# GenBot - AI Chat Assistant

## Complete Source Code Documentation

## Table of Contents



## Project Structure

```
Chat Bot/
├── frontend/
│   ├── src/
│   │   ├── App.js
│   │   ├── Login.js
│   │   ├── Signup.js
│   │   ├── MessageFormatter.js
│   │   ├── components/
│   │   │   └── LoadingScreen.js
│   │   └── App.css
│   ├── package.json
│   └── .env
├── backend/
│   ├── src/main/java/com/chatbot/
│   │   ├── ChatController.java
│   │   ├── GroqService.java
│   │   ├── ImageGenerationService.java
│   │   ├── FileProcessingService.java
│   │   ├── User.java
│   │   ├── ChatSession.java
│   │   ├── Message.java
│   │   ├── FileUpload.java
│   │   ├── UserRepository.java
│   │   ├── ChatSessionRepository.java
│   │   ├── MessageRepository.java
│   │   ├── FileUploadRepository.java
│   │   └── ChatbotApplication.java
│   ├── src/main/resources/
│   │   └── application.properties
│   └── pom.xml
├── .env
└── README.md
```

# Frontend Source Code

## App.js (Main Application Component)

```
import React, { useState, useEffect, useRef } from 'react';
import { motion, AnimatePresence } from 'framer-motion';
import { Send, Paperclip, Menu, Search, Sun, Moon, Plus } from 'lucide-react';
import axios from 'axios';
import MessageFormatter from './MessageFormatter';
import Login from './Login';
import Signup from './Signup';
import './App.css';

// Use environment variable or fallback to current host
const API_BASE_URL = process.env.REACT_APP_API_URL ||
`${window.location.protocol}//${window.location.hostname}:8080`;

function App() {
  const [messages, setMessages] = useState([]);
  const [input, setInput] = useState('');
  const [isLoading, setIsLoading] = useState(false);
  const [darkMode, setDarkMode] = useState(false);
  const [typingText, setTypingText] = useState('');
  const [isTyping, setIsTyping] = useState(false);
  const [sessions, setSessions] = useState([]);
  const [currentSession, setCurrentSession] = useState(null);
  const [showSidebar, setShowSidebar] = useState(false);
  const [editingSession, setEditingSession] = useState(null);
  const [editTitle, setEditTitle] = useState('');
  const [currentUser, setCurrentUser] = useState(null);
  const [authMode, setAuthMode] = useState('login'); // 'login' or 'signup'
  const [isAuthenticated, setIsAuthenticated] = useState(false);
  const [responseLimit, setResponseLimit] = useState(2000);
  const [showSettings, setShowSettings] = useState(false);
  const [replyingTo, setReplyingTo] = useState(null);
  const [showSearch, setShowSearch] = useState(false);
  const [searchQuery, setSearchQuery] = useState('');
  const [searchResults, setSearchResults] = useState([]);
  const [isSearching, setIsSearching] = useState(false);
  const [editingMessage, setEditingMessage] = useState(null);
  const [editText, setEditText] = useState('');
  const [regeneratingMessage, setRegeneratingMessage] = useState(null);
  const [uploadingFile, setUploadingFile] = useState(false);
  const [isGeneratingImage, setIsGeneratingImage] = useState(false);
  const [generatingImagePrompt, setGeneratingImagePrompt] = useState('');
  const [aiThinking, setAiThinking] = useState(false);
  const [thinkingMessage, setThinkingMessage] = useState('');
  const messagesEndRef = useRef(null);
  const messagesContainerRef = useRef(null);
  const fileInputRef = useRef(null);
```

```
  const Logo = () => (
    <div className="flex items-center gap-2">
      <img src="/logo.svg" alt="GenBot" className="w-8 h-8" />
      <span className="text-xl font-bold bg-gradient-to-r from-blue-600 to-blue-
800 bg-clip-text text-transparent">
        GenBot
      </span>
    </div>
  );

  // ... (Rest of the component code continues)
  // Note: Full App.js code is extensive - this shows the structure
}

export default App;
```

## Login.js (Login Component)

```
import React, { useState } from 'react';
import { motion } from 'framer-motion';
import { User, Sparkles } from 'lucide-react';
import axios from 'axios';
import LoadingScreen from './components/LoadingScreen';

const API_BASE_URL = process.env.REACT_APP_API_URL ||
`${window.location.protocol}//${window.location.hostname}:8080`;

function Login({ onLogin, onSwitchToSignup }) {
  const [username, setUsername] = useState('');
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState('');

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!username.trim()) {
      setError('Please enter a username');
      return;
    }

    setIsLoading(true);
    setError('');

    try {
      const response = await axios.get(`${API_BASE_URL}/api/users`);
      const existingUser = response.data.find(user => user.username === username);

      if (existingUser) {
        await new Promise(resolve => setTimeout(resolve, 1500));
        onLogin(existingUser);
      } else {
        setError('User not found. Please sign up first.');
```

```
      }
    } catch (error) {
      console.error('Login error:', error);
      setError('Login failed. Please try again.');
    } finally {
      setIsLoading(false);
    }
  };

  if (isLoading) {
    return <LoadingScreen />;
  }

  return (
    <div className="min-h-screen bg-gradient-to-br from-indigo-900 via-purple-900
to-pink-900 flex items-center justify-center p-4 relative overflow-hidden">
      {/* Login form JSX */}
    </div>
  );
}

export default Login;
```

Signup.js (Signup Component)

```
import React, { useState } from 'react';
import axios from 'axios';

const API_BASE_URL = process.env.REACT_APP_API_URL ||
`${window.location.protocol}//${window.location.hostname}:8080`;

function Signup({ onSignup, onSwitchToLogin }) {
  const [formData, setFormData] = useState({
    username: '',
    displayName: '',
    avatar: '👤'
  });
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);

  const avatars = ['👤', '👨', '👩', '🧒', '👦', '👧', '🤖', '🐱', '🐶',
'🦊'];

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!formData.username.trim() || !formData.displayName.trim()) return;

    setLoading(true);
    setError('');

    try {
```

```
      const response = await axios.post(`${API_BASE_URL}/api/users`, formData);
      onSignup(response.data);
    } catch (error) {
      if (error.response?.status === 500) {
        setError('Username already exists. Please choose a different one.');
      } else {
        setError('Signup failed. Please try again.');
      }
    } finally {
      setLoading(false);
    }
  };

  // ... (Rest of signup form JSX)
}

export default Signup;
```

MessageFormatter.js (Message Formatting Component)

```
import React from 'react';
import { Prism as SyntaxHighlighter } from 'react-syntax-highlighter';
import { vscDarkPlus, vs } from 'react-syntax-highlighter/dist/esm/styles/prism';

const MessageFormatter = ({ content, darkMode }) => {
  const decodeHtml = (text) => {
    return text
      .replace(/&quot;/g, '"')
      .replace(/&#39;/g, "'")
      .replace(/&lt;/g, '<')
      .replace(/&gt;/g, '>')
      .replace(/&amp;/g, '&')
      .replace(/\\u003e/g, '>')
      .replace(/\\u003c/g, '<')
      .replace(/\\n/g, '\n')
      .replace(/\\t/g, '\t')
      .replace(/\\\\/g, '/');
  };

  // ... (Message formatting logic)

  // Handle image content
  if (content.startsWith('![') && content.includes('](data:image/')) {
    const imageMatch = content.match(/!\[.*?\]\((data:image\/[^)]+)\)/);
    if (imageMatch) {
      const downloadImage = () => {
        const link = document.createElement('a');
        link.href = imageMatch[1];
        link.download = `generated-image-${Date.now()}.png`;
        document.body.appendChild(link);
        link.click();
```

```
        document.body.removeChild(link);
      };

      return (
        <div className="formatted-message relative group">
          <img
            src={imageMatch[1]}
            alt="Generated Image"
            className="max-w-full h-auto rounded-lg shadow-lg"
            style={{ maxHeight: '400px' }}
          />
          <button
            onClick={downloadImage}
            className="absolute top-2 right-2 opacity-0 group-hover:opacity-100
bg-black/50 hover:bg-black/70 text-white p-2 rounded-lg transition-all duration-
200"
            title="Download Image"
          >
            <svg className="w-4 h-4" fill="none" stroke="currentColor" viewBox="0
0 24 24">
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth={2}
d="M12 10v6m0 0l-4-4m4 4l4-4m-6 8h8a2 2 0 002-2V7a2 2 0 00-2-2H8a2 2 0 00-2 2v11a2
2 0 002 2z" />
            </svg>
          </button>
        </div>
      );
    }
  }

  return <div className="formatted-message whitespace-pre-wrap leading-relaxed">
{formatMessage(content)}</div>;
};

export default MessageFormatter;
```

---

## Backend Source Code

### ChatController.java (Main REST Controller)

```
package com.chatbot;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.http.ResponseEntity;
import org.springframework.web.multipart.MultipartFile;
import java.util.List;
import java.util.Map;
import java.time.LocalDateTime;
```

```
@RestController
@RequestMapping("/api")
@CrossOrigin(origins = "*")
public class ChatController {

    @Autowired
    private MessageRepository messageRepository;

    @Autowired
    private ChatSessionRepository sessionRepository;

    @Autowired
    private GroqService groqService;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private FileProcessingService fileProcessingService;

    @Autowired
    private FileUploadRepository fileUploadRepository;

    @Autowired
    private ImageGenerationService imageGenerationService;

    @GetMapping("/users")
    public List<User> getUsers() {
        return userRepository.findAll();
    }

    @PostMapping("/users")
    public User createUser(@RequestBody UserRequest request) {
        if (userRepository.existsByUsername(request.getUsername())) {
            throw new RuntimeException("Username already exists");
        }
        User user = new User(request.getUsername(), request.getDisplayName());
        user.setAvatar(request.getAvatar());
        return userRepository.save(user);
    }

    @PostMapping("/sessions/{sessionId}/messages")
    public Message sendMessage(@PathVariable Long sessionId, @RequestBody
MessageRequest request) {
        ChatSession session = sessionRepository.findById(sessionId).orElseThrow();

        Message userMessage = new Message(request.getContent(), "user");
        userMessage.setSession(session);

        if (request.getParentMessageId() != null) {
            Message parentMessage =
messageRepository.findById(request.getParentMessageId()).orElse(null);
            if (parentMessage != null) {
                userMessage.setParentMessage(parentMessage);
```

```
            }
        }

        messageRepository.save(userMessage);

        String aiPrompt = request.getContent();
        if (request.getParentMessageId() != null && userMessage.getParentMessage()
!= null) {
            aiPrompt = "Replying to: \"" +
userMessage.getParentMessage().getContent() + "\"\n\n" + request.getContent();
        }

        int maxTokens = request.getMaxTokens() != null ? request.getMaxTokens() :
2000;
        String botResponse = groqService.getChatResponse(aiPrompt, maxTokens);
        Message botMessage = new Message(botResponse, "bot");
        botMessage.setSession(session);

        if (userMessage.getParentMessage() != null) {
            botMessage.setParentMessage(userMessage.getParentMessage());
        }

        messageRepository.save(botMessage);

        session.setUpdatedAt(java.time.LocalDateTime.now());
        sessionRepository.save(session);

        return botMessage;
    }

    @PostMapping("/sessions/{sessionId}/generate-image")
    public ResponseEntity<Message> generateImage(@PathVariable Long sessionId,
@RequestBody ImageRequest request) {
        try {
            System.out.println("Image generation request for session: " +
sessionId + ", prompt: " + request.getPrompt());

            ChatSession session =
sessionRepository.findById(sessionId).orElseThrow();

            Message userMessage = new Message("🎨 Generate image: " +
request.getPrompt(), "user");
            userMessage.setSession(session);
            messageRepository.save(userMessage);

            String base64Image =
imageGenerationService.generateImage(request.getPrompt());
            String imageContent = "![Generated Image](" +
base64Image + ")";

            Message botMessage = new Message(imageContent, "bot");
            botMessage.setSession(session);
            messageRepository.save(botMessage);
```

```
            session.setUpdatedAt(LocalDateTime.now());
            sessionRepository.save(session);

            return ResponseEntity.ok(botMessage);
        } catch (Exception e) {
            System.err.println("Image generation error: " + e.getMessage());
            e.printStackTrace();
            return ResponseEntity.badRequest().body(null);
        }
    }

    // ... (Additional controller methods)
}
```

### GroqService.java (AI Service)

```
package com.chatbot;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.reactive.function.client.WebClient;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.JsonNode;
import java.util.Map;
import java.util.List;
import java.time.Duration;

@Service
public class GroqService {

    @Value("${groq.api.key}")
    private String apiKey;

    @Value("${groq.api.url}")
    private String apiUrl;

    private final WebClient webClient;
    private final ObjectMapper objectMapper;

    public GroqService() {
        this.webClient = WebClient.builder()
            .codecs(configurer -> configurer.defaultCodecs().maxInMemorySize(10 *
1024 * 1024))
            .build();
        this.objectMapper = new ObjectMapper();
    }

    public String getChatResponse(String message, int maxTokens) {
        try {
```

```
            Map<String, Object> requestBody = Map.of(
                "model", "llama-3.1-8b-instant",
                "messages", List.of(
                    Map.of("role", "user", "content", message)
                ),
                "max_tokens", maxTokens
            );

            System.out.println("Request body: " +
objectMapper.writeValueAsString(requestBody));

            String response = webClient.post()
                .uri(apiUrl)
                .header(HttpHeaders.AUTHORIZATION, "Bearer " + apiKey)
                .header(HttpHeaders.CONTENT_TYPE,
MediaType.APPLICATION_JSON_VALUE)
                .bodyValue(requestBody)
                .retrieve()
                .bodyToMono(String.class)
                .timeout(Duration.ofSeconds(30))
                .block();

            System.out.println("Full API response: " + response);

            JsonNode jsonResponse = objectMapper.readTree(response);
            String content =
jsonResponse.path("choices").get(0).path("message").path("content").asText();

            System.out.println("Extracted content length: " + content.length());

            return content;
        } catch (Exception e) {
            System.err.println("Error calling Groq API: " + e.getMessage());
            e.printStackTrace();
            return "I apologize, but I'm having trouble processing your request
right now. Please try again.";
        }
    }
}
```

ImageGenerationService.java (Image Generation Service)

```
package com.chatbot;

import org.springframework.stereotype.Service;
import org.springframework.web.reactive.function.client.WebClient;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import java.time.Duration;
```

```
@Service
public class ImageGenerationService {

    private final WebClient webClient;
    private static final String POLLINATIONS_URL =
"https://image.pollinations.ai/prompt/";

    public ImageGenerationService() {
        this.webClient = WebClient.builder()
            .codecs(configurer -> configurer.defaultCodecs().maxInMemorySize(10 *
1024 * 1024))
            .build();
    }

    public String generateImage(String prompt) {
        try {
            System.out.println("Generating image for prompt: " + prompt);

            String encodedPrompt = URLEncoder.encode(prompt,
StandardCharsets.UTF_8);
            String imageUrl = POLLINATIONS_URL + encodedPrompt + "?
width=512&height=512&nologo=true&enhance=true";

            System.out.println("Image URL: " + imageUrl);

            byte[] imageBytes = webClient.get()
                .uri(imageUrl)
                .retrieve()
                .bodyToMono(byte[].class)
                .timeout(Duration.ofSeconds(30))
                .block();

            if (imageBytes != null && imageBytes.length > 0) {
                System.out.println("Image generated successfully, size: " +
imageBytes.length + " bytes");
                return Base64.getEncoder().encodeToString(imageBytes);
            }

            throw new RuntimeException("No image data received from Pollinations
API");
        } catch (Exception e) {
            System.err.println("Primary image generation failed: " +
e.getMessage());

            try {
                String encodedPrompt = URLEncoder.encode(prompt,
StandardCharsets.UTF_8);
                String altImageUrl = "https://image.pollinations.ai/prompt/" +
encodedPrompt + "?nologo=true&enhance=true";

                System.out.println("Trying alternative URL: " + altImageUrl);

                byte[] imageBytes = webClient.get()
                    .uri(altImageUrl)
```

```
                    .retrieve()
                    .bodyToMono(byte[].class)
                    .timeout(Duration.ofSeconds(30))
                    .block();

                if (imageBytes != null && imageBytes.length > 0) {
                    System.out.println("Alternative image generated successfully,
size: " + imageBytes.length + " bytes");
                    return Base64.getEncoder().encodeToString(imageBytes);
                }
            } catch (Exception altE) {
                System.err.println("Alternative image generation also failed: " +
altE.getMessage());
            }

            throw new RuntimeException("Failed to generate image: " +
e.getMessage());
        }
    }
}
```

---

## Configuration Files

### application.properties (Backend Configuration)

```
# Database Configuration
spring.datasource.url=${DATABASE_URL:jdbc:postgresql://localhost:5432/chatbot}
spring.datasource.username=postgres
spring.datasource.password=password123
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=false

# Server Configuration
server.port=${PORT:8080}
server.address=0.0.0.0

# Groq Configuration
groq.api.key=${GROQ_API_KEY:your_groq_api_key_here}
groq.api.url=https://api.groq.com/openai/v1/chat/completions

# File Upload Configuration
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB
```

### package.json (Frontend Dependencies)

```
{
  "name": "genbot-frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.4",
    "@testing-library/react": "^13.3.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.4.0",
    "framer-motion": "^10.12.16",
    "lucide-react": "^0.263.1",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "react-syntax-highlighter": "^15.5.0",
    "tailwindcss": "^3.3.2",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  }
}
```

pom.xml (Backend Dependencies)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.0</version>
        <relativePath/>
    </parent>

    <groupId>com.example</groupId>
    <artifactId>chatbot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>chatbot</name>
    <description>AI Chat Assistant with Spring Boot</description>

    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
        <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-webflux</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
            <version>5.2.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.pdfbox</groupId>
            <artifactId>pdfbox</artifactId>
            <version>2.0.28</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

---

## How to Convert to PDF

### Method 1: Using Markdown to PDF Converters

1. **Pandoc** (Recommended):

   ```
   pandoc SOURCE_CODE_DOCUMENTATION.md -o GenBot_Source_Code.pdf
   ```

2. **Online Converters**:
   - Markdown to PDF online converters

- GitHub's built-in PDF export

### Method 2: Using IDE/Editor

1. **VS Code**: Install "Markdown PDF" extension
2. **IntelliJ IDEA**: Built-in markdown to PDF export
3. **Typora**: Direct PDF export feature

### Method 3: Print to PDF

1. Open the markdown file in any markdown viewer
2. Use browser's "Print to PDF" function
3. Adjust margins and formatting as needed

---

*This documentation contains the complete source code structure and key components of the GenBot AI Chat Assistant project.*