**Национальный исследовательский технологический университет «МИСИС»**

**Институт информационных технологий и компьютерных наук**

**Курсовая работа**

по курсу «Разработка клиент-северных приложений»

**Тема:**

«Автомастерская, марка авто, неисправность»

Выполнил: студент группы БИВТ-21-6

Шавлидзе Давид Бадриевич

Проверил: Рзазаде Ульви Азар оглы

г. Москва

2023 год

**Содержание**

Предметная область 3

Постановка задачи 4

Описание архитектуры 5

Описание структуры БД 6

Описание серверной части 13

Описание клиентской части 17

Заключение 19

Список литературы 20

# Предметная область

Клиент-серверное приложение для станции технического обслуживания (СТО) является важным инструментом для эффективного управления и автоматизации бизнес-процессов в области технического обслуживания автомобилей. СТО является местом, где проводятся различные виды ремонта, диагностика и техническое обслуживание автомобилей, а также осуществляется учет и управление всеми процессами, связанными с этими операциями.

Традиционно, СТО часто оперировали на основе бумажных документов, ручного ввода данных и организации информации в виде таблиц и файлов. Однако, с ростом сложности и объема работ, такой подход ограничивает эффективность работы и усложняет процессы учета, планирования и контроля.

Клиент-серверное приложение для СТО предлагает новый уровень автоматизации и интеграции всех аспектов работы станции технического обслуживания. Оно позволяет централизованно управлять всеми процессами, начиная от приема и регистрации заказов на ремонт, до отслеживания выполнения работ, учета запасных частей и контроля финансовых операций.

Такое приложение должно предоставлять серверную и клиентскую части, которые взаимодействуют друг с другом для обмена данных и выполнения операций. Серверная часть отвечает за хранение и обработку информации, а также предоставляет API для взаимодействия с клиентской частью. Клиентская часть предоставляет удобный пользовательский интерфейс для ввода данных, просмотра информации и управления процессами на станции технического обслуживания. Частично данные требования были воплощены в моей работе.

Клиент-серверное приложение для СТО позволяет значительно упростить и оптимизировать процессы работы, улучшить контроль над выполнением заказов, сократить время на поиск и обработку информации, а также повысить качество обслуживания клиентов. Это становится особенно важным в условиях растущей конкуренции и повышенных требований к качеству и эффективности работы СТО.

# Постановка задачи

Мною был выбран 16-й вариант темы для курсовой работы. Целью данного проекта является разработка клиент-серверного приложения для станции технического обслуживания (СТО), которое позволит оптимизировать и автоматизировать основные процессы, связанные с обслуживанием автомобилей.

В рамках постановки задачи были выделены следующие основные задачи, которые приложение должно решать:

1. Управление заказами на ремонт: Разработать функционал, позволяющий принимать, отслеживать и управлять заказами на ремонт автомобилей. Это включает возможность регистрации новых заказов, установку приоритетов, отслеживание статуса выполнения работ и генерацию отчетов о выполненных работах.
2. Планирование и учет ресурсов: Разработать механизм планирования ресурсов, таких как доступность рабочей силы, запасных частей и оборудования. Приложение должно предоставлять возможность эффективного планирования расписания работ, учета доступных ресурсов и оптимизации их использования.
3. Учет финансовых операций: Реализовать функционал для учета финансовых операций, связанных с ремонтом автомобилей. Это включает фиксацию затрат на запасные части, оказанные услуги и расчет общей стоимости ремонта для клиента.
4. Взаимодействие с клиентами: Создать механизм взаимодействия с клиентами, который позволит им получать информацию о состоянии заказов, расписании работ, стоимости и выполненных работах.
5. Безопасность и контроль данных: Обеспечить высокий уровень безопасности и контроля данных, чтобы предотвратить несанкционированный доступ к информации о заказах, клиентах и финансовых операциях.

Путем разработки и внедрения клиент-серверного приложения, решающего указанные задачи, я стремился повысить эффективность работы станции технического обслуживания, улучшить качество обслуживания клиентов, оптимизировать использование ресурсов и обеспечить надежность и безопасность данных.

# Описание архитектуры

Для реализации клиент-серверного приложения станции технического обслуживания (СТО) была выбрана архитектура, основанная на принципе разделения обязанностей между клиентской и серверной частями системы. Это позволяет достичь гибкости, масштабируемости и легкости сопровождения приложения.

Архитектура проекта включает следующие ключевые компоненты:

1. Клиентская часть (Frontend): Отвечает за взаимодействие с пользователем и предоставление графического интерфейса для работы с приложением. Реализована с использованием современных веб-технологий, таких как HTML, CSS и фреймворк Bootstrap. Клиентская часть обеспечивает удобный интерфейс для регистрации заказов, просмотра статуса работ, взаимодействия с клиентами и генерации отчетов.
2. Серверная часть (Backend): Обрабатывает запросы от клиентской части, выполняет бизнес-логику и взаимодействует с базой данных. Реализована с использованием фреймворка NestJS, который предоставляет мощные инструменты для разработки серверных приложений. Серверная часть обеспечивает обработку заказов, управление ресурсами, учет финансовых операций и взаимодействие с клиентами.
3. База данных: Используется для хранения информации о заказах, клиентах, ресурсах и финансовых операциях. Для реализации базы данных выбрана СУБД PostgreSQL, которая обеспечивает надежность, масштабируемость и возможности для эффективного управления данными. Взаимодействие с базой данных осуществляется с помощью ORM (Object-Relational Mapping) инструмента TypeORM, который облегчает работу с базой данных и предоставляет высокий уровень абстракции.

Архитектура проекта обеспечивает модульность, легкость расширения и обновления, а также возможность работы с различными типами клиентских устройств (например, десктопные компьютеры, планшеты, мобильные телефоны). Такая структура позволяет создать гибкую и надежную систему управления станцией технического обслуживания, способную эффективно решать задачи по обслуживанию автомобилей и улучшать взаимодействие с клиентами.

# Описание структуры БД

В базе данных для клиент-серверного приложения станции технического обслуживания (СТО) присутствуют следующие таблицы:

1. Таблица "cars":
   * id: Автоматически генерируемый идентификатор автомобиля.
   * make: Марка автомобиля.
   * model: Модель автомобиля.
   * year: Год выпуска автомобиля.
   * vin: VIN-номер автомобиля.
   * mileage: Пробег автомобиля.
   * status: Статус автомобиля.
   * repairs: Связь с таблицей "repairs" через отношение OneToMany.
2. Таблица "customers":
   * id: Автоматически генерируемый идентификатор клиента.
   * firstName: Имя клиента.
   * lastName: Фамилия клиента.
   * phoneNumber: Номер телефона клиента.
   * email: Адрес электронной почты клиента.
3. Таблица "repairs":
   * id: Автоматически генерируемый идентификатор ремонта.
   * car: Связь с таблицей "cars" через отношение ManyToOne.
   * clientId: Идентификатор клиента, связанный с ремонтом.
   * startDate: Дата начала ремонта.
   * endDate: Дата окончания ремонта.
   * description: Описание ремонта.
   * works: Необходимые работы по автомобилю.
   * cost: Стоимость ремонта.
4. Таблица "warehouse":
   * id: Автоматически генерируемый идентификатор товара на складе.
   * name: Наименование товара на складе.
   * count: Количество товара на складе.

Связи между таблицами:

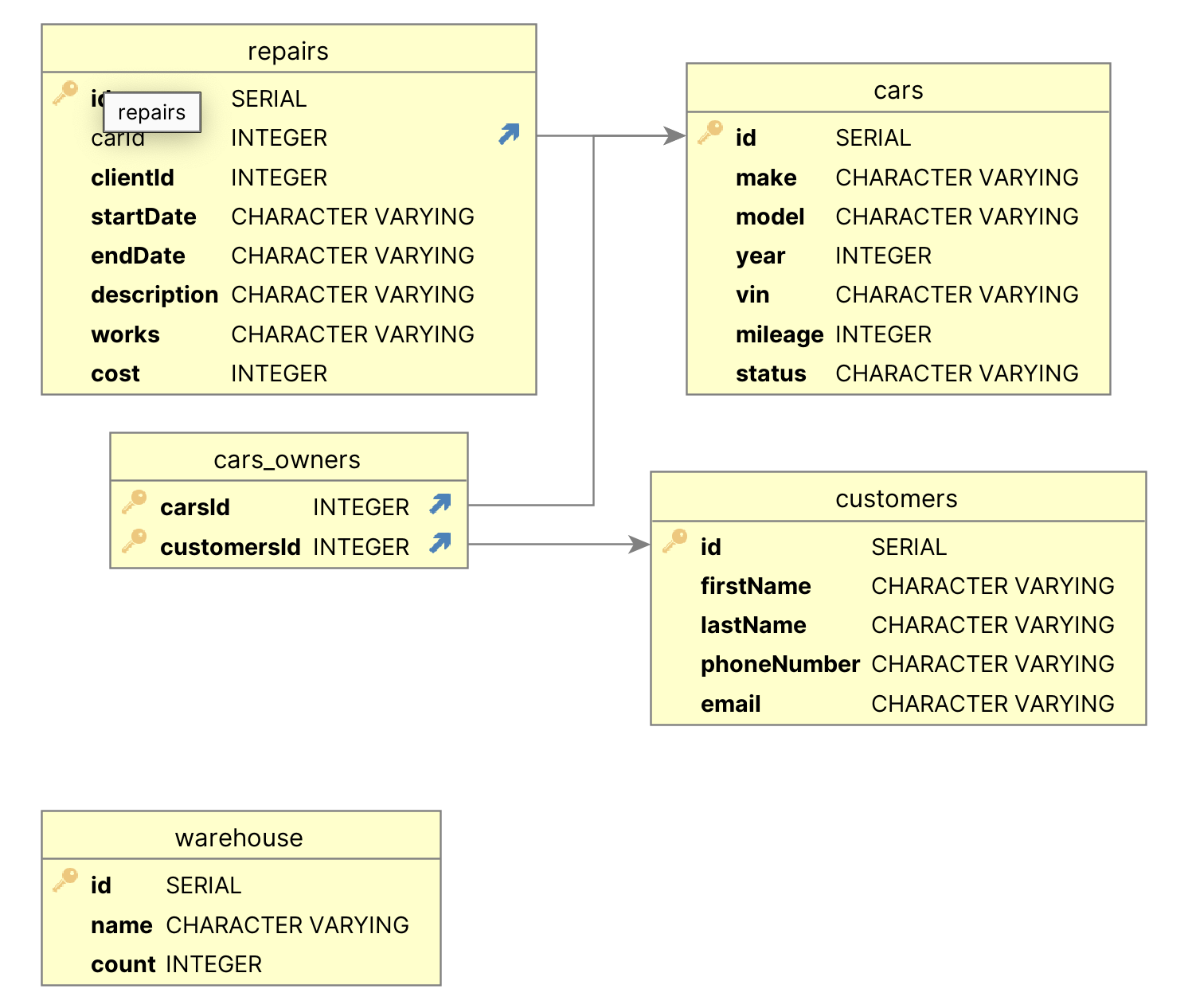


Схема 1. Графическое представление базы данных СТО

* Таблица "cars" и "customers" связаны отношением ManyToMany через таблицу "cars\_owners", которая хранит связи между автомобилями и их владельцами.
* Таблица "repairs" имеет связь ManyToOne с таблицей "cars", обозначающую, что каждый ремонт относится к определенному автомобилю.

Такая структура БД позволяет эффективно хранить и управлять данными о клиентах, автомобилях, ремонтах и товарах на складе в контексте станции технического обслуживания.

*Представления*:

Представление для получения списка автомобилей и их владельцев:

CREATE VIEW car\_owners\_view AS

SELECT c.id AS car\_id, c.make, c.model, c.year, c.vin, c.mileage, c.status,

cu.id AS customer\_id, cu.firstName, cu.lastName, cu.phoneNumber, cu.email

FROM cars c

JOIN cars\_owners co ON c.id = co.carId

JOIN customers cu ON co.customerId = cu.id;

Представление для получения списка ремонтов и информации об автомобиле:

CREATE VIEW repair\_car\_view AS

SELECT r.id AS repair\_id, r.startDate, r.endDate, r.description, r.works, r.cost,

c.id AS car\_id, c.make, c.model, c.year, c.vin, c.mileage, c.status

FROM repairs r

JOIN cars c ON r.carId = c.id;

Представление для получения списка товаров на складе:

CREATE VIEW warehouse\_items\_view AS

SELECT id, name, count

FROM warehouse;

*Функции*:

Функция для расчета общей стоимости ремонтов для заданного автомобиля:

CREATE FUNCTION calculate\_total\_cost(carId INTEGER) RETURNS NUMERIC AS $$

DECLARE

total\_cost NUMERIC := 0;

BEGIN

SELECT SUM(cost) INTO total\_cost

FROM repairs

WHERE carId = $1;

RETURN total\_cost;

END;

Функция для получения списка клиентов, владеющих определенным автомобилем:

CREATE FUNCTION get\_car\_owners(carId INTEGER) RETURNS SETOF customers AS $$

BEGIN

RETURN QUERY

SELECT cu.\*

FROM customers cu

JOIN cars\_owners co ON cu.id = co.customerId

WHERE co.carId = $1;

END;

Функция для получения списка автомобилей, владеемых определенным клиентом:

CREATE FUNCTION get\_customer\_cars(customerId INTEGER) RETURNS SETOF cars AS $$

BEGIN

RETURN QUERY

SELECT c.\*

FROM cars c

JOIN cars\_owners co ON c.id = co.carId

WHERE co.customerId = $1;

END;

*Хранимые процедуры:*

Процедура для добавления нового автомобиля и его владельцев:

CREATE PROCEDURE add\_car\_with\_owners(make VARCHAR, model VARCHAR, year INTEGER, vin VARCHAR, mileage INTEGER, status VARCHAR, owners INTEGER[]) AS $$

DECLARE

carId INTEGER;

BEGIN

INSERT INTO cars (make, model, year, vin, mileage, status)

VALUES (make, model, year, vin, mileage, status)

RETURNING id INTO carId;

FOREACH owner IN ARRAY owners

LOOP

INSERT INTO cars\_owners (carId, customerId)

VALUES (carId, owner);

END LOOP;

END;

Процедура для добавления нового ремонта:

CREATE PROCEDURE add\_repair(carId INTEGER, clientId INTEGER, startDate DATE, endDate DATE, description VARCHAR, works VARCHAR, cost NUMERIC) AS $$

BEGIN

INSERT INTO repairs (carId, clientId, startDate, endDate, description, works, cost)

VALUES (carId, clientId, startDate, endDate, description, works, cost);

END;

Процедура для обновления информации об автомобиле:

CREATE PROCEDURE update\_car(carId INTEGER, make VARCHAR, model VARCHAR, year INTEGER, vin VARCHAR, mileage INTEGER, status VARCHAR) AS $$

BEGIN

UPDATE cars

SET make = make, model = model, year = year, vin = vin, mileage = mileage, status = status

WHERE id = carId;

END;

*Триггеры*:

Триггер для автоматического обновления общей стоимости ремонтов при добавлении или изменении ремонта:

CREATE OR REPLACE FUNCTION update\_total\_cost()

RETURNS TRIGGER AS $$

BEGIN

UPDATE cars

SET totalCost = calculate\_total\_cost(NEW.carId)

WHERE id = NEW.carId;

RETURN NEW;

END;

CREATE TRIGGER update\_total\_cost\_trigger

AFTER INSERT OR UPDATE ON repairs

FOR EACH ROW

EXECUTE FUNCTION update\_total\_cost();

Триггер для автоматического обновления количества автомобилей у клиента при добавлении или удалении связи "автомобиль-владелец":

CREATE OR REPLACE FUNCTION update\_car\_count()

RETURNS TRIGGER AS $$

BEGIN

UPDATE customers

SET carCount = (SELECT COUNT(\*) FROM cars\_owners WHERE customerId = NEW.customerId)

WHERE id = NEW.customerId;

RETURN NEW;

END;

CREATE TRIGGER update\_car\_count\_trigger

AFTER INSERT OR DELETE ON cars\_owners

FOR EACH ROW

EXECUTE FUNCTION update\_car\_count();

Триггер для автоматического обновления статуса автомобиля при изменении его пробега:

CREATE OR REPLACE FUNCTION update\_car\_status()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.mileage >= 100000 THEN

NEW.status = 'Old';

ELSE

NEW.status = 'New';

END IF;

RETURN NEW;

END;

CREATE TRIGGER update\_car\_status\_trigger

BEFORE UPDATE ON cars

FOR EACH ROW

EXECUTE FUNCTION update\_car\_status();

# Описание серверной части

*Описание основных сущностей:*

* Car: Представляет сущность автомобиля. Содержит информацию о марке (make), модели (model), годе выпуска (year), VIN-коде (vin), пробеге (mileage), статусе (status) и связь с ремонтами (repairs). Связан с сущностью Customer через Many-to-Many отношение по владельцам автомобиля.
* Customer: Представляет сущность клиента. Содержит информацию о имени (firstName), фамилии (lastName), номере телефона (phoneNumber) и электронной почте (email). Связан с сущностью Car через Many-to-Many отношение по владельцам автомобиля.
* Repair: Представляет сущность ремонта. Содержит информацию о связи с автомобилем (car), идентификаторе клиента (clientId), датах начала (startDate) и окончания (endDate) ремонта, описании (description), проведенных работах (works) и стоимости (cost). Связан с сущностью Car через Many-to-One отношение по автомобилю.
* Warehouse: Представляет сущность склада. Содержит информацию о наименовании (name) и количестве (count) на складе.

*Описание контроллеров:*

* CarsController: Обрабатывает запросы, связанные с автомобилями. Взаимодействует с сервисом CarsService для выполнения операций с данными.
* CustomersController: Обрабатывает запросы, связанные с клиентами. Взаимодействует с сервисом CustomersService для выполнения операций с данными.
* RepairsController: Обрабатывает запросы, связанные с ремонтами. Взаимодействует с сервисом RepairsService для выполнения операций с данными.

*Описание моделей:*

* Car: Модель автомобиля. Содержит свойства, соответствующие полям таблицы cars.
* Customer: Модель клиента. Содержит свойства, соответствующие полям таблицы customers.
* Repair: Модель ремонта. Содержит свойства, соответствующие полям таблицы repairs.
* Warehouse: Модель склада. Содержит свойства, соответствующие полям таблицы warehouse.

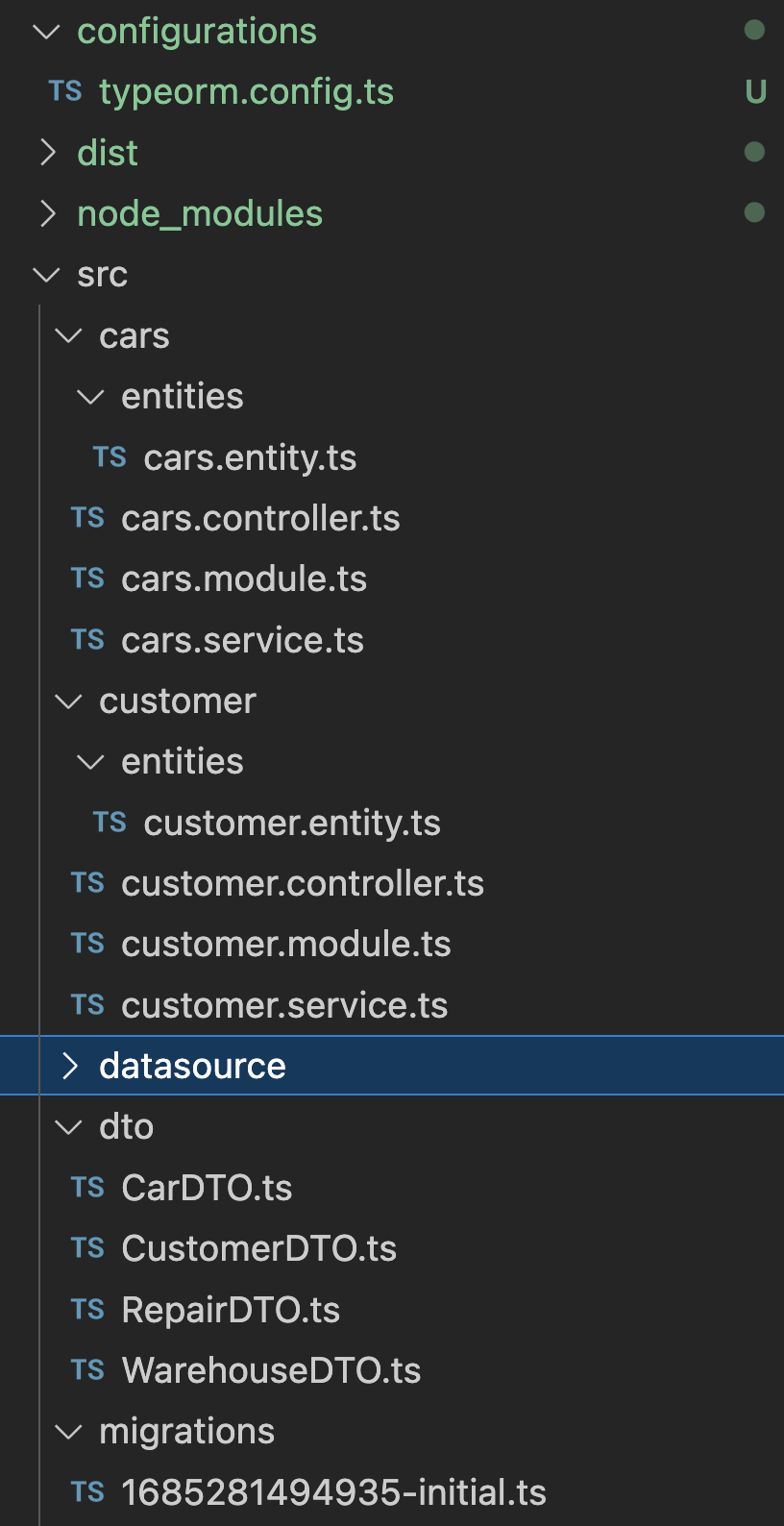


Рисунок 2.2. Архитектура сервера

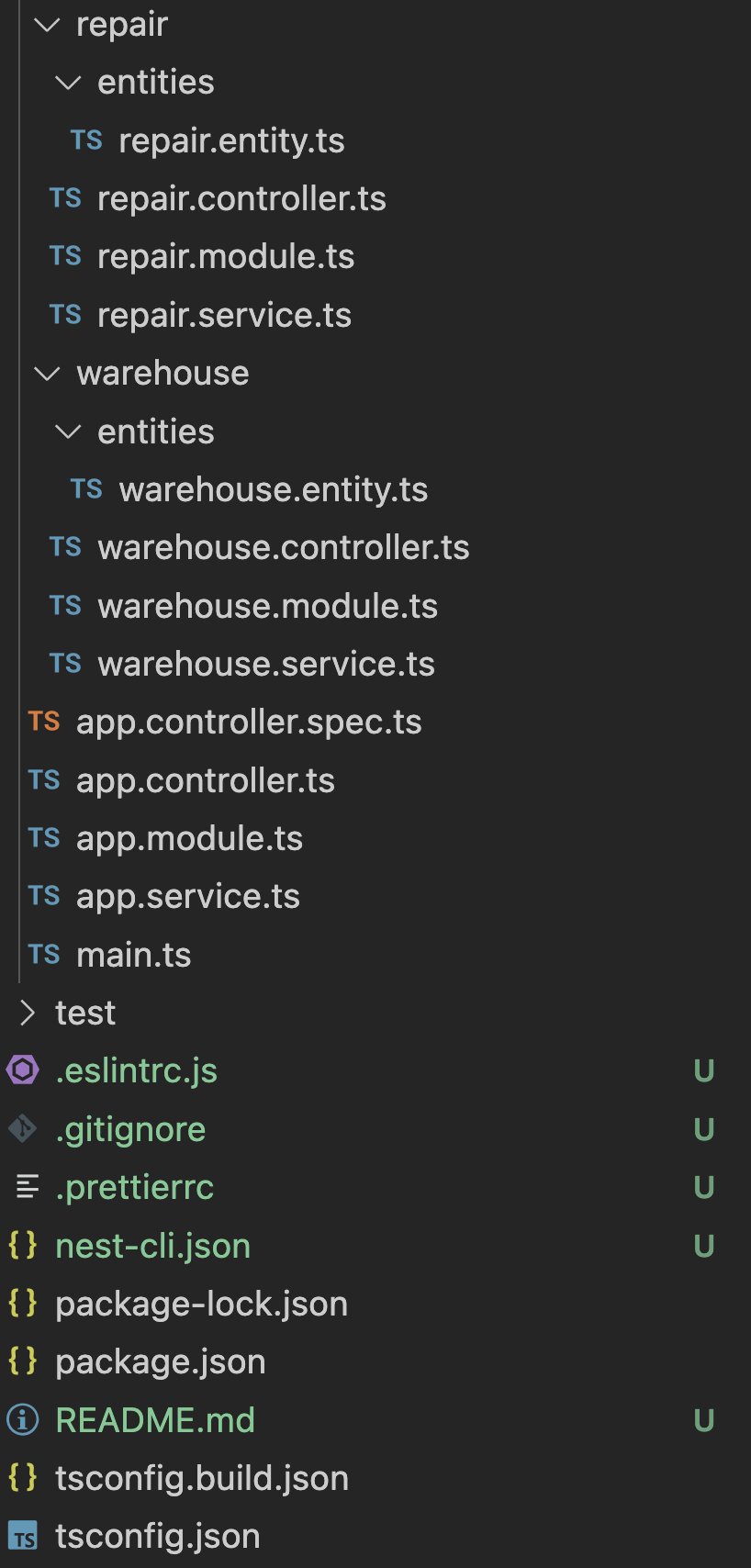


Рисунок 2.1. Архитектура сервера

Данное описание представляет основные компоненты серверной части клиент-серверного приложения для станции технического обслуживания (СТО) на основе предоставленных сущностей. Реализация контроллеров и сервисов, а также логика обработки запросов и взаимодействия с базой данных, требует более детального анализа требований и функциональности вашего приложения.

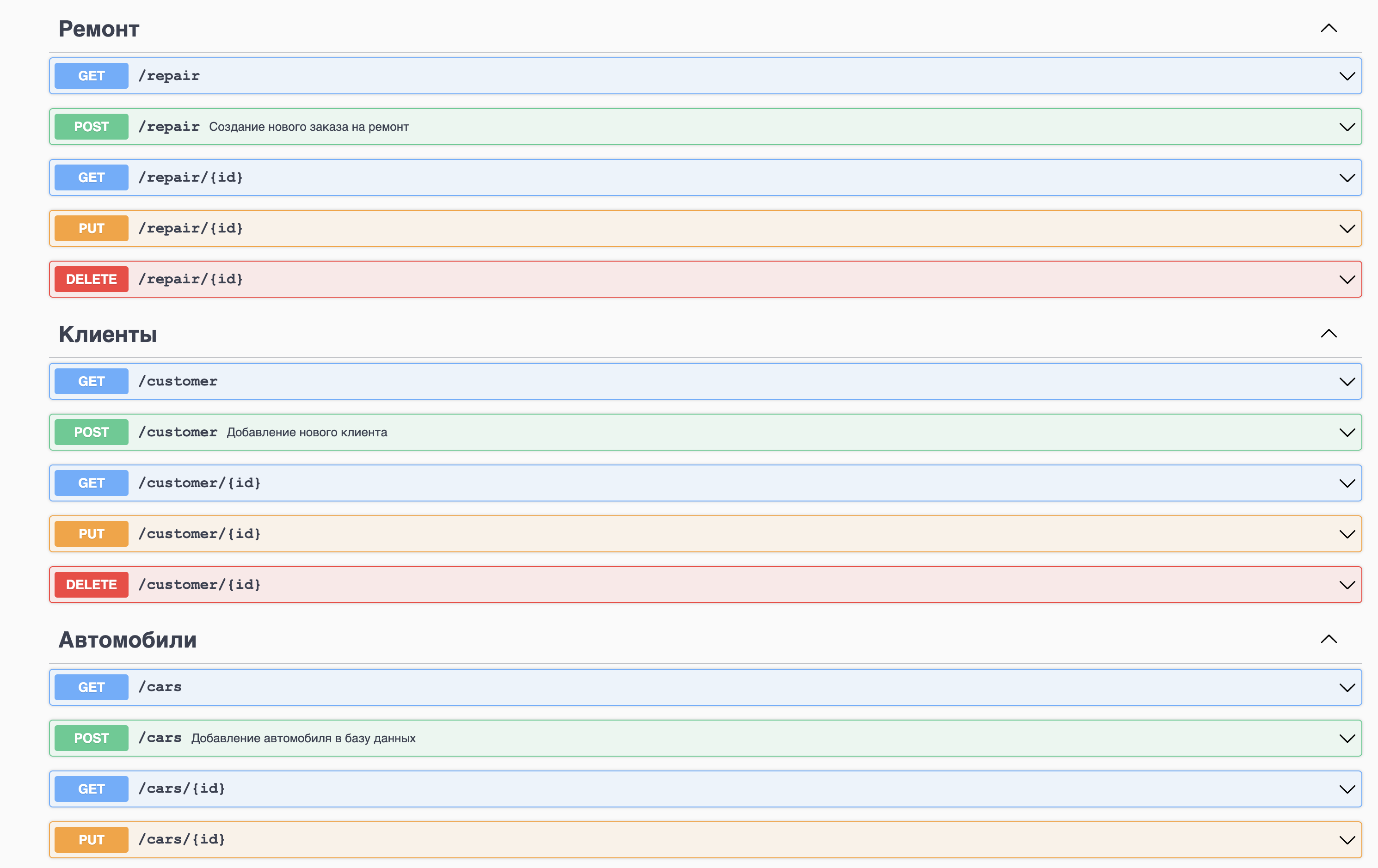


Рисунок 1. Пример использования Swagger для тестирования моделей проекта

Swagger - инструмент для создания, документирования и тестирования API. Он позволяет описать структуру и функциональность API с использованием спецификации OpenAPI (ранее известной как Swagger Specification). Swagger предоставляет возможность автоматически генерировать интерактивную документацию API, которая может быть использована разработчиками для понимания и использования API.



Рисунок 2. Пример выполнения POST запроса

Swagger поддерживает различные языки программирования и фреймворки. Для каждого API, описанного в спецификации OpenAPI, Swagger генерирует интерактивную документацию в формате Swagger UI. В Swagger UI можно просмотреть доступные маршруты, параметры запросов, модели данных, а также выполнять тестирование API прямо из интерфейса.

Для работы со сущностями, представленными в коде, можно использовать Swagger для описания соответствующих маршрутов и моделей данных.

# Описание клиентской части

Клиентская часть сайта СТО была разработана с использованием HTML, CSS и Bootstrap. Однако, учитывая изменение программы курса, разработка клиентской части сайта была прервана до завершения.

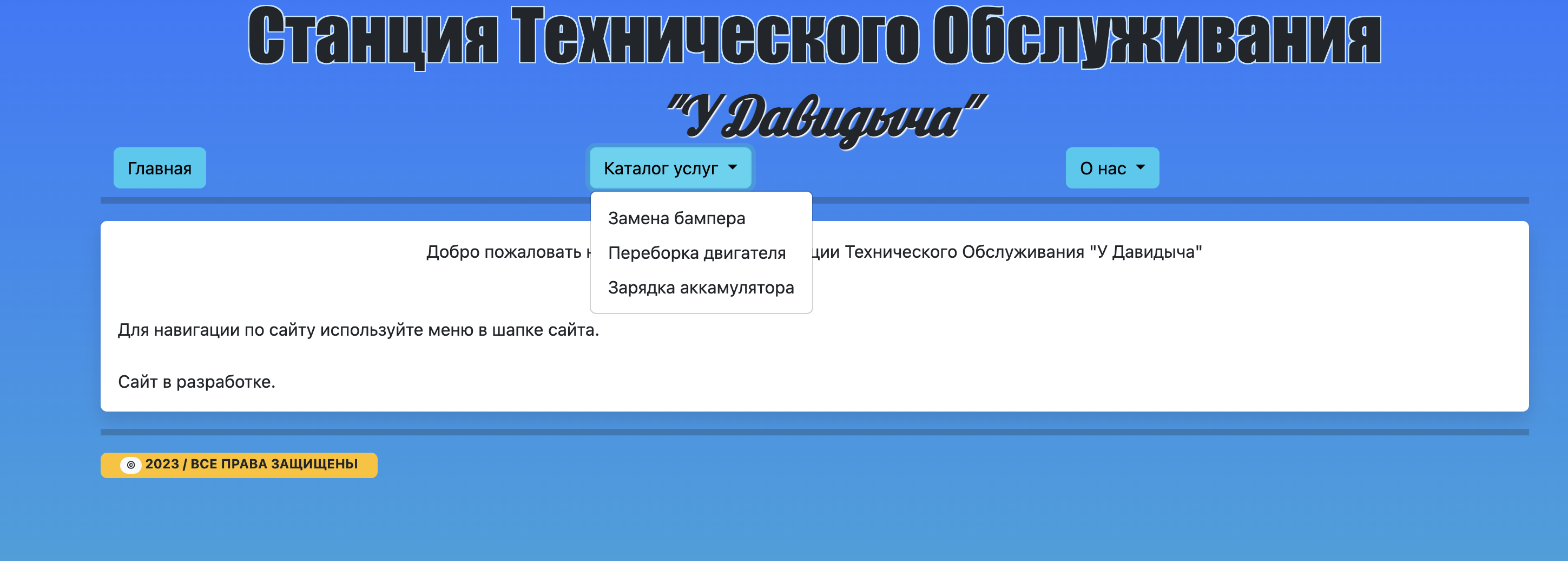


Рисунок 3. Главная страница сайта СТО с открытым каталогом услуг

Давайте рассмотрим возможности сайта, представленного в предоставленном коде, и как пользователь может с ним взаимодействовать:

1. Навигация по сайту: В шапке сайта присутствуют несколько кнопок-ссылок, которые позволяют пользователю перемещаться по различным разделам сайта. Например, кнопка "Главная" перенаправляет пользователя на главную страницу сайта.
2. Каталог услуг: Путем нажатия на кнопку "Каталог услуг" открывается выпадающее меню, в котором представлены различные услуги, предлагаемые СТО. Пользователь может выбрать нужную услугу, например, "Замена бампера" или "Переборка двигателя", и перейти на соответствующую страницу с подробной информацией о выбранной услуге.
3. О нас: При нажатии на кнопку "О нас" также открывается выпадающее меню, где пользователь может найти различную информацию о СТО. Например, разделы "Контакты", "Администрация" и "Поддержка" могут содержать соответствующие страницы с контактными данными, информацией о персонале СТО и возможностях получения поддержки.
4. Описание сайта: Под меню расположен блок с текстом, который представляет собой приветствие и описание СТО. Текст приглашает пользователя ознакомиться с сайтом и использовать предоставленное меню для навигации. Также указано, что сайт находится в разработке.
5. Копирайт: Внизу страницы присутствует копирайт, указывающий год и защиту прав на содержимое сайта.

Хотя сайт был остановлен в разработке, представленный код демонстрирует базовые элементы, такие как шапка сайта с навигацией, выпадающие меню и блок с описанием. Комбинация HTML, CSS и Bootstrap позволяет создать эстетически приятный и удобный интерфейс для пользователей.

# Заключение

Проблема, которая рассматривалась, заключалась в неэффективной организации и управлении бизнес-процессами на станции технического обслуживания автомобилей. Традиционные методы, такие как бумажные документы и ручной ввод данных, ограничивали возможности учета, планирования и контроля, а также затрудняли процессы автоматизации и интеграции.

Для решения этой проблемы было разработано клиент-серверное приложение для станции технического обслуживания. Оно предоставляет централизованное управление всеми процессами на станции, начиная от приема заказов на ремонт и диагностику, до учета запасных частей, контроля выполнения работ и финансовых операций.

Инструменты, использованные для разработки приложения, включают язык SQL для работы с базой данных, языки программирования для серверной и клиентской частей, такие как JavaScript или TypeScript, а также фреймворки и библиотеки, такие как Node.js и NestJS, для упрощения разработки и создания пользовательского интерфейса.

Результаты, полученные в результате разработки и внедрения клиент-серверного приложения для СТО, включают улучшенную организацию бизнес-процессов, более эффективное управление работами и ресурсами, повышение качества обслуживания клиентов, сокращение времени на обработку информации, улучшенный контроль и анализ данных, а также повышение общей производительности и эффективности работы станции технического обслуживания.

# Список литературы

1. Смирнов И.А. "TypeScript в действии: Построение клиент-серверных приложений". Санкт-Петербург: Питер Пресс, 2020.
2. Карпов А.А. "Разработка клиент-серверных приложений на языке TypeScript". Москва: ДМК Пресс, 2019.
3. Корнелюк А. "PostgreSQL. Проектирование и разработка баз данных". Москва: ДМК Пресс, 2019.
4. Чакрыгин М. "Git. Система контроля версий". Москва: ДМК Пресс, 2019.
5. Кузнецов В. "NestJS. Разработка масштабируемых веб-приложений на Node.js". Москва: ДМК Пресс, 2020.
6. Петров Д. "TypeORM. Проектирование и управление базами данных на TypeScript". Санкт-Петербург: Питер Пресс, 2021.
7. Баранов М., Поляков В. "Node.js. Полное руководство по созданию веб-приложений". Санкт-Петербург: Питер Пресс, 2020.
8. Макфарланд А., Бутстрап Т. "Bootstrap 4. Руководство разработчика". Москва: ДМК Пресс, 2018.
9. Даков В. "HTML и CSS. Разработка и дизайн веб-сайтов". Москва: ДМК Пресс, 2019.с.
10. ReactJS. Официальная документация - https://reactjs.org/docs/getting-started.html
11. Git репазиторий. [github.com/DaveShavl](http://github.com/DaveShavl)