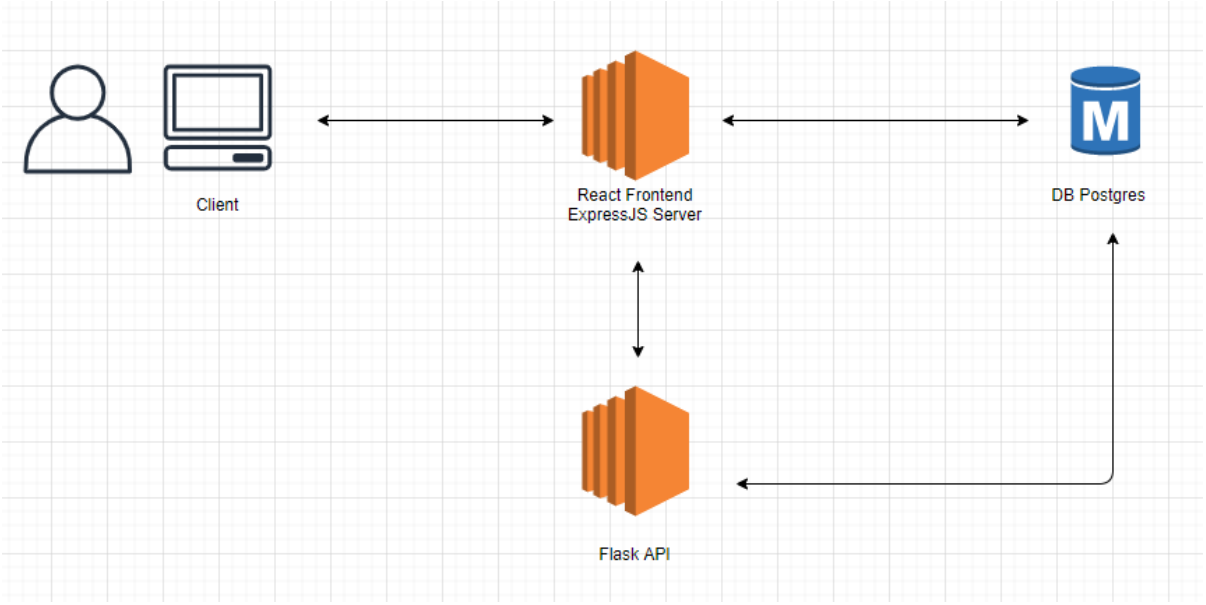
Technical Documentation

AutoSense

Contents

1) IT Landscape

This documentation provides an overview of the IT landscape for the backend system, including the infrastructure setup using Amazon Web Services (AWS) and the technology stack employed.



Infrastructure

The backend system is hosted on the AWS cloud platform and utilizes the following services:

Amazon EC2 (Elastic Compute Cloud)

- Instance Type: t3.micro
- Description: EC2 provides virtual servers in the cloud, allowing for the deployment and management of applications.

Amazon RDS (Relational Database Service)

- Database Engine: PostgreSQL
- Description: RDS is a managed database service that simplifies database administration tasks. In this case, it hosts a PostgreSQL database to store the backend application's data.

Technology Stack

The backend system is built using the following technologies:

Database: Postgres

Backend: ExpressJS, NodeJS, Sequelize, AWS SDK

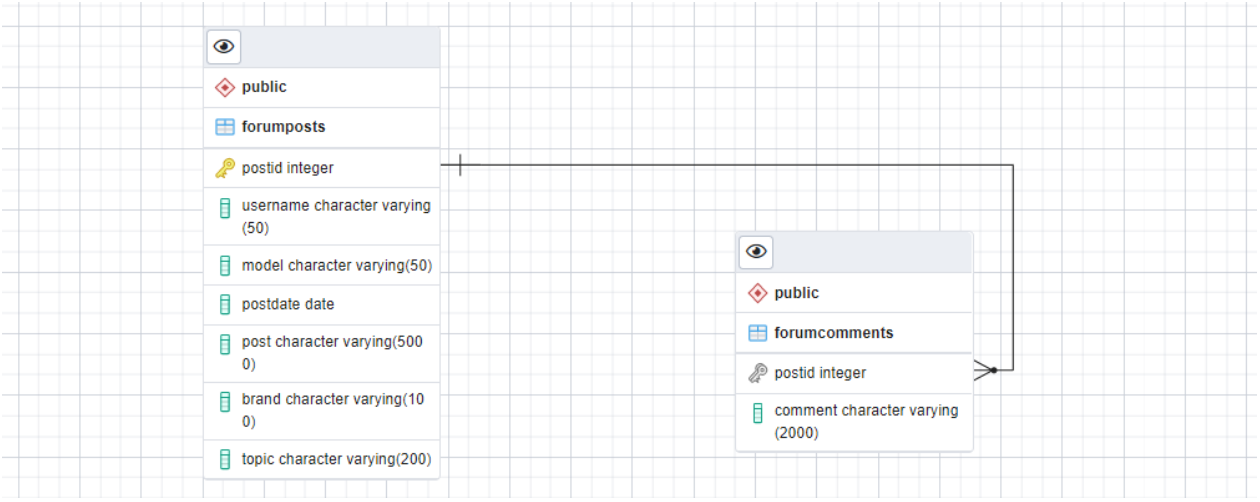
Frontend: ReactJS

Conclusion

This IT landscape documentation has provided an overview of the infrastructure setup and technology stack used in the backend system. The system is hosted on an EC2 t3.micro instance, utilizing AWS RDS with the latest version of PostgreSQL as the database. The backend application is developed using Node.js and Express.js, with additional dependencies such as AWS SDK, Moment, and Sequelize. This combination of technologies ensures a reliable, scalable, and efficient backend solution for the application.

2) Database

The database contains two tables as below:



The *forumpost* table entails information about all posts within the Online Car forum, whereas the *forumcomments* table contains all comments associated to the posts. The columns are described as below:

Forumpost:

- Postid: primary key
- Username: name of the user who created the post
- Model: car model of the post
- Postdate: initial date of post
- Post: the content of the post
- Brand: the car brand

- Topic: associated topic of the post

Forumcomments

- Postid: foreign key, the association to the related post
- Comment: the comment in text format

3) Backend

This technical documentation provides an overview of the backend system built with Node.js and Express.js. The backend serves as an API to handle various operations related to car malfunctions and information retrieval. It utilizes additional Node modules such as AWS SDK, Moment, and Sequelize as an ORM (Object-Relational Mapping) for interacting with the database.

Dependencies

Node.js

- Version: 20.4.0
- Description: Node.js is an open-source JavaScript runtime environment that allows for server-side execution of JavaScript code. It provides a scalable and efficient platform for building network applications.

Express.js

- Version: 4.18.2
- Description: Express.js is a web application framework for Node.js that simplifies the development of server-side applications. It provides a robust set of features for building APIs and handling HTTP requests.

AWS SDK

- Version: 2.1095.0
- Description: The AWS SDK for JavaScript enables interaction with various AWS services, such as Amazon S3, Amazon DynamoDB, and more. In this backend system, it is utilized to interface with AWS services, as per the requirements.

Moment

- Version: 2.29.4
- Description: Moment.js is a JavaScript library that simplifies date and time manipulation, parsing, and formatting. It is employed in the backend system to handle date-related operations efficiently.

Sequelize

- Version: 6.6.2
- Description: Sequelize is a powerful Object-Relational Mapping (ORM) tool for Node.js. It provides an abstraction layer for database operations, allowing developers to interact with databases using JavaScript objects and methods. In this backend system, it is used as the ORM to interact with the PostgreSQL database hosted on AWS RDS.

All the dependencies are listed in the package.json file, and running npm install will automatically install the required versions.

Prerequisites

Before setting up and running the backend, ensure that the following prerequisites are met:

- Node.js and npm (Node Package Manager) are installed on the system.
- A compatible database is available and properly configured.
- AWS SDK credentials are set up (if AWS services are utilized).

Setup

To set up the backend system, follow these steps:

1. Clone the repository from the project source.
2. Install the required dependencies by running the following command in the project directory: *npm install*
3. Configure the database connection settings in the config/db.config.js file.
4. Configure the allowed origins for CORS (if required) in the config/general.config.js file.

API Endpoints

The backend API provides several endpoints that can be accessed using the POST method. Each endpoint serves a specific purpose and requires certain parameters. The details of each endpoint are described below:

Endpoint	Method	Description	Parameters
/getTopics	POST	Returns all possible topics for car malfunctions.	None
/getBrands	POST	Returns an array of all available car brands.	None
/getModels	POST	: Returns all available car models for a given brand.	brand (string): The brand of the car for which models are requested.
/getCountProblems	POST	Returns the number of occurrences for all topics related to a specified brand and model.	brand (string): The brand of the car for which problem counts are requested.

			model (string): The model of the car for which problem counts are requested.
/getCountProblems	POST	Returns the number of occurrences for all topics related to a specified brand and model.	brand (string): The brand of the car for which problem counts are requested. model (string): The model of the car for which problem counts are requested.
/ getCountProblemsYear	POST	Returns all occurrences per year for a given topic, brand, and model. Makes predictions by using the ML API	topic (string): The topic for which occurrences per Quarter among Years are requested. brand (string): The brand of the car for which occurrences are requested. model (string): The model of the car for which occurrences are requested.

Conclusion

This technical documentation has provided an overview of the backend system built with Node.js and Express.js. It has described the API endpoints, their functionalities, and the required parameters. Additionally, it has outlined the setup process, database configuration, and the necessary dependencies. This information should assist in understanding and utilizing the backend system effectively.

4) Frontend Documentation

This documentation provides an overview of the frontend implementation using React.js. It describes the structure of the landing page and the two main components: the Quantity Estimation site and the Chatbot.

The frontend is designed responsive and can be installed as a progressive web application an all devices with all operating systems.

Th System can be accessed via: http://44.201.230.15/

Technology Stack

The frontend is built using the following technologies:

React.js

- Version: 18.2.0

- Description: React.js is a popular JavaScript library for building user interfaces. It enables the development of interactive and reusable components, making it efficient and scalable for frontend development.

Bootstrap

- Version: 2.8.0
- Description: Bootstrap is a widely-used CSS framework that provides pre-built components and styles for creating responsive web applications. It simplifies the process of designing and styling the frontend, ensuring a consistent and visually appealing user interface.

jQuery

- Version: 3.7.0
- Description: jQuery is a fast, small, and feature-rich JavaScript library. It provides a concise and simplified way to manipulate HTML documents, handle events, create animations, and perform various other tasks. Although React.js handles most of the dynamic behavior, jQuery can be useful for certain DOM manipulations or interactions.

Axios

- Version: 1.4.0
- Description: Axios is a widely-used JavaScript library for making HTTP requests from the browser. It simplifies the process of sending API requests and handling responses in frontend applications.

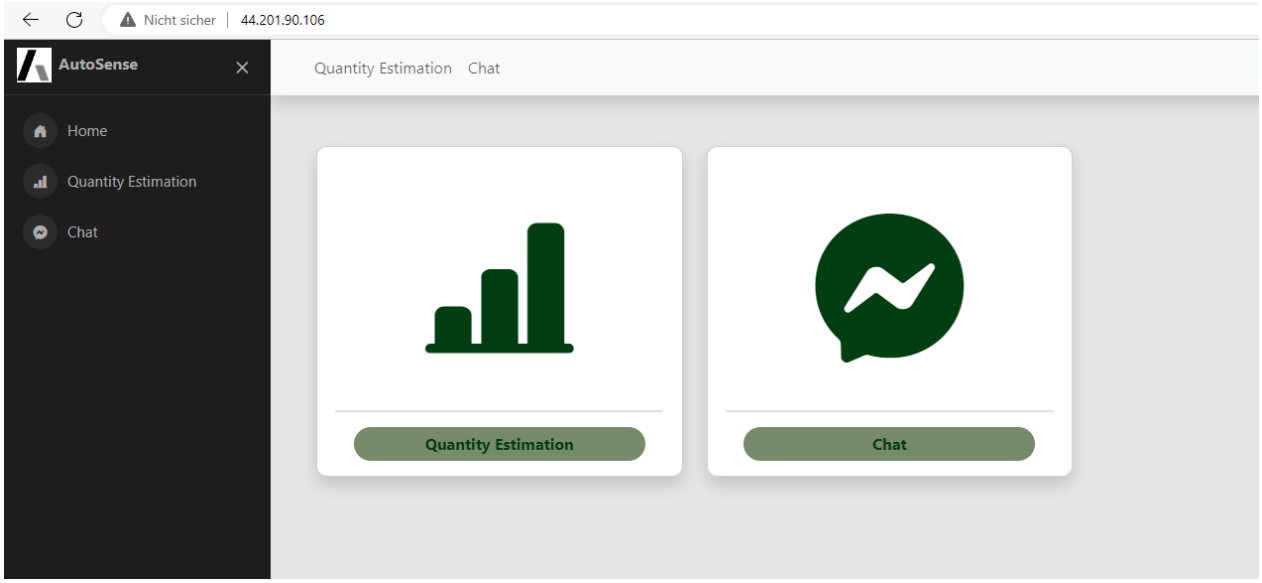
Google Charts

- Version: 4.0.0
- Description: Google Charts is a widely-used JavaScript library to display data in graphs.

Pages

Landing Page

The landing page serves as the entry point to the application. It consists of a set of cards that lead to two different sections: Quantity Estimation and Chatbot.



Quantity Estimation

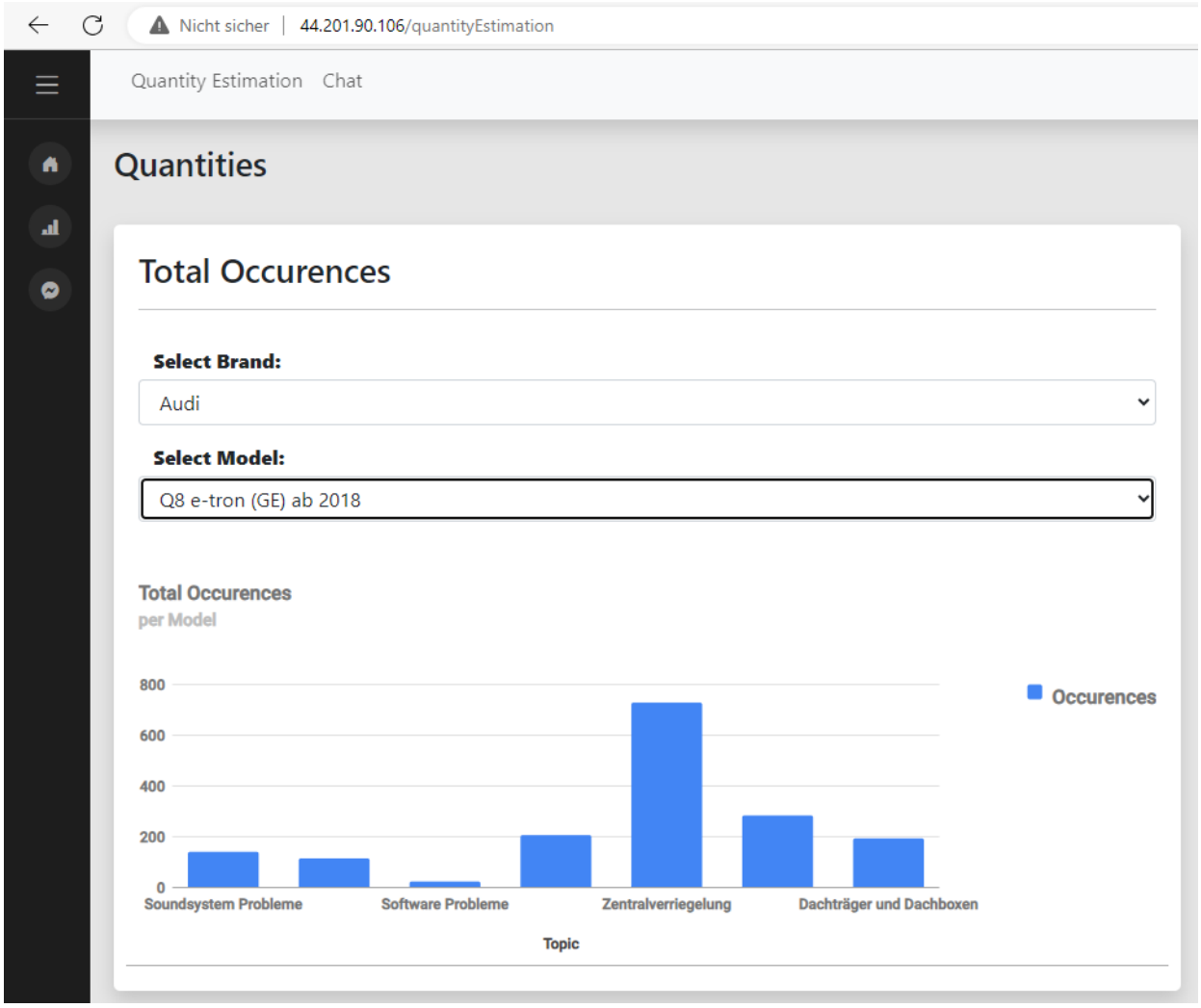
The Quantity Estimation section includes two pages that display graphs based on user selections.

Graph 1: Bar Chart, Total Occurences by Brand and Model

Description: This page allows the user to select a car brand and model from dropdown fields. Upon selection, a bar chart is generated, displaying the total occurrences of each topic on the x-axis and the number of occurrences on the y-axis since system initiation.

Components used:

- Dropdown fields for car brand and model selection.
- Bar chart component for visualizing the data.

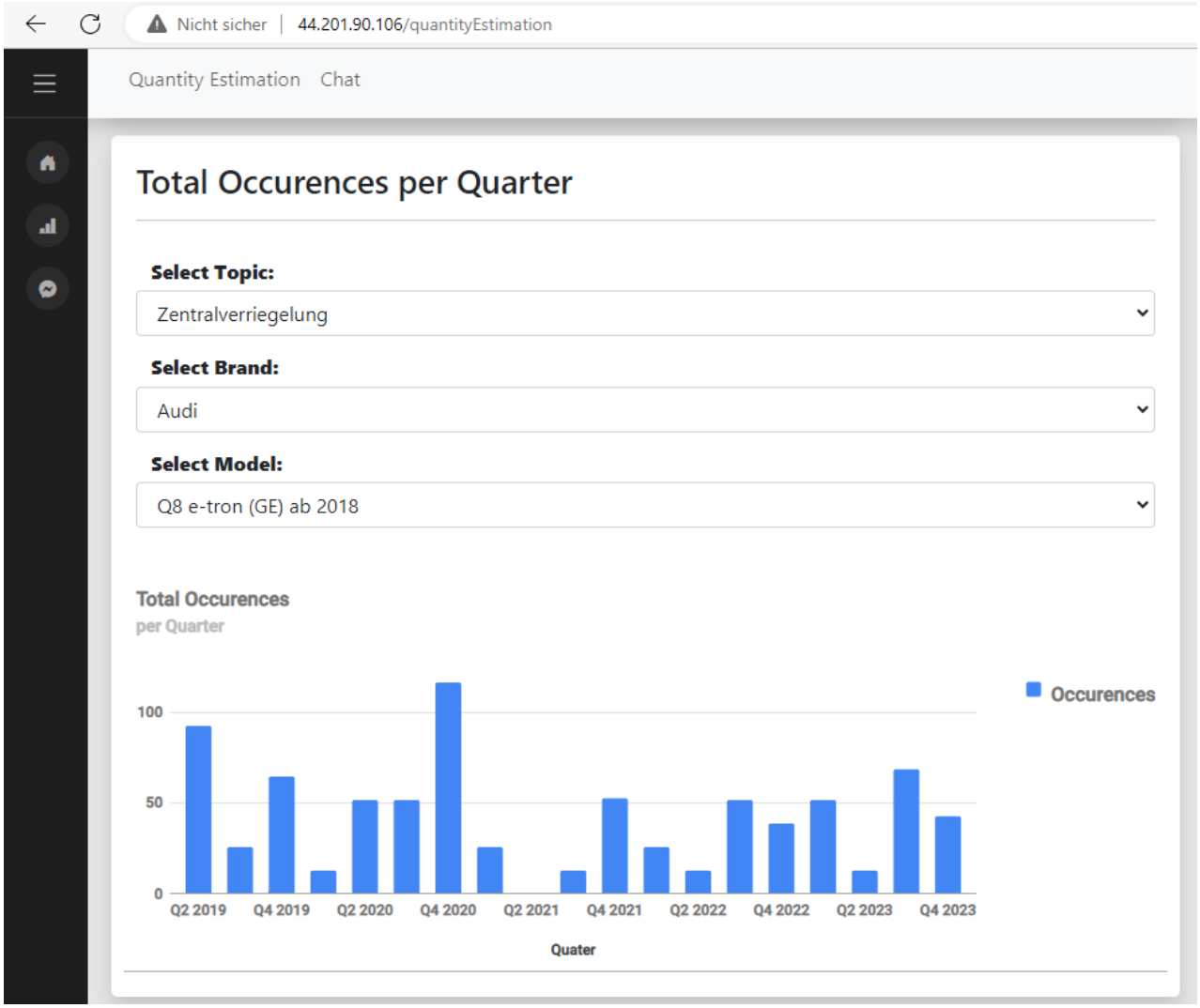


Graph 2: Bar Chart, Total Occurrences per Quarter by Topic, Brand, and Model

Description: This page enables the user to select a topic, car brand, and model from dropdown fields. After making the selections, a bar chart is rendered, showing the occurrences per quarterfor the chosen topic, brand, and model.

Components used:

- Dropdown fields for topic, car brand, and model selection.
- Bar chart component for visualizing the data.



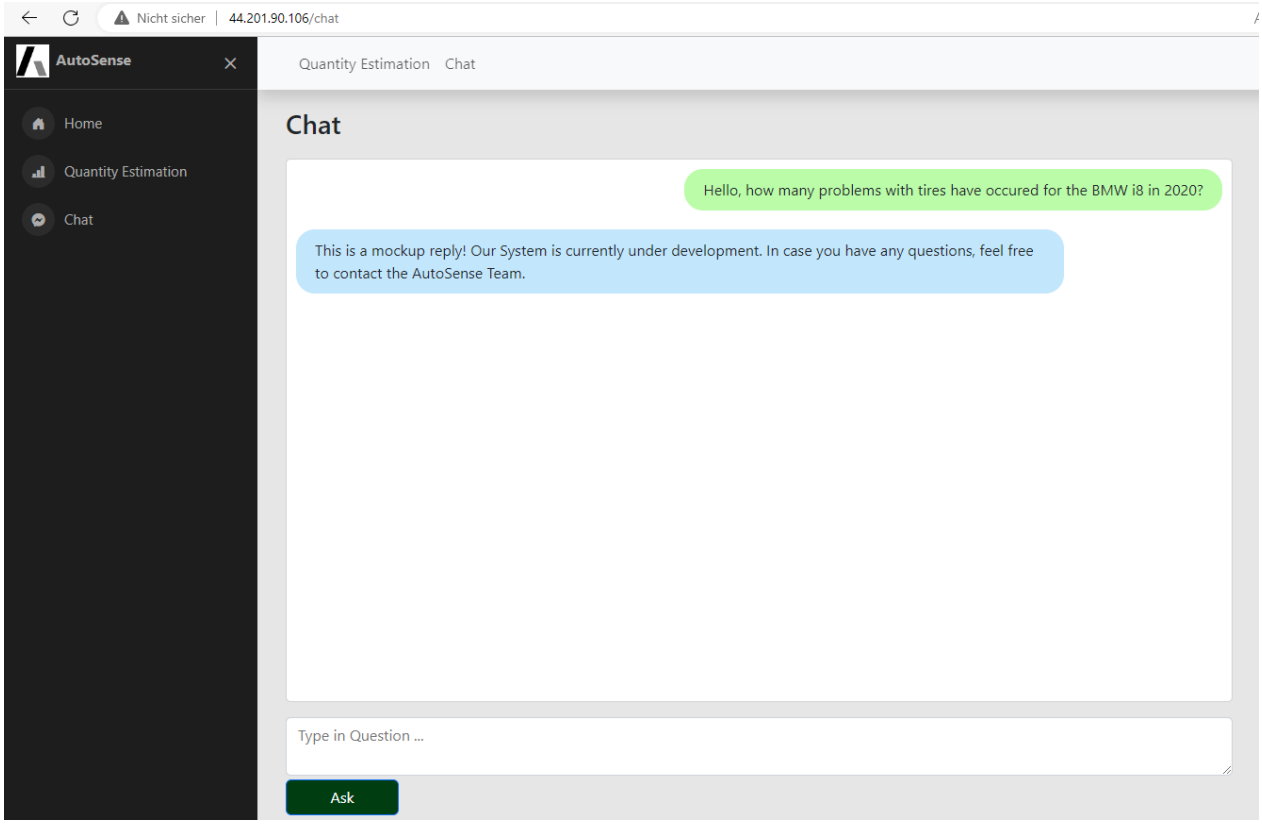
Chatbot

The Chatbot section provides a chat interface where users can type in questions and receive mockup responses.

Description: The chatbot is a text-based interface where users can enter questions or queries. However, note that it is currently in a mockup state and may not provide accurate responses.

Components used:

- Text input field for user input.
- Message display area for showing conversation history.



API Requests

To interact with the backend API, the frontend uses Axios for making HTTP requests. Axios provides a simple and intuitive way to send requests and handle responses.

Description: Axios is utilized to communicate with the backend API endpoints for retrieving data related to car malfunctions, brand and model information, occurrence counts, and more.

Conclusion

This frontend documentation has provided an overview of the frontend implementation using React.js, along with the usage of additional libraries such as Bootstrap and jQuery. It described the structure of the landing page and the two main components: Quantity Estimation and Chatbot. The Quantity Estimation section includes two pages that display bar charts based on user selections. The Chatbot section provides a mockup chat interface for user interaction. Integration with Bootstrap and jQuery allows for enhanced styling and additional functionality if required. This combination of technologies and libraries enables the development of a dynamic and visually appealing frontend application.

5) Natural Language Processing (NLP)

In order to process the forum posts and gain valuable information from it, Natural Language Processing (NLP) was used. To be more specific Natural Language Inference (NLI) as a subtopic of NLP. NLI is used on identifying directional relations between text. Different pretrained NLI models were put to the test in order to find the best fit for the given data. Based on a sample of the data the following tests were made.

all-MiniLM-L6-v2

```
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-Original!) nachrüsten.': 0.9999998807907104
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Ich brauche eine Heckkamera': 0.4904746413230896
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Dashcam ist kaputt': 0.4724730849266052
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Reifen haben ein Loch': 0.45074477791786194
```

distiluse-base-multilingual-cased

```
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-Original!) nachrüsten.': 0.9999999403953552
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Ich brauche eine Heckkamera': 0.40080565214157104
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Dashcam ist kaputt': 0.2627114951610565
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Reifen haben ein Loch': 0.15211904048919678
```

all-mpnet-base-v2

```
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-Original!) nachrüsten.': 1.0000001192092896
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Ich brauche eine Heckkamera': 0.4487485885620117
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Dashcam ist kaputt': 0.39919739961624146
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Reifen haben ein Loch': 0.4274103343486786
```

multi-qa-mpnet-base-dot-v1

```
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-Original!) nachrüsten.': 0.9999999403953552
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
```

```
Original!) nachrüsten.' and 'Ich brauche eine Heckkamera': 0.4428626000881195
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Dashcam ist kaputt': 0.48663175106048584
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Reifen haben ein Loch': 0.32912319898605347
```

multi-qa-distilbert-cos-v1

```
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-Original!) nachrüsten.': 0.9999997615814209
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Ich brauche eine Heckkamera': 0.31338033080101013
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Dashcam ist kaputt': 0.3035987615585327
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Reifen haben ein Loch': 0.03462108224630356
```

german-roberta-sentence-transformer-v2

```
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-Original!) nachrüsten.': 1.0000000238418579
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Ich brauche eine Heckkamera': 0.4691472053527832
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Dashcam ist kaputt': 0.26795658469200134
Similarity between 'Jetzt möchte ich mir gerne eine Original-
Rückfahrkamera (von Audi-
Original!) nachrüsten.' and 'Meine Reifen haben ein Loch': 0.12866975367069244
```

Comparing the different pretrained models it becomes clear, that “german-roberta-sentence-transformer-v2” performes the best. This comes at no surprise, because the Roberta model is finetuned with german data. Other models were traind with data from different languages, sometimes even multiple languages at the same time. Since the data in the database is from a german forum the german Roberta model performs best.

With OpenAI the most occurring topics were identified and utterances for each topics were generated.

Topics with highest occurrence:

- Zentralverriegelung
- Software Probleme
- Qualitätsmangel
- Reifen und Felgen
- Schiebedach Probleme
- Dachträger und Dachboxen

The text of each forum post is compared against all utterances of all topics and a confidence score is calculated from the average utterance confidence per topic. If the highest confididence surpasses a confidence level of 0.3 the topic is considered to be identified and written into the database.

Depending on the car that is discussed in the forum post, between 10 and 15 percent of forum posts can be matched to one of the mentioned six topics.

Further improvements to the NLI can be made by training a custom model for car related social media content and by defining more utterances per topic.

6) Machine Learning API

The machine learning api is a Flask application, which receives processed data from the forums and returns predictions for the demands of the next years.

The ML API was built using the following technologies:

Python 3.9.12

Description: Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility.

Flask 2.3.2

Description: Flask is a lightweight and flexible web framework for Python that allows for rapid development of web applications.

Werkzeug 2.3.4

Description: Werkzeug is a comprehensive WSGI utility library for Python that provides essential components for building web applications, such as request handling, routing, and response generation.

The Python library used for the Machine Learning Algorithm was:

- sklearn.linear_model

from sklearn.linear_model import LinearRegression

Other libraries in the protoype were:

- flask
- flask_cors
- Numpy

```
from flask import Flask, request, jsonify

from flask_cors import CORS

import numpy as np
```

The Machine Learning model used in the protype was Multiple Linear Regression. We chose this method, because it achieved a higher prediction accuracy for our available data than methods like Random Forest and Extreme Gradient Boosting.

Example:

Let's predict the demand of headlights for an Audi Q7 (2015) in 2023 based on data from the past three years.

When send sending a POST request to the API, with the following header

```
{

"occurances": [14, 13, 443]

}
```

...the API returns:

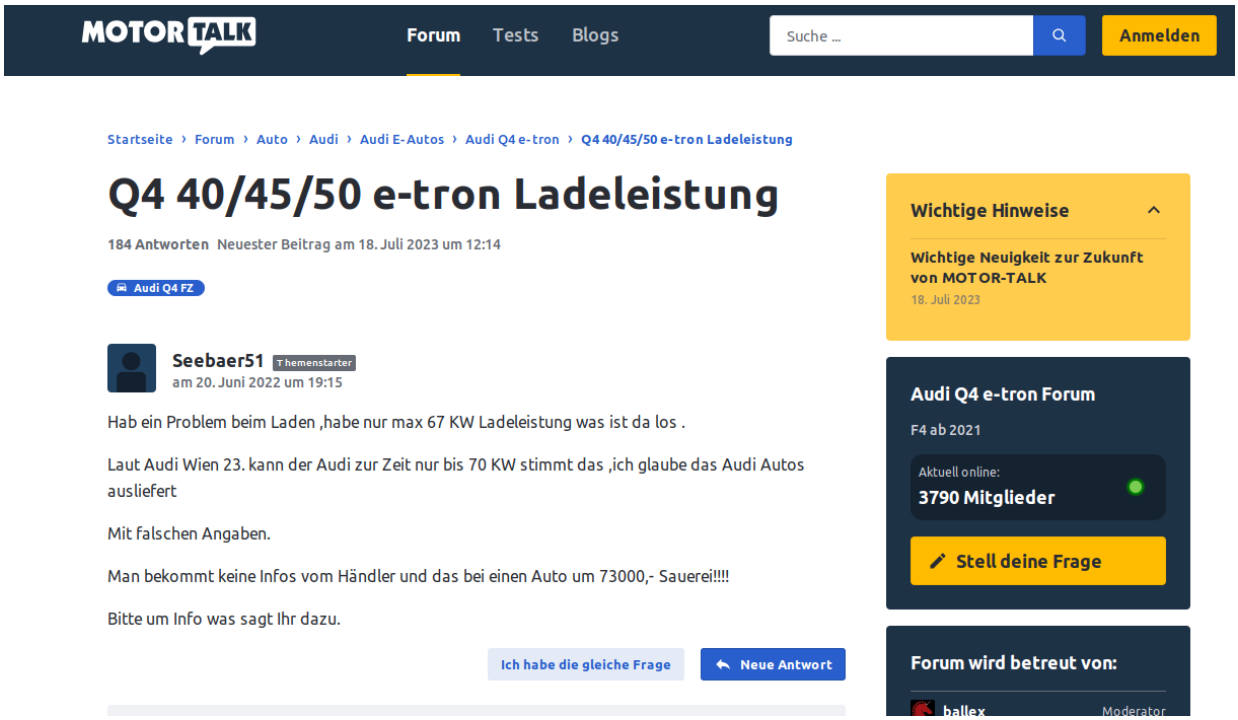
```
{

"predictions": 186.27272727272762

}
```

Interpretation of example: The demand in 2020, 2021 and 2022 were 14, 13 and 443 million headlights. Therefore the model predicts a demand of 186 million for 2023.

7) Data Extraction

In order to collect data from the web, web scraping was used. At the moment, data is extracted from only one German forum – MotorTalk (https://www.motor-talk.de/forum.html). This forum contains user entered data about the problems faced by them regarding cars of different models and brands.

The forum contains more than a million entries for Germany. The data from this forum is extracted using selenium in python.



Brands for which data is extracted currently -

- Alfa Romeo
- Audi
- BMW
- Mercedes

Fields extracted from each forum post-

- Post Date
- Username
- Brand
- Model
- Comments

This data is then stored in the database from where it is used for NLP processing, machine learning api and the Web UI. At the moment, data from around 25k forum posts is extracted and being used downstream.

8) Homepage

The Homepage can be accessed via [Home | AutoSense (jimdosite.com).]
It contains the following sites:

- Home
- Services
- Contact

- Team

9) Source Code

The source code of out prototype can be found under:

ArslanThobani/tc-autosense (github.com)