Car Management System - Technical Approach Document

## Overview

This document outlines the step-by-step technical approach to build a Car Management System using Next.js, Tailwind CSS, and Firebase. The system includes role-based access, post approval workflows, and basic chat functionality.

# 1. Project Initialization

## 1.1 Create Next.js App

npx create-next-app@latest car-management-system  
cd car-management-system

## 1.2 Install and Configure Tailwind CSS

npm install -D tailwindcss postcss autoprefixer  
npx tailwindcss init -p  
  
Update tailwind.config.js:  
content: [  
 "./pages/\*\*/\*.{js,ts,jsx,tsx}",  
 "./components/\*\*/\*.{js,ts,jsx,tsx}"  
]  
  
Add Tailwind to styles/globals.css:  
@tailwind base;  
@tailwind components;  
@tailwind utilities;

# 2. Firebase Setup

## 2.1 Create Project

Go to Firebase Console and create a new project. Enable Firestore and Authentication (Email/Password).

## 2.2 Install Firebase SDK

npm install firebase

## 2.3 Initialize Firebase in App

Create lib/firebase.js:  
  
import { initializeApp } from "firebase/app";  
import { getAuth } from "firebase/auth";  
import { getFirestore } from "firebase/firestore";  
  
const firebaseConfig = { /\* Firebase config \*/ };  
const app = initializeApp(firebaseConfig);  
const auth = getAuth(app);  
const db = getFirestore(app);  
export { auth, db };

# 3. Authentication

## 3.1 Login & Signup Pages

Create /pages/login.js and /pages/signup.js using Firebase Auth functions.

## 3.2 Store Role in Firestore

import { doc, setDoc } from "firebase/firestore";  
await setDoc(doc(db, "users", user.uid), { role: "renter" });

## 4. Role-Based Access

Roles: admin, renter, purchaser  
Store role in Firestore, redirect after login, restrict access using middleware or guards.

# 5. Post Management

## 5.1 Renter Creates Post

Save post to Firestore with approved: false

## 5.2 Admin Approves Post

Admin panel lists all posts and sets approved: true

## 5.3 Purchasers View Approved Posts

Display only posts where approved == true

# 6. Chat Functionality

## 6.1 Chat Data Structure

{  
 "user1": "uid1",  
 "user2": "uid2",  
 "messages": [  
 {  
 "text": "Hello",  
 "sender": "uid1",  
 "timestamp": "..."  
 }  
 ]  
}

## 6.2 Real-Time Messaging

Use Firestore onSnapshot() to listen for updates

## 7. Role Switching Logic

Purchaser requests role change, admin approves and updates in Firestore.

## 8. Admin Panel

View users, approve/reject posts, handle role requests, optional chat.

# 9. Deployment

## 9.1 Using Vercel

npx vercel  
Connect Firebase project to deployed frontend

## Future Improvements

Role toggling support  
Notification system  
Post filtering and search  
Enhanced chat features with file sharing