any examination or assessment.,
- the imposition of a fine and
- the requirement that a student to attend additional or other lectures or courses or undertake additional academic work.

National College of Ireland

BSc in Computing

2013/2014

Octavian Morcov

X13101242

X13101242@student.ncirl.ie

Interactive Cars

Technical Report

Final Report

Table of Contents

Introduction

1.1 Executive Summary

The love of cars brings people around the world together. Auto shows are the biggest events in the automotive industry which attract millions of people around the world. There are thousands of such events every year, some are massive, and some only attract a few hundreds of visitors. But this is a great chance for people to check their favourite models, explore new ones or even to buy a car.

Car companies use these events as a chance to promote their cars or ideas and also get inspired by competitors' cars.

As these events are massive, with many spectators, it is important that people get enough information about the car they explore there. All the flyers and brochures still work but they were never useful. Beside taking great pictures of those cars, it is important that user also get all the information about the cars on events, share their experience and photos with friends and save everything they saw for further references. No doubts that websites do their work great, but it is a hassle to go to the website, searching for a car, find the right section, etc. Users simply avoid doing all these steps. Also, another problem I discovered is that there is never enough staff to assist people and provide more detailed information.

1.2 Background

I came with an idea to combine IoT technologies with software in order to build and app that will give users the right information at the right time with minimum effort. My idea is to use beacons which behave as its name says. Beacons are small chips that broadcast information about themselves every half second or more. A beacon works in the same way as a light house. It kind of says: “Hey, I am beacon number 123 and I am about 5 meters away from you”. So I thought that using this concept will change the way users explore cars.

By installing a beacon to every car in a show room, the cars will start kind of interacting with people. Cars will start giving users information about themselves. And this is where the name of the app came from: “Interactive Cars”. When user will approach a car and look at the device, the information about the car will already displayed on the screen. The car now kind of says: “Hey, I am BMW i3. I have this, and that, and that”.

1.3 Project’s Purpose

The aim of the project is to develop an Android beacon app that will help car shows and car salons visitors to get the most from attendance. A beacon will be attached to each car in a gallery and the app will scan beacons. The app will detect in to which car the user is closer and will display detailed information about that car.

User will also be able to call assistance by clicking a button. This is useful when user needs more information like the sale, company can do or all the requirements for buying the car in credit.

Users will also be able to save their favourite cars or share them with friends. The app will also provide access to mobile’s camera so it will be easier to take pictures and share them.

Using this app, users will get the most from car shows, car shops and staff’ work will be easier.

1.4 Definitions, Acronyms, and Abbreviations

GA Gallery App (The app)

CA Car Gallery (The building in which the app is used)

UI User Interface

1.5 Technologies

UI

The Interactive Cars is a hybrid application built using Ionic SDK. Ionic is an open-source SDK built on top of AngularJS and Apache Cordova. I chose this technology as it allows me to develop hybrid mobile apps using Web technologies like HTML, CSS, Angular. This way, I was able to build my app for iOS and Android by working only with web technologies.

Database

For the database I used MongoDB and Couch DB. The reason I used 2 databases is that being familiar with MongoDB I used it for the Staff website. But I used Couch DB and Pouch DB for the mobile app because it is very easy and fast to build local databases on mobile devices that doesn't require internet. Pouch DB is an extension of Couch DB which allows to build local offline database which can sync later with cloud database. It is very important to use this feature because it will improve performance a lot.

The non-sql databases I chose are also good for storing IoT data. For analysis purpose I will collect data every time user approaches a car. This way, managers will know which cars were most visited, where it was the most crowded, etc

Beacons

Beacons is the main technology that makes this app unique and useful. Beacons are small Bluetooth Low Energy (BLE) devices that broadcast signals. There is actually nothing magic behind beacons. A Beacon sends very small information:

- Universal ID (UUID) – this helps identify groups of beacons. For example, Microsoft bought 100 beacons. So, Microsoft will set every beacon's UUID to be "123xx-123xx-000rre0wqw". Then Microsoft builds an app that will scan beacons. It is useful to use UUID when developer wants to make sure that the app will scan only the beacons Microsoft owns.
- Major – Major is usually used to group some beacons in an organization. For example, all beacons on floor 1 will have the major: 1111 but beacons on floor 2 will have major: 2222
- Minor – Minor is used to identify each beacon. It is used as the actual id f every single beacon
- RSI – is the strength of the signal.

Node.js

Node.js is the JavaScript for server side applications. I used Node.js to make the Interactive Cars app exchanging information with the staff website.

Socket.io

Socket.io is a JavaScript library for creating real-time applications. In my case I used it notify clients and staff for new incoming messages. For example, when user taps o button to get assistance, a message is sent to the server and the server sends the message to the right end-point(staff) to notify that somebody requires help. The socket.io allows to create a channel between server and client, and exchange messages in real-time.

2 User Requirements Definition

The user requirements gathering was the second step in developing my app. After finishing my research about how possible it was to implement my idea, I went to Hyundai and VW garages in Co. Kildare and collected some information. When I met the company's staff I asked them the following questions:

- How many cars they have in their center?
- How many centers are worldwide?
- What is the biggest number of cars you have/had in a center?
- Do your company attend any car shows/events? If yeas, how many a year and how many cars they usually show at the event?
- Does it get busy in car centers or car shows?
- How staff deal with many visitor's questions?
- How often the questions asked are the same?
- Did it happen to not be able to help all visitors?

I got answers to all the questions listed above. And based on them I created the user requirements. But I still needed more information about car shows, galleries and centers so I have to do an online research.

I discovered that centers can be small and very large. You can see from the picture below the Nissan center which has over 5 cars. And it is quite difficult for staff to walk every client and ask the same questions. Instead, attaching a beacon to every car would be a good idea for making customers happy and staff' work more productive.

With the Interactive Guide users will not need any assistance for the most common questions as the app will provide them with all information needed to make a decision. However, if user will need some more specific details that the app can't give, user will be able to tap on a button to ask for assistance. The staff will be notified and will know where the user is located and will assist him/her.

Below, I listed all the user requirements for implementing the app.

As user walks through building or in the parking, the app should automatically change content/page when approaching a car. On the page should be displayed all the information available about a car.

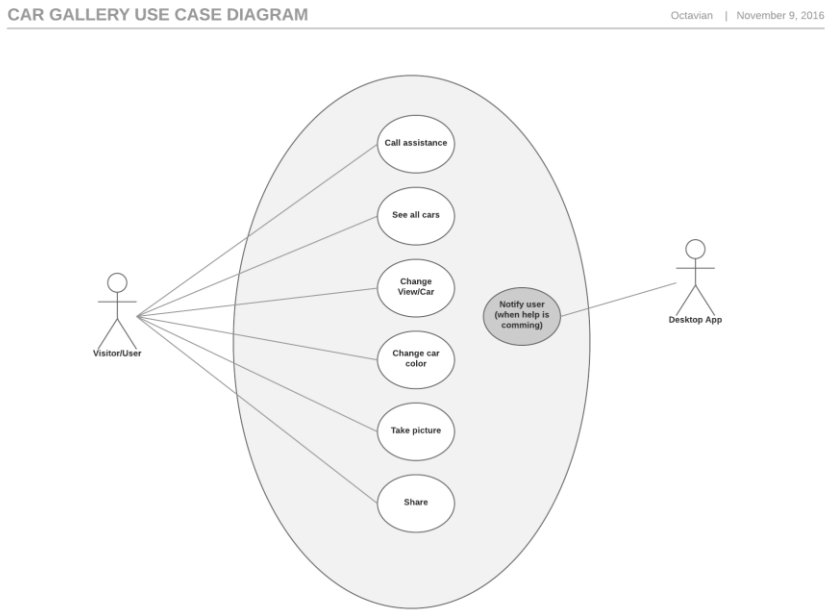
- User shall be able to see information about other cars without.
- The app shall to see a list of all cars available in the center user is located.
- User shall be able to see information about a car even if user doesn't approach any.

- User shall be able to get assistance by tapping a button.
- User shall also be able to cancel the assistance request.
- User shall be able to open camera from the app in order to take pictures
- The app shall display the taken picture within the app
- User shall be able to share pictures
- The User shall be able to delete pictures
- The user shall be able to swipe both sides in order to change car images.
- Users shall be able to see car themselves in different colors, but no only the color names.

3 Requirements Specification

3.1 Functional requirements

3.1.1 Use Case Diagram



3.1.2 Requirement 1: See all cars

3.1.2.1 Description & Priority

The user wants to see all the cars in the centre. It is very important that users.

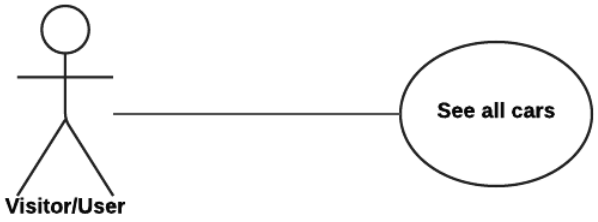
The importance of this requirement is high because users need to see what cars are in the showroom.

3.1.2.2 Use Case

Scope

The scope of this use case is to demonstrate how users can get all cars displayed on the screen.

Use Case Diagram



Flow Description

Precondition

Before this use case can be initiated, user doesn't need to be connected to the internet. It is not necessary to have Bluetooth for this use case.

Activation

This use case starts when the user clicks on "All cars" button from the menu.

Main flow

1. The User clicks on "All Cars" tab from navigation bar on the bottom of the screen.
2. It goes to another page where all the cars are displayed.

Termination

The system displays all cars in a grid.

Post condition

The system goes into a wait state

3.1.3 Requirement 2: Call assistance

3.1.3.1 Description & Priority

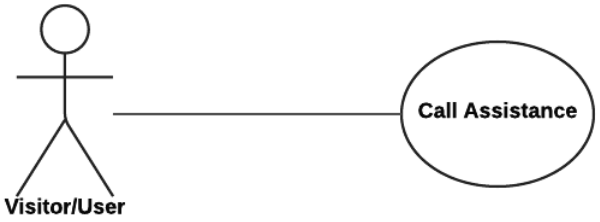
The user needs more information that is not in the app and calls for assistance.

3.1.3.2 Use Case

Scope

The scope of this use case is to send a message to the desktop application to let staff know that a user needs assistance.

Use Case Diagram



Flow Description

Precondition

The system shall have access to internet.

Activation

This use case starts when user clicks on “Assistance” icon from the top navigation bar. User must be in range of at least a beacon. This will make sure that when user is far away from the show area, it will not be possible to ask for assistance.

Main flow

1. When the assistance icon was taped, the Assistance Page will show up.
2. The user taps on the “Get Assistance”.
3. The system sends the request to the server using websocket.
4. The websocket sends to the staff website a message that contains the id of the car where assistance is needed, the name of the car, and client’s socket id.

Termination

New information is added to the staff’ webpage with the information about the car where assistance is needed.

On the client's page the "Get Assistance" button is disabled and another button "Cancel request" appeared below the other button.

Post condition

The system goes into a wait state

3.1.4 Requirement 4: (Staff) Accept user request

3.1.4.1 Description & Priority

Staff received an assistance request and must accept it.

3.1.4.2 Use Case

Scope

The scope of this case is for staff to answer to user requests.

Flow Description

Precondition

The system must have access to internet.

Activation

This use case starts when user(staff) opens the webpage and the list with requests shows up.

Main flow

1. User chooses a request from the list and clicks on the "Accept Request button". The choice is made based on First – Come – First – Served.
2. User goes to the right car and taps on "Assistance Arrived" button to notify the client that staff is around.
3. User taps on "Done" button when client is happy with the help provided

Post condition

After the "Accept Request" button clicked, other two buttons "Cancel" and "Done" buttons appear next to the first one. The "Accept Request" button becomes disabled until the "Cancel" or "Done" button is pressed.

3.1.5 Requirement 5: View other pictures of the selected car

3.1.5.1 Description & Priority

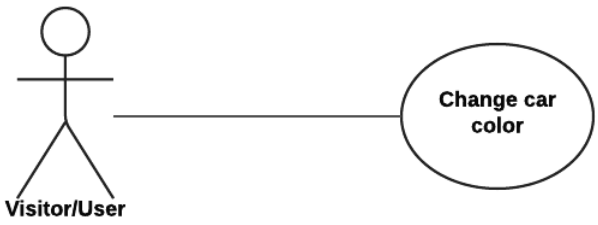
User wants to see other pictures of the car. Every car should have at least two pictures so that user could see the car from different sides.

3.1.5.2 Use Case

Scope

The scope of this use case is to show other pictures of the selected car.

Use Case Diagram



Flow Description

Precondition

The system shall not have access to internet.

Activation

This use case starts when user swipes the car's picture to a side.

Main flow

4. The user swipes the picture to whether left or right
5. The picture changes.

Post condition

The system goes into a wait state

3.1.6 Requirement 4: Take photo

3.1.6.1 Description & Priority

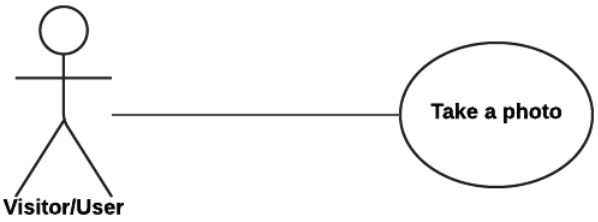
User takes a photo of the car and the image is save in user's own gallery.

3.1.6.2 Use Case

Scope

The scope of this use case is to save the image of a car to the gallery.

Use Case Diagram



Flow Description

Precondition

The system shall not have access to internet.

Activation

This use case starts when user clicks on “Capture” button. As at the moment of writing this report I hadn’t had a decision about where to place the “Capture” button, it might be whether on the side menu or the button will appear by tapping and holding on the screen.

Main flow

1. The user clicks on the “Capture” button.
2. The system opens the camera.
3. User takes a photo of the car.
4. The system adds the picture to the list.
5. The system waits for other action. After taking a picture, the camera closes and going back where user left.

Post condition

The system goes into a wait state

3.1.7 Requirement 6: See all pictures

3.1.7.1 Description & Priority

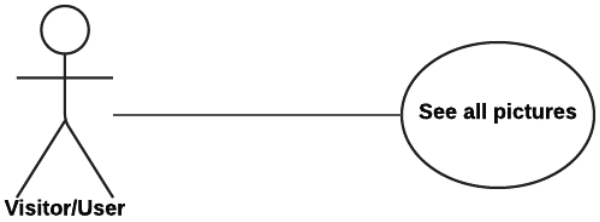
User opens the gallery of pictures that were taken.

3.1.7.2 Use Case

Scope

The scope of this use case is to access the photos that were taken at the gallery.

Use Case Diagram



Flow Description

Precondition

The system shall not have access to internet.

Main flow

1. User clicks on Gallery tab from the bottom menu.
2. It goes to the Gallery page where all photos are displayed in a grid

Post condition

The system goes into a wait state

3.2 Amended Requirements

Here is described the requirements that are incomplete or never done due to time constraints.

- Change colour of the car – User can tap on any colour from the list and see the selected car in that colour.
- Display cars in Augmented Reality – User can point the camera to a poster next to the car and see the car in different colours and animated.

3.3 Non-Functional Requirements

3.3.1 Availability requirement

The app is to be uploaded on Google Play store on the day of presentation. I uploaded the first version of the app but it hasn't all functionalities listed in Requirements Specifications.

The staff webpage is running 24 hours on a server.

The only constraint for the client to send requests is that client must be in the range of beacons which means in the show area.

3.3.2 Security requirement

3.3.3 Reliability requirement

As planned, all the functionalities of the application work in depending if user meets the constraints of the app.

To have most of the functionalities available, user must have at least Bluetooth switched on. This way user will be able to walk around the room and get car information.

User will have all the functionalities if he/she will also have the Internet switched on. Having internet on, user can get help from staff, share cars information and taken photos with friends.

The system will not behave properly if a beacon will not function or its battery will be down. If that will happen, when approaching a car, there will be no information displayed about it.

The application shouldn't crash if there will be no internet connection. However, if there will be no internet connection, only partial content will be displayed.

3.3.4 Maintainability requirement

All beacons batteries shall be replaced every 4 months and shall be checked every day. The checking consists in using the app to see if app can detect all the beacons.

Any error should be fixable by only one developer.

4 Interface requirements

The Interface requirements were met by using Ionic. It allows to user predefined html tags and CSS classes to have responsive pages and components. The interface on Android slightly defer from the one on iOS but that is not a drawback. It is because both platforms display interfaces a little different.

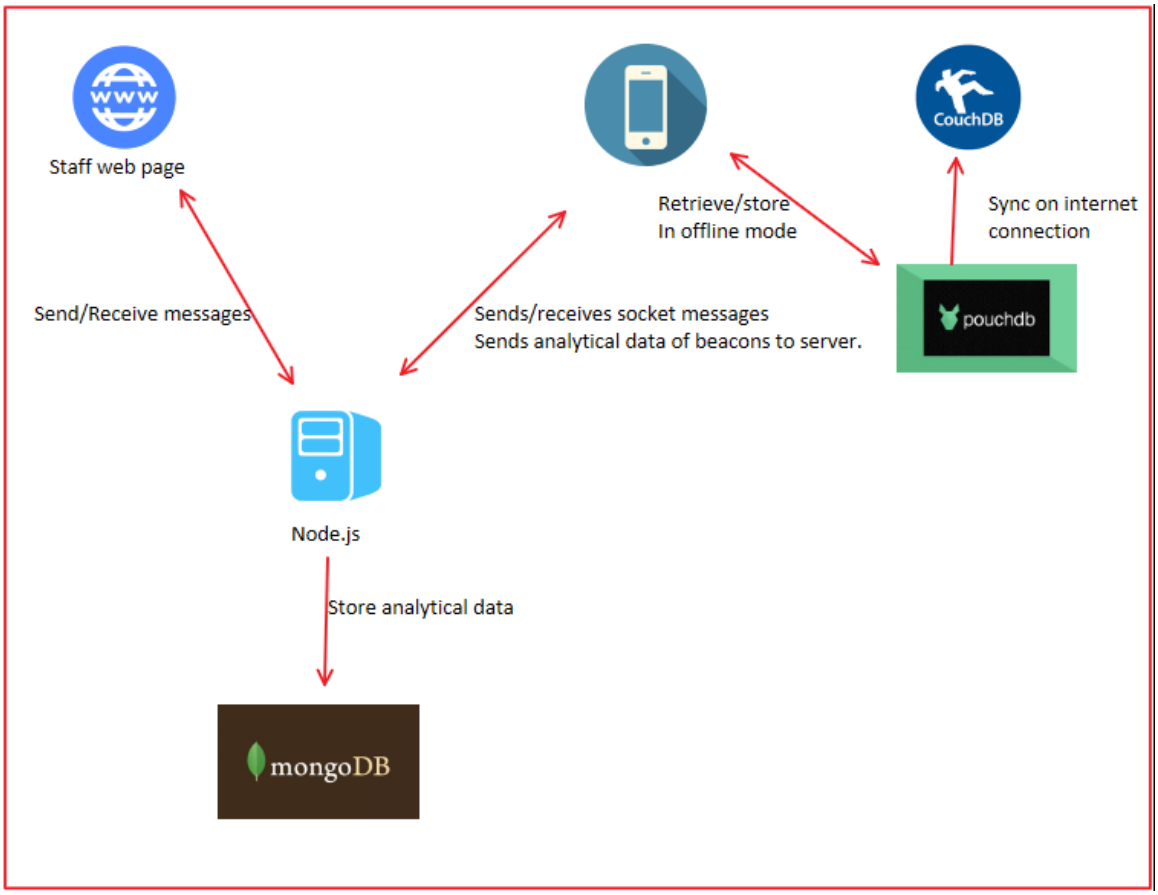
5 Design Architecture

The app consists of 3 components:

1. Server – which handles all the communication between staff and client.

2. Client's Front-End – is the mobile app for clients that scans beacons and send/receive requests to/from the staff (through the server).
3. Staff's Front-End – is the web page for staff that receives the requests from users and sends back answers.

The picture below shows the architecture of the entire project.



As you can see from the picture, the clients send and receive requests and answers from the mobile phone. As you can also see, client's devices send analytical data to the server. The data contains information about when client approach to a car, when client distanced the car, and what car was that. Below is the flow of Client architecture:

- When Client app starts, it retrieves all the information about cars from the PouchDB.
- When Client approaches a car, the time is recorded. When Client distances the car, the data is sent to the server which in turn is stored in MongoDB for analysis purposes.
- When user taps on button to ask for request, the device connects to the server.
- Once the connection is established, the message is sent.

- On the server, it looks for sockets that represent the staff and message is forwarded to them.
- When an answer is sent back, the client will receive it automatically thank to socket.io.
- When there is internet connection, the data from PouchDB is synced with the CouchDB database.

As you also see from the picture, staff have their own page where they receive and send request and responses. The flow of the Staff architecture is as follows:

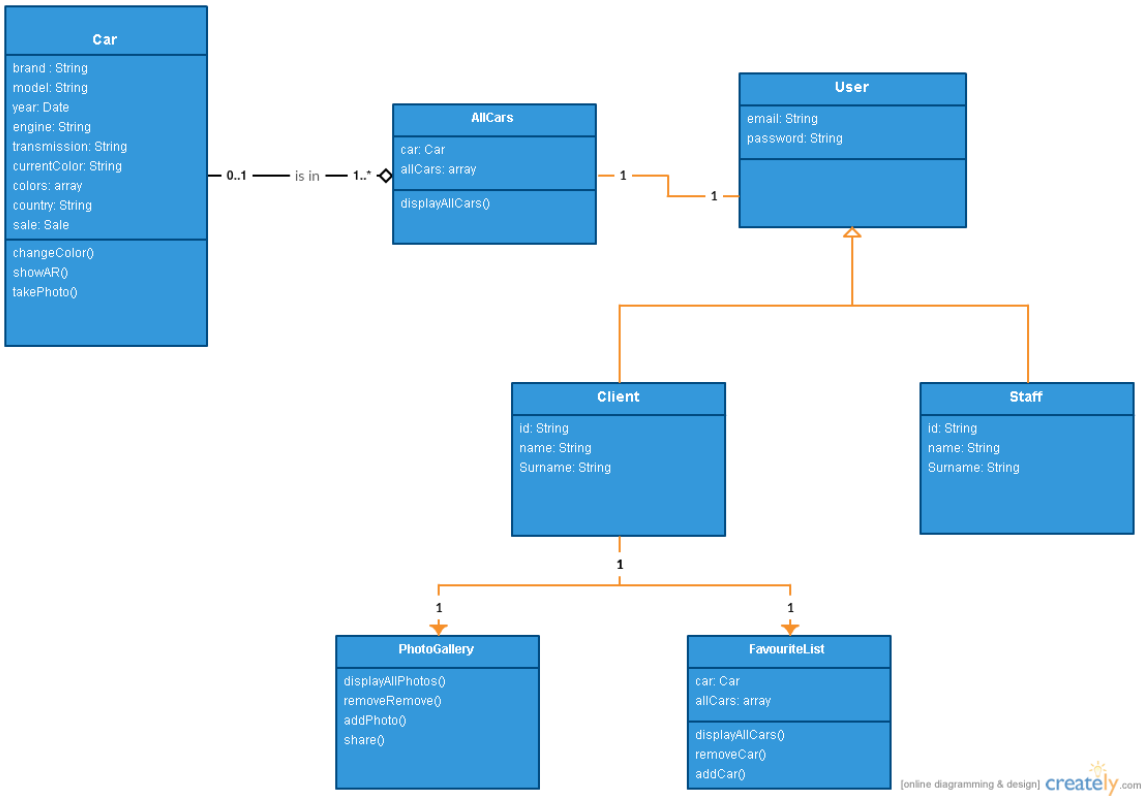
- When, page is loaded a socket connection is created with the server in order to receive real-time messages from the clients.
- When staff accepts a request, a message is sent back to the client through the server saying that assistance is on its way.

The information about cars is stored in the database on personal device. The database (Pouch DB) works in offline mode. But, when there is a connection, the local database syncs with the cloud one. That is the reason I use 2 databases: MongoDB and CouchDB (PouchDB is part of CouchDB)

6 Implementation

6.1 Architecture

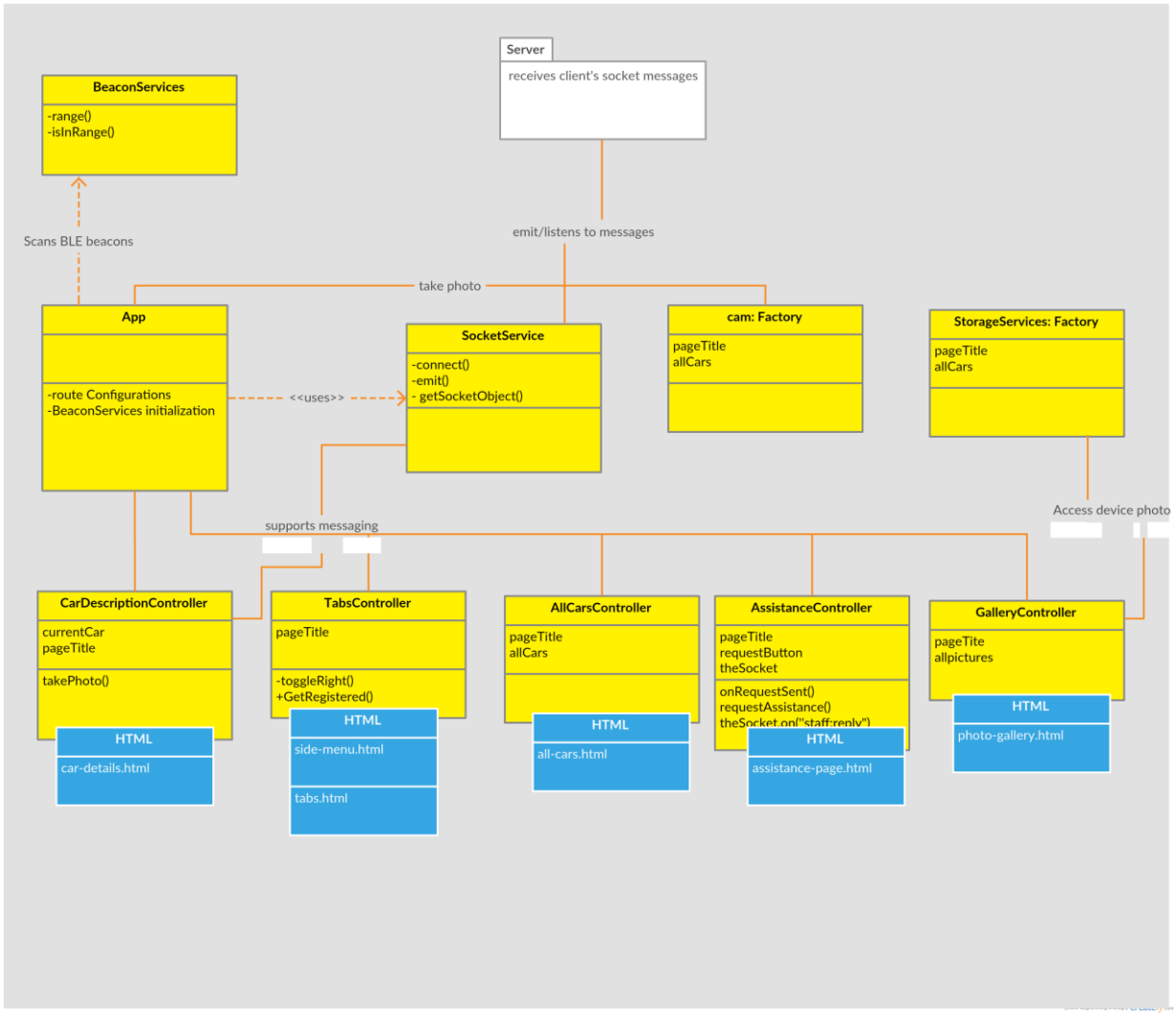
The picture below shows the objects I used in my application.



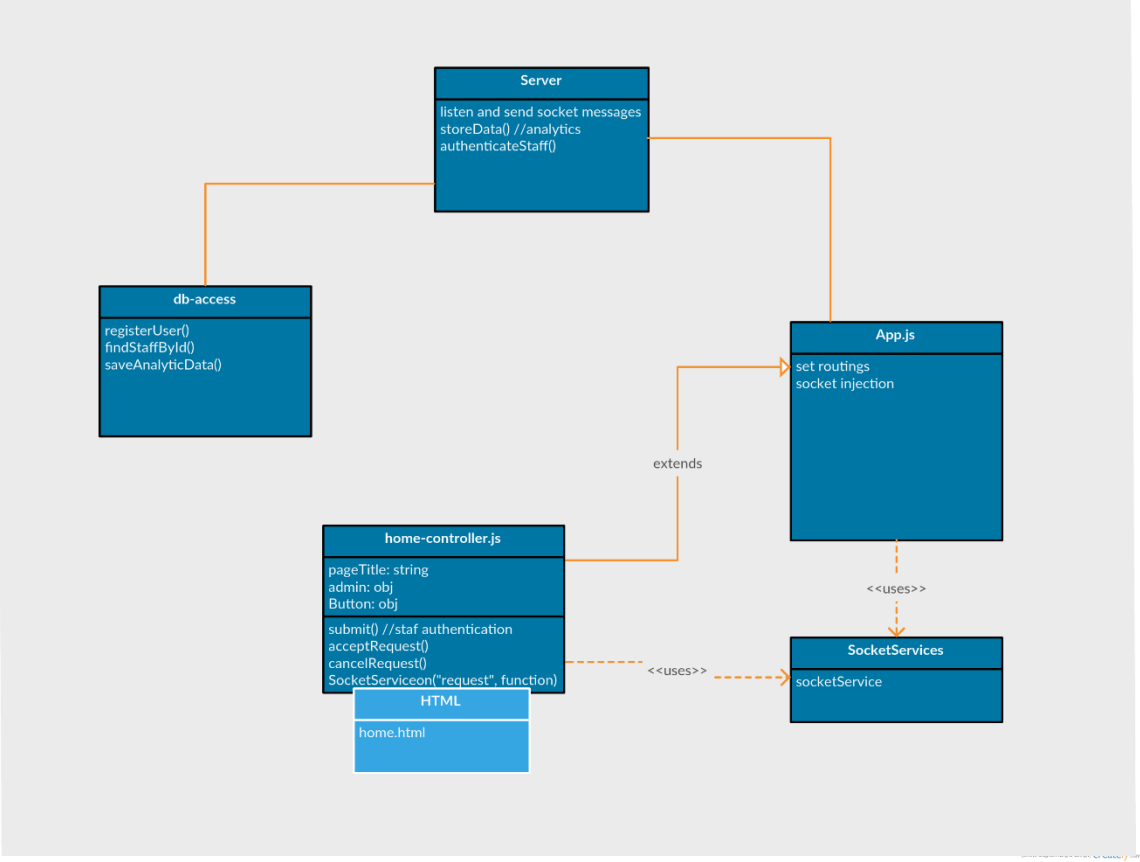
The following diagram shows the architecture of the client's mobile application. As you can see from the picture, the app has some dependencies such as:

- BeaconServices – which access the device's Bluetooth in order to scan beacons
- SocketServices – the module that connects to a server and create a channel through which the messages are sent and received in real-time.
- Cam – is the service that access device's camera in order to take picture and then return to the app where user left.

From the diagram, you also can see that every page has a controller as Angular is an MVC framework.



The diagram below shows the architecture of the server part of the application and staff's front-end.



As you can see, the server handles all the socket messages that come from the staff front-end app. As you remember, this server also receives all the client's requests from the client's mobile app.

When client enters and leaves a region (a beacon's signal range), this information is collected and sent to the database which in turn is stored in MongoDB database.

In this version of my application the staff's website has only one page and authentication.

6.2 Implementation

6.2.1 Sockets

The screenshot below shows the code that injects socket.io module on the client side of the Staff website, sends messages and listens to incoming messages from server. The angular socket wrapper connects to "http://localhost:3000" and returns itself to be used within the app to emit messaged and listen to incoming ones.

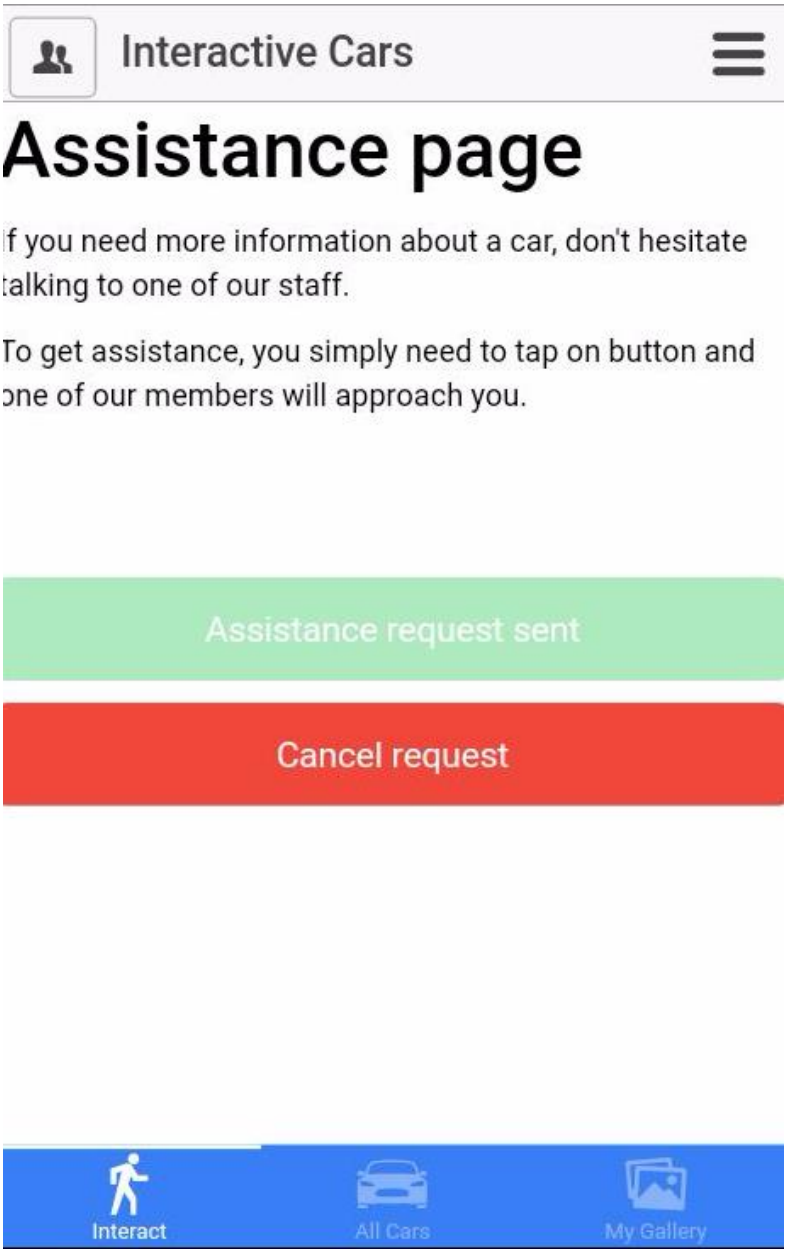
```
(function(){

  angular.module('myApp')
      .service('SocketService', ['socketFactory', SocketService]);

    function SocketService(socketFactory){
        return socketFactory({

            ioSocket: io.connect('http://localhost:3000')

        });
    }
})();
```

In the code below it is shown what happens when client taps on “Request Assistance”. On click, it will create the message object which will have 2 parameters: carId and carName. The carId is the beacon’s minor (As I mentioned in the beginning, the minor is the unique ID of a beacon). The name of the car I get from a function that takes as parameter the closest beacon’s minor and find the car name based on that.

Now that I have the message ready, it is time to connect to the server. The mobile app doesn’t connect the same as the staff’s website. I made the app to connect to server only when user does a request because otherwise there is no need to have the device connected as it might slow down the performance of the app (the staff’s website connects when the page is loaded). As you see, the socketFactory connects to localhost where the server runs.

```
$scope.requestAssistance = function () {
  var msg = {
    carId: $rootScope.nearestBeacon.minor, //116
    carName: getCar($rootScope.nearestBeacon.minor)  //VW Polo
  };
  $scope.theSocket = socketFactory({ioSocket: io.connect('http://localhost:3000')});
  $scope.theSocket.emit('send:request', msg);
  onRequestSent();
  $scope.theSocket.on("staff:reply", function (data) {
    console.log(data);
  })
};
```

Then I am sending the created message to the server using socket.io. The message passes as a parameter the Object. After sending the socket message, the onRequestSent() message is sent which updates the client’s screen view by disabling the “Request Assistance” button and displays next to it the “Cancel Request” button.



Then I set the theSocket object to listen to “staff:reply” messages. It means that, if staff will accept this particular request, a message will be sent back to client saying that staff is around and will make the phone buzzing.

The following codes demonstrates how server handles all the socket messages from client and staff apps.

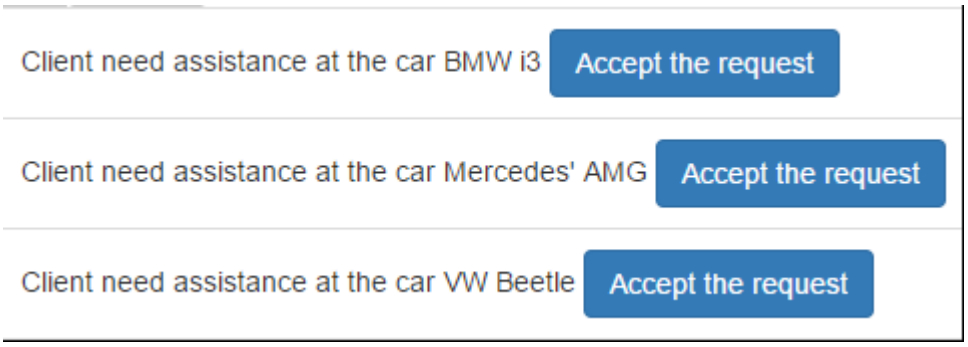
```
io.on('connection', function(socket){
    console.log('a user connected: ' + socket.id);
    // //register client
    // if ( userSockets[socket.id] == null) {
    //     userSockets[socket.id] = socket.id;
    // }

    socket.on("register:admin", function (data, fn) {
        var admin = data.admin_id
        console.log(admin);
        fn(true)
    });

    socket.on("notify:accepted", function (data) {
        console.log(data);
        console.log(typeof data);
    });

    socket.on("notify:canceled", function (data) {
        console.log(data);
        console.log(typeof data);
    })

    socket.on("notify:done", function (data) {
        io.emit("staff:reply", true);
        console.log(data);
        console.log(typeof data);
    });

    socket.on('client:cancel:request', function (data) {
        console.log("client canceled request: " + data)
    })

    socket.on('send:request', function(data){
        // var request = data.car;
        console.log("arrived request from car===::::: " + data);
        io.emit('request', data)
    });

    socket.on('disconnect', function(){
        console.log('user disconnected');
    });
});
```

The following code shows what happened when staff's web application receives a message from the server (server passed client's request message to the staff's website). Here, the SocketService.on() is the function that listens to incoming messages of type 'request' (this is the assistance request that is sent by client who needs more information about a car). In this case, when a request message arrives, it will create a Button object that will contain the parameters received from socket

message such as carId (where the user sent the request from) and carName. Then this button will be attached to the list in html file. Now, staff knows that there is a person who needs assistance and staff also know where the client is located.

```
//listen for request, and add it to the list
SocketService.on("request", function (data) {
    var request = new Button(data.carId, data.carName);
    $scope.assistanceRequests.push(request)
})
```

Below is the screenshot showing the html page result when an assistance request arrives from the client.



All the requests from clients have the following form:

```
$scope.assistanceRequests = [
    new Button("111", "BMW i3"),
    new Button("112", "Mercedes' AMG"),
    new Button("113", "VW Beetle"),
    new Button("114", "VW Scirocco"),
    new Button("114", "VW Scirocco")
];
```

As you can see, when a client taps on button to require assistance, a message with the car's beacon id and car's name is sent to the server which in turn is forwarded to the staff's front-end app and stored in the assistanceRequests list.

The next screenshot shows the code that is executed when staff clicks on the "Accept the request" button. On click, the instance of that button is passed to the $scope.acceptRequest function which sets the 'accepted' parameter to true. And the important part I wanted to show is how the message is sent back to the server. SocketService.emit("notify:accepted") emits a message of type "notify:accepted". The second parameter can be anything from primitive data up to JSON files. In this screenshot I am passing the Boolean "true" because this part wasn't finished at the time of writing this report. However, instead of true it will be passed the client's socket id so that server will know to which socket end-point to forward the message.

```
$scope.acceptRequest = function (req) {
    req.accepted = true;
    SocketService.emit("notify:accepted", true)
};
```

6.2.2 Beacons

This is the most interesting part of the app because it is what makes the app unique.

```
//listen to the broadcast.....
$rootScope.$on("$cordovaBeacon:didRangeBeaconsInRegion", function (event, data) {
  var uniqueBeaconKey;

  for(var i = 0; i < data.beacons.length; i++) {
    uniqueBeaconKey = data.beacons[i].uuid + ":" + data.beacons[i].major + ":" + data.beacons[i].minor;
    $rootScope.beacons[uniqueBeaconKey] = data.beacons[i];

    //set nereast beacon to be any beacon which is detected first by the app.
    if($rootScope.nearestBeacon == null) {
      $rootScope.nearestBeacon = data.beacons[i];
    }
    else {
      console.log('it is not null' + $rootScope.nearestBeacon.minor);
    }
    updateNearestBeacon(data.beacons, $rootScope.nearestBeacon, function (result) {
      $rootScope.nearestBeacon = result;
      if(prevClosest != $rootScope.nearestBeacon) {
        $window.location.href = "#/tab/home/"+ $rootScope.nearestBeacon.minor;


        // console.log("NEW BEACON");
        // AnalyticsManager.exitAnalytics(prevClosest.minor);
        // AnalyticsManager.arrivalAnalytics();
        prevClosest = $rootScope.nearestBeacon;
      }

      $compile($('#nearestB'))($rootScope)
    });

    //update nearest RSSI
    trackAndUpdateNearestRSSI(data.beacons[i]);
  }
```

Everything that is related to scanning beacons is done in one function. When the app starts, I make a call to run all the codes from the screenshot above. And what it does is that it starts listening to “$cordovaBeacon:didRangeBeaconsInRegion” event which means that if device detects a beacon nearby, that function will be called. The function passes 2 parameters: event and data. The data parameter is the important thing as it holds the information about the detected beacon such as minor and major numbers, and signal strength.

Then I check if whether it is the nearest beacon to the mobile device. If it is the nearest beacon, then it updates client’s page to display the car that represents the beacon.

When the page is changed, beacons information is passed to the page's controller in order to find the right car's details and picture.

All the code above will be executed every half-second or a second in order to compare every beacon's signal's strength and determine which one is the closest.

As you can see, I commented out the analytical functions as it was not finished at the time of writing this report.

6.2.3 Camera

In the following code it shows how the cordova accesses device's camera. The Service I created uses cordova to access camera and StorageService that will access mobile's local storage where image's paths are stored.

```
.factory("cam", function ($cordovaCamera, StorageService) {
var allPic = [];
var tp = function() {
  var options = {
    quality : 90,
    destinationType : Camera.DestinationType.DATA_URL,
    sourceType : Camera.PictureSourceType.CAMERA,
    allowEdit : false,
    encodingType: Camera.EncodingType.JPEG,
    correctOrientation: true,
    popoverOptions: CameraPopoverOptions,
    saveToPhotoAlbum: true
  };

  $cordovaCamera.getPicture(options).then(function(imageData) {
    allPic.push("" + imageData);
    StorageService.add("" + imageData);
  }, function(err) {
    // An error occured. Show a message to the user
  });
}

return{
  takePhoto:tp,
  getAllFotos:allPic
}
```

In the code above, an array is created that will hold paths to pictures that were taken by this app.

Then when user taps on "Take Photo" button, the "tp" function is called. The function will create an object where all the parameters of a picture is set such as: where the

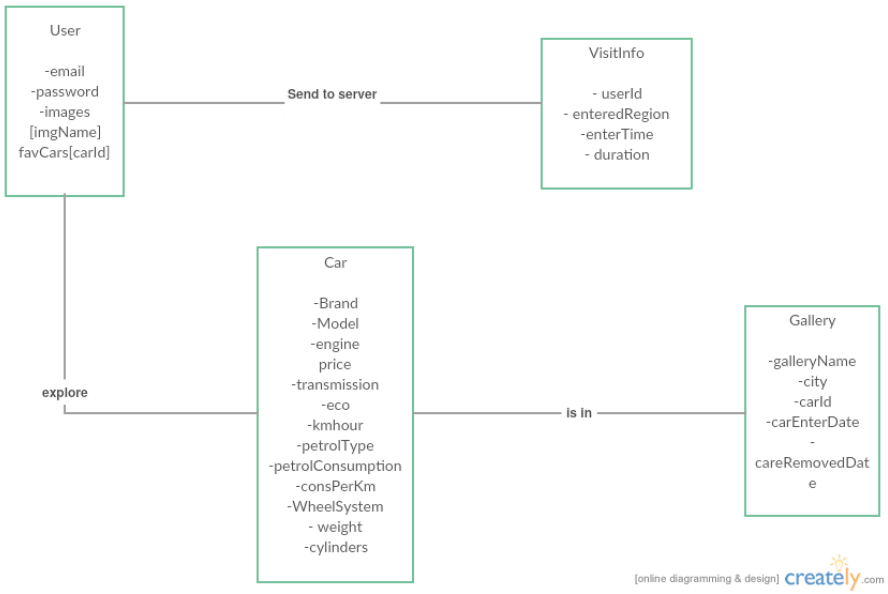
picture should come from (it is set to camera), the format of the picture (JPEG in this case), where to save the photo, its quality, etc.

Once the object is created, the $cordovaCamera.getPicture() is called which will take the ‘options’ object as a parameter and will switch on to camera. Once, the picture is taken, it will switch back to where it left.

Then the picture’s path is store to local storage (see the StorageService.add() function in the code).

7 ERD

This is the database of the app. The visitInfo and Gallery table will be used for statistics. The rest will be used for app to display car information



8 Testing

Due to time constraints and the thing that we never covered it in college, I found testing a little bit difficult. The only testing I did was the Unit Testing using Mocha. I was able to test all the functions except socket’s emit and on functions as it works and are called little bit differently. I also tested all the APIs using Postman tool. However it helped me to find the bugs in my app and fix them.

9 System Evolution

As I mentioned in the introduction of this document, I plan to add Augmented Reality to the application. When user will want to see a particular car in a different color, user

will not simply see an image. The app will access the camera (with user's permission) and will generate a virtual car with the chosen color.

10 References

Beacons with Ionic

https://dzone.com/articles/support-beacons-in-your-ionic-framework-mobile-app

https://www.thepolyglotdeveloper.com/2014/12/understanding-tabs-ionic-framework/

Ionic

https://forum.ionicframework.com/t/best-way-to-add-a-ion-view-to-ion-tabs-without-having-a-tab/10469

Socket.io

http://jbavari.github.io/blog/2014/06/17/building-a-chat-client-with-ionic/

https://devdactic.com/images-videos-fullscreen-ionic/

Cordova

https://www.thepolyglotdeveloper.com/2014/09/use-android-ios-camera-ionic-framework/

Node.js

https://www.sitepoint.com/creating-restful-apis-express-4/

Interactive Cars

Technical Report

OctavianMorcov x13101242
11/17/2016

Requirements Specification (RS)

Table of Contents

Introduction

10.1 Purpose

The purpose of this document is to set out the requirements for the development of “Car Gallery” app.

It will explain system constraints, tools and who the users are.

10.2 Project Scope

The scope of the project is to develop an Android beacon app that will help car gallery visitors to get the most from attendance. A beacon will be attached to each car in a gallery and the app will scan beacons. The app will detect in to which car the user is closer and will display detailed information about that car.

Users will also be able to call assistance by clicking a button. The staff will have a Java desktop application that will receive assistance requests and will notify the staff. Gallery staff will be able to see to which car they should go.

The software needs both Internet and Bluetooth. The Bluetooth is used to scan for beacons. With Internet, assistance requests will be displayed on staff’s desktop computers in real time by using Socket.

The desktop application will be developed in Java.

All the data will stored on MySQL or MongoDB.

OPTIONAL: Augmented Reality will be integrated in the android app to display different colours of a car.

10.3 Definitions, Acronyms, and Abbreviations

GA Gallery App (The app)

CA Car Gallery (The building in which the app is used)

UI User Interface

10.4 Technologies

The application will be built on Android platform and it will use web sockets to and receive real messages when user will want to get assistance.

The app will also be scanning for Beacons and will execute some actions depending on a specific beacon.

The server will be Java based.

For storing data I will be using MongoDB. After doing a small research I realized that MongoDB (NoSql database) will be a good choice even if it doesn't have much support for Android. However, MySQL will be the other option in case if it will not be possible for my app to use MongoDB by plan.

11 User Requirements Definition

This section describes the set of objectives and requirements for the system from the customer's perspective. What are the clients saying they want?

The app should automatically change content/page when approaching a car. On the page should be displayed all the information available about a car.

The app shall have the option to display a list of all cars even if user doesn't approach any.

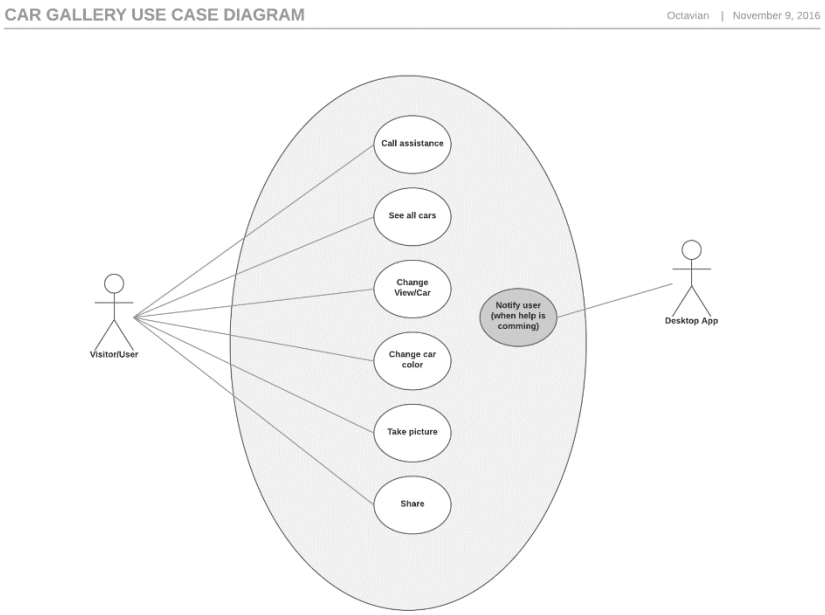
User shall be able to click a button to get assistance. User shall also be able to cancel it.

Users shall be able to see car themselves in different colors, but no only the color names.

12 Requirements Specification

12.1 Functional requirements

12.1.1 Use Case Diagram



12.1.2 Requirement 1: See all cars

12.1.2.1 Description & Priority

The user wants to see all the cars in the showroom. It is very important that users.

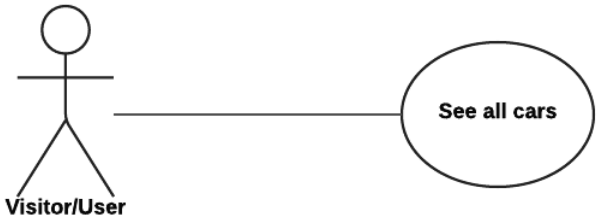
The importance of this requirement is high because users need to see what cars are in the showroom.

12.1.2.2 Use Case

Scope

The scope of this use case is to demonstrate how users can display on the screen all the cars available in the car gallery

Use Case Diagram



Flow Description

Precondition

Before this use case can be initiated, user has to be connected to the internet. It is not necessary to have Bluetooth for this use case.

Activation

This use case starts when the user clicks on “All cars” button from the menu.

Main flow

3. The User clicks on collapsible menu button from navigation bar.
4. A list of menu options drops down.
5. The user clicks on the “All cars” button.

Termination

The system displays all cars in a grid.

Post condition

The system goes into a wait state

12.1.3 Requirement 2: Call assistance

12.1.3.1 Description & Priority

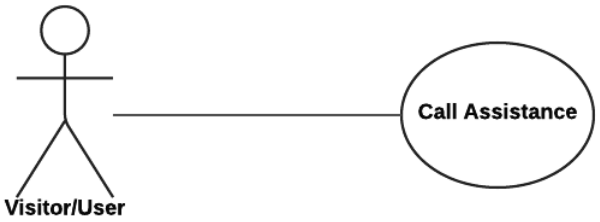
The user needs more information that is not in the app and calls for assistance.

12.1.3.2 Use Case

Scope

The scope of this use case is to send a message to the desktop application to let staff know that a user needs assistance.

Use Case Diagram



Flow Description

Precondition

The system shall have access to internet.

Activation

This use case starts when user clicks on “Assistance” button. User must be in range of at least a beacon. This will make sure that when user is far away from the CA, it will not be possible to ask for assistance.

Main flow

5. The system sends the request to the server using websocket.
6. The server updates desktop application interface displaying a text that a user needs assistance.
7. The Staff clicks on the request to send back a message that assistance is coming.

Termination

The system he system receives a notification from the server.

Post condition

The system goes into a wait state

12.1.4 Requirement 3: View other colours of car

12.1.4.1 Description & Priority

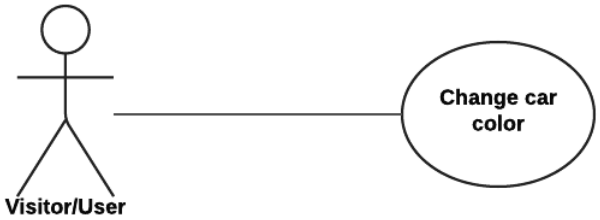
User wants to see a car in different colours. The user will not simply get the all the colours available but will see car itself in the chosen colour.

12.1.4.2 Use Case

Scope

The scope of this use case is to display a car in a different colour than it is by default

Use Case Diagram



Flow Description

Precondition

The system shall have access to internet.

Activation

This use case starts when user clicks on “More colours...” button.

Main flow

6. The user clicks on the “More colours…” button.
7. A small list shows up with all the colours available.
8. User clicks on a colour.
9. The System changes the car image with another image of the same car but different colour.

Post condition

The system goes into a wait state

12.1.5 Requirement 4: Take photo

12.1.5.1 Description & Priority

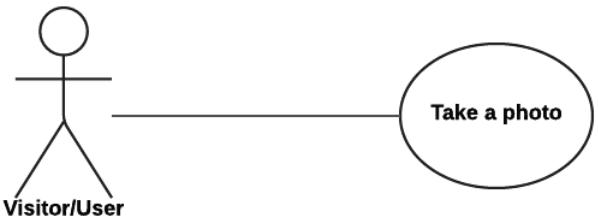
User takes a photo of the car and the image is save in user's own gallery.

12.1.5.2 Use Case

Scope

The scope of this use case is to save the image of a car to the gallery.

Use Case Diagram



Flow Description

Precondition

The system shall have access to internet.

Activation

This use case starts when user clicks on "Capture" button.

Main flow

6. The user clicks on the "Capture" button.
7. The system opens the camera.
8. User takes a photo of the car.
9. The system adds the picture to the list.
10. The system waits for other action. After taking a picture, the camera will not close.
11. User clicks on "Back" button.
12. The System changes to the Gallery view and displays all pictures.

Post condition

The system goes into a wait state

12.1.6 Requirement 6: See all pictures

12.1.6.1 Description & Priority

User opens the gallery of pictures that were taken.

12.1.6.2 Use Case

Scope

The scope of this use case is to access the photos that were taken at the gallery.

Use Case Diagram

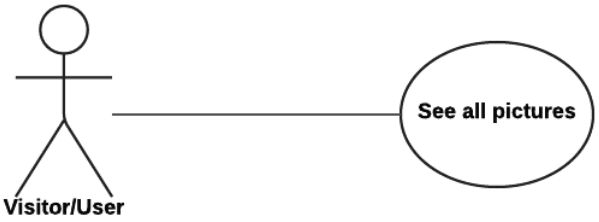


Diagram should highlight actors and uses cases……..

Flow Description

Precondition

The system shall have access to internet.

Main flow

3. User clicks on menu button.
4. System display a dropdown list with all menu options.
5. User clicks on “Gallery”.
6. The system displays the gallery that contains all the pictures.

Post condition

The system goes into a wait state

12.2 Non-Functional Requirements

12.2.1 Availability requirement

The system shall work 24 hours, only in 2 different ways. When the app is used in the Gallery, the car content are changed depending in which beacon region user is located. When the app is used away from gallery, cars content are changed by swiping. Also “call assistance” button will not function if there is no beacon around. No beacon around means that the app is not used within the gallery.

12.2.2 Security requirement

12.2.3 Reliability requirement

To guarantee the reliability of the application, a device should have at least internet connection. Also, to properly work, a device should have Bluetooth turned on. However, the app can run without Bluetooth, but the user will not get the maximum of the app.

The system will not behave properly if a beacon will not function or its battery will be down.

The application shouldn’t crash if there will be no internet connection. However, if there will be no internet connection, only partial content will be displayed.

12.2.4 Maintainability requirement

All beacons batteries shall be replaced every 4 months and shall be checked every day. The checking consist in using the app to see if app can detect all the beacons.

Any error should be fixable by only one developer.

12.2.5 Portability requirement

User shall be able to use this app only from Android OS. After deploying first version, the app will be transported to iOS.

Migrating the desktop application (java application designed for staff) from Windows to iOS shouldn’t take more than 1 day and shouldn’t require more than 1% of changes.

Porting the app from android to iOS shouldn’t take more than 2 weeks. The time also includes testing.

13 Interface requirements

As my project will consist of two apps, each app will have its own UI.

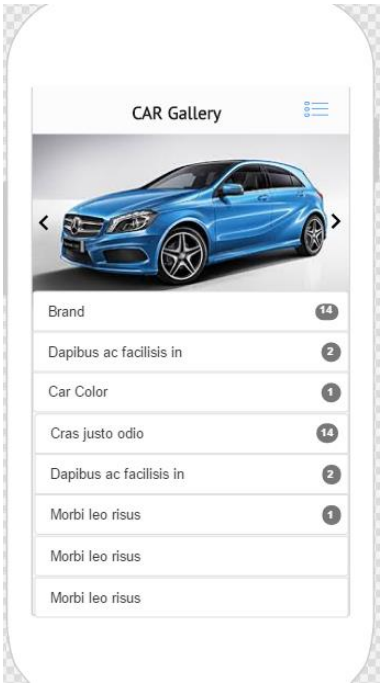
The interface of the Android app will be of Activity views. Views will have many fragments on them that will change depending on user's click events. A prototype will be provided next month. The android UI will be able to update desktop app when an assistance request is sent. Desktop app will have an icon that will change to different color when there will be incoming requests.

The UI for the desktop app will be a simple Java interface built using JFX. The interface will have only one view.

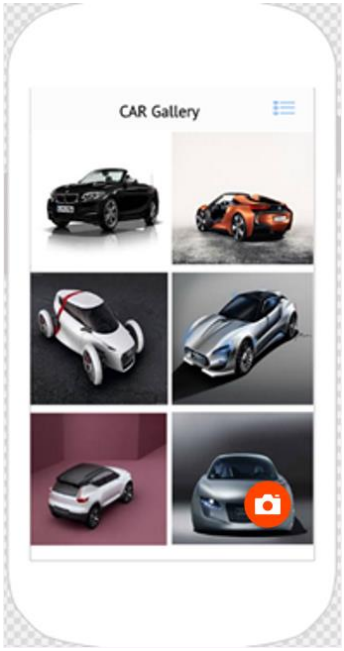
13.1 GUI

This is the first view that user will see when running an app. The start button will bring user to the Car gallery page. When user clicks on the collapsable menu in top right corner, a list will drop down with the following options: Login, Go to Your photos, search car, all cars (Note: there is going to be another button that will display only the cars in a showroom depending where user is accessing the app).

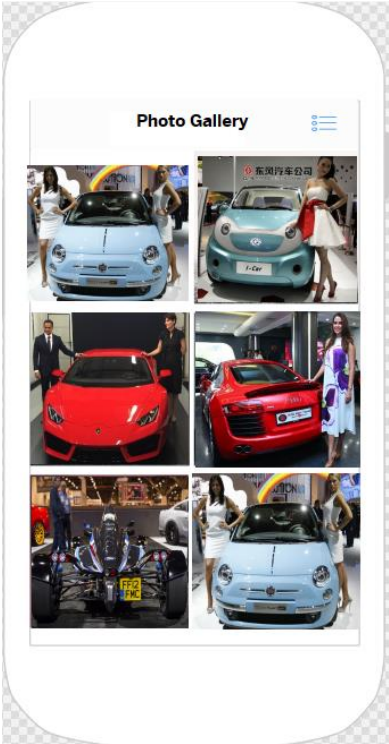
The user clicked on start and a page appeared of a car. The app chose which page to display based on the closest beacon. User can change pages manually. On this page user will change car's colors and will find all the information about a car.



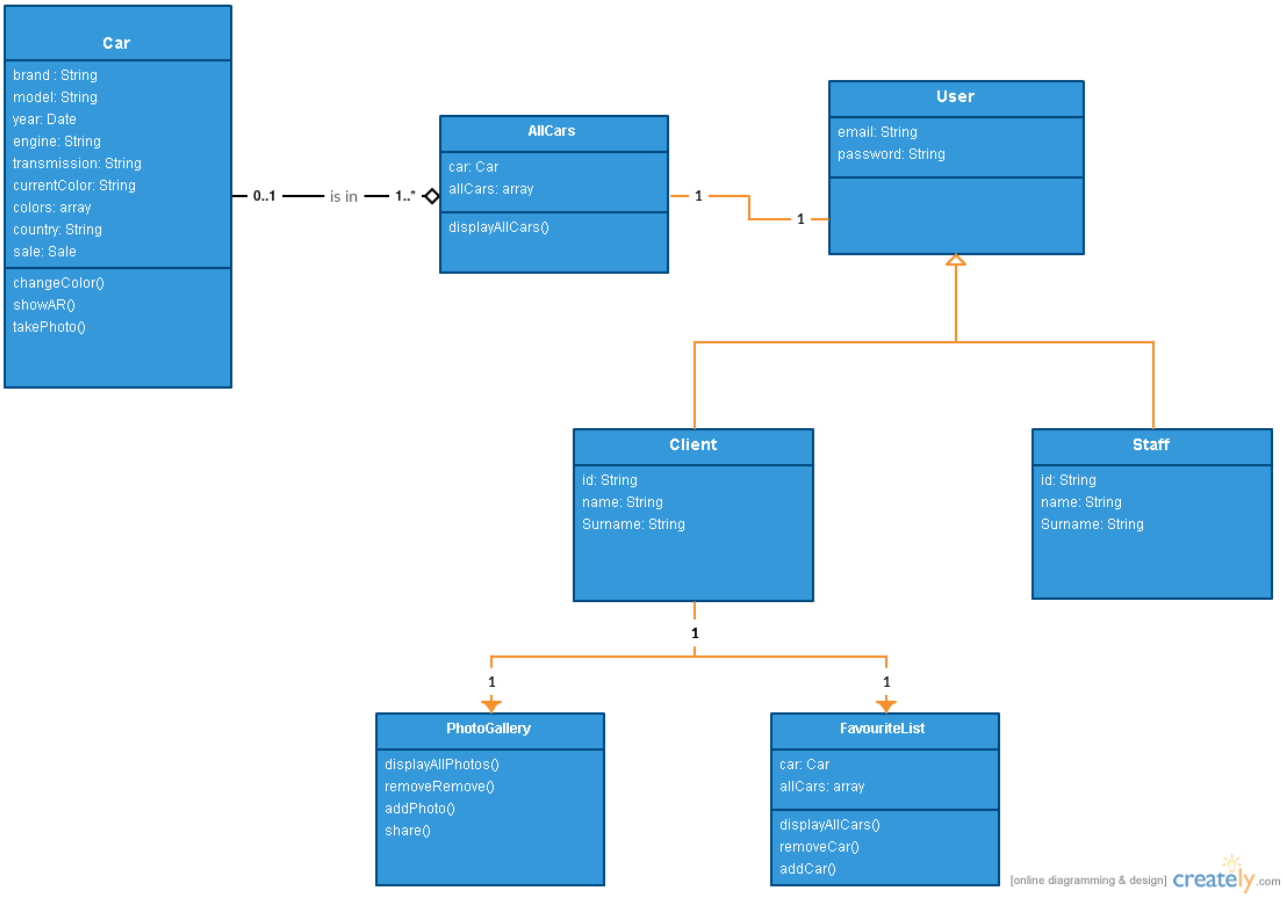
This page displays all the cars available in the showroom. User can access this page by clicking on the collapsable menu and choose "All cars around"



This page will display all the photos taken by the user. The photo will also have information about where the photos taken, with which car, and a link to the car information.

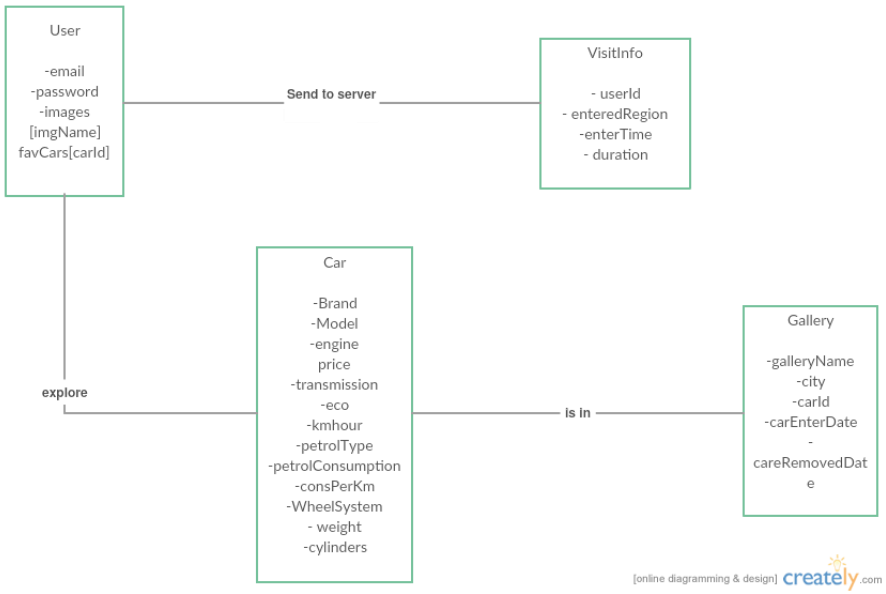


14 System Architecture



15 ERD

This is the database of the app. The visitInfo and Gallery table will be used for statistics. The rest will be used for app to display car information



16 System Evolution

As I mentioned in the introduction of this document, I plan to add Augmented Reality to the application. When user will want to see a particular car in a different color, user will not simply see an image. The app will access the camera (with user's permission) and will generate a virtual car with the chosen color.