Version 1.0

**Nov 10, 2022**

CAR SELLING WEBSITE

SYSTEM

REQUIREMENT

**SPECIFICATION**

Table of contents

[1. Overview 2](#__RefHeading___Toc563_4161806273)

[1.1 Purpose of this document 2](#__RefHeading___Toc565_4161806273)

[1.2 Project Scope 2](#__RefHeading___Toc945_825886457)

[2. Functional Requirements 3](#__RefHeading___Toc567_4161806273)

[2.1 Get All Products 3](#__RefHeading___Toc569_4161806273)

[2.2 Search cars 3](#__RefHeading___Toc571_4161806273)

[2.3 Search filters 3](#__RefHeading___Toc715_4161806273)

[2.4 Filtering cars by brand 4](#__RefHeading___Toc717_4161806273)

[2.5 Register sell car 4](#__RefHeading___Toc573_4161806273)

[2.6 Car details page 5](#__RefHeading___Toc577_4161806273)

[2.7 Car order page 5](#__RefHeading___Toc579_4161806273)

[2.8 My orders 5](#__RefHeading___Toc719_4161806273)

[3. Nonfunctional Requirements 6](#__RefHeading___Toc947_825886457)

[3.1 Performance 6](#__RefHeading___Toc949_825886457)

[3.2 Maintainability 6](#__RefHeading___Toc951_825886457)

[3.3 Security 6](#__RefHeading___Toc953_825886457)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Document Version History** | | | | | |
| **Version** | **Effective Date** | **Author** | **Details** | **Reviewer** | **Approvers** |
| 0.1 | Jul 08 2022 | Dao Minh Hieu | Created | All members | Le Cao |
| 1.0 | Nov 10 2022 | Tung Nguyen | Add all system requirements | All members | Le Cao |

Confidentiality

This document is distributed on a restricted basis, is commercial in confidence to the recipient, and may not be used for any purpose other than that associated with a NashTech project. The contents of this document may not be disclosed to any third parties without the expressed advance written authorisation of NashTech.

# Overview

## Purpose of this document

This document provides the functional requirements covered in the project “Car Selling Website”. All the features will include a feature name, a short description and React and Next.js techniques that the developer are applying to their works.

## Project Scope

Build a car selling website that allows users to do the following once it is launched

* Search for a car by brand, model, year, name, or category
* View details information of a car
* Register to sell a car
* Order a car
* View order list

Deployment is excluded.

# Functional Requirements

Describe some main functional requirements and how the solution has covered each of the items.

## Get All Products

### Description

* As a client / user when they go to the home page then they can see all cars existing in the system.
* Cars are displayed as grid. Every item should contain
  + A picture
  + Name
  + Price
  + Country
  + Year
* Order by registered date, descending.
* Pagination

### Techniques

* For SEO optimization and a minimal load time, we use **Static Generator**.
* The products will be loaded at build time via the **getStaticProps** function. Following that, the product list will be supplied as a prop to the Product component and saved as a state of that component.
* That state will be used by the component to render them on the browser.

## Search cars

### Description

* On the hone page, users can search for cars by typing keywords into the search box.
* The system allows searching by name, brand, model, and year.

### Techniques

* Create two components named SearchBox and CarList
* In the index page, use “Static Generator” to pre-render the list of cars at build time, which is good for SEO.
* The index page contains the SearchBox and the CarList components. Whenever the user triggers a search, the index page calls the API to get the cars then passes to the CarList to render.
* Use the State hook to store the current cars.

## Search filters

### Description

* Allow users to narrow down search results.
* Whenever the user clicks on the Filter button, it will show a dialog that allow the user to specify
  + Category: a drop-down list that contains all categories in the system
  + Price range
  + Order by: a radio group with the following options
    - Newest
    - Lowest price
* There are the “reset” and “apply” buttons at the bottom of the dialog.
* Users click the “reset” button to reset the criteria
* Users click the “apply” button to apply the filter

### Techniques

* Taking advantage of Material Dialog component to create base Modal component.
* Create the FilterSearch component that contains all the filter criteria
* Use the State hook to store the filter criteria.
* Embed the FilterSearch component into the SearchBox component and let the SearchBox component handles the search action.

## Filtering cars by brand

### Description

* In addition to the standard filter, users are able to narrow the scope of search result to a specific brand.
* The brand filter will be presented in form of a carousel with list of brands per page. The carousel is allowed to swipe left or right to reveal other pages. Whenever users pick up a brand, they will be navigating to a sub-page for that brand with list of belonging cars.

### Techniques

* Taking advantage of **Static Generator** for SEO optimization, the brand selector and sub-page for each brand will be generated at build time.
* For brand selector component, list of brands will be fetched at build time to render all pages of carousel. A third-party library [Swiper](https://github.com/nolimits4web/swiper) for Reactjs is used to render and bootstrap carousel at client-side.
* For sub-page per brand, dynamic routes is applied to handle all brands, or pages in equivalent, then prerendered all the pages at build time. Brands will also be retrieved at build time to identify all possible value. The default search result, which is list of cars belongs to a specific brand will be processed at client-side, using common Reactjs features such as Component, Hooks(useState, useEffect).

## Register sell car

### Description

* Users want to sell their cars online and they need a place to post information to sell them.
* To advertise cars for sale, user must be signed in and go to sell car page by pressing sell car button. On the sell car page, the user must fill out all of the information required to sell the car and submit the form. After that, the selling car will appear on the homepage where the buyers can find them.

### Techniques

* Use functional components to create question field components (TextField, AutoComplete, DatePicker…) each field are required value, if the input is invalid, the error will be displayed. These components are built with “@mui/material” library.
* The form data after submission will send to API by using fetch API.
* Use useSession() to check user is signed in after reaching the register sell car page.

## Car details page

### Description

* When clicking on a car on the Home or the Brand page, it should redirect to the Car Details page.
* This page display all information of the car.

### Techniques

* Use useRouter to get the car ID in the URL.
* Using State Hook, Effect Hook to query the car detail.

## Car order page

### Description

* When clicking on the “Order” button in the Car Details page, it should redirect to the Order page.
* After filling the order form, the order is created.

### Techniques

* In order to go to this page, the user must be signed in. In order to check this, useSession() is used. It returns the status of authentication. If the status is unauthenticated, it redirects to the login page.
* Use useRouter to get the car ID in the URL.
* Using State Hook, Effect Hook to query the car detail.

## My orders

### Description

* A user can view his order list
* Every item should contain the order ID, car name, price, and order date

### Techniques

* This is a user-specific page and SEO optimization is not needed. Therefore, we just use client fetching.
* Create a function component named OrderList
* Use the State hook to store the order list

# Nonfunctional Requirements

## Performance

* Each page must load within 2 seconds.

## Maintainability

* Use a micro-services architecture
* Be able to develop and deploy in a Azure DevOps environment

## Security

* Only authenticated users can sell or buy a car