CHATBOT V1 PROJESİ RAPORU

Web Tabanlı Yapay Zeka Chatbot Uygulaması

Proje: ChatBot V1
Teknolojiler: React, Node.js, TypeScript, T5 AI Model, MySQL
Tarih: 2024

---

1. PROJE ÖZETİ

Bu proje, modern web teknolojileri kullanılarak geliştirilmiş yapay zeka destekli chatbot uygulamasıdır. Kullanıcılar hem ChatGPT API'si hem de özel T5 modeli ile etkileşim kurabilir.

Teknik Özellikler:

- **Frontend:** React 18 + TypeScript + Material-UI
- **Backend:** Node.js + Express.js + TypeScript
- **AI Model:** FLAN-T5 CodeParrot Fine-tuned Model
- **Veritabanı:** MySQL
- **Özellikler:** Gerçek zamanlı chat, model seçimi, konuşma geçmişi

---

2. SİSTEM MİMARİSİ

```
Frontend (React:3000) ↔ Backend (Node.js:5000) ↔ AI Service (Python:8000)
                                    ↓
                              Database (MySQL:3306)
```

Katmanlar:

1. **Sunum Katmanı:** React + Material-UI
2. **İş Mantığı Katmanı:** Express.js API
3. **Veri Katmanı:** MySQL Database
4. **AI Servisi:** Python Flask + T5 Model

---

3. KAYNAK KODLARI

3.1 Frontend - App.tsx

```tsx
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { ThemeProvider, createTheme, CssBaseline } from '@mui/material';
import { useState } from 'react';

// Pages
import Login from './pages/Login';
import Register from './pages/Register';
import Chat from './pages/Chat';
import Profile from './pages/Profile';

// Components
import Header from './components/Header';

// Context
import { AuthProvider } from './context/AuthContext';
import { ThemeContext } from '../src/context/ThemeContext';

function App() {
  const [darkMode, setDarkMode] = useState(localStorage.getItem('darkMode') === 'true');

  const theme = createTheme({
    palette: {
      mode: darkMode ? 'dark' : 'light',
      primary: {
        main: '#3f51b5',
      },
      secondary: {
        main: '#f50057',
      },
    },
    components: {
      MuiListItem: {
        defaultProps: {
          disablePadding: true,
        },
      },
    },
  });

  const toggleDarkMode = () => {
```

```
    setDarkMode(!darkMode);
    localStorage.setItem('darkMode', (!darkMode).toString());
  };

  return (
    <ThemeContext.Provider value={{ darkMode, toggleDarkMode }}>
      <ThemeProvider theme={theme}>
        <CssBaseline />
        <AuthProvider>
          <Router>
            <Header />
            <Routes>
              <Route path="/login" element={<Login />} />
              <Route path="/register" element={<Register />} />
              <Route
                path="/chat/:conversationId?"
                element={
                  <PrivateRoute>
                    <Chat />
                  </PrivateRoute>
                }
              />
              <Route
                path="/profile"
                element={
                  <PrivateRoute>
                    <Profile />
                  </PrivateRoute>
                }
              />
              <Route path="/" element={<Navigate to="/chat" replace />} />
              <Route path="*" element={<Navigate to="/login" replace />} />
            </Routes>
          </Router>
        </AuthProvider>
      </ThemeProvider>
    </ThemeContext.Provider>
  );
}

// Private route component
const PrivateRoute = ({ children }: { children: React.ReactNode }) => {
  const isAuthenticated = localStorage.getItem('token') !== null;

  if (!isAuthenticated) {
    return <Navigate to="/login" replace />;
  }

  return <>{children}</>;
};

export default App;
```

3.2 Backend - index.ts

```
import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';
import authRoutes from './routes/auth';
import chatRoutes from './routes/chat';

dotenv.config();

const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(cors());
app.use(express.json());

// Global error handler
app.use((err: any, req: express.Request, res: express.Response, next: express.NextFunction) => {
  console.error('Global error handler caught:', err);
  res.status(500).json({
    message: 'Internal server error',
    error: process.env.NODE_ENV === 'production' ? null : err.message
  });
});

// Test endpoint
app.get('/api/test', (req, res) => {
  res.json({ message: 'API test endpoint is working' });
});

// Routes
app.use('/api/auth', authRoutes);
app.use('/api/chat', chatRoutes);

// Root route
app.get('/', (req, res) => {
  res.send('ChatBot API is running');
});

// Start server
try {
  app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
    console.log(`- Root: http://localhost:${PORT}`);
    console.log(`- API Test: http://localhost:${PORT}/api/test`);
  });
} catch (error) {
  console.error('Failed to start server:', error);
}
```

3.3 AI Model Services

3.3.1 T5 CodeParrot Service - app.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import torch
from transformers import T5ForConditionalGeneration, T5Tokenizer
import os
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app = Flask(__name__)
CORS(app)

BASE_MODEL = "google/flan-t5-base"
CUSTOM_MODEL_PATH = "../flan-t5-codeparrot-model"
model = None
tokenizer = None

def load_model():
    global model, tokenizer

    try:
        logger.info("Loading T5 Model...")

        # Load base tokenizer
        tokenizer = T5Tokenizer.from_pretrained(BASE_MODEL)

        # Try custom model, fallback to base
        if os.path.exists(CUSTOM_MODEL_PATH):
            model = T5ForConditionalGeneration.from_pretrained(
                CUSTOM_MODEL_PATH,
                torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
                ignore_mismatched_sizes=True
```

```
            )
            logger.info("Custom model loaded successfully!")
        else:
            model = T5ForConditionalGeneration.from_pretrained(BASE_MODEL)
            logger.info("Base model loaded successfully!")

        model.eval()
        device = "cuda" if torch.cuda.is_available() else "cpu"
        logger.info(f"Using device: {device}")

        if not torch.cuda.is_available():
            model = model.to("cpu")

        return True

    except Exception as e:
        logger.error(f"Error loading model: {str(e)}")
        return False

def generate_response(question: str, max_length: int = 512) -> str:
    try:
        if model is None or tokenizer is None:
            return "Model not loaded"

        input_text = f"Answer this question: {question}"

        inputs = tokenizer.encode(
            input_text,
            return_tensors="pt",
            max_length=512,
            truncation=True,
            padding=True
        )

        device = next(model.parameters()).device
        inputs = inputs.to(device)

        with torch.no_grad():
            outputs = model.generate(
                inputs,
                max_length=max_length,
                min_length=10,
                num_beams=3,
                early_stopping=True,
                temperature=0.8,
                do_sample=True,
                top_p=0.9,
                no_repeat_ngram_size=2,
                pad_token_id=tokenizer.pad_token_id,
                eos_token_id=tokenizer.eos_token_id
            )

        response = tokenizer.decode(outputs[0], skip_special_tokens=True)
        response = response.strip()

        if response.lower().startswith("answer:"):
            response = response[7:].strip()

        return response if response else "Need more context for a helpful answer."

    except Exception as e:
        logger.error(f"Error generating response: {str(e)}")
        return f"Error: {str(e)}"

@app.route('/health', methods=['GET'])
def health_check():
    return jsonify({
        "status": "healthy",
        "model_loaded": model is not None,
        "tokenizer_loaded": tokenizer is not None,
        "base_model": BASE_MODEL
    })

@app.route('/chat', methods=['POST'])
def chat():
    try:
        data = request.get_json()

        if not data or 'message' not in data:
            return jsonify({"error": "Message required"}), 400

        message = data['message']
        max_length = data.get('max_length', 512)

        response = generate_response(message, max_length)
```

```
        return jsonify({
            "response": response,
            "model": "FLAN-T5-CodeParrot",
            "success": True
        })

    except Exception as e:
        logger.error(f"Error in chat: {str(e)}")
        return jsonify({
            "error": f"Server error: {str(e)}",
            "success": False
        }), 500

if __name__ == '__main__':
    load_model()
    app.run(host='0.0.0.0', port=8000, debug=False)
```

3.3.2 Mistral AI Service - ollama_mistral_service.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import requests
import logging
import json

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app = Flask(__name__)
CORS(app)

# Ollama configuration
OLLAMA_BASE_URL = "http://localhost:11434"
MODEL_NAME = "mistral:7b"

def check_ollama_status():
    """Check if Ollama is running"""
    try:
        response = requests.get(f"{OLLAMA_BASE_URL}/api/tags", timeout=5)
        return response.status_code == 200
    except Exception as e:
        logger.error(f"Ollama not accessible: {str(e)}")
        return False

def check_model_exists():
    """Check if Mistral model is downloaded"""
    try:
        response = requests.get(f"{OLLAMA_BASE_URL}/api/tags", timeout=5)
        if response.status_code == 200:
            models = response.json()
            for model in models.get('models', []):
                if model['name'].startswith('mistral'):
                    return True
        return False
    except Exception as e:
        logger.error(f"Error checking models: {str(e)}")
        return False

def pull_model():
    """Download Mistral model if not exists"""
    try:
        logger.info("Downloading Mistral 7B model...")

        payload = {"name": MODEL_NAME}
        response = requests.post(
            f"{OLLAMA_BASE_URL}/api/pull",
            json=payload,
            stream=True,
            timeout=600
        )

        if response.status_code == 200:
            logger.info("☑ Mistral 7B model downloaded successfully!")
            return True
        else:
            logger.error(f"Failed to download model: {response.text}")
            return False

    except Exception as e:
        logger.error(f"Error downloading model: {str(e)}")
        return False

def generate_response(prompt: str, max_tokens: int = 512) -> str:
    """Generate response using Ollama Mistral"""
    try:
        payload = {
```

```
            "model": MODEL_NAME,
            "prompt": prompt,
            "stream": False,
            "options": {
                "temperature": 0.7,
                "max_tokens": max_tokens,
                "top_p": 0.9
            }
        }

        response = requests.post(
            f"{OLLAMA_BASE_URL}/api/generate",
            json=payload,
            timeout=180
        )

        if response.status_code == 200:
            result = response.json()
            return result.get('response', 'No response generated')
        else:
            logger.error(f"Ollama API error: {response.text}")
            return f"Error: Ollama API returned {response.status_code}"

    except Exception as e:
        logger.error(f"Error generating response: {str(e)}")
        return f"Error: {str(e)}"

@app.route('/health', methods=['GET'])
def health_check():
    """Health check endpoint"""
    ollama_status = check_ollama_status()
    model_status = check_model_exists() if ollama_status else False

    return jsonify({
        "status": "healthy" if ollama_status and model_status else "unhealthy",
        "service": "Ollama Mistral 7B",
        "ollama_running": ollama_status,
        "model_downloaded": model_status,
        "model_name": MODEL_NAME,
        "ollama_url": OLLAMA_BASE_URL
    })

@app.route('/chat', methods=['POST'])
def chat():
    """Main chat endpoint"""
    try:
        data = request.get_json()

        if not data or 'message' not in data:
            return jsonify({"error": "Message is required"}), 400

        message = data['message']
        max_tokens = data.get('max_length', 512)

        # Check if Ollama is running
        if not check_ollama_status():
            return jsonify({
                "error": "Ollama service is not running."
            }), 503

        # Check if model exists, download if needed
        if not check_model_exists():
            logger.info("Model not found, downloading...")
            if not pull_model():
                return jsonify({
                    "error": "Failed to download Mistral model"
                }), 500

        # Generate response
        response = generate_response(message, max_tokens)

        return jsonify({
            "response": response,
            "model": "Mistral 7B (Ollama)",
            "success": True
        })

    except Exception as e:
        logger.error(f"Error in chat endpoint: {str(e)}")
        return jsonify({
            "error": f"Internal server error: {str(e)}",
            "success": False
        }), 500

if __name__ == '__main__':
    logger.info("🚀 Starting Ollama Mistral 7B Service...")
```

```
    app.run(host='0.0.0.0', port=8002, debug=False)
```

3.4 TypeScript Types - types.ts

```typescript
export type ModelType = 'api' | 'custom' | 'mistral';

export interface User {
  id: number;
  username: string;
  email: string;
  profilePicture?: string;
}

export interface UserSettings {
  theme: 'light' | 'dark';
  language: string;
  preferredModel: ModelType;
  notificationsEnabled: boolean;
}

export interface Message {
  id: number;
  message: string;
  response: string;
  modelType: ModelType;
  isMathRelated: boolean;
  createdAt: string;
}

export interface Conversation {
  id: number;
  title: string;
  modelType: ModelType;
  messageCount: number;
  isPinned: boolean;
  lastMessage?: string;
  lastResponse?: string;
  createdAt: string;
  updatedAt: string;
}
```

---

4. YAZILIM MİMARİSİ VE DESIGN PATTERN'LER

4.1 Kullanılan Mimari Yaklaşımlar

4.1.1 Layered Architecture (Katmanlı Mimari)

Proje, katmanlı mimari prensiplerine göre tasarlanmıştır:

```
┌─────────────────────────────────────┐
│          Presentation Layer         │ ← React Components
├─────────────────────────────────────┤
│          Business Logic Layer       │ ← Controllers & Services
├─────────────────────────────────────┤
│          Data Access Layer          │ ← Database Operations
├─────────────────────────────────────┤
│         External Services Layer     │ ← AI APIs, Third-party
└─────────────────────────────────────┘
```

4.1.2 Microservices Architecture

Her servis bağımsız port'larda çalışır:

- **Frontend Service:** Port 3000 (React)
- **Backend API:** Port 5000 (Node.js)
- **T5 AI Service:** Port 8000 (Python)
- **Mistral AI Service:** Port 8002 (Python)

4.2 Kullanılan Design Pattern'ler

4.2.1 Context Pattern - React State Management

```tsx
// AuthContext.tsx - Global State Management
import React, { createContext, useState, useEffect, ReactNode } from 'react';

interface AuthContextType {
  user: User | null;
  isAuthenticated: boolean;
  isLoading: boolean;
  login: (email: string, password: string) => Promise<void>;
  register: (username: string, email: string, password: string) => Promise<void>;
  logout: () => void;
  error: string | null;
}

export const AuthContext = createContext<AuthContextType>({
  user: null,
  isAuthenticated: false,
  isLoading: true,
  login: async () => {},
  register: async () => {},
  logout: () => {},
  error: null
});

export const AuthProvider = ({ children }: AuthProviderProps) => {
  const [user, setUser] = useState<User | null>(null);
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  // Authentication logic implementation
  const login = async (email: string, password: string) => {
    try {
      setError(null);
      const response = await axios.post(`${API_URL}/auth/login`, { email, password });
      const { token, user } = response.data;

      localStorage.setItem('token', token);
      localStorage.setItem('user', JSON.stringify(user));

      setUser(user);
      setIsAuthenticated(true);
    } catch (error: any) {
      setError(error.response?.data?.message || 'Login failed');
      throw error;
    }
  };

  return (
    <AuthContext.Provider value={{ user, isAuthenticated, isLoading, login, register, logout, error }}>
      {children}
    </AuthContext.Provider>
  );
};
```

4.2.2 Strategy Pattern - Model Selection

```tsx
// ModelSelector.tsx - Strategy Pattern Implementation
interface ModelSelectorProps {
  selectedModel: 'api' | 'custom' | 'mistral';
  onModelChange: (model: 'api' | 'custom' | 'mistral') => void;
}

const ModelSelector: React.FC<ModelSelectorProps> = ({ selectedModel, onModelChange }) => {
  const handleChange = (newModel: 'api' | 'custom' | 'mistral' | null) => {
    if (newModel !== null) {
      onModelChange(newModel); // Strategy değişimi
    }
  };

  return (
    <ToggleButtonGroup value={selectedModel} exclusive onChange={handleChange}>
      <ToggleButton value="api">
        <Box display="flex" alignItems="center" gap={1}>
          <ApiIcon fontSize="small" />
          <Typography variant="caption">ChatGPT</Typography>
        </Box>
      </ToggleButton>

      <ToggleButton value="custom">
        <Box display="flex" alignItems="center" gap={1}>
          <CustomModelIcon fontSize="small" />
          <Typography variant="caption">CodeParrot</Typography>
        </Box>
      </ToggleButton>

      <ToggleButton value="mistral">
        <Box display="flex" alignItems="center" gap={1}>
          <CustomModelIcon fontSize="small" />
          <Typography variant="caption">Mistral 7B</Typography>
        </Box>
      </ToggleButton>
    </ToggleButtonGroup>
  );
};
```

4.2.3 Router Pattern - Backend API Organization

```tsx
// chat.ts - Express Router Pattern
import express from 'express';
import {
  getChatHistory,
  getConversationMessages,
  createConversation,
  sendMessage,
  sendStreamingMessage,
  clearConversation,
  deleteConversation,
  clearChatHistory,
  getUserSettings,
  updateUserSettings,
  getUserStatistics
} from '../controllers/chatController';
import { authMiddleware } from '../middleware/authMiddleware';

const router = express.Router();

// Middleware application
router.use(authMiddleware as any);

// Conversation routes - RESTful pattern
router.get('/conversations', getChatHistory as any);
router.post('/conversations', createConversation as any);
router.get('/conversations/:conversationId', getConversationMessages as any);
router.delete('/conversations/:conversationId', deleteConversation as any);
router.delete('/conversations/:conversationId/messages', clearConversation as any);

// Message routes
router.post('/messages', sendMessage as any);
router.post('/messages/stream', sendStreamingMessage as any);

// User management routes
router.get('/settings', getUserSettings as any);
router.put('/settings', updateUserSettings as any);
router.get('/statistics', getUserStatistics as any);

export default router;
```

4.2.4 Factory Pattern - AI Service Creation

```python
# AI Service Factory Pattern Implementation
class AIServiceFactory:
    @staticmethod
    def create_service(service_type: str):
        if service_type == "t5":
            return T5Service()
        elif service_type == "mistral":
            return MistralService()
        elif service_type == "chatgpt":
            return ChatGPTService()
        else:
            raise ValueError(f"Unknown service type: {service_type}")

class T5Service:
    def __init__(self):
        self.model = load_t5_model()

    def generate_response(self, prompt: str) -> str:
        return self.model.generate(prompt)

class MistralService:
    def __init__(self):
        self.ollama_url = "http://localhost:11434"

    def generate_response(self, prompt: str) -> str:
        payload = {"model": "mistral:7b", "prompt": prompt}
        response = requests.post(f"{self.ollama_url}/api/generate", json=payload)
        return response.json().get('response', '')
```

4.2.5 Repository Pattern - Data Access

```typescript
// Database Repository Pattern
interface ConversationRepository {
  findById(id: number): Promise<Conversation | null>;
  findByUserId(userId: number): Promise<Conversation[]>;
  create(conversation: CreateConversationDto): Promise<Conversation>;
  update(id: number, conversation: UpdateConversationDto): Promise<Conversation>;
  delete(id: number): Promise<void>;
}

class MySQLConversationRepository implements ConversationRepository {
  constructor(private db: mysql.Connection) {}

  async findById(id: number): Promise<Conversation | null> {
    const [rows] = await this.db.execute(
      'SELECT * FROM conversations WHERE id = ?',
      [id]
    );
    return rows.length > 0 ? rows[0] as Conversation : null;
  }

  async findByUserId(userId: number): Promise<Conversation[]> {
    const [rows] = await this.db.execute(
      'SELECT * FROM conversations WHERE user_id = ? ORDER BY updated_at DESC',
      [userId]
    );
    return rows as Conversation[];
  }

  async create(conversation: CreateConversationDto): Promise<Conversation> {
    const [result] = await this.db.execute(
      'INSERT INTO conversations (user_id, title, model_type) VALUES (?, ?, ?)',
      [conversation.userId, conversation.title, conversation.modelType]
    );

    return this.findById((result as any).insertId);
  }
}
```

4.2.6 Middleware Pattern - Express.js

```
// Authentication Middleware
export const authMiddleware = (req: Request, res: Response, next: NextFunction) => {
  try {
    const token = req.header('Authorization')?.replace('Bearer ', '');

    if (!token) {
      return res.status(401).json({ message: 'Access denied. No token provided.' });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET!) as JwtPayload;
    req.user = decoded;
    next();
  } catch (error) {
    res.status(400).json({ message: 'Invalid token.' });
  }
};

// Error Handling Middleware
export const errorHandler = (err: Error, req: Request, res: Response, next: NextFunction) => {
  console.error('Global error handler:', err);

  res.status(500).json({
    message: 'Internal server error',
    error: process.env.NODE_ENV === 'production' ? null : err.message
  });
};
```

4.3 Architectural Benefits

4.3.1 Separation of Concerns

- **Frontend:** UI/UX logic ayrımı
- **Backend:** Business logic centralization
- **AI Services:** Model-specific implementations
- **Database:** Data persistence isolation

4.3.2 Scalability

- **Horizontal Scaling:** Microservices approach
- **Load Balancing:** Service-specific scaling
- **Resource Optimization:** Independent deployment

4.3.3 Maintainability

- **Modular Code:** Component-based architecture
- **Type Safety:** TypeScript implementation
- **Error Handling:** Centralized error management
- **Logging:** Structured logging system

---

5. PROJE YAPISI

```
ChatBotV1/
├── client/                 # React Frontend
│   ├── src/
│   │   ├── components/     # UI bileşenleri
│   │   ├── pages/          # Sayfa bileşenleri
│   │   ├── context/        # React Context
│   │   ├── types.ts        # TypeScript tipleri
│   │   └── App.tsx         # Ana uygulama
│   └── package.json
├── server/                 # Node.js Backend
│   ├── src/
│   │   ├── routes/         # API route'ları
│   │   ├── controllers/    # İş mantığı
│   │   └── index.ts        # Server entry point
│   └── package.json
├── model_service/          # Python AI Service
│   ├── app.py             # Flask uygulaması
│   └── requirements.txt    # Python bağımlılıkları
└── START_CHATBOT.bat      # Başlatma scripti
```

---

5. ÖZELLİKLER VE AI MODEL KARŞILAŞTIRMASI

5.1 Ana Özellikler

- **Kullanıcı Kimlik Doğrulama:** Login/Register sistemi
- **Çoklu AI Model Desteği:** 3 farklı AI model seçeneği
    - **ChatGPT API:** Genel amaçlı model (OpenAI)
    - **T5 CodeParrot:** Kod odaklı fine-tuned model
    - **Mistral 7B:** Local Ollama tabanlı model
- **Gerçek Zamanlı Chat:** Anlık mesajlaşma arayüzü

- **Konuşma Geçmişi:** Önceki sohbetleri kaydetme
- **Tema Desteği:** Light/Dark mode
- **Responsive Tasarım:** Mobile-friendly arayüz

5.2 AI Model Karşılaştırması

Özellik	ChatGPT API	T5 CodeParrot	Mistral 7B
Tip	Cloud API	Local Model	Local Model
Boyut	N/A	~1.2GB	~4GB
Uzmanlık	Genel	Kod/Programlama	Genel + Kod
Port	External	8000	8002
Yanıt Süresi	1-3s	3-8s	2-6s
İnternet	Gerekli	Gereksiz	Gereksiz
Maliyet	API Ücreti	Ücretsiz	Ücretsiz
Offline	Hayır	Evet	Evet
GPU Desteği	N/A	CUDA	CUDA
Bağımlılık	OpenAI API	Transformers	Ollama

5.3 AI Model Entegrasyonu Detayları

5.3.1 Model Selection Strategy Pattern

```
// ModelSelector Component with 3 AI Models
const ModelSelector: React.FC<ModelSelectorProps> = ({ selectedModel, onModelChange }) => {
  return (
    <ToggleButtonGroup value={selectedModel} exclusive onChange={handleChange}>
      <ToggleButton value="api">
        <Box display="flex" alignItems="center" gap={1}>
          <ApiIcon fontSize="small" />
          <Box textAlign="left">
            <Typography variant="caption">ChatGPT</Typography>
            <Typography variant="caption" display="block" color="textSecondary">
              API Model
            </Typography>
          </Box>
        </Box>
      </ToggleButton>

      <ToggleButton value="custom">
        <Box display="flex" alignItems="center" gap={1}>
          <CustomModelIcon fontSize="small" />
          <Box textAlign="left">
            <Typography variant="caption">CodeParrot</Typography>
            <Typography variant="caption" display="block" color="textSecondary">
              Code Model
            </Typography>
          </Box>
        </Box>
      </ToggleButton>

      <ToggleButton value="mistral">
        <Box display="flex" alignItems="center" gap={1}>
          <CustomModelIcon fontSize="small" />
          <Box textAlign="left">
            <Typography variant="caption">Mistral 7B</Typography>
            <Typography variant="caption" display="block" color="textSecondary">
              Local Model
            </Typography>
          </Box>
        </Box>
      </ToggleButton>
    </ToggleButtonGroup>
  );
};
```

5.3.2 Backend Model Routing

```
// Chat Controller - Model routing implementation
const sendMessage = async (req: Request, res: Response) => {
  const { message, modelType } = req.body;
  let response: string;

  try {
    switch (modelType) {
      case 'api':
        response = await sendToChatGPT(message);
        break;
      case 'custom':
        response = await sendToT5Model(message);
        break;
      case 'mistral':
        response = await sendToMistralModel(message);
        break;
      default:
        throw new Error('Unsupported model type');
    }

    res.json({ success: true, response, modelType });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
};
```

5.4 Teknik İmplementasyon

- **State Management:** React Context API
- **Routing:** React Router v6
- **HTTP İstemcisi:** Fetch API + Axios
- **UI Framework:** Material-UI v5
- **Database:** MySQL ile sohbet verisi
- **AI Integration:** REST API üzerinden model erişimi
- **Model Switching:** Runtime'da dinamik model değiştirme
- **Error Handling:** Model-specific error management

---

6. TEST VE PERFORMANS

6.1 Test Senaryoları

- Frontend component testleri
- API endpoint testleri
- AI model response testleri
- Database CRUD işlem testleri

6.2 Performans Metrikleri

- **Frontend Load:** ~2-3 saniye
- **API Response:** ~100-500ms
- **AI Model Response:** ~2-5 saniye
- **Database Query:** ~10-50ms

---

7. SONUÇ VE DEĞERLENDİRME

Bu ChatBot V1 projesi, modern web geliştirme teknolojilerini ve yapay zeka modellerini başarıyla entegre ederek kullanıcı dostu bir chatbot platformu oluşturmuştur.

Başarılar:

- Full-stack web uygulaması geliştirme
- AI model entegrasyonu ve deployment
- Modern UI/UX tasarım prensipleri
- TypeScript ile tip güvenliği
- Modüler kod yapısı

Teknik Kazanımlar:

- React ecosystem mastery
- Node.js backend geliştirme
- AI model deployment
- Database design ve optimization
- API design best practices

Gelecek Geliştirmeler:

- WebSocket ile real-time messaging
- Voice chat özelliği
- Advanced analytics dashboard
- Mobile app development
- Kubernetes deployment

Proje Tamamlanma Tarihi: 2024
Geliştirme Süresi: [Süre belirtilecek]
Kod Satırı: ~2000+ lines
Kullanılan Teknoloji Sayısı: 15+

Bu rapor, ChatBot V1 projesinin kapsamlı teknik dokümantasyonunu ve kod implementasyonunu içermektedir.