Bosch Car Service AI Assistant

An intelligent diagnostic assistant that helps users troubleshoot vehicle issues and interpret DTC (Diagnostic Trouble Code) errors using AI-powered responses with adaptive learning from user feedback.

Features

Intelligent Car Diagnostics

- DTC code interpretation and analysis
- Vehicle symptom-based troubleshooting
- Comprehensive diagnostic information
- Expert-level automotive assistance

Chat Interface

- Clean and intuitive chat-based interface
- Real-time AI responses
- Chat history management
- Ability to rename and organize conversations

NEW: Adaptive Feedback System

The AI Assistant now learns from your feedback to provide personalized responses:

How It Works:

1. **Rate Responses**: After each AI response, you can provide feedback using 👍 (thumbs up) or 👎 (thumbs down)
2. **Specify What You Like/Dislike**: Select from predefined options:

Positive Feedback Options:

- Perfect length and detail
- Technical accuracy was excellent
- Great video/resource links
- Clear step-by-step instructions
- Quick and to the point
- Exactly what I needed

Negative Feedback Options:

- Too long/verbose
- Too short/not enough detail
- Too technical/hard to understand
- Not technical enough
- Missing resources/videos
- Incorrect/inaccurate information
- Didn't address my question
- Poor formatting/hard to read

3. **Instant Adaptation**: The system intelligently adjusts the AI's response style based on your feedback patterns using a weighted aggregate approach

Intelligent Learning System:

The feedback system uses **aggregate weighted learning** rather than just the latest feedback:

- **Analyzes last 10 feedbacks** for each preference parameter
- **Recent feedback weighted higher** using exponential decay (most recent = 100%, 10th = ~20%)
- **Positive feedback weighted 1.5x** more than negative feedback
- **Pattern recognition** identifies consistent preferences over random clicks
- **Stability threshold** requires confidence score > 0.5 to change from defaults

- **Prevents flip-flopping** when you give contradictory feedback
- **Smooth transitions** as preferences evolve over time

What Gets Adjusted:

- **Response Length**: Concise, balanced, or detailed responses
- **Technical Level**: Simple language, balanced, or advanced technical terminology
- **Resource Density**: More or fewer video links and documentation
- **Structure Style**: Step-by-step focus, narrative format, or bullet points
- **Verification Level**: Standard or high-confidence responses with extra verification
- **Relevance Focus**: Direct answers or comprehensive explanations

Benefits:

- **Personalized Experience**: Each user gets responses tailored to their preferences
- **Continuous Improvement**: The AI learns and adapts to your specific needs over time
- **Better Results**: Over time, responses become more aligned with what you find helpful
- **Privacy-First**: All preferences are stored locally in your browser
- **Stable Learning**: Aggregate approach prevents erratic behavior from single feedback
- **Pattern Recognition**: System identifies your consistent preferences, not random clicks
- **Smart Weighting**: Recent feedback matters more, but history still influences decisions

Structured Response Format

Each response includes:

1. Error Code Meaning / Issue Description
2. Top 5 Most Common Causes
3. Step-by-Step Diagnostic Process

4. Verified Solutions with Difficulty Levels
5. Relevant Resources (Documentation & Videos)
6. Preventive Maintenance Advice

Key Capabilities

- Interprets DTC codes with detailed explanations
- Provides percentage-based frequency of common causes
- Includes estimated repair times and difficulty levels
- Links to official Bosch documentation and relevant resources
- Offers preventive maintenance suggestions

Technical Features

- Built with React and Material-UI
- Redux state management for chat and feedback
- Responsive design for mobile and desktop
- Integration with Perplexity AI API
- Persistent chat history
- **Dynamic prompt generation based on user feedback**
- **Local storage for user preferences and feedback history**
- **Real-time prompt adaptation system**

Architecture

Feedback System Architecture

```
User Feedback → FeedbackButtons Component → promptBuilder.js
                                                ↓
                                    Updates User Preferences (localStorage)
                                                ↓
                                    Next Query → buildSystemPrompt()
```

```
                                        ↓
                        AI Response with Adjusted Style
```

Key Components

1. **FeedbackButtons.js**: Interactive feedback UI component
2. **promptConfig.js**: Configuration for different prompt styles and feedback options
3. **promptBuilder.js**: Dynamic prompt generation engine
4. **chatSlice.js**: Redux state management including feedback storage

Data Flow

1. User provides feedback on AI response
2. Feedback is saved to history (last 50 feedbacks kept)
3. System analyzes last 10 feedbacks for each preference parameter
4. Calculates weighted scores with recency bias (exponential decay)
5. Updates preferences only if confidence threshold is met
6. User preferences are saved in localStorage
7. Next query uses the updated preferences to build a customized system prompt
8. AI generates response based on the learned personalized prompt

Getting Started

Prerequisites

- Node.js (v14 or higher)
- npm or yarn package manager

Installation

1. Clone the repository:

```
git clone [repository-url]
```

2. Install dependencies:

```
npm install
```

3. Start the development server:

```
npm start
```

The application will be available at `http://localhost:3000`

Usage

1. Start a New Chat
    - Click the "New Chat" button in the sidebar
    - Enter a DTC code or describe your car issue
2. Getting Help
    - Click the "Help & FAQ" button in the sidebar
    - Review usage tips and example queries
3. Managing Chats
    - Rename chats for better organization
    - Delete unnecessary conversations
    - Access chat history from the sidebar
4. Providing Feedback
    - After each AI response, click 👍 or 👎 to rate the response
    - Select one or more specific feedback options
    - Click "Submit Feedback" to apply your preferences
    - Your next query will use the updated response style

5. Resetting Preferences
    - Feedback preferences are stored locally
    - Clear browser data to reset to default preferences
    - Each feedback submission immediately updates the AI's behavior

Project Structure

```
src/
├── components/
│   └── Chat/
│       ├── ChatInterface.js       # Main chat component
│       ├── Message.js              # Individual message displ
│       ├── FeedbackButtons.js      # Feedback UI component
│       ├── ChatHistoryItem.js      # Sidebar chat items
│       └── HelpDialog.js           # Help information
├── redux/
│   ├── store.js                    # Redux store configurati
│   └── chatSlice.js                # Chat state & feedback ma
├── utils/
│   ├── promptConfig.js             # Prompt templates & feedb
│   └── promptBuilder.js            # Dynamic prompt generati
└── App.js                          # Main application compone
```

Feedback System Implementation

Implementation Details

Files Created:

1. **`src/utils/promptConfig.js`**
    - Configuration file containing all prompt templates and feedback options
    - Base prompt sections for different response styles
    - Feedback option definitions (positive and negative)

- Mapping between feedback selections and prompt adjustments

2. **`src/utils/promptBuilder.js`**
   - Dynamic prompt generation engine with **aggregate weighted learning**
   - Key Functions:
     - `getUserPreferences()` : Retrieve user preferences from localStorage
     - `saveUserPreferences()` : Save preferences to localStorage
     - `buildSystemPrompt()` : Generate customized prompt based on preferences
     - `processFeedback()` : Apply feedback adjustments using aggregate approach
     - `calculateAggregatePreferences()` : Analyze last 10 feedbacks with weighted scoring
     - `getFeedbackStats()` : Get feedback analytics
     - `resetPreferences()` : Reset to default settings
   - **Aggregate Learning Features:**
     - Exponential decay weighting (recent feedback = higher weight)
     - Positive feedback weighted 1.5x more than negative
     - Confidence threshold (0.5) prevents premature changes
     - Analyzes patterns across last 10 feedbacks
3. **`src/components/Chat/FeedbackButtons.js`**
   - Interactive feedback UI component with:
     - Thumbs up/down buttons
     - Expandable feedback options panel
     - Multi-select checkboxes for detailed feedback
     - Submit feedback button
     - Visual confirmation after submission

Files Modified:

- **`src/redux/chatSlice.js`** : Added `addFeedback` action to store feedback with messages
- **`src/components/Chat/Message.js`** : Integrated FeedbackButtons component for AI messages
- **`src/components/Chat/ChatInterface.js`** : Replaced static prompt with dynamic prompt generation

Adjustable Parameters

The system can adjust 6 different aspects of responses:

1. **responseLength**: `concise` | `balanced` | `detailed`
2. **technicalLevel**: `simple` | `balanced` | `technical`
3. **resourceDensity**: `low` | `medium` | `high`
4. **structureEmphasis**: `steps` | `narrative` | `bullets`
5. **verificationLevel**: `standard` | `high`
6. **relevanceFocus**: `direct` | `comprehensive`

Data Storage

localStorage Keys:

- `userPromptPreferences` : Current user preferences
- `feedbackHistory` : Last 50 feedback submissions
- `chats` : Chat history including feedback data

Example Preference Structure:

```
{
  responseLength: "concise",
  technicalLevel: "simple",
  resourceDensity: "high",
  structureEmphasis: "steps",
```

```
  verificationLevel: "standard",
  relevanceFocus: "direct"
}
```

UI Features

- **Color Coding**:
  - Green background for positive feedback panel
  - Orange background for negative feedback panel
  - Bosch blue for submit buttons
- **Visual States**:
  - Outlined thumbs (default)
  - Filled thumbs (selected)
  - Disabled state after submission
  - Success confirmation message
- **Animations**:
  - Smooth collapse/expand of options panel
  - Fade-in confirmation message
  - Hover effects on buttons

Technical Flow

```
User Provides Feedback
        ↓
FeedbackButtons Component captures selection
        ↓
Feedback saved to history (last 50 kept)
        ↓
calculateAggregatePreferences() analyzes last 10 feedbacks
        ↓
Applies weighted scoring with recency bias
        ↓
Positive feedback: weight × 1.5
```

```
Recent feedback: weight × (0.85^index)
        ↓
Calculates confidence scores for each option
        ↓
Updates preference if confidence > 0.5
        ↓
Saves to localStorage (userPromptPreferences)
        ↓
Stores in Redux (addFeedback action)
        ↓
Next Query → buildSystemPrompt() generates custom prompt
        ↓
Perplexity AI receives personalized system prompt
        ↓
AI Response matches user's preferred style (learned from patte
```

Feedback Impact Examples

User Feedback History	System Response
2× "Too long" + 1× "Perfect length"	Stays **concise** (weighted: 2 "too long" > 1 "perfect")
1× "Too long" + 1× "Too short"	Stays **balanced** (conflicting feedback = default)
3× "Too technical" (recent)	Switches to **simple language** (consistent pattern detected)
1× "Not technical" + 2× "Too technical"	Stays **simple** (most recent wins with higher weight)
5× "Missing resources" (positive)	Switches to **high resource density** (strong positive pattern, 1.5x weight)
1× "Great step-by-step" + old feedback	Reinforces **step-by-step structure** (positive feedback locks preference)

Key Learning Principles:

- **Consistency wins**: 2-3 similar feedbacks create pattern

- **Recent matters more**: Last feedback weighted highest
- **Positive reinforces**: Positive feedback has 1.5× impact
- **Conflicts = Balanced**: Contradictory feedback keeps defaults
- **Gradual evolution**: Preferences shift smoothly, not abruptly

Aggregate Learning Algorithm

The system uses a sophisticated weighted scoring algorithm:

Weighting Formula:

```
Weight = 0.85^index × (feedbackType === 'positive' ? 1.5 : 1.

Example for last 10 feedbacks:
Feedback #1 (most recent):  1.00 × type_weight
Feedback #2:                0.85 × type_weight
Feedback #3:                0.72 × type_weight
Feedback #10 (oldest):      0.20 × type_weight
```

Decision Logic:

1. For each preference parameter (responseLength, technicalLevel, etc.)
2. Analyze last 10 relevant feedbacks
3. Calculate weighted score for each option (concise/balanced/detailed)
4. Select option with highest score
5. Apply change only if score > 0.5 (confidence threshold)

Example Calculation:

```
Parameter: responseLength
Recent feedbacks:
  - "Too long" (1 day ago, negative)  → concise: +1.00
  - "Too long" (3 days ago, negative) → concise: +0.85
  - "Perfect" (5 days ago, positive)  → balanced: +1.08 (0.72
```

```
Scores: concise: 1.85, balanced: 1.08, detailed: 0
Winner: concise (score > 0.5) ✓
```

This approach ensures stable, intelligent learning from user behavior patterns!

Built With

- React.js - Frontend framework
- Material-UI - UI components
- Redux - State management
- Perplexity AI - AI model integration

Contributing

1. Fork the repository
2. Create your feature branch (`git checkout -b feature/AmazingFeature`)
3. Commit your changes (`git commit -m 'Add some AmazingFeature'`)
4. Push to the branch (`git push origin feature/AmazingFeature`)
5. Open a Pull Request

License

Acknowledgments

- Bosch Car Service for automotive expertise
- Perplexity AI for natural language processing capabilities
- Material-UI team for the component library