**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

**Отчет по технологической (производственной) практике**

на тему: «Web приложение для работы автосалона»

Исполнитель: студент гр. ИП-31

Зайцев М.С.

Руководитель от предприятия:

Гуд С.Н.

Руководитель: преподаватель Богданова Н. С.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Гомель 2021**

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc75868567)

[**1 ОБЩИЕ СВЕДЕНИЯ ОБ ОРГАНИЗАЦИИ** 4](#_Toc75868568)

[1.1 История организации 4](#_Toc75868569)

[1.2 Охрана труда и техника безопасности на рабочем месте 5](#_Toc75868570)

[**2 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ** 10](#_Toc75868571)

[2.1 Постановка задачи 10](#_Toc75868572)

[2.2 Обзор существующих технологий для моделирования базы данных 10](#_Toc75868573)

[2.3 Описание технологии решения поставленной задачи 11](#_Toc75868574)

[2.4 Структура приложения 12](#_Toc75868575)

[2.5 Общие сведения для работы с приложением 14](#_Toc75868576)

[2.6 Описание интерфейса пользователя 14](#_Toc75868577)

[**ЗАКЛЮЧЕНИЕ** 18](#_Toc75868578)

[**Список использованных источников** 19](#_Toc75868579)

[**Приложение A** 20](#_Toc75868580)

# 

# ВВЕДЕНИЕ

Сегодня невозможно представить себе нашу жизнь без автомобильного транспорта. Ежедневно мы едем в школу, родители на работу. Автомобиль уже очень давно стал не роскошью, а средством передвижения.

Продукты в магазины привозят на автомобилях, лекарства в аптеки также везут на авто. Без автомобильного транспорта невозможно было бы спасать людей. Пожарные, полицейские, аварийные – все они должны быть очень быстрыми, чтобы вовремя успеть до опасного города.

Автосалон – это место продажи, хранения и обслуживания автомобилей. И он имеет место быть в нашем современном обществе исходя из того, что автомобильный транспорт занимает важное место в единой транспортной системе страны. Он перевозит более 80% народнохозяйственных грузов, что обусловлено высокой маневренностью автомобильного транспорта, возможностью доставки грузов «от двери до двери» без дополнительных перегрузок в пути, а, следовательно, высокой скоростью доставки и сохранностью грузов. Большая протяженность автомобильных дорог обеспечивает возможность их повсеместной эксплуатации при значительной провозной способности. Высокая мобильность, способность оперативно реагировать на изменения пассажиропотоков ставят автомобильный транспорт «вне конкуренции» при организации местных перевозок пассажиров. На его долю приходится почти половина пассажирооборота. Автомобильный транспорт сыграл огромную роль в формировании современного характера расселения людей, в распространении дальнего туризма, в территориальной децентрализации промышленности и среды обслуживания.

Данная задание, выданное на предприятии направлена на то, чтобы изучить структуру работы автосалона. Цель задания – спроектировать деятельность автосалона для повышения качества и упрощения управления некоторыми аспектами, разработать приложения для автоматизации деятельности автосалона.

# 1 ОБЩИЕ СВЕДЕНИЯ ОБ ОРГАНИЗАЦИИ, ЭТАПЫ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНОГО ПРОДУКТА

## **1.1 История организации**

EPAM (Effective Programming for America) основали одноклассники Аркадий Добкин и Леонид Лознер. История компании началась в двух комнатах: штаб устроили в квартире Добкина в Нью-Джерси, минский офис — в подвале дома на Куйбышева.

В 1997 году компания начала сотрудничать с [SAP](https://www.sap.com/index.html) — мировым лидером поставщиков программных решений для бизнеса. По легенде, руководитель SAP [Хассо Платтнер](https://en.wikipedia.org/wiki/Hasso_Plattner" \t "_blank) случайно [увидел](https://www.kp.by/daily/26371.4/3252197/) систему, которую разработали в EPAM, и решил, что её сделала крупная компания. Когда узнал, что это дело рук группы людей из Минска, встретился с руководством и предложил сделать аналогичный продукт для SAP.

В 2003 году SAP [выбрала](http://www.cnews.ru/news/line/sap_vybral_epam_systems_klyuchevym_partnerom) EPAM основным технологическим партнёром в разработке приложений xApps, которые объединяли в разнородной корпоративной среде сотрудников, информацию и бизнес-процессы. SAP на тот момент была одной из крупнейших софтовых компаний в мире, сотрудничество открыло перед EPAM новые двери. «У нас появилась возможность научиться делать профессиональный софтовый продукт у профессиональной софтовой компании», — [вспоминает](https://probusiness.io/strategy/3276-arkadiy-dobkin-o-novoy-celi-epam-idealnykh-sotrudnikakh-i-energii-osnovatelya.html) Добкин. В ноябре 2010 года EPAM [получила](http://www.tadviser.ru/index.php/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F:EPAM_Systems_(%D0%AD%D0%9F%D0%90%D0%9C_%D0%A1%D0%B8%D1%81%D1%82%D1%8D%D0%BC%D0%B7)#.22.D0.92.D0.A2.D0.91_.D0.9A.D0.B0.D0.BF.D0.B8.D1.82.D0.B0.D0.BB.22_.D0.BF.D1.80.D0.B8.D0.BE.D0.B1.D1.80.D0.B5.D0.BB_.D0.BC.D0.B8.D0.BD.D0.BE.D1.80.D0.B8.D1.82.D0.B0.D1.80.D0.BD.D1.8B.D0.B9_.D0.BF.D0.B0.D0.BA.D0.B5.D1.82_EPAM_Systems) статус SAP Gold Channel Partner (наивысший партнёрский статус в программе SAP PartnerEdge).

Фонд [Siguler Guff](http://www.sigulerguff.com/" \t "_blank) приобрёл акции EPAM за $20 млн. «Органический рост компании в течение последних пяти лет поставил их в положение регионального лидера. Мы надеемся ускорить рост EPAM, полностью используя возможности для региональной консолидации», — управляющий директор фонда Дрю Гафф.  Через три года ещё три компании инвестировали в EPAM — за пакет акций $50 млн [заплатили](https://www.sigulerguff.com/sites/default/files/5.15.08EPAMSystemsAnnounces50MillionCapitalRaise.pdf) фонды [Renaissance Investment Management](https://www.reninv.com/" \t "_blank) (инвестиционно-банковская фирма специализируется на Европе, Ближнем Востоке и Африке), [Da Vinci Capital](http://www.dvcap.com/" \t "_blank) (группа компаний, в основном работают в СНГ) и [Euroventures Capital](http://www.euroventures.hu/en" \t "_blank) (одна из старейших и ведущих независимых частных компаний в Центральной Европе).

Выручка компании в 2010 году составила $222 млн — по сравнению с предыдущим годом [выросла](http://www.cnews.ru/news/top/epam_privlechet_do_133_mln_na_birzhe) на 40%.

Белорусский флаг — впервые на Нью-Йоркской фондовой бирже. 25 января 2012 года EPAM объявила о начале подготовки к IPO. Готовиться начали еще в 2000-м. IPO состоялось 8 февраля, было оценено аналитиками как неудачное: вместо ожидаемых $16–18 за бумагу акции продавались за $12. Стоимость всей компании составила $488 млн: на $160-$240 млн ниже предварительных оценок, которые давали EPAM организаторы IPO. Добкин [комментировал](http://www.cnews.ru/news/top/osnovatel_epam_otdacha_ot_nashego_ipo) ситуацию так: «Прогнозирование цены IPO, к сожалению, напоминает рулетку. И тут мы попали в не очень удачный момент. Из Европы было много негативной общеэкономической информации. 50% наших клиентов находятся в Европе. Инвесторы оказались настроены достаточно консервативно». Аналитики отмечали, что в том же году Facebook объявила о предстоящем размещении на фондовой бирже, и интерес к другим пакетам значительно упал. Другие [говорят](http://www.cnews.ru/news/top/izza_facebook_otsenka_epam_v_hode_ipo), EPAM слишком резко сообщила о дате размещения акций — даже те, кто был заинтересован, не успели толком ознакомиться с предложением.

В июне 2014 капитализация компании выросла более чем в четыре раза и составила $2,14 млрд. В настоящее время одну акцию EPAM можно приобрести примерно за $130. «У нас добавилось около 10% сотрудников в США и Западной Европе, то есть мы увеличили пропорцию в пять раз. Основная масса этих сотрудников не говорит по-русски. Компания стала реально международной», — [говорит](https://probusiness.io/strategy/3276-arkadiy-dobkin-o-novoy-celi-epam-idealnykh-sotrudnikakh-i-energii-osnovatelya.html) Добкин. В 2015 EPAM меняет логотип, русский язык перестаёт быть принципиально базовым для компании. По [данным](http://www.cnews.ru/news/top/smena_firmennogo_stilya_obojdetsya_epam) cNews, на обновление и продвижение фирменного стиля компания потратила около $20 млн.

Прибыль EPAM в третьем квартале 2018 года выросла на 27,2% в годовом исчислении до $1,17 на акцию. Выручка за квартал составила $468,2 млн, что на 24% больше по сравнению с аналогичным периодом прошлого года. 14 аналитиков Уолл-стрит [опубликовали](https://www.marketbeat.com/stocks/NYSE/EPAM/price-target/) 12-месячные целевые цены на акции EPAM. Их прогнозы колеблются от $130 до $157. В среднем они ожидают, что цена акций EPAM достигнет $142,4 в следующем году. Аналитики также опубликовали рейтинги EPAM «покупать», «держать» и «продавать». Только трое рекомендуют удерживать акции и 11 считают акции EPAM выгодным вложением и рекомендуют «покупать».

## **1.2 Охрана труда и техника безопасности на рабочем месте**

Общие требования охраны труда:

* К работе программистом допускаются лица не моложе 18 лет, имеющие соответствующую выполняемой работе квалификацию, прошедшие вводный и первичный на рабочем месте инструктажи по охране труда, обученные безопасности труда при работе с персональным компьютером.
* Для выполнения работ на персональном компьютере программист должен изучить Инструкцию по эксплуатации персонального компьютера, на котором работник выполняет работы, пройти инструктаж по электробезопасности и получить I группу.
* Программист, выполняющий работу на персональном компьютере, независимо от квалификации и стажа работы, не реже одного раза в шесть месяцев должен проходить повторный инструктаж по безопасности труда; в случае нарушения требований безопасности труда, при перерыве в работе более чем на 60 календарных дней программист должен пройти внеплановый инструктаж.
* Программист, не прошедший инструктажи по охране труда и не имеющий I группы по электробезопасности, к самостоятельной работе не допускается.
* Программист, показавший неудовлетворительные навыки и знания требований безопасности при работе на персональном компьютере, к самостоятельной работе не допускается.
* Программист, допущенный к постоянной работе на персональном компьютере, перед поступлением на работу и в дальнейшем периодически (не реже 1 раза в год) должен проходить медицинские осмотры.
* Программист, допущенный к самостоятельной работе, должен знать: правила эксплуатации и требования безопасности при работе с персональным компьютером, способы рациональной организации рабочего места, санитарно- гигиенические требования к условиям труда, опасные и вредные производственные факторы, которые могут оказывать неблагоприятное воздействие на программиста.
* Программист, направленный для участия в несвойственных его профессии работах, должен пройти целевой инструктаж по безопасному выполнению предстоящих работ.
* Программисту запрещается пользоваться инструментом, приспособлениями и оборудованием, безопасному обращению с которыми он не обучен.

Требование охраны труда перед началом работы:

* Перед началом работы программисту следует рационально организовать свое рабочее место.
* Программист должен знать о том, что, если в помещении расположены несколько персональных компьютеров, то для обеспечения безопасности расстояние между ними должно быть не менее 1,5 м.
* Не рекомендуется располагать монитор экраном к окну.
* Для того чтобы в процессе работы не возникало перенапряжение зрительного анализатора, программисту следует проверить, чтобы на клавиатуре и экране монитора не было бликов света.
* Для повышения контрастности изображения перед началом работы программист должен очистить экран монитора от пыли, которая интенсивно оседает на нем под воздействием зарядов статического электричества.
* Программист должен убрать с рабочего места все лишние предметы, не используемые в работе.
* Перед включением персонального компьютера программисту следует визуально проверить исправность электропроводки, вилки, розетки, а также электрических подсоединений между собой всех устройств, входящих в комплект персонального компьютера.
* Перед началом выполнения работы программист должен проверить исправность персонального компьютера и подготовить его к работе.

Требования храны труда в аварийных ситуаиях

* При обнаружении каких-либо неполадок в работе персонального компьютера программист должен прекратить работу, выключить компьютер и сообщить об этом непосредственному руководителю для организации ремонта.
* Программисту не следует самому устранять технические неполадки персонального компьютера.
* Программист не должен производить работу при снятом корпусе компьютера.
* При несчастном случае, отравлении, внезапном заболевании необходимо немедленно оказать первую помощь пострадавшему, вызвать врача или помочь доставить пострадавшего к врачу, а затем сообщить руководителю о случившемся.
* Программист должен уметь оказывать первую помощь при ранениях; при этом он должен знать, что всякая рана легко может загрязниться микробами, находящимися на ранящем предмете, коже пострадавшего, а также в пыли, на руках оказывающего помощь и на грязном перевязочном материале.
* Оказывая первую помощь при ранении, необходимо соблюдать следующие правила:
* Нельзя промывать рану водой или даже каким-либо лекарственным препаратом, засыпать порошком и смазывать мазями, так как это препятствует заживлению раны, вызывает нагноение и способствует занесению в нее грязи с поверхности кожи.
* Нужно осторожно снять грязь с кожи вокруг раны, очищая ее от краев наружу, чтобы не загрязнять рану; очищенный участок кожи нужно смазать йодом и наложить повязку.
* Для оказания первой помощи при ранении необходимо вскрыть имеющийся в аптечке перевязочный пакет.
* При наложении перевязочного материала не следует касаться руками той его части, которая должна быть наложена непосредственно на рану; если перевязочного пакета почему-либо не оказалось, то для перевязки можно использовать чистый платок, чистую ткань и т.п.; накладывать вату непосредственно на рану нельзя.
* На то место ткани, которое накладывается непосредственно на рану, нужно накапать несколько капель йода, чтобы получить пятно размером больше раны, а затем положить ткань на рану; оказывающий помощь должен вымыть руки или смазать пальцы йодом; прикасаться к самой ране даже вымытыми руками не допускается.
* Первая помощь пострадавшему должна быть оказана немедленно и непосредственно на месте происшествия, сразу же после устранения причины, вызвавшей травму, используя медикаменты и перевязочные материалы, которые должны храниться в аптечке.
* Аптечка должна быть укомплектована перевязочными материалами и медикаментами, у которых не истек срок реализации; аптечка должна находиться на видном и доступном месте.
* Если произошла травма вследствие воздействия электрического тока, то меры оказания первой помощи зависят от состояния, в котором находится пострадавший после освобождения его от действия электрического тока:
* Если пострадавший находится в сознании, но до этого был в состоянии обморока, его следует уложить в удобное положение и до прибытия врача обеспечить полный покой, непрерывно наблюдая за дыханием и пульсом; ни в коем случае нельзя позволять пострадавшему двигаться.
* Если пострадавший находится в бессознательном состоянии, но с сохранившимся устойчивым дыханием и пульсом, его следует удобно уложить, расстегнуть одежду, создать приток свежего воздуха, дать понюхать нашатырный спирт, обрызгать водой и обеспечить полный покой.
* Если пострадавший плохо дышит (очень редко и судорожно), ему следует делать искусственное дыхание и массаж сердца; при отсутствии у пострадавшего признаков жизни (дыхания и пульса) нельзя считать его мертвым, искусственное дыхание следует производить непрерывно как до, так

и после прибытия врача; вопрос о бесцельности дальнейшего проведения искусственного дыхания решает врач.

* При обнаружении пожара или признаков горения (задымление, запах гари, повышение температуры и т.п.) необходимо немедленно уведомить об этом пожарную охрану по телефону 01.
* До прибытия пожарной охраны нужно принять меры по эвакуации людей, имущества и приступить к тушению пожара.
* При возгорании персонального компьютера программист должен отключить его от источника тока и приступить к тушению своими силами; при этом следует помнить, что для тушения установок, находящихся под напряжением, применяют углекислотные или порошковые огнетушители.

Большое значение имеет также характер работы. В частности, при организации рабочего места программиста должны быть соблюдены следующие основные условия:

* оптимальное размещение оборудования, входящего в состав рабочего места;
* достаточное рабочее пространство, позволяющее осуществлять все необходимые движения и перемещения;
* необходимо естественное и искусственное освещение для выполнения поставленных задач;
* уровень акустического шума не должен превышать допустимого значения.

Главными элементами рабочего места программиста являются письменный стол и кресло. Основным рабочим положением является положение сидя. Рабочее место для выполнения работ в положении сидя организуется в соответствии с ГОСТ 12.2.032-78.

# 2 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

## **2.1 Постановка задачи**

Необходимо разработать web-приложение, выполняющее следующие функции:

* вывод информации о машинах;
* авторизация пользователя;
* регистрация нового пользователя;
* добавление новых машин;
* изменение цен;
* форма для вопросов.

Эти функции работают на основе следующий справочников:

* пользователи;
* машины.

Приложение должно иметь разделение на роли. Для каждой роли должен быть предусмотрен свой уникальный функционал:

Необходимые роли:

* администратор;
* клиент.

# 2.2 Обзор существующих технологий для моделирования базы данных

База данных, БД – совокупность взаимосвязанных, хранящихся вместе сведениях о различных сущностях одной предметной области (реальных объектах, процессах, явлениях или событиях), обеспечивающая наличие такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений или пользователей; одним из основных свойств баз данных можно считать независимость данных от использующих их прикладных программ. Под независимостью данных подразумевается то, что изменения в данных не приводит к изменению программ. Разработка программ длительный, трудоемкий и дорогостоящий процесс, поэтому при возникновении потребности модифицировать структуру данных, необходимости сохранять уже созданные прикладные программы.

В настоящее время существуют различные СУБД – MS SQL Server, MySQL, Interbase, Oracle, DB2, Paradox, FoxPro и множество других, менее известных. В большинстве случаев базы данных используются для создания автоматизированных информационных систем. База данных является ядром любой информационной системы и позволяет хранить информацию.

При разработке приложения использовалась база данных IndexedDb*.*

IndexedDB транзакционная система базы данных, как SQL – основанная RDBMS. Однако, в отличие от RDBMS, которая использует таблицы с фиксированными колонками, IndexedDB — JavaScript – основанная объектно-ориентированная база данных. IndexedDB позволяет сохранять и возвращать объекты, которые были проиндексированы с **ключом**; любой объект, поддерживаемый [структурированным алгоритмом клонирования](https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Structured_clone_algorithm) может быть сохранён.

## **2.3 Описание технологии решения поставленной задачи**

Для создания приложения было выбрано несколько языков. Основная логика была написана на TypeScript.

TypeScript– язык программирования, представленный [Microsoft](https://ru.wikipedia.org/wiki/Microsoft" \o "Microsoft) в [2012 году](https://ru.wikipedia.org/wiki/2012_%D0%B3%D0%BE%D0%B4) и позиционируемый как средство разработки [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), расширяющее возможности [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript).

TypeScript является обратно совместимым с JavaScript и компилируется в последний. Фактически, после компиляции программу на TypeScript можно выполнять в любом современном браузере или использовать совместно с серверной платформой [Node.js](https://ru.wikipedia.org/wiki/Node.js). Код экспериментального компилятора, транслирующего TypeScript в JavaScript, распространяется под [лицензией Apache](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D1%8F_Apache). Его разработка ведётся в публичном репозитории через сервис [GitHub](https://ru.wikipedia.org/wiki/GitHub).

TypeScript отличается от JavaScript возможностью [явного](https://ru.wikipedia.org/wiki/%D0%AF%D0%B2%D0%BD%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) [статического](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) назначения типов, поддержкой использования полноценных классов (как в традиционных объектно-ориентированных языках), а также поддержкой подключения модулей, что призвано повысить скорость разработки, облегчить читаемость, [рефакторинг](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3" \o "Рефакторинг) и повторное использование кода, помочь осуществлять поиск ошибок на этапе разработки и компиляции, и, возможно, ускорить выполнение программ.

Для отрисовки страницы используется язык JavaScript.

JavaScript — [мультипарадигменный](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BF%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F).

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам [приложений](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0). Наиболее широкое применение находит в [браузерах](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) как язык сценариев для придания [интерактивности](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C) [веб-страницам](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0).

Основные архитектурные черты: [динамическая типизация](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [слабая типизация](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%B0%D0%B1%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [автоматическое управление памятью](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [прототипное программирование](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Прототипное программирование), функции как [объекты первого класса](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_%D0%BF%D0%B5%D1%80%D0%B2%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B0).

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

Для стилизации использовался препроцессор Scss.

SCSS — диалект языка [SASS](http://sass-lang.com/). Sass — это [метаязык](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B0%D1%8F%D0%B7%D1%8B%D0%BA) на основе [CSS](https://ru.wikipedia.org/wiki/CSS), предназначенный для увеличения уровня абстракции CSS-кода и упрощения файлов [каскадных таблиц стилей](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%81%D0%BA%D0%B0%D0%B4%D0%BD%D1%8B%D0%B5_%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8B_%D1%81%D1%82%D0%B8%D0%BB%D0%B5%D0%B9).

Для сборки проекта использовался Webpack.

Webpack – это сборщик модулей [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Он создан в первую очередь для JavaScript, но может преобразовывать внешние ресурсы, такие как HTML, CSS и изображения, если включены соответствующие загрузчики. Webpack принимает модули с зависимостями и генерирует статические ресурсы, представляющие эти модули.

Webpack принимает зависимости и генерирует граф зависимостей, позволяющий веб-разработчикам использовать модульный подход для разработки своих веб-приложений. Его можно использовать из командной строки или настроить с помощью файла конфигурации с именем [*webpack.config.js*](https://webpack.js.org/configuration/). Этот файл используется для определения правил, плагинов и т.д. для проекта. (цebpack сильно расширяем с помощью правил, которые позволяют разработчикам писать задачи, которые они хотят выполнять при объединении файлов.) Для использования webpack требуется [Node.js](https://ru.wikipedia.org/wiki/Node.js).

## **2.4 Структура приложения**

Приложение можно разделить на 3 части: работа с базой данных, прелставление, классы страниц. Общая структура классов представлена на рисунке 2.1:

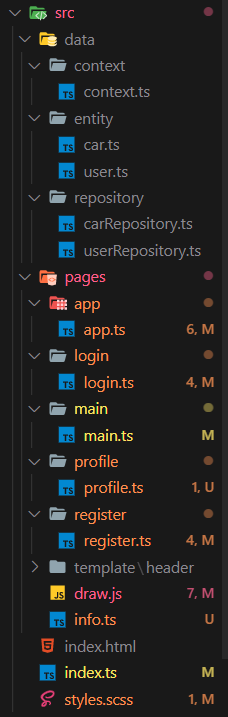


Рисунок 2.1 – Структура классов приложения

Структура приложения состоит из папок. Вся основная логика храниться в папке *src*. В этой папке находятся папки *data, pages, template* и некоторые файлы. В файле *index.html* будет находиться код отображаемой страницы. В файле *index.ts* находится логика для запуска приложения. В файле styles.scss находится стили страниц.

Рассмотрим папки *data*, *pages* и *template.* В папке *data* храниться сущности и основная логика для работы с базой данных. В этой папке находятся еще несколько папок:  *context, entity, repository.* Файл *context.ts* в папке *context* отвечает за подключения базы данных. В папке *entity* находятся сущности для базы данных. Файлы *carRepository.ts* и *userRepository.ts* в папке *repository* отвечают за доступ к данным в базе данных.

Рассмотрим папку *pages*. Тут хранятся все страницы приложения. Логика каждой страницы находится в отдельных файлах и папках. Отрисовка страниц производится в файле *draw.js*.

Рассмотрим папку template. В этой папке хранятся элементы страниц которые могут повторяться, например *header*.

Класс App – основной класс приложения. При создании его объекта и вызова функции *render()* происходит генерирование страницы Main. При нажатии разных кнопок будет происходить создание объекта соответствующего класса и генерация соответствующей страницы. Страницы получают данные из базы с помощью контекста данных и репозиториев.

На рисунке 2.2 показаны файлы настройки Webpack

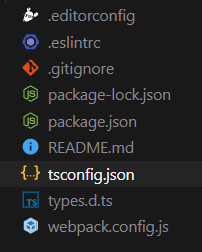


Рисунок 2.2 – файлы настройки Webpack

## **2.5 Общие сведения для работы с приложением**

При разработке приложения были использованы следующие инструменты:

* среда разработки Visual Studio Code;

## **2.6 Описание интерфейса пользователя**

После запуска приложения гость увидит страницу:

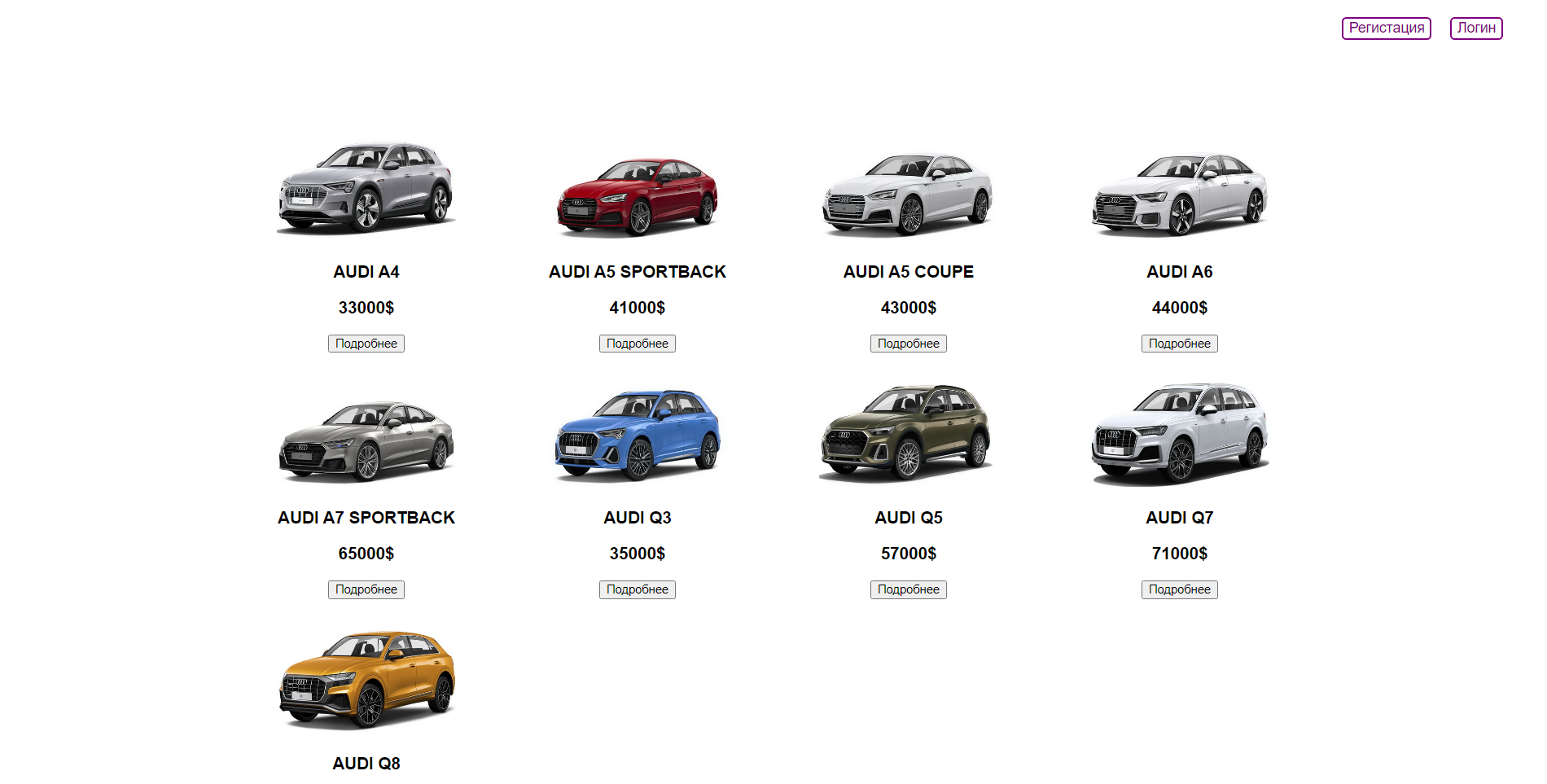


Рисунок 2.3 – Главная страница

На рисунке 2.4 показана форма регистрации:

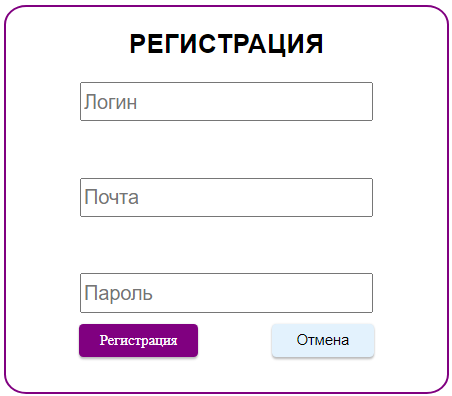
****

Рисунок 2.4 – Форма регистрации

На рисунке 2.5 показана форма авторизации:

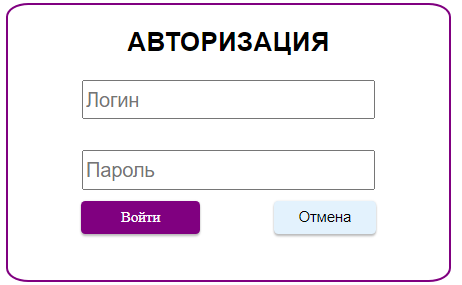
****

Рисунок 2.5 – Форма авторизации

Если гость авторизуется под пользователем, то увидит страницу:

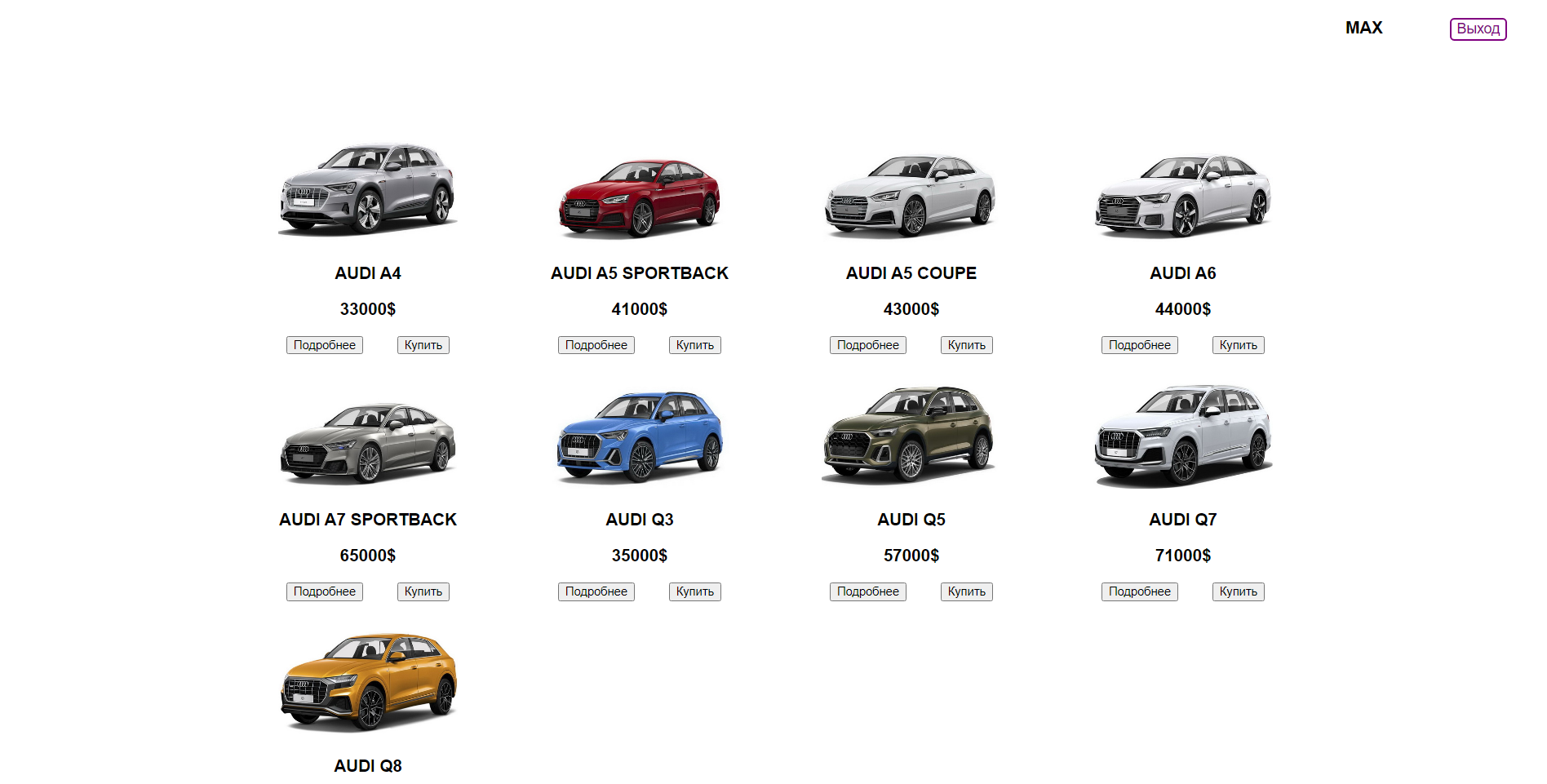
****

Рисунок 2.6 – Главная страница

Если гость авторизуется под администратором, то увидит страницу:

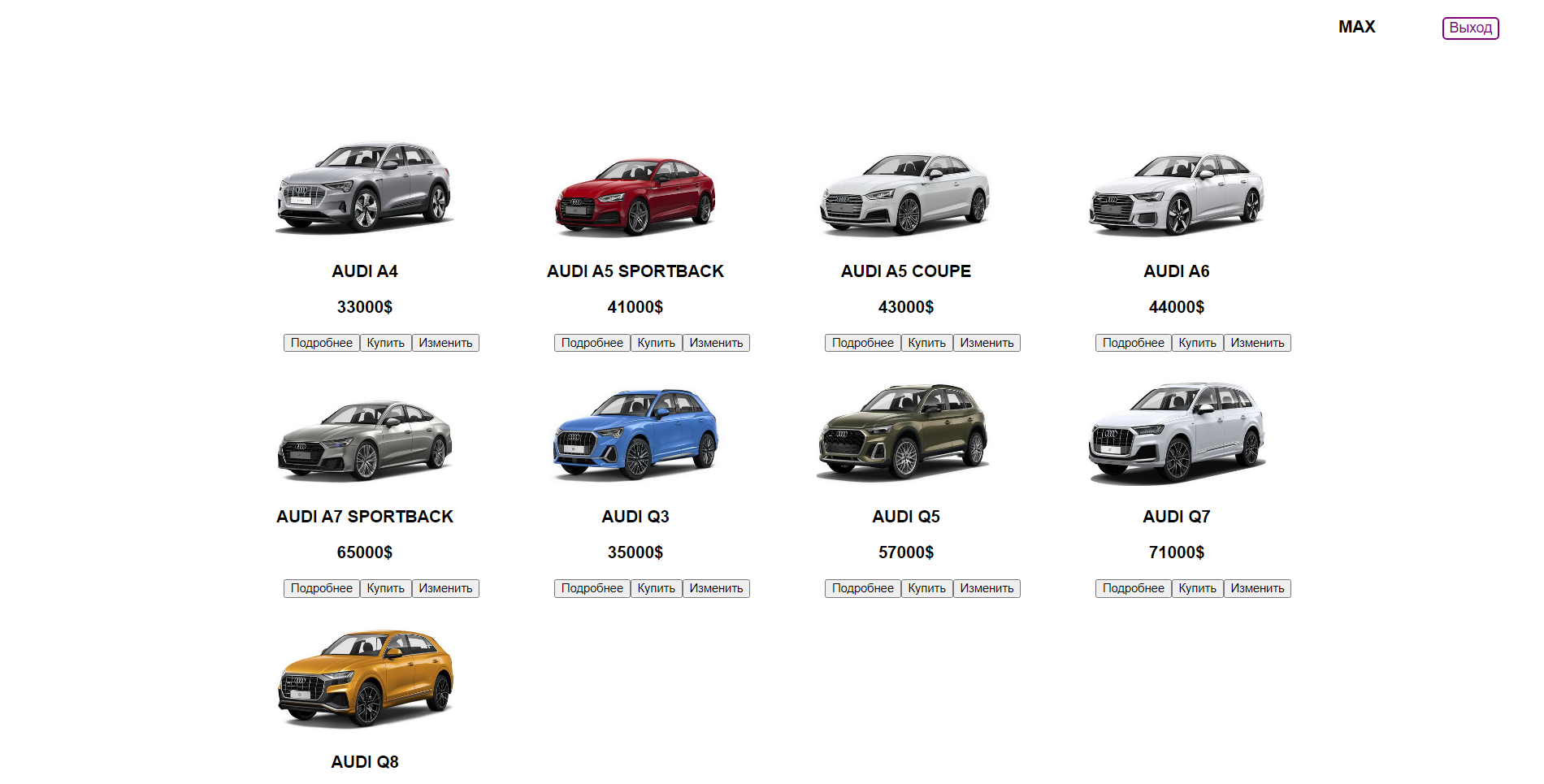


Рисунок 2.7 – Главная страница

# ЗАКЛЮЧЕНИЕ

В рамках производственной практики было спроектировано SPA Web приложение для работы автосалона. Это приложение имеет следующий функционал:

* ведение справочников;
* авторизация и регистрация пользователей;
* хранение данных;

В процессе работы были поставлены основные задачи и подготовлены решения для выполнения основных технических требований. Поставленная задача была выполнена при помощитаких языков как: TypeScript, JavaScript. Сборка проекта проводилась с помощью Webpack. Для хранения данных была использована база данных IndexedDb.

В ходе производственной практики выл получен опыт работы с TypeScript, JavaScript, работы с IndexedDb, Webpack.

Данный проект требует значительных доработок. Из добавок можно отметить добавление новых функций, добавление мобильной версии, доработать дизайн.

# Список использованных источников

1. [Розенталс](https://chitatel.by/searchnew?query=%D0%A0%D0%BE%D0%B7%D0%B5%D0%BD%D1%82%D0%B0%D0%BB%D1%81+%D0%9D.), Изучаем TypeScript 3. – [ДМК Пресс](https://chitatel.by/searchnew?query=%D0%94%D0%9C%D0%9A+%D0%9F%D1%80%D0%B5%D1%81%D1%81), 2019. – 608 c.

2.Экспорт и импорт – [Электронный ресурс]. – Режим доступа: <https://learn.javascript.ru/import-export>, – Дата доступа: 17.06.2021.

3. Настройки компиляции TypeScript – [Электронный доступ]. – Режим доступа: <https://metanit.com/web/typescript/1.4.php>, – Дата доступа: 17.06.2021.

4. Промисы: обработка ошибок – [Электронный доступ]. – Режим доступа: <https://learn.javascript.ru/promise-error-handling>, – Дата доступа 18.06.2021.

## **Приложение A**

Листинг: context.ts

class DbContext {

  private dbAccess: DbContext | undefined;

  private db: IDBDatabase | undefined;

  async connect(dbName: string, storeName: string) {

    if (this.db) {

      return this.db;

    }

    const request = indexedDB.open(dbName, 1);

    return new Promise<IDBDatabase>((resolve, reject) => {

      request.onsuccess = () => {

        this.db = request.result;

        resolve(this.db);

      };

      request.onupgradeneeded = () => {

        this.db = request.result;

        request.result.createObjectStore(storeName, { keyPath: 'id' });

        resolve(this.db);

      };

    });

  }

  get instance() : DbContext {

    if (!this.dbAccess) {

      this.dbAccess = new DbContext();

    }

    return this.dbAccess;

  }

}

export default DbContext;

Листинг: car.ts

class Car {

  public id: number;

  public name: string;

  public price: number;

  public src: string;

  constructor(\_id: number, \_name: string, \_price: number, \_src: string) {

    this.id = \_id;

    this.name = \_name;

    this.price = \_price;

    this.src = \_src;

  }

}

export default Car;

Листинг: user.ts

class User {

  public id: string;

  public email: string;

  public password: string;

  public role: string;

  constructor(\_id: string, \_email: string, \_password: string, \_role: string) {

    this.id = \_id;

    this.email = \_email;

    this.password = \_password;

    this.role = \_role;

  }

}

export default User;

Листинг: carReposityry.ts

import DbContext from '../context/context';

import Car from '../entity/car';

class CarRepository {

  private connection: Promise<IDBDatabase>;

  constructor(dbName: string, private storeName: string) {

    this.connection = new DbContext().instance.connect(dbName, storeName);

  }

  async add(item: Car) : Promise<Car> {

    const db = await this.connection;

    const request = db.transaction([this.storeName], 'readwrite')

      .objectStore(this.storeName)

      .add(item);

    return CarRepository.requestHandler(request);

  }

  async remove(id: string) : Promise<Car> {

    const db = await this.connection;

    const request = db.transaction([this.storeName], 'readwrite')

      .objectStore(this.storeName)

      .delete(id);

    return CarRepository.requestHandler(request);

  }

  private static requestHandler(request: IDBRequest) : Promise<Car> {

    return new Promise<Car>((resolve, reject) => {

      request.onsuccess = () => resolve(request.result);

      request.onerror = () => reject(request.result);

    });

  }

  async retrieve() : Promise<Car[]> {

    const db = await this.connection;

    const store = db.transaction([this.storeName], 'readonly')

      .objectStore(this.storeName);

    return new Promise<Car[]>((resolve, reject) => {

      const result: any[] = [];

      store.openCursor().onsuccess = (event) => {

        const cursor = (event.target as any).result;

        if (cursor) {

          result.push(cursor.value);

          cursor.continue();

        } else {

          return resolve(result);

        }

      };

    });

  }

}

export default CarRepository;

Листинг: userReposityry.ts

import DbContext from '../context/context';

import User from '../entity/user';

class UserRepository {

  private connection: Promise<IDBDatabase>;

  constructor(dbName: string, private storeName: string) {

    this.connection = new DbContext().instance.connect(dbName, storeName);

  }

  async add(item: User) : Promise<User> {

    const db = await this.connection;

    const request = db.transaction([this.storeName], 'readwrite')

      .objectStore(this.storeName)

      .add(item);

    return UserRepository.requestHandler(request);

  }

  async remove(id: string) : Promise<User> {

    const db = await this.connection;

    const request = db.transaction([this.storeName], 'readwrite')

      .objectStore(this.storeName)

      .delete(id);

    return UserRepository.requestHandler(request);

  }

  private static requestHandler(request: IDBRequest) : Promise<User> {

    return new Promise<User>((resolve, reject) => {

      request.onsuccess = () => resolve(request.result);

      request.onerror = () => reject(request.result);

    });

  }

  async retrieve() : Promise<User[]> {

    const db = await this.connection;

    const store = db.transaction([this.storeName], 'readonly')

      .objectStore(this.storeName);

    return new Promise<User[]>((resolve, reject) => {

      const result: *any*[] = [];

      store.openCursor().onsuccess = (event) => {

        const cursor = (event.target as any).result;

        if (cursor) {

          result.push(cursor.value);

          cursor.continue();

        } else {

          return resolve(result);

        }

      };

    });

  }

}

export default UserRepository;

Листинг: app.ts

import User from '../../data/entity/user';

import { headerDraw } from '../draw';

import Info from '../info';

import LoginPage from '../login/login';

import Main from '../main/main';

import ProfilePage from '../profile/profile';

import Register from '../register/register';

import Header from '../template/header/header';

class App {

  private container: string;

  private header: HTMLElement;

  private page: HTMLElement;

  constructor() {

    Info.currentUser = new User('0', 'test@gmail.com', 'pass', 'admin');

    this.header = new Header().render();

    this.container = '';

    this.page = new Main().render();

  }

  public render() {

    const headerBlock = document.createElement('div');

    headerBlock.append(this.header);

    document.body.append(headerBlock);

    const mainBlock = document.createElement('div');

    document.body.append(this.page);

  //  document.body.append(this.page);

  }

}

export default App;

Листинг: main.ts

import Car from "../../data/entity/car";

import CarRepository from "../../data/repository/carRepository";

import { mainDraw } from '../draw';

import Info from "../info";

class Main {

  public container: HTMLElement;

  constructor() {

    this.container = document.createElement('main');

    const rep = new CarRepository('showroom', 'Car');

    const prom = rep.retrieve().then((res) => {

      for (let i = 0; i < res.length; i++) {

        Info.car.push(new Car(res[i].id, res[i].name, res[i].price, res[i].src));

      }

      this.container.innerHTML = mainDraw();

      document.body.append(this.container);

    });

  }

  public render() : HTMLElement {

    return this.container;

  }

}

export default Main;

Листинг: draw.ts

import Info from "./info";

export function profileDraw() {

  return `<div class="profile-wrapper">

            <div class="profile-block">

              <h3>Логин:</h3>

              <h3>Max</h3>

            </div>

            <div class="profile-block">

              <h3>Почта:</h3>

              <h3>${Info.currentUser.email}</h3>

            </div>

            <div class="profile-block">

              <h3>Пароль:</h3>

              <h3>${Info.currentUser.password}</h3>

            </div>

          </div>`;

}

export function loginDraw() {

  return `<form class="form">

  <h2 class="register\_title">Авторизация</h2>

  <div class="register\_info">

    <div class="register\_block">

      <input placeholder="Логин" class="register\_input" id="name">

    </div>

    <div class="register\_block">

      <input placeholder="Пароль" class="register\_input" id="lastname">

    </div>

  </div>

  <div class="register\_buttons">

    <button class="register\_register">Войти</button>

    <button class="register\_cancel">Отмена</button>

  </div>

</form>`;

}

export function registerDraw() {

  return `<form class="form">

    <h2 class="register\_title">Регистрация</h2>

    <div class="register\_info">

      <div class="register\_block">

        <input placeholder="Логин" class="register\_input" id="name">

      </div>

      <div class="register\_block">

        <input placeholder="Почта" class="register\_input" id="lastname">

      </div>

      <div class="register\_block">

        <input placeholder="Пароль" class="register\_input" id="email">

      </div>

    </div>

    <div class="register\_buttons">

      <button class="register\_register">Регистрация</button>

      <button class="register\_cancel">Отмена</button>

    </div>

  </form>`;

}

//#region Header

function headerUserDraw() {

  return `<h3>${Info.currentUser.email}</h3>

          <button>Выход</button>`;

}

export function headerDraw() {

  if (Info.currentUser) {

    return headerUserDraw();

  }

  return `<button>Регистация</button>

    <button>Логин</button>`;

}

//#endregion

function drawAdminCar(car) {

  return `<div class="car">

  <img src="${car.src}" width="200" height="150">

  <h3>${car.name}</h3>

  <h3>${car.price}$</h3>

  <div class="button-wrapper">

    <button>Подробнее</button>

    <button>Купить</button>

    <button>Изменить</button>

  </div>

</div>`;

}

function drawUserCar(car) {

  return `<div class="car">

            <img src="${car.src}" width="200" height="150">

            <h3>${car.name}</h3>

            <h3>${car.price}$</h3>

            <div class="button-wrapper">

              <button>Подробнее</button>

              <button>Купить</button>

            </div>

          </div>`;

}

function drawCar(car) {

  return `<div class="car">

            <img src="${car.src}" width="200" height="150">

            <h3>${car.name}</h3>

            <h3>${car.price}$</h3>

            <button>Подробнее</button>

          </div>`;

}

function drawCars() {

  let container = '';

  for (let i = 0; i < Info.car.length; i++) {

    if (Info.currentUser.role === 'user') {

      container += drawUserCar(Info.car[i]);

    } else if (Info.currentUser.role === 'admin') {

      container += drawAdminCar(Info.car[i]);

    } else {

      container += drawCar(Info.car[i]);

    }

  }

  return container;

}

export function mainDraw() {

  return `<div class="car-warpper">

            ${drawCars()}

          </div>`;

}