# MoMIQ (Meeting Notes Made Intelligent & Quick) Technical Documentation

GitHub Repo : <https://github.com/bharatalok11/MoMIQ.git>

Deployed Link : <https://momiq-summarizer.vercel.app/>

This document provides a comprehensive technical overview of the MoMIQ application, including its architecture, technology stack, development processes, and deployment strategies.

## 1. Project Overview

**MoMIQ (Meeting Notes Made Intelligent & Quick)** is an AI-powered application designed to transform raw meeting transcripts into actionable insights. It leverages intelligent summarization and provides instant sharing capabilities to streamline post-meeting workflows.

## 2. Architecture Overview

The system is designed with a decoupled frontend and backend architecture, ensuring scalability and maintainability. The frontend communicates with the backend via a RESTful API over HTTP/HTTPS.

### System Architecture Diagram

**┌─────────────────┐ HTTP/HTTPS ┌─────────────────┐**

**│ Frontend │ ◄──────────────► │ Backend │**

**│ (React/Vite) │ │ (Node.js) │**

**│ │ │ │**

**│ - File Upload │ │ - AI Processing │**

**│ - UI Components │ │ - Email Service │**

**│ - State Mgmt │ │ - API Routes │**

**└─────────────────┘ └─────────────────┘**

**│ │**

**│ │**

**▼ ▼**

**┌─────────────────┐ ┌─────────────────┐**

**│ Vercel │ │ onrender │**

**│ (Frontend) │ │ (Backend) │**

**└─────────────────┘ └─────────────────┘**

* **Frontend**: A modern React application built with Vite, responsible for the user interface and client-side logic. Hosted on **Vercel**.
* **Backend**: A Node.js and Express.js server that handles business logic, AI processing with Groq, and email services. Hosted on **OnRender**.

## 3. Technology Stack

### Frontend Technologies

* **Core Framework**:
  + **React 19.1.1**: The latest version of React, utilizing modern features and hooks.
  + **Vite 7.1.2**: A high-performance build tool and development server.
  + **JavaScript**
* **Styling & UI**:
  + **Tailwind CSS 3.4.17**: A utility-first CSS framework for rapid UI development.
  + **PostCSS & Autoprefixer**: For CSS processing and automatic vendor prefixing.
* **State Management**:
  + **React Hooks**: useState and useEffect for managing local component state.
  + **Custom Hooks**: useApi for streamlined API communication.
  + **Context API**: For sharing state across components where needed.
* **HTTP Client**:
  + **Axios 1.11.0**: A promise-based HTTP client for making API requests.
  + **Interceptors**: Configured for centralized request/response handling and error management.

### Backend Technologies

* **Runtime & Framework**:
  + **Node.js 18+**: A JavaScript runtime environment for building server-side applications.
  + **Express.js 5.1.0**: A minimal and flexible web application framework for Node.js.
  + **Body Parser**: Middleware for parsing incoming request bodies.
* **AI Integration**:
  + **Groq AI SDK**: For accessing high-performance AI inference capabilities.
  + **Model**: llama-3.1-8b-instant (configurable).
  + **Prompt Engineering**: Utilizes structured templates for consistent and effective AI responses.
* **Email Service**:
  + **Nodemailer 7.0.5**: A module for sending emails from Node.js applications.
  + **Gmail SMTP**: Used as the email transport service.
* **Development Tools**:
  + **Nodemon**: Automatically restarts the server during development on file changes.
  + **Dotenv**: Manages environment variables for secure configuration

### Infrastructure & Deployment

* **Frontend Hosting**: **Vercel** ( Auto Deploy).
* **Backend Hosting**: **OnRender** (Environment Variables, Logging).
* **Version Control**: **Git** & **GitHub**.

## 4. Development Process

### 1. Project Setup Phase

#### Frontend Initialization

*# Create React project with Vite*

npm create vite@latest frontend -- --template react

cd frontend

*# Install dependencies*

npm install axios react react-dom

npm install -D tailwindcss postcss autoprefixer

*# Configure Tailwind CSS*

npx tailwindcss init -p

#### Backend Initialization

*# Create backend directory*

mkdir backend

cd backend

*# Initialize Node.js project*

npm init -y

*# Install dependencies*

npm install express cors body-parser dotenv groq-sdk nodemailer

npm install -D nodemon

### 2. Development Workflow

* **Component Development**: New components are created in src/components/, styled with Tailwind CSS, and integrated into the main application.
* **API Development**: New API routes are defined in the routes/ directory, with business logic handled by services in the services/ directory.

### 3. Testing & Quality Assurance

* **Frontend Testing**: Includes manual UI/UX testing, cross-browser compatibility checks, responsive design verification, and Lighthouse performance audits.
* **Backend Testing**: Involves API endpoint validation with tools like Postman, error handling tests, load testing, and integration tests for AI and email services.

## 5. Project Structure

### Frontend Structure

frontend/  
├── public/ # Static assets  
├── src/  
│ ├── components/ # React components  
│ │ ├── Header.jsx  
│ │ ├── FileUpload.jsx  
│ │ ├── SummaryDisplay.jsx  
│ │ └── ...  
│ ├── hooks/ # Custom React hooks  
│ │ └── useApi.js  
│ ├── App.jsx # Main application component  
│ └── main.jsx # Application entry point  
├── package.json  
└── vite.config.js

### Backend Structure

backend/  
├── routes/ # API route handlers  
│ ├── summaryRoutes.js  
│ └── emailRoutes.js  
├── services/ # Business logic services  
│ ├── aiService.js  
│ └── emailService.js  
├── utils/ # Utility functions  
│ └── promptBuilder.js  
├── index.js # Main server file  
├── package.json  
└── .env # Environment variables

## 6. API Design

The backend exposes a RESTful API for frontend communication.

### RESTful Endpoints

#### Summary Generation

* **Endpoint**: POST /generate-summary
* **Request Body**:  
  JSON

{  
 "transcript": "Meeting transcript text...",  
 "prompt": "Custom summarization instructions..."  
}

* **Success Response (200)**:  
  JSON  
  {  
   "success": true,  
   "summary": "Generated summary text...",  
   "message": "Summary generated successfully"  
  }

#### Email Sharing

* **Endpoint**: POST /share-summary
* **Request Body**:  
  JSON  
  {  
   "summary": "Summary text to share...",  
   "emails": ["user1@example.com", "user2@example.com"]  
  }
* **Success Response (200)**:  
  JSON  
  {  
   "success": true,  
   "message": "Summary shared via email successfully"  
  }

#### Health Check

* **Endpoint**: GET /health
* **Success Response (200)**:  
  JSON  
  {  
   "status": "healthy",  
   "timestamp": "2025-01-17T10:30:00.000Z"  
  }

### Error Handling

A standardized error response format is used across the API.

* **Standard Error Response**:  
  JSON  
  {  
   "success": false,  
   "error": "Error description",  
   "details": "Additional error information"  
  }
* **HTTP Status Codes**:
  + 200: Success
  + 400: Bad Request (e.g., missing parameters)
  + 404: Route Not Found
  + 500: Internal Server Error

## 7. Security Implementation

* **CORS Configuration**: The backend is configured to accept requests only from the deployed frontend origin (https://momiq-summarizer.vercel.app).
* **Environment Variables**: All sensitive keys (API keys, email credentials) are stored securely as environment variables and are not hardcoded.
* **Input Validation**:
  + **File Size & Type**: Limits uploads to 10MB and restricts them to .txt files.
  + **Content Sanitization**: Inputs are cleaned to prevent injection attacks.
* **Rate Limiting**: API request throttling is implemented to prevent abuse and ensure service stability.

## 8. Performance Optimization

### Frontend

* **Code Splitting**: Components are lazy-loaded to reduce initial bundle size.
* **Bundle Optimization**: Vite's build process handles tree shaking and minification.
* **Caching**: Browser and CDN caching are leveraged for faster asset delivery.

### Backend

* **Response Caching**: Caching strategies are in place for frequently requested, non-dynamic data.
* **Compression**: Gzip compression is used to reduce the size of API responses.
* **AI Processing**: Prompt engineering is optimized for faster and more efficient responses from the AI model.

## 9. Deployment Pipeline

The deployment process is fully automated via GitHub integration.

1. **Code Push**: Developers push code changes to the main branch.
2. **Auto Build**: Vercel and OnRender detect the push and trigger a new build.
3. **Deployment**: The new build is automatically deployed to production.