Slovak University of Technology
Faculty of informatics and information Technology
Ilkovičova 2, 842 16 Bratislava 4

Mobile Technologies

Android application

Auto bazaar

Sultan Numyalai, Nikola Karakas

Instructor: **Ing. Marek Galinsky**
Study Field: INFO3
Year: 3. Bc
Academic Year: 2018/2019

Documentation Author: Sultan Numyalai

Semester: Summer

Contents

Project Description

Car-Dealer Android Application is the perfect solution for listing your Car Inventory. It Engages your client with sharp design and features like Favorite, Push notification to keep your customer engagement to you.

Technologies used in the Project.
Front-End : Java
Back-end : NoSQL Firebase and Java.
Environment: Android Studio

Platform: Android
Time Approximation : 3 Months

Functionalities of the Project

Inventory Listing

Listing of your Car's Inventory most appealing way.

Favourite

Easy book mark your Favourite Car's

Live Search

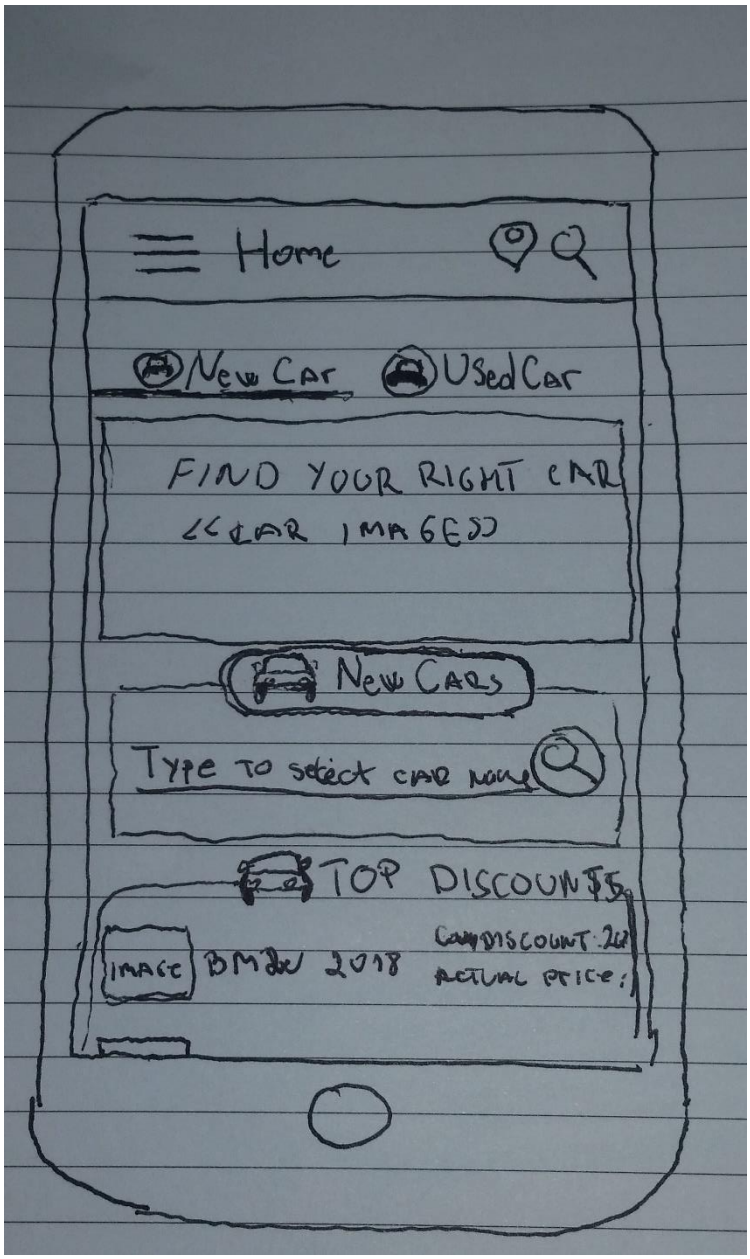
Easy Searching with live search feature with Car Name.

Login

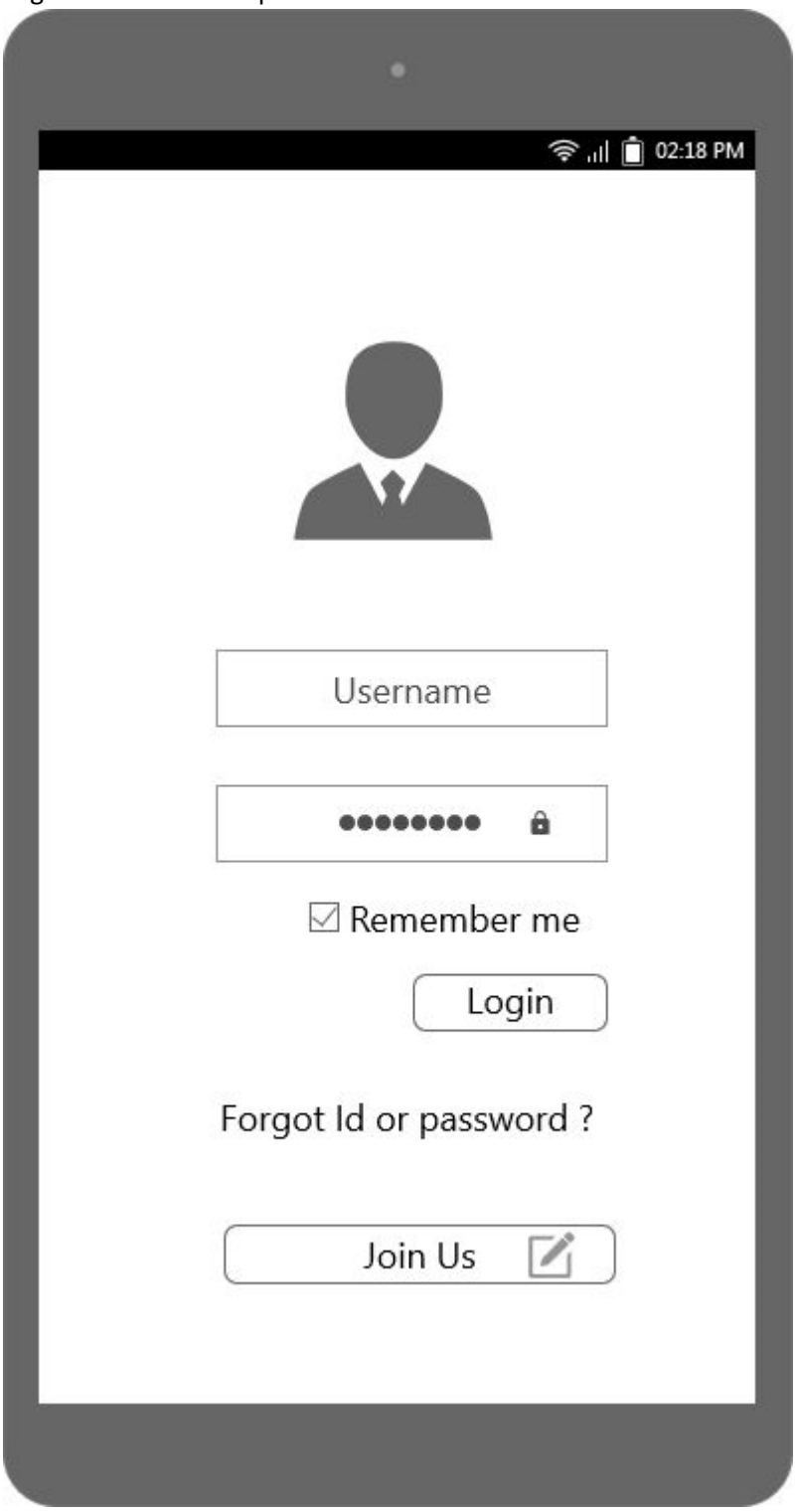
Users can login and check their Favorite lis

1. Assignment
2. Low-fidelity prototype

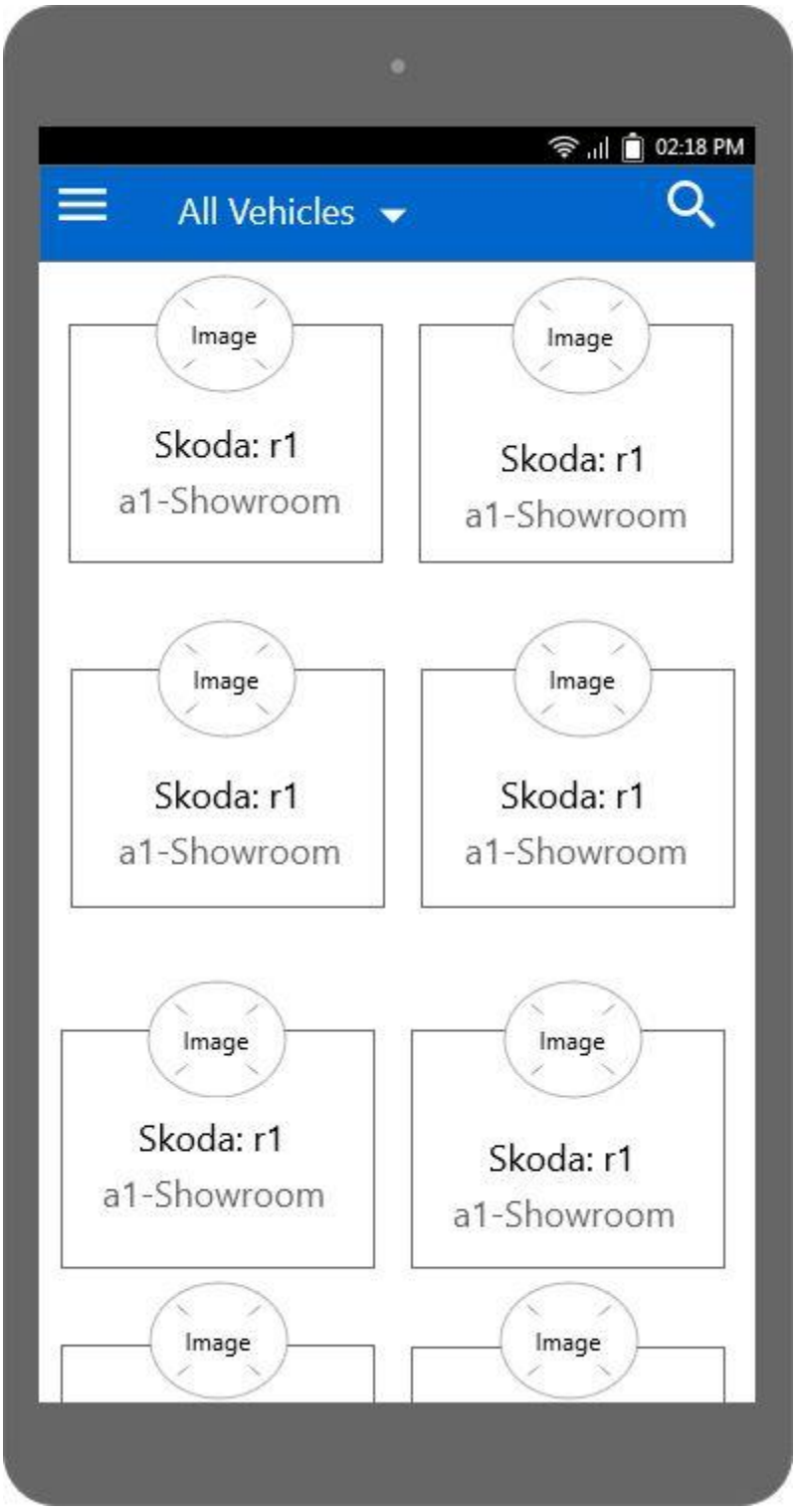
Home Screen



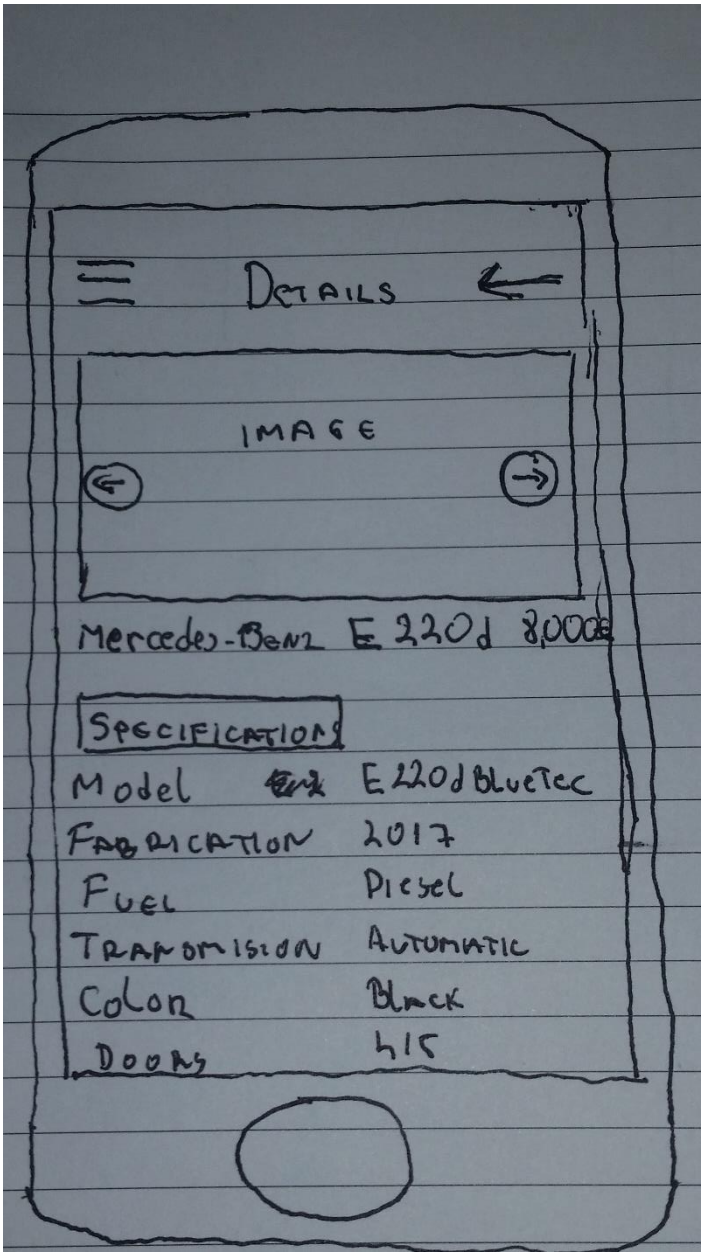
Login screen for the aplication



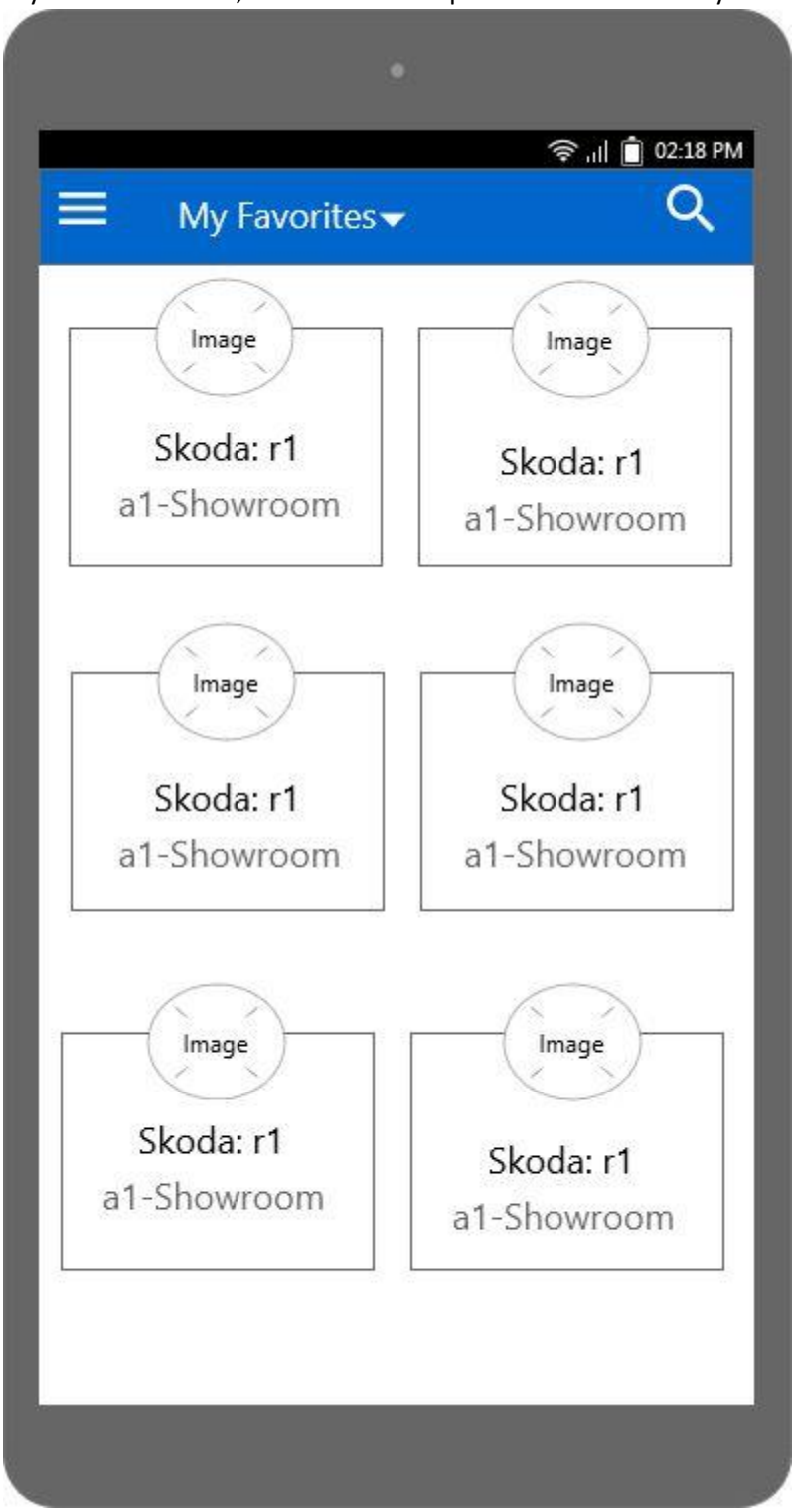
Filter of all vehicles screen



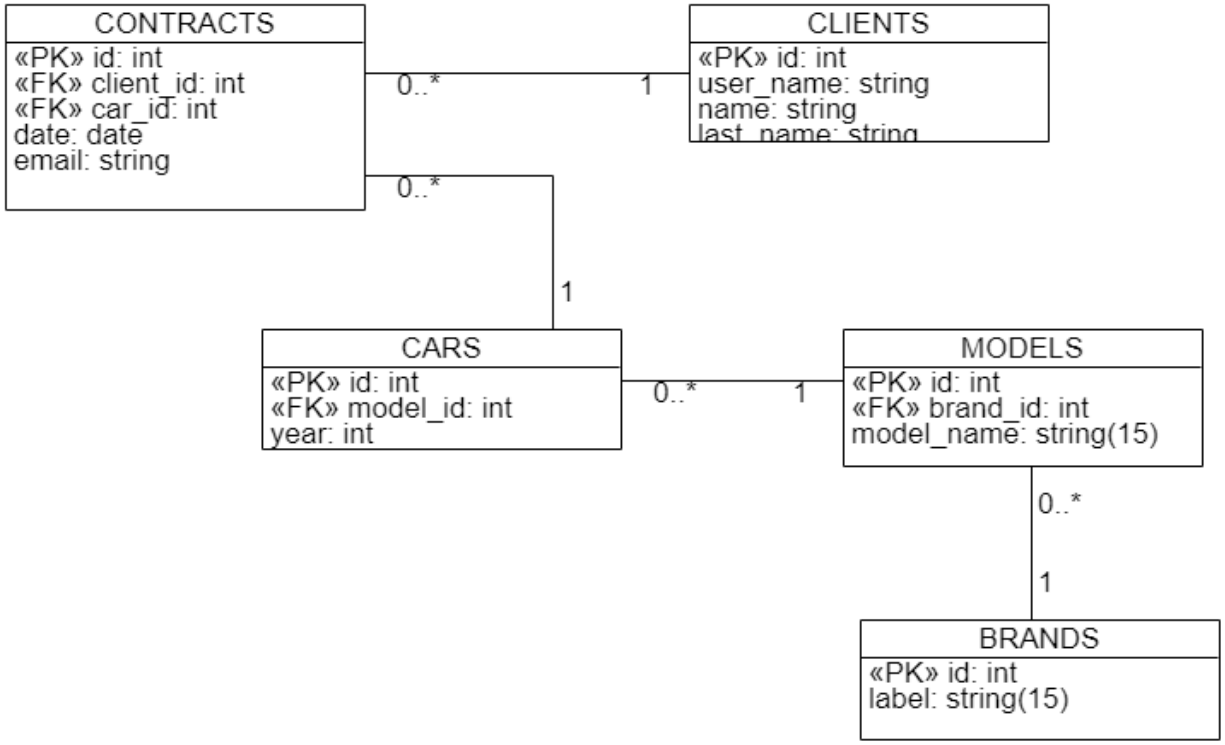
Vehicle Detail screen



My favorites screen, consist of all the products marked as my favorite browsing



3. Data-Model


CONTRACTS
«PK» id: int
«FK» client_id: int
«FK» car_id: int
date: date
email: string
0..*
1
CLIENTS
«PK» id: int
user_name: string
name: string
last_name: string
0..*
1
CARS
«PK» id: int
«FK» model_id: int
year: int
0..*
1
MODELS
«PK» id: int
«FK» brand_id: int
model_name: string(15)
0..*
1
BRANDS
«PK» id: int
label: string(15)

4. API CALLS:

API calls for "Auto Bazaar"

1: Adding a new Vehicle POST request : Creating a new instance of the Vehicle and adding it to the showroom.

URL: HTTP POST http://localhost /vehicle/1

2: Viewing vehicles GET request

URL: HTTP GET http://localhost/vehicle/123

3: Updating vehicles PUT And PATCH Request

UR:: HTTP PUT http://localhost/vehicles/123

4: Deleting a vehicle DELETE

URL: HTTP DELETE http://localhost/vehicle/{id}

5: Login Authentication POST

URL: http://localhost/login/?postlogin

6: Detail of a vehicle GET

URL: GET http://localhost/vehicle/vehicleDetails/{id}

7: Contact Dealer Send Email POST

URL: http://localhost/contact/?postlogin/form

8: My Favorites POST Request

URL: http://localhost/?add-to-favorites=25

POST request and Status Code

Status Code	Description
200 (OK)	This is the standard response for successful HTTP requests.

201 Created Typically a response to a POST request. The request has been completed, and a new resource has been created.

204 (No Content) The server successfully processed the request, but is not returning any content.

GET request and Status Code

200 OK if the resource is found on the server along with response body which is usually either XML or JSON content

404 (NOT FOUND) In case resource is NOT found on server

400 (BAD REQUEST) if it is determined that GET request itself is not correctly formed

PUT request and status code

200 OK if an existing resource is modified

204 (No Content) response codes SHOULD be sent to indicate successful completion of the request but no content

DELETE request and status code

A successful response of DELETE requests SHOULD be HTTP response code 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has been queued, or 204 (No Content) if the action has been performed but the response does not include an entity.

JSON format examples for HTTP requests:

Creating a vehicle and updating will return the following

```
{
  id: 101,
  name: ‘avensis’,
  category: ‘toyota’,
  model : ‘ac2’,
  description: 'this is a good car',
  AdminId: 1
```

Acceptance Tests

Functions that the software is going to perform

Input data based on function's specifications

Determine output based on functions specifications

Execute test case

Compare actual and expected outputs.

Car-Dealer:

1. Test Name: Search

Goals: Retrieve information about specific car, where infos about same car are previously given in fields for that purpose. Results have to be relevant based on every filter previously selected.

Precondition:

a) Functional front-end & back-end
b) Vehicles must exist in database
c) Vehicles must have different specifications so they could be filtered separately of each other

Steps:

a) Click on filtered search
b) Select filters for the desired fields
c) Click on Search button
d) Show cars

Test Input: Filter by Volkswagen

Expected Output: Vehicles relevant to Volkswagen brand would show on the screen

Result : Pass / Fail

2. Test Name: Show detailed information of vehicle

When user select one of the shown vehicles from available vehicle list, new screen should display detailed information about selected vehicle, with pictures on the top of the screen.

Precondition:

a) Functional front-end & back-end
b) Vehicle must exist in database
c) Vehicle is previously displayed by the search filters

Steps:

a) Click on the desired vehicle from the list
b) Have insight in detailed informations
c) Have insight in pictures

Test Input: Selected Vehicle BMW 5,2015 35000e

Expected output: Navigating to detailed screen of BMW 5, 2015 35000e

Result : Pass / Fail

3. Test Name: Watchlist

Watchlist is actually an alternative for cart in the ecommerce applications but in our applications it is actually showing vehicles based on the favor of the user and the user can add a browsed vehicle to the watchlist and it can be then shown on the watchlist based on the concrete user.

Functionality:

Add/Remove vehicle to/From the Watchlist.

Show Watchlist

Precondition:

a) Functional front-end & back-end
b) User has been signed in

Steps:

1: Add vehicle(s) to the watchlist for better comparison later
2: Click on the watchlist to show the vehicles which are already added.
3: Open watchlist screen and show all vehicles added before

Input: Add BMW 5 to the watchlist

Expected Output: added vehicle in Watchlist

Result : Pass / Fail

4. Test Name: Login

Goals: After entering of correct information on the login screen, user should get full access to its own account.

Precondition:

a) Functional front-end & back-end
b) User has to be already registered
c) User exists in user's database

Steps:

a) User enter correct user name
b) User enter correct password
c) Application display message about successfully log in
d) Application navigate to home screen

Input:

Username: sultan96

Password: Sultan123

Expected Output: Message about successful log in

Result: Pass / Fail

5. Test Name: Homepage

Homepage is the basic page of the application which is opened just after opening the application

Functionality:

Can user navigate to the right page from homepage ?

Can the application load the contents ?

Preconditions:

a) Functional front-end & back-end

Steps:

a) Open application
b) Show homepage
c) Auto scroll
d) Navigate to any page from homepage

Input:

Expected Output:

Result : Pass / Fail

Back-End Tests

Back-End acceptance and functional tests for Car Dealer Android Application:

Request type mapping:

GET	Read or retrieve data
POST	Add new data
PUT	Update data that already exists
DELETE	Remove data

- Sending a GET request to /vehicle would retrieve vehicle list from the database.
- Sending a GET request to /vehicle/{Id} would retrieve vehicle with a specified ID from the database.
- Sending a POST request to /vehicle/{Id} would add a new vehicle to the database.
- Sending a PUT request to /vehicle/{Id} would update the attributes of an existing Vehicle, identified by a specified id.
- Sending a DELETE request to /Vehicle/{Id} would delete a specified vehicle from the database.

Method	Endpoint	Resource	Parameters
GET	http://server.com	/vehicle/findbyBrand	?brand= Toyota
GET	http://server.com	/vehicle	

1) GET Request for Vehicle List:
   Author: Sultan Numyalai

Message

[Step 1] GET VehicleList passed Response

HOST : http://127.0.0.0

GET /vehicle

Connection: keep-alive

Keep-alive: 300

Content-Type : application/json

Accept : application/json

HTTP/1.x 200 OK

Content-Type: application/json

Content-Length : x

Status code: 200

Transfer-Encoding: chunked

Server: xxxx

Connection: close

Data : JsonObject`

Response HTTP Method GET

{ “data”: [{

“Id” : “1”

“name” : “abc”

“Brand” : “Toyota”

“Price” : “12222.0”

},

“Id” : “2”

“name” : “abcfsdfdss”

“Brand” : “Toyota”

“Price” : “12222.0”

},

“Id” : “3”

“name” : “abdfsdfsdfsdfc”

“Brand” : “Toyota”

“Price” : “12222.0”

}, }]

Bad Request

HTTP/1.1 400 Bad Request

Content-Type: application/json


```
{
  "errors": [
    {
      "source": { "parameter": "include" },
      "title":  "Invalid Query Parameter",
      "detail": "parameter x required dadasd."
    }
  ]
}
```


2) Creating a Product
   Author: Sultan Numyalai

URL /vehicle HTTP Method POST

REQUEST

“data”: [{

"id": 3,

"name": "ASDasDasdas",

"Price": 500000

“Brand”: “daas”

},

“brand” :{

“id”: 1

“name”: daas

}]

HOST : http://127.0.0.0

POST /vehicle/create

Connection: keep-alive

Keep-alive: 300

Content-Type : application/json

Accept : application/json

Response

HTTP/1.1 201 Created

Content-Type: application/json

Successful Product Creation JSON

{

“data”: [{

"id": 3,

"name": "ASDasDasdas",

"Price": 500000

“Brand”: “daas”

},

“brand” :{

“id”: 1

“name”: daas

}]

Failure

HTTP/1.1 422 Unprocesssable Entity

Content-Type: application/json

{

"errors":

[{ "source": {

"pointer": ""

},

"detail": "Missing `data` Member at document's top level." }

] }

3) GET Request for Vehicle by ID
   Author: Sultan Numyalai

Message

[Step 1] GET Vehicle/{id}passed Response

HOST : http://127.0.0.0

GET /vehicle/{1}

Connection: keep-alive

Keep-alive: 300

Content-Type : application/json

Accept : application/json

HTTP/1.x 200 OK

Content-Type: application/json

Content-Length : x

Status code: 200

Server: xxxx

Connection: close

Data : JsonObject`

Response HTTP Method GET

{

“Id” : “1”

“name” : “abc”

“Brand” : “Toyota”

“Price” : “12222.0”

}

4) PUT Request for Vehicle update by ID
Author: Nikola Karakas

Step 1] GET Vehicle/{id}passed

HOST : http://127.0.0.0

GET /vehicle/{1}

Connection: keep-alive

Keep-alive: 300

Content-Type : application/json

Accept : application/json

Response

HTTP/1.x 200 OK

Content-Type: application/json

Content-Length : x

Status code: 200

Server: xxxx

Connection: close

Data : JsonObject`

Response HTTP Method GET

{

“Id” : “1”

“name” : “abc”

“Brand” : “Toyota”

“Price” : “12222.0”

}

Step 2] PUT attribute/{}passed

Response

HOST : http://127.0.0.0

PUT /vehicle/{1}

“name” : “BCA”

Connection: keep-alive

Keep-alive: 300

Content-Type : application/json

Accept : application/json

HTTP/1.x 200 OK

Content-Type: application/json

Content-Length : x

Status code: 200

Server: xxxx

Connection: close

Data : JsonObject`

5) DELETE Request for Vehicle delete by ID
Author: Nikola Karakas

Step 1] GET Vehicle/{id}passed

Response

HOST : http://127.0.0.0

GET /vehicle/{1}

Connection: keep-alive

Keep-alive: 300

Content-Type : application/json

Accept : application/json

HTTP/1.x 200 OK

Content-Type: application/json

Content-Length : x

Status code: 200

Server: xxxx

Connection: close

Data : JsonObject`

Response HTTP Method GET

{

“Id” : “1”

“name” : “abc”

“Brand” : “Toyota”

“Price” : “12222.0”

}

Step 2] DELETE Vehicle/{id}passed

HOST : http://127.0.0.0

DELETE /vehicle/{1}

Connection: keep-alive

Keep-alive: 300

Content-Type : application/json

Accept : application/json

Response

HTTP/1.x 200 OK

Content-Type: application/json

Content-Length : x

Status code: 200

Server: xxxx

Connection: close

Data : JsonObject`