

02. Juli 2022

CAR UTILITY MANAGEMENT

TECHNISCHE DOKUMENTATION

Elnaz Gharoon, Alexander Hagl, Jasmin Hübner, Anna Weber

Inhaltsverzeichnis

Projekt	1
Struktur der Website	1
Verknüpfungen	2
Technischer Aufbau	2
Fahrzeugübersicht	2
Buchungsseite	3
Kriterien	4
Selbstgeschriebenes HTML	4
Fehlerfreies HTML	5
Selbstgeschriebenes CSS	5
Responsive Design	5
Modernes JavaScript	5
Selbstgeschriebene Frontend-Logik	6
Selbstgeschriebene Backend-Logik	6
Backend-Aufteilung nach Model-2-Entwurfsmuster	6

Projekt

Im Rahmen dieses Projektes, wurde eine Car-Rental und Sharing Webseite entwickelt, auf der man Fahrzeuge von verschiedenen Anbietern, einfach und zentral, jederzeit buchen kann. Zentrales Element ist hier die Fahrzeugübersicht zusammen mit der Fahrzeugspezifischen Buchungsseite, womit Kunden ein Wunschfahrzeug suchen und dieses anschließend buchen können.
Der Sourcecode ist in der mitgelieferten zip oder auf [GitHub](#) zu finden.

Struktur der Website

- Projektstartseite mit Projektinformationen und kleiner User-Doku
- Startseite Car Utility Management (<http://c-u-management.de>)
 - Header:
 - * Logo, Banner
 - * Navigationsleiste (Home, Fahrzeugübersicht, Bewertung, Kontakt, Profil/Login)
 - Werbevideo (<https://youtu.be/FhyaLiANmkE>)
 - Informationen über Car Utility Management
 - User-Doku, wie der Buchungsvorgang funktioniert
 - Footer Links:
 - * Kontakt, Telefon, Instagram, Youtube
 - * Impressum, Datenschutz, AGBs
 - * Partner von CUM, Karriere bei CUM
- Fahrzeugübersicht (<http://c-u-management.de/overview>)
 - Header s.o.
 - Filter zum dynamischen filtern der Fahrzeugangebote
 - Fahrzeugliste (20 Fahrzeuge pro Seite)
Für jedes Fahrzeug: Allgemeine Informationen, Grundpreis, Bild
Button “Zur Buchung”
 - Paging buttons zum Navigieren zwischen Resultseiten
 - Footer s.o.
- Buchungsseite (<http://c-u-management.de/booking/1e278eed7b32464f8222dfd423f7c0ff>)
 - Header s.o.
 - Allgemeine Informationen über das gewählte Fahrzeug
 - Buchungseinstellungen
 - * Buchungszeitraum Datepicker
 - * Optionaler Zusatz: Voll-/Teilkasko, Refuel (nicht tanken müssen, nachdem man das Fahrzeug zurück gibt)
 - Footer s.o.

- Bewertungsseite (<http://c-u-management.de/rating>)
 - Header s.o.
 - Verteilung der Kundenbewertungen in Balkendiagramm (1-5 Sterne)
 - Kommentarfeld und Stern-Auswahl zum Bewerten (Nur wenn angemeldet und noch nicht Bewertet → Jeder User kann nur einmal Bewerten)
 - Auflistung aller Kommentare / Bewertungen
 - Paging buttons zum Navigieren zwischen Resultseiten
 - Footer s.o.
- Kontaktseite (<http://c-u-management.de/contact>)
- Login / Registerseite (<http://c-u-management.de/login>)
- Profilseite (Nur erreichbar wenn angemeldet) (<http://c-u-management.de/profile>)
- Partnerseite (<http://c-u-management.de/partner>)
- Karriereseite (<http://c-u-management.de/karriere>)
- Impressum, AGB, Datenschutzseite im Footer

Verknüpfungen

Navigationsleiste auf jeder Seite, die zur Indexseite, Fahrzeugsübersicht, Bewertungsseite und Kontaktseite führt. In der Fahrzeugübersicht sind die individuellen Buchungsseiten der einzelnen Fahrzeuge verlinkt.

Im Footer findet man Verlinkungen zu den Seiteninformationen (Impressum, AGBs, Datenschutz), sowie die Partner- und Karriereseite.

Technischer Aufbau

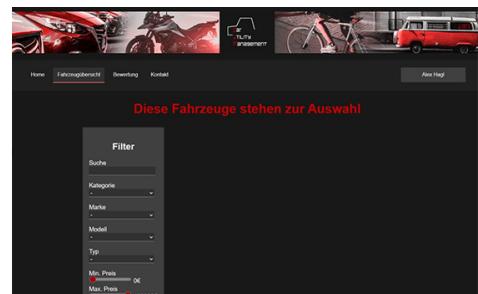
Fahrzeugübersicht

Laden des Main-Layoutes durch den Controller.

```
/// <summary>
/// Endpoint to Overview for all vehicles
/// </summary>
/// <returns></returns>
[HttpGet]
[Route(Routes.OVERVIEW)]

public async Task<IActionResult> Overview() {
    // Get all Vehicles from Database and hand over to View
    List<Vehicle> vehicles = await dbStore.GetAllVehiclesAsync();
    var modelList = vehicles.Select(m => m.Model).Distinct().OrderBy(e => e).ToList();
    var brandList = vehicles.Select(b => b.Brand).Distinct().OrderBy(e => e).ToList();

    return View(new OverviewViewModel(await PrepareBaseViewModel()));
    ModelFilterList = modelList;
    BrandFilterList = brandList
}
```



Einfügen der Fahrzeugliste mithilfe von AJAX. Erstellung der gefilterten und gepageten Liste durch Anfrage an den APIController.

```
//<summary>
// Endpoint to retrieve a filtered, 20 element page from all vehicles
//</summary>
//<param name="page">Page to get</param>
//<param name="searchTxt">Search filter by model and brand</param>
//<param name="categoryFilter">Category filter by sharing and renting</param>
//<param name="brandFilter">Filter by a specific brand</param>
//<param name="modelFilter">Filter by a specific model</param>
//<param name="typeFilter">Filter by a specific vehicle type</param>
//<param name="minPrice">Price minimum filter</param>
//<param name="maxPrice">Price maximum filter</param>
<returns>Partial Html View, that contains the maximum 20 element, filtered page. You can embed the result via Javascript</returns>
[HttpGet]
[Route("Routes.FILTERED_VEHICLES")]
public async Task<PartialViewResult> GetVehicleListView(int? page, string searchTxt, int? categoryFilter, string brandFilter, string modelFilter, int? typeFilter, float? minPrice, float? maxPrice) {
    // Get all vehicles
    Ienumerable<Vehicle> veh = await dbStore.GetAllVehiclesAsync();

    // if search filter is present -> filter
    if (!String.IsNullOrWhiteSpace(searchTxt))
        veh = veh.Where(e => e.Brand.ToLower().Contains(searchTxt)
            || e.Model.ToLower().Contains(searchTxt)
            || e.Firm.ToLower().Contains(searchTxt));

    // if filter is present -> filter
    if (categoryFilter != null)
        veh = veh.Where(e => e.Category == categoryFilter);
    if (String.IsNullOrWhiteSpace(brandFilter))
        veh = veh.Where(e => e.Brand.Equals(brandFilter));
    if (String.IsNullOrWhiteSpace(modelFilter))
        veh = veh.Where(e => e.Model.Equals(modelFilter));
    if (typeFilter != null)
        veh = veh.Where(e => e.Type == typeFilter);
    if (minPrice != null)
        veh = veh.Where(e => e.BasicPrice >= minPrice);
    if (maxPrice != null)
        veh = veh.Where(e => e.BasicPrice <= maxPrice);

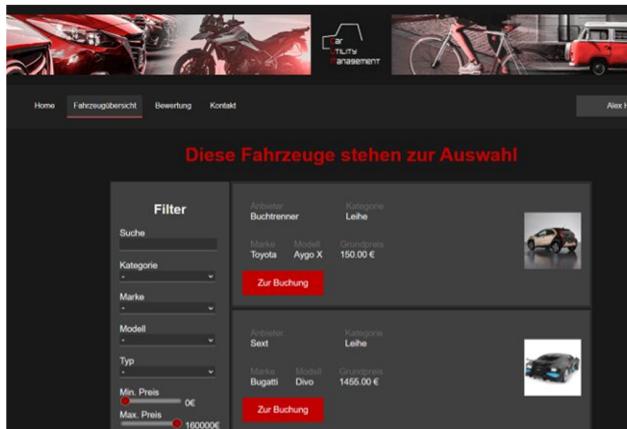
    // Calculate the page
    int itemsPerPage = 20;
    int currentPage = page ?? 1;
    if (currentPage <= 0) currentPage = 1;
    int totalItems = veh == null ? 0 : veh.Count();

    int pageCount = totalItems > 0 ? (int)Math.Ceiling(totalItems / (double)itemsPerPage) : 0;
    if (currentPage > pageCount) currentPage = pageCount;

    // Get filtered page view
    if (veh != null & totalItems > 0)
        veh = veh.Skip((currentPage - 1) * itemsPerPage).Take(itemsPerPage);

    // Set metadata for view
    ViewData["MaxPage"] = pageCount;
    ViewData["CurrentPage"] = currentPage;
    return PartialView("_overviewList", veh.ToList());
}
```

```
function getVehicleList() {
    $.ajax({
        url: apiUrl,
        datatype: 'html',
        method: 'GET',
        data: {
            ...findGetParameters(),
            searchTxt: $('#searchFilter').val(),
            categoryFilter: $('#categoryFilter').val(),
            brandFilter: $('#brandFilter').val(),
            modelFilter: $('#modelFilter').val(),
            typeFilter: $('#typeFilter').val(),
            minPrice: $('#minPriceFilter').val(),
            maxPrice: $('#maxPriceFilter').val()
        },
        success: (res) => {
            $('#listVehicles').html(res);
            $('#nextDaneBtn').prop('disabled', dataName == dataMaxDane);
        }
    });
}
```



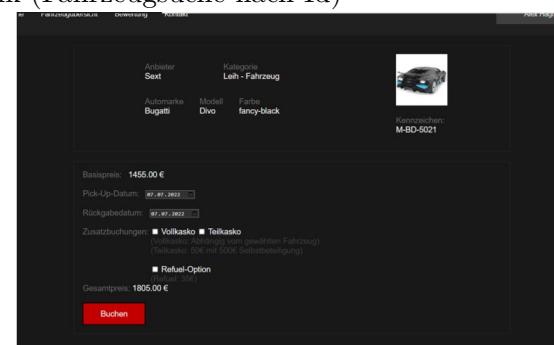
Fahrzeugdaten (sowie sämtliche andere persistente Daten der Seite), werden aus der SQLite Datenbank entnommen, durch den Controller nach Nutzereinstellungen gefiltert, und als HTML PartialView an den Client geschickt, welcher diese per JS in den DOM einfügt.

Buchungsseite

Buchungsroute: /booking/{vehicleId}

Befüllen der View mit den Daten aus der Datenbank (Fahrzeugsuche nach Id)

```
//<summary>
// Endpoint to the bookingview of a vehicle with specific VehicleId
//</summary>
//<param name="id">Vehicle Id</param>
//<returns></returns>
[HttpGet]
[Route("Routes.BOOKING")]
public async Task<IActionResult> Booking(string id) {
    //Get Vehicle from id and hand over to view
    List<Vehicle> vehicles = await dbStore.GetAllVehiclesAsync();
    var veh = vehicles.Where(e => id.Equals(e.VehicleId));
    if (veh.Any())
        return View(new BookingViewModel(await PrepareBaseViewModel(), veh.First()));
    else
        return NotFound(await PrepareBaseViewModel());
}
```



Beim ändern der Buchungseinstellungen wird der darunter stehende Gesamtpreis dynamisch durch Java-Script geupdated. Beim klicken auf "Buchen" werden dann die gewählten Buchungseinstellungen an das Backend übermittelt, der Preis nochmals berechnet (für mehr Sicherheit: Berechnung durch Backend) und das Fahrzeug gebucht.

```
/// <summary>
/// Endpoint to logical booking action
/// </summary>
/// <param name="id">Vehicle Id</param>
/// <param name="orderModel">Order Data</param>
/// <returns></returns>
[Route(Routes.BOOKING + "/bookingaction")]
[HttpPost]
[Authorize]
0 references
public async Task<IActionResult> PlaceOrder(string id, [FromForm] Order orderModel) {

    // if user is not logged in -> Loginpage
    if (!User.Identity.IsAuthenticated)
        return RedirectToRoute("Login", "Home");

    // Get user from Email
    var user = await userManager.FindByEmailAsync(User.Identity.Name);
    if (user == null)
        return RedirectToRoute("Login", "Home");
    var customer = (await dbStore.GetCustomersAsync()).Where(e => e.UserId.Equals(user.Id));
    if (customer == null || !customer.Any())
        return RedirectToRoute("Login", "Home");
    orderModel.User = customer.First();

    // Get vehicle from VehicleId
    var vehicle = (await dbStore.GetAllVehiclesAsync()).Where(e => e.VehicleId.Equals(id));
    if (vehicle == null || !vehicle.Any())
        return NotFound();
    orderModel.Vehicle = vehicle.First();

    // Calculate totalprice
    orderModel.Totalprice = PriceCalc.CalculateTotalprice(orderModel);

    // Create corresponding Order entry
    try {
        await dbStore.AddOrderAsync(orderModel);
    } catch (DatabaseAPIException) {
        return await Error(418);
    }

    // send confirmation mail
    _ = mailer.MailerAsync(configuration.GetValue<string>("MailCredentials:Email"), user.Email, MailTxt.CreateOrderSubject(orderModel),
        MailTxt.CreateOrderResponse(orderModel));
    return RedirectToAction("BookingDone", "Home");
}
```

Dieser Endpunkt ist nur verfügbar, wenn man eingeloggt ist. Es werden die übergebenen Daten überprüft, der Preis berechnet und die Order persistent in der Datenbank gespeichert. Anschließend wird eine Bestätigungsmaile an den Kunden gesendet und auf eine Erfolgsseite weitergeleitet.

Kriterien

Selbstgeschriebenes HTML

Alle HTML Dateien sind selbstgeschrieben und werden durch die ASP.NET Templatingengine zusammengesetzt.

Ordner: Project-Omni-Ride-Network/Views

- /Api → Fahrzeugliste, Bewertungsliste, die durch AJAX in Laufzeit an den Client gesendet werden.
- /Home → Standard Routen/Views.
- /Shared/_Layout.cshtml → Datei, die das Hauptlayout beinhaltet, in das alle Views eingesetzt werden.

Fehlerfreies HTML

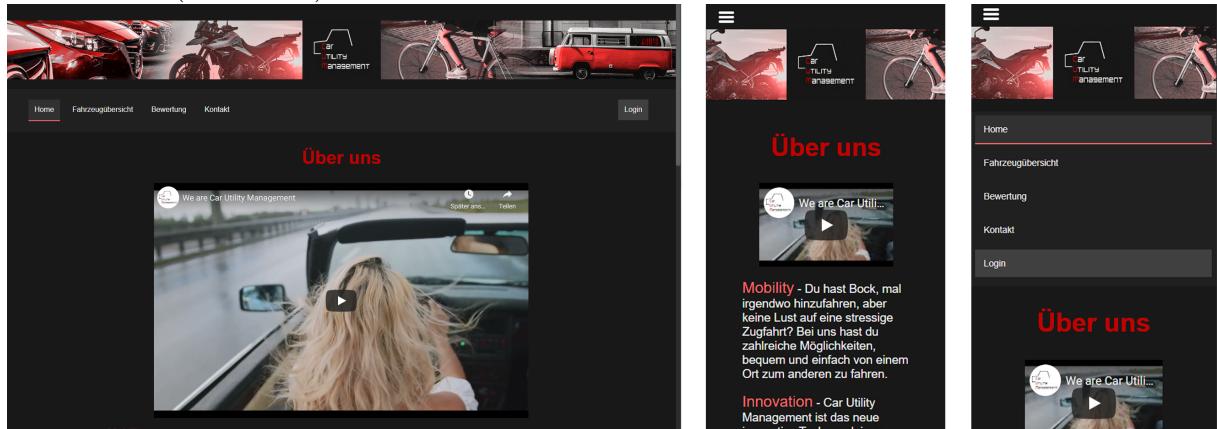
Siehe Ordner /Result/ für die gerenderten HTML dateien
(oder optional auf <http://c-u-management.de>)

Selbstgeschriebenes CSS

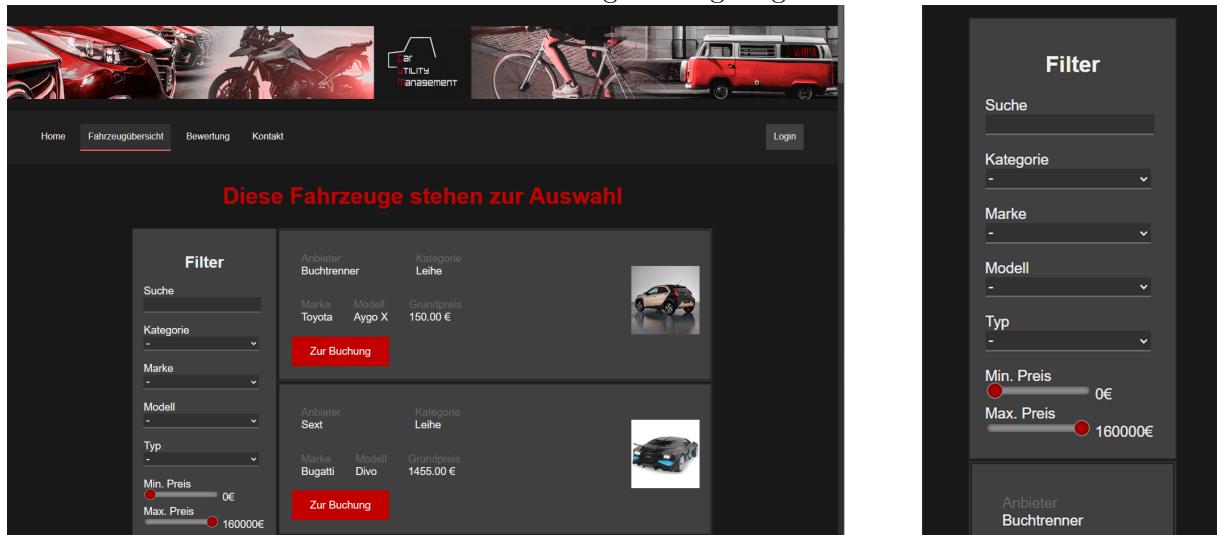
Das komplette CSS wurde selbst geschrieben.
Ordner: Project-Omni-Ride-Network/wwwroot/css/style.css

Responsive Design

Navigationsleiste verschwindet bei mobilen Endgeräten und wird durch einen Dropdown Menübutton (oben links) ersetzt.



Es wurde darauf geachtet, dass jede Seite sowohl am Desktop Computer als auch am Handy benutzbar und übersichtlich ist. Zum Beispiel werden die Filter der Fahrzeugübersicht bei zu kleinem Bildschirm als Block über der Fahrzeugliste angezeigt.



Modernes JavaScript

JavaScript, beispielsweise zum dynamischen Einbinden der Fahrzeugliste / Bewertungsliste oder zur Preisberechnung bei der Buchung, auf modernen JavaScript Standards.

Selbstgeschriebene Frontend-Logik

Alle Scripts wurden selbstgeschrieben. Teilweise wurde noch JQuery miteingebunden, um weitere Funktionalitäten zur Basis zu haben.

Ordner: Project-Omni-Ride-Network/wwwroot/js

Selbstgeschriebene Backend-Logik

Das komplette Backend wurde selbst in C# auf der Basis des ASP.NET Core Frameworks geschrieben. Zusätzlich wurde EntityFramework als SqLite Datenbank Adapter und DNAFramework für die Dependency Injection benutzt.

Backend-Aufteilung nach Model-2-Entwurfsmuster

Die Backendstruktur ist nach Model-2-Entwurfsmuster aufgeteilt, mit den verschiedenen Controllern in 'Project-Omni-Ride-Network/Controllers' und den Views in 'Project-Omni-Ride-Network/Views'