Quick Chat (React)

Introduction:-

In the modern digital world, real-time communication applications play an important role in connecting people across different locations. Chat applications enable users to exchange messages instantly and are widely used for personal communication, educational purposes, and professional collaboration. With the advancement of web technologies, building efficient and user-friendly chat systems has become an essential part of modern web development.

The project titled “QuickChat – Real-Time Chat Application” is developed to demonstrate the implementation of a web-based chat platform using modern frontend technologies. The primary objective of this project is to provide a simple, interactive, and responsive application that allows users to communicate through instant messaging. The application focuses on delivering a clean user interface and smooth user experience.

QuickChat is built using Next.js, a React-based framework that simplifies the development of fast and scalable web applications. The project is implemented using JavaScript with JSX, which allows the creation of dynamic user interfaces by combining JavaScript logic with HTML-like syntax. The application follows a component-based architecture, making the code modular, reusable, and easy to maintain.

To manage shared data efficiently, the application uses React Context for state management, reducing unnecessary data passing between components. The user interface is designed to be responsive, ensuring proper functionality across different devices and screen sizes. Overall, the QuickChat project demonstrates the practical use of modern frontend technologies to build a real-time chat application with an organized structure and user-friendly design.

Scenario-Based Intro:-

Imagine a student or developer working on a project who needs to communicate instantly with teammates. Instead of relying on delayed messaging platforms, they open Quick Chat.

As soon as the application loads, the user is presented with a simple chat interface. Messages typed in the input box are delivered instantly to the receiver. The interface updates in real time without reloading the page, making communication fast and efficient.

Quick Chat eliminates delays and provides a smooth chatting experience, making it ideal for quick discussions and collaboration.

Target Audience:-

Quick Chat is designed for a wide range of users, including:

- Students: For project discussions and quick communication
- Developers: For understanding real-time web applications
- Teams: For fast internal communication
- Beginners: To learn real-time messaging concepts

Project Goals and Objectives:-
The primary goal of Quick Chat is to provide a fast and reliable real-time messaging platform.

Objectives include:

User-Friendly Interface:
Design a simple and intuitive chat interface that allows users to send and receive messages effortlessly.

Real-Time Communication:
Ensure instant message delivery without page refresh using real-time communication techniques.

Efficient Message Handling:
Enable smooth message transmission between sender and receiver with minimal delay.

Key Features:-

- Real-Time Chat:
  Users can send and receive messages instantly.
- Simple Chat Interface:
  Clean and minimal UI for easy interaction.
- Live Message Updates:
  Messages appear in real time without reloading the page.
- Responsive Design:
  The application works smoothly on different screen sizes

PRE-REQUISITES:-

Here are the key prerequisites for developing the Quick Chat application:

Node.js and npm:

Node.js is a JavaScript runtime environment used to execute JavaScript on the server side. npm is used to manage project dependencies.

- Download: https://nodejs.org/en/download/
- Installation guide: https://nodejs.org/en/download/package-manager/

JavaScript:

JavaScript is used for handling client-side logic, message sending, and real-time updates.

Frontend Framework / Library (if used):

React.js or plain JavaScript is used to build the user interface and manage UI updates dynamically.

Version Control:

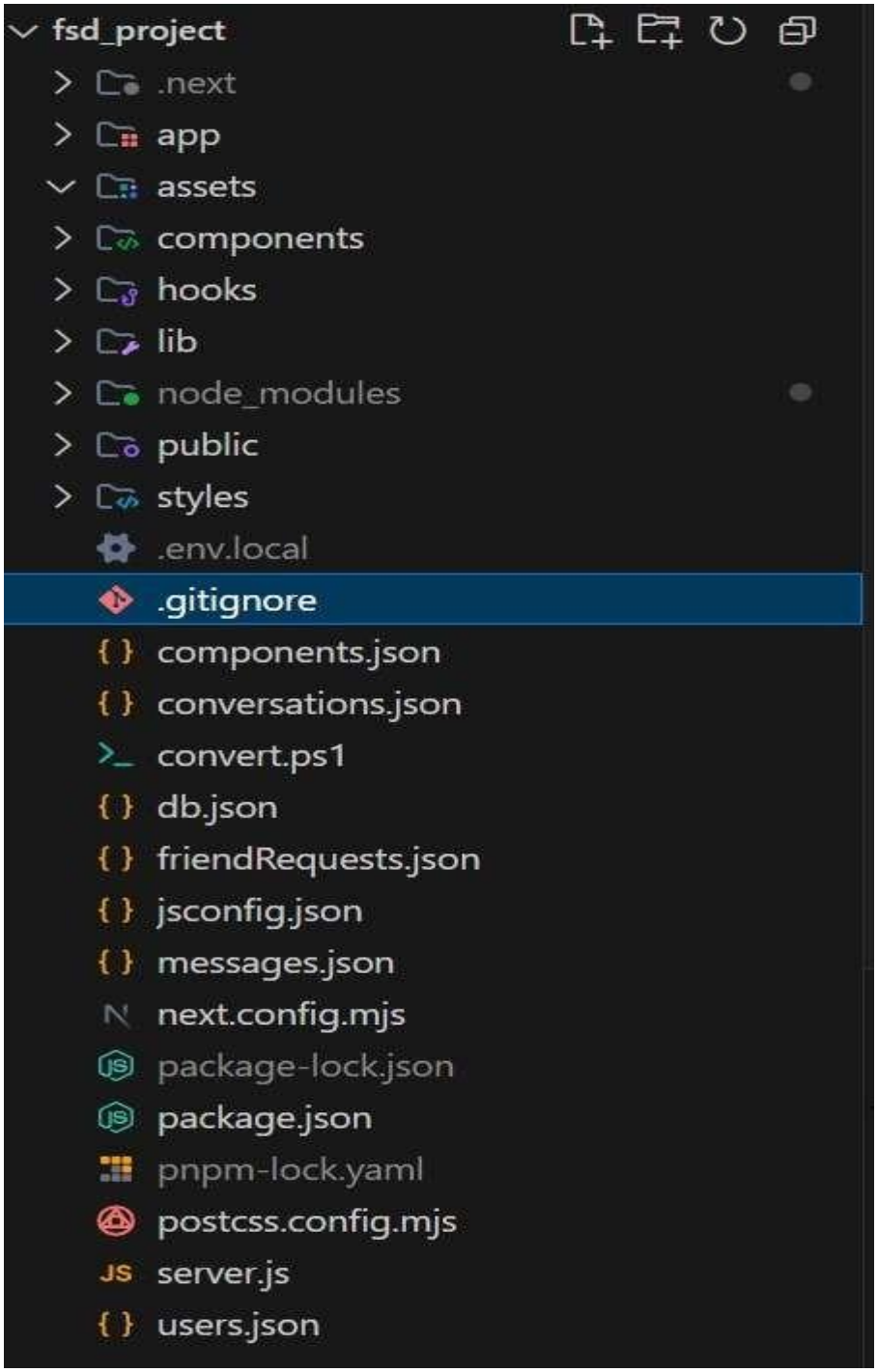
Git is used for version control and code management.

- Download Git: https://git-scm.com/downloads

Development Environment:

- Visual Studio Code https://code.visualstudio.com/download
- Web Browser (Chrome / Edge)

Project structure:



The project follows a structured and modular approach to improve readability and maintainability.

- Frontend Folder – UI and client-side logic

- Backend Folder – Server-side logic
- Configuration Files – Dependency and setup files

Each file and folder is organized logically to support scalability and future enhancements.

PROJECT FLOW:-

Project demo:

Project demo:

Before starting the project, a demo of the application helps in understanding its functionality.

Demo Link: (Attach demo link if available)

Milestone 1: Project Setup and Configuration:

1. Install required tools and software:

- Install Node.js and npm
- Open the project folder
- Install dependencies using npm Technologies used in this project:
- JavaScript
- Node.js
- Frontend framework/library
- Real-time communication module

Milestone 2: Web Development:

App.js component (Next.js equivalent)2. Design UI components:

```
"use client"

import { useState, useEffect, useRef } from "react"
import { useRouter } from "next/navigation"
import { getUserSession, clearUserSession } from "@/lib/session"
import Sidebar from "../../components/Chat/Sidebar"
import ChatWindow from "../../components/Chat/ChatWindow"
import EmptyState from "../../components/Chat/EmptyState"
import NewChatModal from "../../components/Chat/NewChatModal"
import NewGroupModal from "../../components/Chat/NewGroupModal"
import FriendRequestsList from "../../components/Chat/FriendRequestsList"

export default function ChatPage() {
  const router = useRouter()
  const [currentUser, setCurrentUser] = useState(null)
  const [conversations, setConversations] = useState([])
  const [selectedConversation, setSelectedConversation] = useState(null)
  const [messages, setMessages] = useState([])
  const [users, setUsers] = useState([])
  const [friendRequests, setFriendRequests] = useState({ sent: [], received: [] })
  const [showNewChat, setShowNewChat] = useState(false)
  const [showNewGroup, setShowNewGroup] = useState(false)
  const [loading, setLoading] = useState(true)
  const [sidebarOpen, setSidebarOpen] = useState(true)
  const pollingRef = useRef(null)

  useEffect(() => {
    const user = getUserSession()
    if (!user) {
      router.push("/login")
      return
    }
    setCurrentUser(user)
  }, [router])

  useEffect(() => {
    if (currentUser) {
      fetchConversations()
      fetchUsers()
      fetchFriendRequests()

      pollingRef.current = setInterval(() => {
        fetchConversations()
        fetchFriendRequests()
        if (selectedConversation) {
```

Code Description:-

- app/page.jsx: Acts as the main application component similar to App.js in React.
- Imports required components and styles for rendering the Quick Chat interface.

- Defines the main function that returns JSX for displaying the chat layout.
- Serves as the entry point for rendering the application UI.
- Exports the page component as the default export for routing and rendering.

Root Layout Component

```jsx
import { Geist, Geist_Mono } from 'next/font/google'
import './globals.css'

const _geist = Geist({ subsets: ["latin"] });
const _geistMono = Geist_Mono({ subsets: ["latin"] });

export const metadata = {
    title: 'QuickChat - Real-time Messaging',
    description: 'Connect with friends through real-time messaging',
    icons: {
        icon: '/favicon.svg',
    },
}

export default function RootLayout({
    children,
}) {
    return (
        <html lang="en">
            <body className={`font-sans antialiased`}>
                {children}
            </body>
        </html>
    )
}
```

Code Description:-

- app/layout.jsx: Defines the root layout of the Quick Chat application.
- Wraps all pages with a common HTML and body structure.
- Applies global styles using globals.css.
- Uses {children} to render nested route components.

Chat Page Component

```
"use client"

import { useEffect } from "react"
import { useRouter } from "next/navigation"

export default function Home() {
  const router = useRouter()

  useEffect(() => {

    router.push("/login")
  }, [router])

  return (
    <div className="min-h-screen flex items-center justify-center bg-background">
      <div className="text-center">
        <div className="w-16 h-16 relative mx-auto mb-6">
          <div className="absolute inset-0 rounded-full border-4 border-primary/20"></div>
          <div className="absolute inset-0 rounded-full border-4 border-primary border-t-transp
        </div>
        <h2 className="text-xl font-semibold text-foreground">QuickChat</h2>
        <p className="mt-2 text-muted-foreground">Redirecting to login...</p>
      </div>
    </div>
  )
}

```

Code Description:-

- app/chat/page.jsx: Implements the main chat interface of Quick Chat.
- Displays chat messages between users.
- Integrates message input and message display components.
- Handles rendering of real-time conversations.
- Acts as the core functionality of the application.

Login Page Component

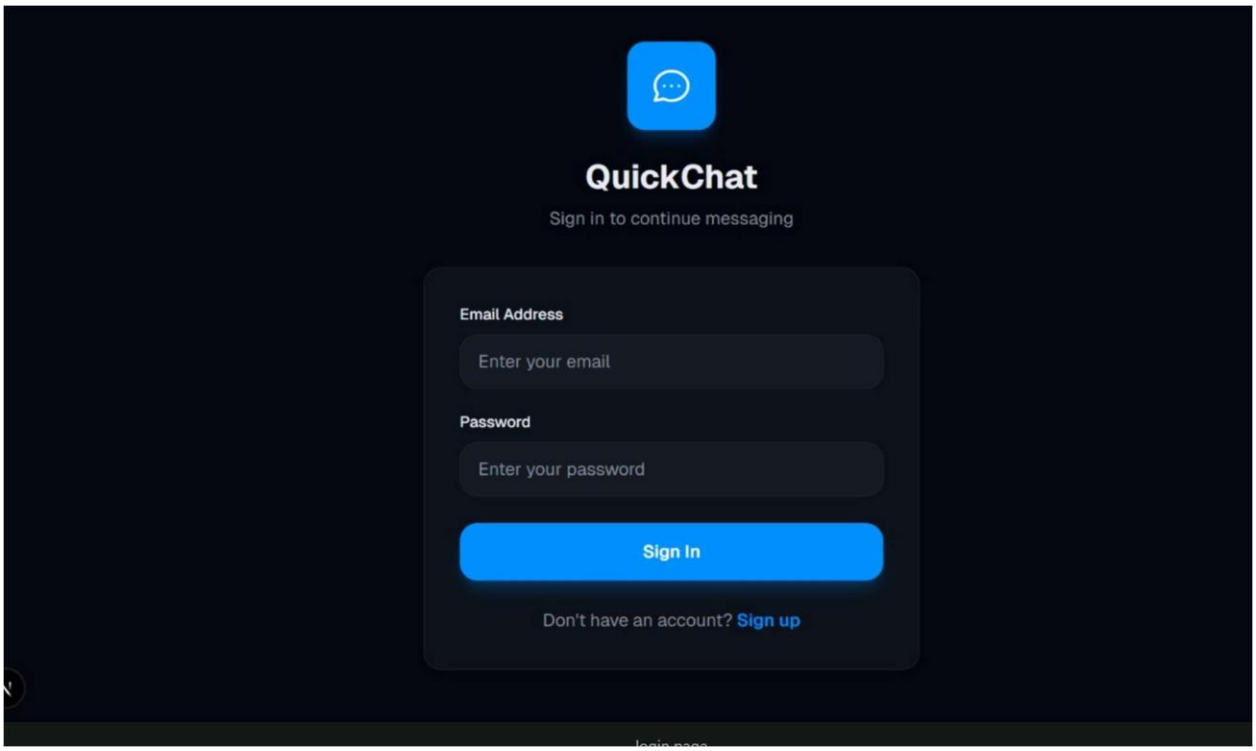
```jsx
"use client"

import { useState } from "react"
import { useRouter } from "next/navigation"
import Link from "next/link"
import { setUserSession } from "@/lib/session"

export default function LoginPage() {
  const router = useRouter()
  const [formData, setFormData] = useState({
    email: "",
    password: "",
  })
  const [error, setError] = useState("")
  const [loading, setLoading] = useState(false)

  const handleChange = (e) => {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value,
    })
  }

  const handleSubmit = async (e) => {
    e.preventDefault()
    setError("")
    setLoading(true)

    try {
      const response = await fetch("/api/auth/login", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(formData),
      })

      const data = await response.json()

      if (response.ok) {
        // Use session cookie instead of localStorage
        setUserSession(data.user)
        router.push("/chat")
      } else {
        setError(data.error || "Login failed")
```

Code Description:-

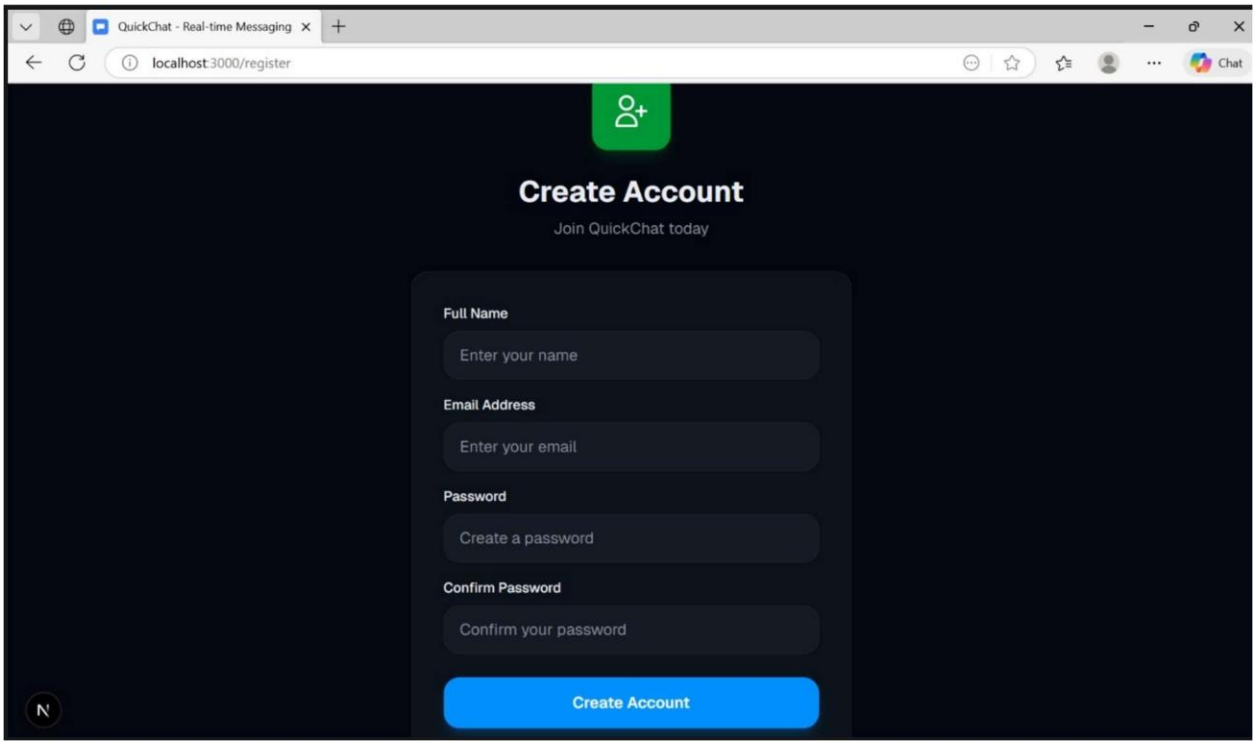
- app/login/page.jsx: Provides the login interface for users.
- Contains form elements for user credentials.
- Handles user authentication input.
- Navigates users to chat page upon successful login.

Project Execution:

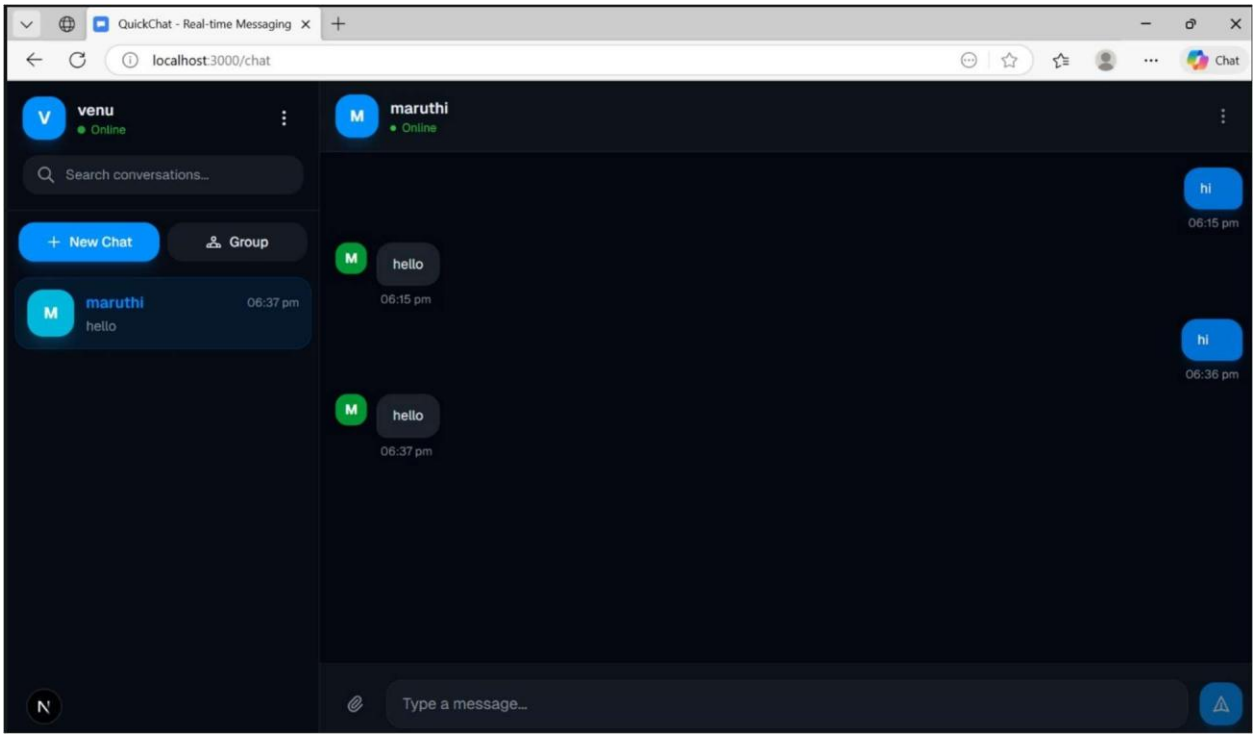
Login page



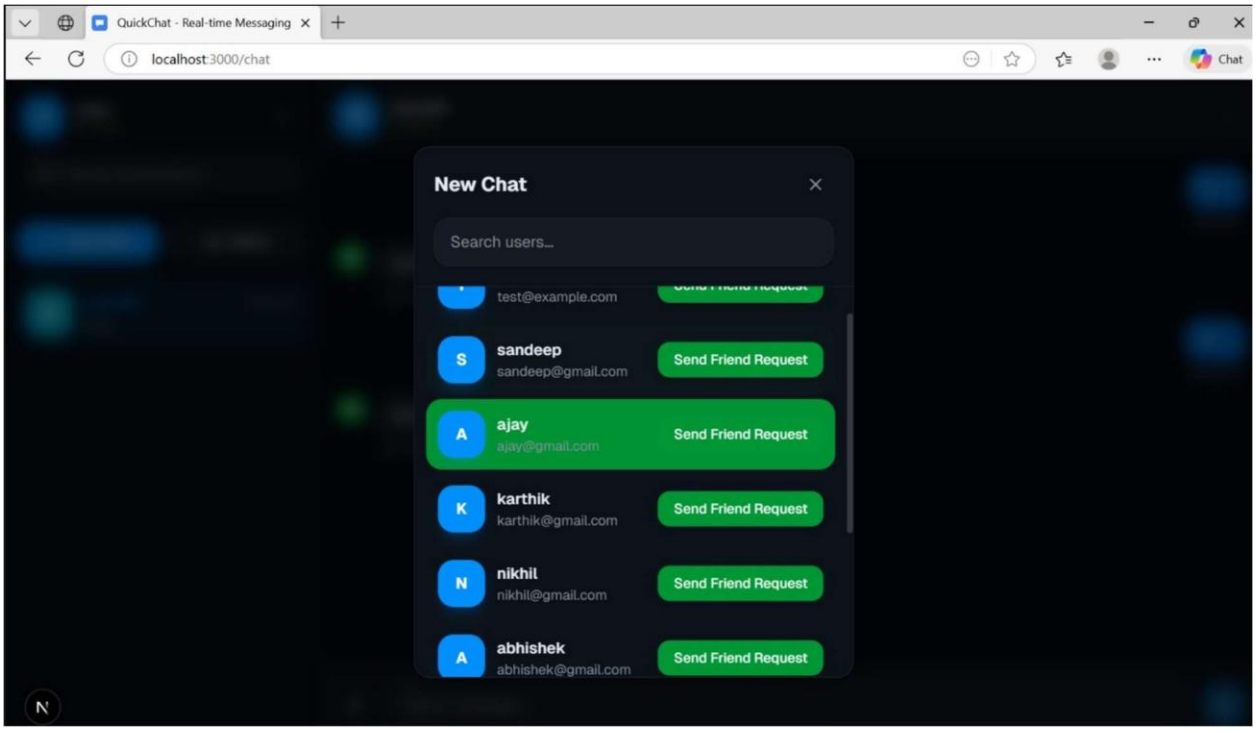
Sign up page



User chat page



Functionality



Project Demo and code explanation link:

https://drive.google.com/drive/u/0/folders/1HdahF_CwCqwtDLYBp1JRhFQnXA7CO3fk