perplexity

Building a Complete YouTube Clone with React, Express, and PostgreSQL

This comprehensive guide will walk you through creating a fully-functional YouTube clone using modern web technologies. We'll build a robust backend with Express.js, create an interactive frontend with React and Vite, and manage our data using Prisma with PostgreSQL. By the end of this tutorial, you'll have a deep understanding of how video streaming platforms work and the technical skills to build one yourself.

Project Overview and Architecture

Our YouTube clone will be a full-stack application with a clear separation between frontend and backend. The application will feature video uploads, streaming, user authentication, comments, likes, subscriptions, and personalized recommendations.

Key Technologies

- **Backend**: Node.js with Express
- **Frontend**: React with Vite (for faster development)
- **Database**: PostgreSQL with Prisma ORM
- **Authentication**: JWT (JSON Web Tokens)
- **Video Storage**: Local storage (can be extended to cloud storage)

Project Structure

```
youtube-clone/
├── server/             # Express backend
│   ├── src/
│   │   ├── routes/     # API endpoints
│   │   ├── controllers/ # Business logic
│   │   ├── middleware/ # Auth and validation
│   │   ├── prisma/     # Database schema and migrations
│   │   └── index.js    # Server entry point
│   └── package.json
└── client/             # React frontend with Vite
    ├── src/
    │   ├── components/ # Reusable UI components
    │   ├── pages/      # Application pages
    │   ├── hooks/      # Custom React hooks
    │   ├── context/    # State management
    │   ├── utils/      # Helper functions
```

```
    │   └── App.jsx     # Main component
    └── package.json
```

Setting Up the Backend

Let's start by creating our Express backend with Prisma integration.

Step 1: Setting Up the Project

First, create the backend directory and initialize our Node.js project:

```
mkdir youtube-clone
cd youtube-clone
mkdir server
cd server
npm init -y
```

Install the necessary dependencies:

```
npm install express prisma @prisma/client cors cookie-parser dotenv jsonwebtoken bcryptjs
npm install -D nodemon
```

Step 2: Modeling Our Data with Prisma

Setting up Prisma and connecting it to our PostgreSQL database:

```
npx prisma init
```

This creates a `prisma` directory with a `schema.prisma` file. Let's modify this file to define our database models:

```
// prisma/schema.prisma

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider = "prisma-client-js"
}

model User {
  id            String         @id @default(uuid())
  createdAt     DateTime       @default(now())
  username      String
  email         String         @unique
  password      String
  avatar        String         @default("default-avatar.png")
```

```
  cover         String         @default("default-cover-banner.png")
  about         String         @default("")
  videos        Video[]
  videoLikes    VideoLike[]
  comments      Comment[]
  subscribers   Subscription[] @relation("subscriber")
  subscribedTo  Subscription[] @relation("subscribedTo")
  views         View[]
}

model Comment {
  id        String   @id @default(uuid())
  createdAt DateTime @default(now())
  text      String
  userId    String
  videoId   String
  user      User     @relation(fields: [userId], references: [id])
  video     Video    @relation(fields: [videoId], references: [id])
}

model Subscription {
  id             String   @id @default(uuid())
  createdAt      DateTime @default(now())
  subscriberId   String
  subscribedToId String
  subscriber     User     @relation("subscriber", fields: [subscriberId], references: [id
  subscribedTo   User     @relation("subscribedTo", fields: [subscribedToId], references:
}

model Video {
  id          String      @id @default(uuid())
  createdAt   DateTime    @default(now())
  title       String
  description String?
  url         String
  thumbnail   String
  userId      String
  user        User        @relation(fields: [userId], references: [id])
  videoLikes  VideoLike[]
  comments    Comment[]
  views       View[]
}

model VideoLike {
  id        String   @id @default(uuid())
  createdAt DateTime @default(now())
  like      Int      @default(0) // 1 for like, -1 for dislike, 0 for neutral
  userId    String
  videoId   String
  user      User     @relation(fields: [userId], references: [id])
  video     Video    @relation(fields: [videoId], references: [id])
}

model View {
  id        String   @id @default(uuid())
  createdAt DateTime @default(now())
```

```
  userId    String?
  videoId   String
  user      User?    @relation(fields: [userId], references: [id])
  video     Video    @relation(fields: [videoId], references: [id])
}
```

After defining our schema, let's create the database tables:

```
npx prisma migrate dev --name init
```

Step 3: Creating the Express Server

Now, let's set up our Express server. Create a file `src/index.js`:

```
// src/index.js
const express = require('express');
const cors = require('cors');
const cookieParser = require('cookie-parser');
const dotenv = require('dotenv');

// Load environment variables
dotenv.config();

// Import routes
const authRoutes = require('./routes/auth');
const videoRoutes = require('./routes/video');
const userRoutes = require('./routes/user');

const app = express();
const PORT = process.env.PORT || 3001;

// Middleware
app.use(express.json());
app.use(cookieParser());
app.use(cors({
  origin: process.env.CLIENT_URL || 'http://localhost:5173',
  credentials: true
}));

// Static file serving for uploaded videos and images
app.use('/uploads', express.static('uploads'));

// Routes
app.use('/api/v1/auth', authRoutes);
app.use('/api/v1/videos', videoRoutes);
app.use('/api/v1/users', userRoutes);

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  const status = err.statusCode || 500;
  const message = err.message || 'Something went wrong';
  res.status(status).json({ message });
```

```
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Step 4: Creating Authentication Middleware

Create a middleware to protect routes that require authentication:

```
// src/middleware/auth.js
const jwt = require('jsonwebtoken');
const { PrismaClient } = require('@prisma/client');

const prisma = new PrismaClient();

exports.protect = async (req, res, next) => {
  try {
    // Get token from cookies
    const token = req.cookies.token;

    if (!token) {
      return res.status(401).json({ message: 'You need to be logged in to access this rou
    }

    // Verify token
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Get user from database
    const user = await prisma.user.findUnique({
      where: {
        id: decoded.id
      }
    });

    if (!user) {
      return res.status(401).json({ message: 'User not found' });
    }

    // Add user to request object
    req.user = user;
    next();
  } catch (error) {
    return res.status(401).json({ message: 'Not authorized to access this route' });
  }
};
```

Step 5: Creating API Routes

Let's create our API routes starting with authentication:

```
// src/routes/auth.js
const express = require('express');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { PrismaClient } = require('@prisma/client');

const router = express.Router();
const prisma = new PrismaClient();

// Register user
router.post('/signup', async (req, res, next) => {
  try {
    const { username, email, password } = req.body;

    // Check if user exists
    const userExists = await prisma.user.findUnique({
      where: {
        email
      }
    });

    if (userExists) {
      return res.status(400).json({ message: 'User already exists' });
    }

    // Hash password
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    // Create user
    const user = await prisma.user.create({
      data: {
        username,
        email,
        password: hashedPassword
      }
    });

    // Create token
    const token = jwt.sign({ id: user.id }, process.env.JWT_SECRET, {
      expiresIn: '30d'
    });

    // Set cookie
    res.cookie('token', token, {
      httpOnly: true,
      maxAge: 30 * 24 * 60 * 60 * 1000, // 30 days
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'strict'
    });

    // Return user data without password
```

```
    const { password: _, ...userData } = user;

    res.status(201).json({
      user: userData
    });
  } catch (error) {
    next(error);
  }
});

// Login user
router.post('/login', async (req, res, next) => {
  try {
    const { email, password } = req.body;

    // Check if user exists
    const user = await prisma.user.findUnique({
      where: {
        email
      }
    });

    if (!user) {
      return res.status(400).json({ message: 'Invalid credentials' });
    }

    // Check password
    const isMatch = await bcrypt.compare(password, user.password);

    if (!isMatch) {
      return res.status(400).json({ message: 'Invalid credentials' });
    }

    // Create token
    const token = jwt.sign({ id: user.id }, process.env.JWT_SECRET, {
      expiresIn: '30d'
    });

    // Set cookie
    res.cookie('token', token, {
      httpOnly: true,
      maxAge: 30 * 24 * 60 * 60 * 1000, // 30 days
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'strict'
    });

    // Return user data without password
    const { password: _, ...userData } = user;

    res.status(200).json({
      user: userData
    });
  } catch (error) {
    next(error);
  }
});
```

```
// Logout user
router.post('/logout', (req, res) => {
  res.clearCookie('token');
  res.status(200).json({ message: 'Logged out successfully' });
});

// Get current user
router.get('/me', async (req, res, next) => {
  try {
    const token = req.cookies.token;

    if (!token) {
      return res.status(401).json({ message: 'Not authorized' });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    const user = await prisma.user.findUnique({
      where: {
        id: decoded.id
      }
    });

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    const { password: _, ...userData } = user;

    res.status(200).json({
      user: userData
    });
  } catch (error) {
    next(error);
  }
});

module.exports = router;
```

Next, let's create the video routes:

```
// src/routes/video.js
const express = require('express');
const { PrismaClient } = require('@prisma/client');
const multer = require('multer');
const path = require('path');
const fs = require('fs');
const { protect } = require('../middleware/auth');

const router = express.Router();
const prisma = new PrismaClient();

// Configure multer for file uploads
const storage = multer.diskStorage({
```

```
  destination: (req, file, cb) => {
    const dest = file.fieldname === 'video' ? 'uploads/videos' : 'uploads/thumbnails';

    // Create directory if it doesn't exist
    if (!fs.existsSync(dest)) {
      fs.mkdirSync(dest, { recursive: true });
    }

    cb(null, dest);
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    const ext = path.extname(file.originalname);
    cb(null, file.fieldname + '-' + uniqueSuffix + ext);
  }
});

const upload = multer({
  storage,
  limits: {
    fileSize: 500 * 1024 * 1024 // 500MB limit
  }
});

// Helper function to get video views
async function getVideoViews(videos) {
  for (const video of videos) {
    const views = await prisma.view.count({
      where: {
        videoId: video.id
      }
    });

    video.views = views;
  }

  return videos;
}

// Get recommended videos (homepage)
router.get('/', async (req, res, next) => {
  try {
    let videos = await prisma.video.findMany({
      include: {
        user: true
      },
      orderBy: {
        createdAt: 'desc'
      }
    });

    if (videos.length === 0) {
      return res.status(200).json({ videos: [] });
    }

    videos = await getVideoViews(videos);
```

```
    res.status(200).json({ videos });
  } catch (error) {
    next(error);
  }
});

// Get trending videos
router.get('/trending', async (req, res, next) => {
  try {
    let videos = await prisma.video.findMany({
      include: {
        user: true
      },
      orderBy: {
        createdAt: 'desc'
      }
    });

    if (videos.length === 0) {
      return res.status(200).json({ videos: [] });
    }

    videos = await getVideoViews(videos);

    // Sort by view count
    videos.sort((a, b) => b.views - a.views);

    res.status(200).json({ videos });
  } catch (error) {
    next(error);
  }
});

// Get video by ID
router.get('/:videoId', async (req, res, next) => {
  try {
    const { videoId } = req.params;

    const video = await prisma.video.findUnique({
      where: {
        id: videoId
      },
      include: {
        user: true
      }
    });

    if (!video) {
      return res.status(404).json({ message: 'Video not found' });
    }

    // Get view count
    const views = await prisma.view.count({
      where: {
        videoId
```

```
      }
    });

    video.views = views;

    // Record a view
    await prisma.view.create({
      data: {
        videoId,
        userId: req.cookies.token ? jwt.verify(req.cookies.token, process.env.JWT_SECRET)
      }
    });

    res.status(200).json({ video });
  } catch (error) {
    next(error);
  }
});

// Upload a video
router.post(
  '/',
  protect,
  upload.fields([
    { name: 'video', maxCount: 1 },
    { name: 'thumbnail', maxCount: 1 }
  ]),
  async (req, res, next) => {
    try {
      const { title, description } = req.body;

      if (!req.files || !req.files.video || !req.files.thumbnail) {
        return res.status(400).json({ message: 'Please upload a video and thumbnail' });
      }

      const videoPath = req.files.video[^0].path;
      const thumbnailPath = req.files.thumbnail[^0].path;

      const video = await prisma.video.create({
        data: {
          title,
          description,
          url: videoPath.replace(/\\/g, '/'),
          thumbnail: thumbnailPath.replace(/\\/g, '/'),
          userId: req.user.id
        }
      });

      res.status(201).json({ video });
    } catch (error) {
      next(error);
    }
  }
);

// Add like/dislike to video
```

```
router.post('/:videoId/like', protect, async (req, res, next) => {
  try {
    const { videoId } = req.params;
    const { like } = req.body; // 1 for like, -1 for dislike, 0 for neutral

    // Check if video exists
    const videoExists = await prisma.video.findUnique({
      where: {
        id: videoId
      }
    });

    if (!videoExists) {
      return res.status(404).json({ message: 'Video not found' });
    }

    // Check if user already liked/disliked the video
    const existingLike = await prisma.videoLike.findFirst({
      where: {
        userId: req.user.id,
        videoId
      }
    });

    if (existingLike) {
      // Update existing like
      const updatedLike = await prisma.videoLike.update({
        where: {
          id: existingLike.id
        },
        data: {
          like
        }
      });

      return res.status(200).json({ like: updatedLike });
    }

    // Create new like
    const newLike = await prisma.videoLike.create({
      data: {
        like,
        userId: req.user.id,
        videoId
      }
    });

    res.status(201).json({ like: newLike });
  } catch (error) {
    next(error);
  }
});

// Add comment to video
router.post('/:videoId/comments', protect, async (req, res, next) => {
  try {
```

```
    const { videoId } = req.params;
    const { text } = req.body;

    // Check if video exists
    const videoExists = await prisma.video.findUnique({
      where: {
        id: videoId
      }
    });

    if (!videoExists) {
      return res.status(404).json({ message: 'Video not found' });
    }

    // Create comment
    const comment = await prisma.comment.create({
      data: {
        text,
        userId: req.user.id,
        videoId
      },
      include: {
        user: true
      }
    });

    res.status(201).json({ comment });
  } catch (error) {
    next(error);
  }
});

// Get comments for a video
router.get('/:videoId/comments', async (req, res, next) => {
  try {
    const { videoId } = req.params;

    const comments = await prisma.comment.findMany({
      where: {
        videoId
      },
      include: {
        user: true
      },
      orderBy: {
        createdAt: 'desc'
      }
    });

    res.status(200).json({ comments });
  } catch (error) {
    next(error);
  }
});

module.exports = router;
```

Finally, let's create the user routes:

```
// src/routes/user.js
const express = require('express');
const { PrismaClient } = require('@prisma/client');
const multer = require('multer');
const path = require('path');
const fs = require('fs');
const { protect } = require('../middleware/auth');

const router = express.Router();
const prisma = new PrismaClient();

// Configure multer for profile images
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    const dest = 'uploads/avatars';

    // Create directory if it doesn't exist
    if (!fs.existsSync(dest)) {
      fs.mkdirSync(dest, { recursive: true });
    }

    cb(null, dest);
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    const ext = path.extname(file.originalname);
    cb(null, 'avatar-' + uniqueSuffix + ext);
  }
});

const upload = multer({
  storage,
  limits: {
    fileSize: 5 * 1024 * 1024 // 5MB limit
  }
});

// Get user profile
router.get('/:userId', async (req, res, next) => {
  try {
    const { userId } = req.params;

    const user = await prisma.user.findUnique({
      where: {
        id: userId
      },
      select: {
        id: true,
        createdAt: true,
        username: true,
        email: true,
        avatar: true,
        cover: true,
        about: true
```

```
      }
    });

    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    // Get subscriber count
    const subscriberCount = await prisma.subscription.count({
      where: {
        subscribedToId: userId
      }
    });

    // Get user's videos
    const videos = await prisma.video.findMany({
      where: {
        userId
      },
      orderBy: {
        createdAt: 'desc'
      }
    });

    // Get view count for each video
    let totalViews = 0;
    for (const video of videos) {
      const viewCount = await prisma.view.count({
        where: {
          videoId: video.id
        }
      });
      video.views = viewCount;
      totalViews += viewCount;
    }

    res.status(200).json({
      user,
      subscriberCount,
      videos,
      totalViews
    });
  } catch (error) {
    next(error);
  }
});

// Update user profile
router.put('/profile', protect, upload.single('avatar'), async (req, res, next) => {
  try {
    const { username, about } = req.body;
    const userId = req.user.id;

    const updateData = {
      username,
      about
```

```
    };

    // If avatar is uploaded, update it
    if (req.file) {
      updateData.avatar = req.file.path.replace(/\\/g, '/');
    }

    const updatedUser = await prisma.user.update({
      where: {
        id: userId
      },
      data: updateData
    });

    const { password: _, ...userData } = updatedUser;

    res.status(200).json({
      user: userData
    });
  } catch (error) {
    next(error);
  }
});

// Subscribe to a channel
router.post('/:userId/subscribe', protect, async (req, res, next) => {
  try {
    const { userId } = req.params;
    const subscriberId = req.user.id;

    // Check if user exists
    const userExists = await prisma.user.findUnique({
      where: {
        id: userId
      }
    });

    if (!userExists) {
      return res.status(404).json({ message: 'User not found' });
    }

    // Check if already subscribed
    const existingSubscription = await prisma.subscription.findFirst({
      where: {
        subscriberId,
        subscribedToId: userId
      }
    });

    if (existingSubscription) {
      return res.status(400).json({ message: 'Already subscribed to this channel' });
    }

    // Create subscription
    const subscription = await prisma.subscription.create({
      data: {
```

```
        subscriberId,
        subscribedToId: userId
      }
    });

    res.status(201).json({ subscription });
  } catch (error) {
    next(error);
  }
});

// Unsubscribe from a channel
router.delete('/:userId/subscribe', protect, async (req, res, next) => {
  try {
    const { userId } = req.params;
    const subscriberId = req.user.id;

    // Check if subscription exists
    const subscription = await prisma.subscription.findFirst({
      where: {
        subscriberId,
        subscribedToId: userId
      }
    });

    if (!subscription) {
      return res.status(400).json({ message: 'Not subscribed to this channel' });
    }

    // Delete subscription
    await prisma.subscription.delete({
      where: {
        id: subscription.id
      }
    });

    res.status(200).json({ message: 'Unsubscribed successfully' });
  } catch (error) {
    next(error);
  }
});

// Get subscribed channels
router.get('/subscriptions', protect, async (req, res, next) => {
  try {
    const subscriptions = await prisma.subscription.findMany({
      where: {
        subscriberId: req.user.id
      },
      include: {
        subscribedTo: true
      }
    });

    const channels = subscriptions.map(sub => {
      const { password: _, ...userData } = sub.subscribedTo;
```

```
      return userData;
    });

    res.status(200).json({ channels });
  } catch (error) {
    next(error);
  }
});

// Get liked videos
router.get('/liked-videos', protect, async (req, res, next) => {
  try {
    const likedVideos = await prisma.videoLike.findMany({
      where: {
        userId: req.user.id,
        like: 1
      },
      include: {
        video: {
          include: {
            user: true
          }
        }
      },
      orderBy: {
        createdAt: 'desc'
      }
    });

    const videos = likedVideos.map(like => like.video);

    // Get view count for each video
    for (const video of videos) {
      const viewCount = await prisma.view.count({
        where: {
          videoId: video.id
        }
      });
      video.views = viewCount;
    }

    res.status(200).json({ videos });
  } catch (error) {
    next(error);
  }
});

module.exports = router;
```

Setting Up the Frontend with Vite

Now, let's move on to setting up our React frontend with Vite.

Step 1: Creating the Vite Project

First, navigate back to the root of our project and create a new Vite app:

```
cd ..
npm create vite@latest client -- --template react
cd client
npm install
```

Step 2: Installing Dependencies

```
npm install react-router-dom axios react-icons react-player styled-components jwt-decode
```

Step 3: Setting Up the Application Structure

Let's start by configuring our routes. Create a new file `src/App.jsx`:

```
// src/App.jsx
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { useState, useEffect } from 'react';
import { AuthProvider } from './context/AuthContext';
import PrivateRoute from './components/PrivateRoute';

// Layout Components
import Navbar from './components/Navbar';
import Sidebar from './components/Sidebar';
import MobileNavbar from './components/MobileNavbar';

// Pages
import Home from './pages/Home';
import WatchVideo from './pages/WatchVideo';
import Channel from './pages/Channel';
import SearchResults from './pages/SearchResults';
import Trending from './pages/Trending';
import Subscriptions from './pages/Subscriptions';
import Library from './pages/Library';
import History from './pages/History';
import LikedVideos from './pages/LikedVideos';
import YourVideos from './pages/YourVideos';
import UploadVideo from './pages/UploadVideo';
import SignIn from './pages/SignIn';
import SignUp from './pages/SignUp';
import NotFound from './pages/NotFound';

function App() {
  const [isSidebarOpen, setSidebarOpen] = useState(false);
```

```
  const toggleSidebar = () => {
    setSidebarOpen(!isSidebarOpen);
  };

  const closeSidebar = () => {
    setSidebarOpen(false);
  };

  // Close sidebar on route change
  useEffect(() => {
    closeSidebar();
  }, [window.location.pathname]);

  return (
    <AuthProvider>
      <Router>
        <div className="app-container">
          <Navbar toggleSidebar={toggleSidebar} />
          <Sidebar isOpen={isSidebarOpen} closeSidebar={closeSidebar} />
          <MobileNavbar />

          <main className="content">
            <Routes>
              <Route path="/" element={<Home />} />
              <Route path="/watch/:videoId" element={<WatchVideo />} />
              <Route path="/channel/:channelId" element={<Channel />} />
              <Route path="/results/:searchQuery" element={<SearchResults />} />
              <Route path="/trending" element={<Trending />} />
              <Route path="/subscriptions" element={<PrivateRoute><Subscriptions /></Priv
              <Route path="/library" element={<PrivateRoute><Library /></PrivateRoute>} /
              <Route path="/history" element={<PrivateRoute><History /></PrivateRoute>} /
              <Route path="/liked-videos" element={<PrivateRoute><LikedVideos /></Private
              <Route path="/your-videos" element={<PrivateRoute><YourVideos /></PrivateRo
              <Route path="/upload" element={<PrivateRoute><UploadVideo /></PrivateRoute>
              <Route path="/signin" element={<SignIn />} />
              <Route path="/signup" element={<SignUp />} />
              <Route path="*" element={<NotFound />} />
            </Routes>
          </main>
        </div>
      </Router>
    </AuthProvider>
  );
}

export default App;
```

Step 4: Creating the Authentication Context

Let's create an auth context to manage authentication state across our app:

```
// src/context/AuthContext.jsx
import { createContext, useState, useEffect } from 'react';
import axios from 'axios';
import jwtDecode from 'jwt-decode';
```

```
export const AuthContext = createContext();

axios.defaults.baseURL = 'http://localhost:3001/api/v1';
axios.defaults.withCredentials = true;

export const AuthProvider = ({ children }) => {
  const [currentUser, setCurrentUser] = useState(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    checkUserLoggedIn();
  }, []);

  // Check if user is logged in
  const checkUserLoggedIn = async () => {
    try {
      const res = await axios.get('/auth/me');
      setCurrentUser(res.data.user);
    } catch (error) {
      console.error('Not authenticated', error);
      setCurrentUser(null);
    } finally {
      setIsLoading(false);
    }
  };

  // Register user
  const register = async (userData) => {
    try {
      const res = await axios.post('/auth/signup', userData);
      setCurrentUser(res.data.user);
      return { success: true, data: res.data };
    } catch (error) {
      return {
        success: false,
        message: error.response?.data?.message || 'An error occurred during registration'
      };
    }
  };

  // Login user
  const login = async (userData) => {
    try {
      const res = await axios.post('/auth/login', userData);
      setCurrentUser(res.data.user);
      return { success: true, data: res.data };
    } catch (error) {
      return {
        success: false,
        message: error.response?.data?.message || 'Invalid credentials'
      };
    }
  };

  // Logout user
```

```
  const logout = async () => {
    try {
      await axios.post('/auth/logout');
      setCurrentUser(null);
      return { success: true };
    } catch (error) {
      return {
        success: false,
        message: 'Error logging out'
      };
    }
  };

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

Step 5: Creating Reusable Components

Let's create some components we'll need for our app. First, the `PrivateRoute` component:

```
// src/components/PrivateRoute.jsx
import { useContext } from 'react';
import { Navigate } from 'react-router-dom';
import { AuthContext } from '../context/AuthContext';

const PrivateRoute = ({ children }) => {
  const { currentUser, isLoading } = useContext(AuthContext);

  if (isLoading) {
    return <div className="loading">Loading...</div>;
  }

  if (!currentUser) {
    return <Navigate to="/signin" />;
  }

  return children;
};

export default PrivateRoute;
```

Next, let's create the Navbar component:

```
// src/components/Navbar.jsx
import { useState, useContext } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { AuthContext } from '../context/AuthContext';
import { FiMenu, FiSearch, FiVideo, FiBell, FiUser } from 'react-icons/fi';
import styled from 'styled-components';

const NavbarContainer = styled.nav`
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 0 16px;
  height: 56px;
  background-color: white;
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  z-index: 100;
  border-bottom: 1px solid #e5e5e5;
`;

const LogoSection = styled.div`
  display: flex;
  align-items: center;

  .menu-icon {
    margin-right: 24px;
    cursor: pointer;
  }

  .logo {
    display: flex;
    align-items: center;
    color: #000;
    text-decoration: none;
    font-weight: bold;
    font-size: 20px;

    svg {
      color: red;
      font-size: 24px;
      margin-right: 5px;
    }
  }
`;

const SearchSection = styled.div`
  display: flex;
  align-items: center;
  flex: 1;
  max-width: 600px;
  margin: 0 40px;

  form {
    display: flex;
```

```
    width: 100%;
  }

  input {
    flex: 1;
    padding: 8px 12px;
    border: 1px solid #ccc;
    border-right: none;
    border-radius: 2px 0 0 2px;
    font-size: 14px;
    height: 38px;
  }

  button {
    padding: 0 16px;
    background-color: #f8f8f8;
    border: 1px solid #ccc;
    border-radius: 0 2px 2px 0;
    cursor: pointer;
    height: 38px;
  }
`;

const UserSection = styled.div`
  display: flex;
  align-items: center;

  .icon-button {
    margin-left: 16px;
    cursor: pointer;
    font-size: 20px;
  }

  .user-avatar {
    width: 32px;
    height: 32px;
    border-radius: 50%;
    margin-left: 16px;
    cursor: pointer;
  }

  .signin-button {
    display: flex;
    align-items: center;
    color: #065fd4;
    border: 1px solid #065fd4;
    padding: 8px 12px;
    border-radius: 2px;
    margin-left: 16px;
    text-decoration: none;
    font-size: 14px;

    svg {
      margin-right: 8px;
    }
  }
```

```
`;

const Navbar = ({ toggleSidebar }) => {
  const [searchTerm, setSearchTerm] = useState('');
  const { currentUser } = useContext(AuthContext);
  const navigate = useNavigate();

  const handleSubmit = (e) => {
    e.preventDefault();
    if (searchTerm.trim()) {
      navigate(`/results/${searchTerm}`);
      setSearchTerm('');
    }
  };

  return (
    <NavbarContainer>
      <LogoSection>
        <FiMenu className="menu-icon" onClick={toggleSidebar} />
        <Link to="/" className="logo">
          <FiVideo /> YouTube Clone
        </Link>
      </LogoSection>

      <SearchSection>
        <form onSubmit={handleSubmit}>
          <input
            type="text"
            placeholder="Search"
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
          />
          <button type="submit">
            <FiSearch />
          </button>
        </form>
      </SearchSection>

      <UserSection>
        {currentUser ? (
          <>
            <Link to="/upload" className="icon-button">
              <FiVideo />
            </Link>
            <div className="icon-button">
              <FiBell />
            </div>
            <Link to={`/channel/${currentUser.id}`}>
              <img
                src={currentUser.avatar}
                alt="User Avatar"
                className="user-avatar"
              />
            </Link>
          </>
        ) : (
```

```
          <Link to="/signin" className="signin-button">
            <FiUser /> SIGN IN
          </Link>
        )}
      </UserSection>
    </NavbarContainer>
  );
};

export default Navbar;
```

Let's also create a VideoCard component to display video thumbnails:

```
// src/components/VideoCard.jsx
import { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import styled from 'styled-components';
import axios from 'axios';

const Card = styled.div`
  width: 100%;
  margin-bottom: 20px;
`;

const Thumbnail = styled.div`
  position: relative;
  width: 100%;
  height: 0;
  padding-top: 56.25%; // 16:9 aspect ratio
  background-color: #f9f9f9;
  overflow: hidden;

  img {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    object-fit: cover;
  }

  .duration {
    position: absolute;
    right: 5px;
    bottom: 5px;
    background-color: rgba(0, 0, 0, 0.7);
    color: white;
    padding: 2px 4px;
    border-radius: 2px;
    font-size: 12px;
  }
`;

const VideoInfo = styled.div`
  display: flex;
```

```
  margin-top: 10px;
`;

const ChannelAvatar = styled.div`
  width: 36px;
  height: 36px;
  margin-right: 12px;

  img {
    width: 100%;
    height: 100%;
    border-radius: 50%;
  }
`;

const VideoDetails = styled.div`
  flex: 1;

  h3 {
    margin: 0 0 4px 0;
    font-size: 16px;
    font-weight: 500;
    line-height: 1.4;
    color: #030303;
    overflow: hidden;
    text-overflow: ellipsis;
    display: -webkit-box;
    -webkit-line-clamp: 2;
    -webkit-box-orient: vertical;
  }

  .channel-name {
    color: #606060;
    font-size: 14px;
    margin-bottom: 2px;
  }

  .video-meta {
    color: #606060;
    font-size: 14px;
  }
`;

// Function to format view count
const formatViewCount = (count) => {
  if (count >= 1000000) {
    return `${(count / 1000000).toFixed(1)}M`;
  } else if (count >= 1000) {
    return `${(count / 1000).toFixed(1)}K`;
  }
  return count;
};

// Function to format time since upload
const timeAgo = (date) => {
  const seconds = Math.floor((new Date() - new Date(date)) / 1000);
```

```
  let interval = seconds / 31536000;
  if (interval > 1) return `${Math.floor(interval)} years ago`;

  interval = seconds / 2592000;
  if (interval > 1) return `${Math.floor(interval)} months ago`;

  interval = seconds / 86400;
  if (interval > 1) return `${Math.floor(interval)} days ago`;

  interval = seconds / 3600;
  if (interval > 1) return `${Math.floor(interval)} hours ago`;

  interval = seconds / 60;
  if (interval > 1) return `${Math.floor(interval)} minutes ago`;

  return `${Math.floor(seconds)} seconds ago`;
};

const VideoCard = ({ video }) => {
  const [views, setViews] = useState(0);

  useEffect(() => {
    if (video.views) {
      setViews(video.views);
    } else {
      // If views aren't already included, fetch them
      const fetchViews = async () => {
        try {
          const res = await axios.get(`/videos/${video.id}/views`);
          setViews(res.data.views);
        } catch (error) {
          console.error('Error fetching views', error);
        }
      };

      fetchViews();
    }
  }, [video]);

  return (
    <Card>
      <Link to={`/watch/${video.id}`}>
        <Thumbnail>
          <img src={`http://localhost:3001/${video.thumbnail}`} alt={video.title} />
        </Thumbnail>
      </Link>

      <VideoInfo>
        <ChannelAvatar>
          <Link to={`/channel/${video.userId}`}>
            <img src={`http://localhost:3001/${video.user.avatar}`} alt={video.user.userr
          </Link>
        </ChannelAvatar>

        <VideoDetails>
```

```
          <Link to={`/watch/${video.id}`} style={{ textDecoration: 'none' }}>
            <h3>{video.title}</h3>
          </Link>

          <Link to={`/channel/${video.userId}`} style={{ textDecoration: 'none' }}>
            <div className="channel-name">{video.user.username}</div>
          </Link>

          <div className="video-meta">
            {formatViewCount(views)} views • {timeAgo(video.createdAt)}
          </div>
        </VideoDetails>
      </VideoInfo>
    </Card>
  );
};

export default VideoCard;
```

Step 6: Creating Main Pages

Let's create some of the main pages for our application, starting with the Home page:

```
// src/pages/Home.jsx
import { useState, useEffect } from 'react';
import axios from 'axios';
import styled from 'styled-components';
import VideoCard from '../components/VideoCard';

const HomeContainer = styled.div`
  padding: 76px 20px 20px 20px;

  @media (min-width: 1200px) {
    padding-left: 250px;
  }
`;

const VideoGrid = styled.div`
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
  gap: 20px;
`;

const Home = () => {
  const [videos, setVideos] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchVideos = async () => {
      try {
        const res = await axios.get('/videos');
        setVideos(res.data.videos);
        setLoading(false);
      } catch (error) {
```

```
        console.error('Error fetching videos', error);
        setError('Failed to fetch videos. Please try again later.');
        setLoading(false);
      }
    };

    fetchVideos();
  }, []);

  if (loading) {
    return (
      <HomeContainer>
        <h2>Loading...</h2>
      </HomeContainer>
    );
  }

  if (error) {
    return (
      <HomeContainer>
        <h2>Error</h2>
        <p>{error}</p>
      </HomeContainer>
    );
  }

  if (videos.length === 0) {
    return (
      <HomeContainer>
        <h2>No videos found</h2>
        <p>Try uploading a video or check back later!</p>
      </HomeContainer>
    );
  }

  return (
    <HomeContainer>
      <VideoGrid>
        {videos.map((video) => (
          <VideoCard key={video.id} video={video} />
        ))}
      </VideoGrid>
    </HomeContainer>
  );
};

export default Home;
```

Next, let's create the WatchVideo page:

```
// src/pages/WatchVideo.jsx
import { useState, useEffect, useContext } from 'react';
import { useParams, Link } from 'react-router-dom';
import ReactPlayer from 'react-player';
import axios from 'axios';
```

```
import styled from 'styled-components';
import { FiThumbsUp, FiThumbsDown, FiShare, FiSave } from 'react-icons/fi';
import { AuthContext } from '../context/AuthContext';
import VideoCard from '../components/VideoCard';

const WatchContainer = styled.div`
  padding: 76px 20px 20px 20px;
  display: grid;
  grid-template-columns: 1fr;
  gap: 20px;

  @media (min-width: 1200px) {
    grid-template-columns: 2fr 1fr;
    padding-left: 250px;
  }
`;

const MainContent = styled.div`
  width: 100%;
`;

const VideoPlayer = styled.div`
  position: relative;
  padding-top: 56.25%; // 16:9 aspect ratio
  width: 100%;

  .react-player {
    position: absolute;
    top: 0;
    left: 0;
  }
`;

const VideoInfo = styled.div`
  margin-top: 20px;
`;

const VideoTitle = styled.h1`
  font-size: 20px;
  margin: 0 0 10px 0;
`;

const VideoMeta = styled.div`
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding-bottom: 16px;
  border-bottom: 1px solid #e5e5e5;
  color: #606060;
  font-size: 14px;
`;

const VideoActions = styled.div`
  display: flex;
  gap: 20px;
```

```
  button {
    background: none;
    border: none;
    display: flex;
    align-items: center;
    gap: 5px;
    color: #606060;
    cursor: pointer;

    &:hover {
      color: #000;
    }
  }

  .active {
    color: #065fd4;
  }
`;

const ChannelInfo = styled.div`
  display: flex;
  margin: 16px 0;
  padding-bottom: 16px;
  border-bottom: 1px solid #e5e5e5;
`;

const ChannelAvatar = styled.div`
  width: 48px;
  height: 48px;
  margin-right: 16px;

  img {
    width: 100%;
    height: 100%;
    border-radius: 50%;
  }
`;

const ChannelDetails = styled.div`
  flex: 1;

  h3 {
    margin: 0 0 5px 0;
    font-size: 16px;
  }

  .subscribers {
    color: #606060;
    font-size: 14px;
    margin-bottom: 10px;
  }

  .description {
    font-size: 14px;
    white-space: pre-wrap;
  }
```

```
`;

const SubscribeButton = styled.button`
  background-color: ${props => props.subscribed ? '#e5e5e5' : 'red'};
  color: ${props => props.subscribed ? '#606060' : 'white'};
  padding: 10px 16px;
  border: none;
  border-radius: 2px;
  font-size: 14px;
  font-weight: 500;
  cursor: pointer;
  margin-left: auto;
`;

const Comments = styled.div`
  margin-top: 24px;
`;

const CommentForm = styled.form`
  display: flex;
  margin-bottom: 24px;

  input {
    flex: 1;
    padding: 10px;
    border: none;
    border-bottom: 1px solid #e5e5e5;
    font-size: 14px;
    outline: none;
  }

  button {
    padding: 10px 16px;
    background-color: #065fd4;
    color: white;
    border: none;
    border-radius: 2px;
    margin-left: 16px;
    cursor: pointer;
  }
`;

const Comment = styled.div`
  display: flex;
  margin-bottom: 16px;

  .avatar {
    width: 40px;
    height: 40px;
    margin-right: 16px;

    img {
      width: 100%;
      height: 100%;
      border-radius: 50%;
    }
```

```
  }

  .comment-details {
    flex: 1;

    .username {
      font-weight: 500;
      margin-right: 5px;
    }

    .time {
      color: #606060;
      font-size: 12px;
    }

    .text {
      margin-top: 5px;
    }
  }
`;

const RelatedVideos = styled.div`
  h3 {
    margin-bottom: 16px;
  }
`;

const WatchVideo = () => {
  const { videoId } = useParams();
  const { currentUser } = useContext(AuthContext);

  const [video, setVideo] = useState(null);
  const [relatedVideos, setRelatedVideos] = useState([]);
  const [comments, setComments] = useState([]);
  const [commentText, setCommentText] = useState('');
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [likeStatus, setLikeStatus] = useState(0);
  const [subscribed, setSubscribed] = useState(false);

  useEffect(() => {
    const fetchVideo = async () => {
      try {
        const res = await axios.get(`/videos/${videoId}`);
        setVideo(res.data.video);

        // Check if user is subscribed to this channel
        if (currentUser) {
          const subscribedRes = await axios.get(`/users/${currentUser.id}/subscribed-to/$
          setSubscribed(subscribedRes.data.subscribed);

          // Check user's like status for this video
          const likeRes = await axios.get(`/videos/${videoId}/like-status`);
          setLikeStatus(likeRes.data.status);
        }
```

```
        // Fetch related videos
        const relatedRes = await axios.get('/videos');
        // Filter out the current video and limit to 10 videos
        const filtered = relatedRes.data.videos.filter(v => v.id !== videoId).slice(0, 10
        setRelatedVideos(filtered);

        // Fetch comments
        const commentsRes = await axios.get(`/videos/${videoId}/comments`);
        setComments(commentsRes.data.comments);

        setLoading(false);
      } catch (error) {
        console.error('Error fetching video', error);
        setError('Failed to fetch video. Please try again later.');
        setLoading(false);
      }
    };

    fetchVideo();
  }, [videoId, currentUser]);

  const handleLike = async (likeValue) => {
    if (!currentUser) {
      // Redirect to sign in page or show sign in modal
      return;
    }

    try {
      const res = await axios.post(`/videos/${videoId}/like`, { like: likeValue });
      setLikeStatus(res.data.like.like);
    } catch (error) {
      console.error('Error liking video', error);
    }
  };

  const handleSubscribe = async () => {
    if (!currentUser) {
      // Redirect to sign in page or show sign in modal
      return;
    }

    try {
      if (subscribed) {
        await axios.delete(`/users/${video.userId}/subscribe`);
        setSubscribed(false);
      } else {
        await axios.post(`/users/${video.userId}/subscribe`);
        setSubscribed(true);
      }
    } catch (error) {
      console.error('Error subscribing', error);
    }
  };

  const handleCommentSubmit = async (e) => {
    e.preventDefault();
```

```
    if (!currentUser) {
      // Redirect to sign in page or show sign in modal
      return;
    }

    if (!commentText.trim()) return;

    try {
      const res = await axios.post(`/videos/${videoId}/comments`, {
        text: commentText
      });

      setComments([res.data.comment, ...comments]);
      setCommentText('');
    } catch (error) {
      console.error('Error posting comment', error);
    }
  };

  if (loading) {
    return (
      <WatchContainer>
        <h2>Loading...</h2>
      </WatchContainer>
    );
  }

  if (error || !video) {
    return (
      <WatchContainer>
        <h2>Error</h2>
        <p>{error || 'Video not found'}</p>
      </WatchContainer>
    );
  }

  return (
    <WatchContainer>
      <MainContent>
        <VideoPlayer>
          <ReactPlayer
            url={`http://localhost:3001/${video.url}`}
            controls
            width="100%"
            height="100%"
            className="react-player"
          />
        </VideoPlayer>

        <VideoInfo>
          <VideoTitle>{video.title}</VideoTitle>

          <VideoMeta>
            <span>{video.views} views • {new Date(video.createdAt).toLocaleDateString()}<
```

```
            <VideoActions>
              <button
                className={likeStatus === 1 ? 'active' : ''}
                onClick={() => handleLike(likeStatus === 1 ? 0 : 1)}
              >
                <FiThumbsUp /> Like
              </button>

              <button
                className={likeStatus === -1 ? 'active' : ''}
                onClick={() => handleLike(likeStatus === -1 ? 0 : -1)}
              >
                <FiThumbsDown /> Dislike
              </button>

              <button>
                <FiShare /> Share
              </button>

              <button>
                <FiSave /> Save
              </button>
            </VideoActions>
          </VideoMeta>
        </VideoInfo>

        <ChannelInfo>
          <ChannelAvatar>
            <Link to={`/channel/${video.userId}`}>
              <img src={`http://localhost:3001/${video.user.avatar}`} alt={video.user.use
            </Link>
          </ChannelAvatar>

          <ChannelDetails>
            <Link to={`/channel/${video.userId}`} style={{ textDecoration: 'none', color:
              <h3>{video.user.username}</h3>
            </Link>

            <div className="description">{video.description}</div>
          </ChannelDetails>

          {currentUser && currentUser.id !== video.userId && (
            <SubscribeButton
              subscribed={subscribed}
              onClick={handleSubscribe}
            >
              {subscribed ? 'SUBSCRIBED' : 'SUBSCRIBE'}
            </SubscribeButton>
          )}
        </ChannelInfo>

        <Comments>
          <h3>{comments.length} Comments</h3>

          {currentUser && (
            <CommentForm onSubmit={handleCommentSubmit}>
```

```
              <input
                type="text"
                placeholder="Add a comment..."
                value={commentText}
                onChange={(e) => setCommentText(e.target.value)}
              />
              <button type="submit">Comment</button>
            </CommentForm>
          )}

          {comments.map((comment) => (
            <Comment key={comment.id}>
              <div className="avatar">
                <Link to={`/channel/${comment.userId}`}>
                  <img src={`http://localhost:3001/${comment.user.avatar}`} alt={comment.
                </Link>
              </div>

              <div className="comment-details">
                <Link to={`/channel/${comment.userId}`} style={{ textDecoration: 'none',
                  <span className="username">{comment.user.username}</span>
                </Link>

                <span className="time">{new Date(comment.createdAt).toLocaleDateString()}

                <div className="text">{comment.text}</div>
              </div>
            </Comment>
          ))}
        </Comments>
      </MainContent>

      <RelatedVideos>
        <h3>Related Videos</h3>

        {relatedVideos.map((video) => (
          <VideoCard key={video.id} video={video} />
        ))}
      </RelatedVideos>
    </WatchContainer>
  );
};

export default WatchVideo;
```

Let's also create a user authentication page:

```
// src/pages/SignIn.jsx
import { useState, useContext } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import styled from 'styled-components';
import { AuthContext } from '../context/AuthContext';

const SignInContainer = styled.div`
  display: flex;
```

```
  justify-content: center;
  align-items: center;
  height: 100vh;
  background-color: #f9f9f9;
`;

const FormContainer = styled.div`
  background-color: white;
  padding: 40px;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 400px;
`;

const Title = styled.h1`
  text-align: center;
  margin-bottom: 24px;
  font-size: 24px;
`;

const Form = styled.form`
  display: flex;
  flex-direction: column;
  gap: 16px;
`;

const FormGroup = styled.div`
  display: flex;
  flex-direction: column;
  gap: 8px;

  label {
    font-size: 14px;
    font-weight: 500;
  }

  input {
    padding: 12px;
    border: 1px solid #ccc;
    border-radius: 4px;
    font-size: 14px;
  }
`;

const Button = styled.button`
  padding: 12px;
  background-color: red;
  color: white;
  border: none;
  border-radius: 4px;
  font-size: 16px;
  font-weight: 500;
  cursor: pointer;
  margin-top: 16px;
```

```
  &:hover {
    background-color: #cc0000;
  }

  &:disabled {
    background-color: #ccc;
    cursor: not-allowed;
  }
`;

const ErrorMessage = styled.div`
  color: red;
  margin-bottom: 16px;
`;

const SignUpLink = styled.div`
  text-align: center;
  margin-top: 24px;
  font-size: 14px;

  a {
    color: #065fd4;
    text-decoration: none;

    &:hover {
      text-decoration: underline;
    }
  }
`;

const SignIn = () => {
  const [formData, setFormData] = useState({
    email: '',
    password: ''
  });
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(false);

  const { login } = useContext(AuthContext);
  const navigate = useNavigate();

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

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

    try {
      const { success, message } = await login(formData);
```

```
      if (success) {
        navigate('/');
      } else {
        setError(message);
      }
    } catch (error) {
      setError('An error occurred. Please try again.');
    } finally {
      setLoading(false);
    }
  };

  return (
    <SignInContainer>
      <FormContainer>
        <Title>Sign In</Title>

        {error && <ErrorMessage>{error}</ErrorMessage>}

        <Form onSubmit={handleSubmit}>
          <FormGroup>
            <label htmlFor="email">Email</label>
            <input
              type="email"
              id="email"
              name="email"
              value={formData.email}
              onChange={handleChange}
              required
            />
          </FormGroup>

          <FormGroup>
            <label htmlFor="password">Password</label>
            <input
              type="password"
              id="password"
              name="password"
              value={formData.password}
              onChange={handleChange}
              required
            />
          </FormGroup>

          <Button type="submit" disabled={loading}>
            {loading ? 'Signing In...' : 'Sign In'}
          </Button>
        </Form>

        <SignUpLink>
          Don't have an account? <Link to="/signup">Sign Up</Link>
        </SignUpLink>
      </FormContainer>
    </SignInContainer>
  );
};
```

```
export default SignIn;
```

Implementing Video Upload Functionality

The video upload feature is a critical part of our YouTube clone. Let's create the UploadVideo page:

```
// src/pages/UploadVideo.jsx
import { useState, useContext } from 'react';
import { useNavigate } from 'react-router-dom';
import styled from 'styled-components';
import axios from 'axios';
import { FiUpload } from 'react-icons/fi';
import { AuthContext } from '../context/AuthContext';

const UploadContainer = styled.div`
  padding: 76px 20px 20px 20px;
  max-width: 800px;
  margin: 0 auto;

  @media (min-width: 1200px) {
    padding-left: 250px;
  }
`;

const Title = styled.h1`
  margin-bottom: 24px;
  font-size: 24px;
`;

const UploadForm = styled.form`
  display: flex;
  flex-direction: column;
  gap: 20px;
`;

const FormGroup = styled.div`
  display: flex;
  flex-direction: column;
  gap: 8px;

  label {
    font-size: 14px;
    font-weight: 500;
  }

  input, textarea {
    padding: 12px;
    border: 1px solid #ccc;
    border-radius: 4px;
    font-size: 14px;
  }
```

```
  textarea {
    min-height: 100px;
    resize: vertical;
  }
`;

const FileInput = styled.div`
  border: 2px dashed #ccc;
  padding: 40px;
  border-radius: 4px;
  text-align: center;
  cursor: pointer;
  transition: border-color 0.3s;

  &:hover {
    border-color: #065fd4;
  }

  input {
    display: none;
  }

  .icon {
    font-size: 48px;
    color: #606060;
    margin-bottom: 16px;
  }

  p {
    margin: 0;
    color: #606060;
  }

  .selected-file {
    margin-top: 16px;
    color: #065fd4;
  }
`;

const ThumbnailPreview = styled.div`
  margin-top: 16px;

  img {
    max-width: 100%;
    max-height: 180px;
    border-radius: 4px;
  }
`;

const Button = styled.button`
  padding: 12px;
  background-color: red;
  color: white;
  border: none;
  border-radius: 4px;
  font-size: 16px;
```

```
  font-weight: 500;
  cursor: pointer;

  &:hover {
    background-color: #cc0000;
  }

  &:disabled {
    background-color: #ccc;
    cursor: not-allowed;
  }
`;

const ProgressBar = styled.div`
  width: 100%;
  height: 10px;
  background-color: #f5f5f5;
  border-radius: 5px;
  margin-top: 10px;

  .progress {
    height: 100%;
    background-color: #065fd4;
    border-radius: 5px;
    width: ${props => props.progress}%;
  }
`;

const ErrorMessage = styled.div`
  color: red;
  margin-top: 16px;
`;

const UploadVideo = () => {
  const { currentUser } = useContext(AuthContext);
  const navigate = useNavigate();

  const [formData, setFormData] = useState({
    title: '',
    description: ''
  });
  const [videoFile, setVideoFile] = useState(null);
  const [thumbnailFile, setThumbnailFile] = useState(null);
  const [thumbnailPreview, setThumbnailPreview] = useState('');
  const [uploading, setUploading] = useState(false);
  const [uploadProgress, setUploadProgress] = useState(0);
  const [error, setError] = useState('');

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

  const handleVideoChange = (e) => {
```

```
    const file = e.target.files[^0];
    if (file) {
      if (file.size > 500 * 1024 * 1024) { // 500MB limit
        setError('Video file size should be less than 500MB');
        return;
      }
      setVideoFile(file);
      setError('');
    }
  };

  const handleThumbnailChange = (e) => {
    const file = e.target.files[^0];
    if (file) {
      if (file.size > 5 * 1024 * 1024) { // 5MB limit
        setError('Thumbnail file size should be less than 5MB');
        return;
      }
      setThumbnailFile(file);

      // Create preview
      const reader = new FileReader();
      reader.onloadend = () => {
        setThumbnailPreview(reader.result);
      };
      reader.readAsDataURL(file);
      setError('');
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    if (!videoFile) {
      setError('Please select a video file');
      return;
    }

    if (!thumbnailFile) {
      setError('Please select a thumbnail image');
      return;
    }

    setUploading(true);
    setError('');

    const formDataToSend = new FormData();
    formDataToSend.append('title', formData.title);
    formDataToSend.append('description', formData.description);
    formDataToSend.append('video', videoFile);
    formDataToSend.append('thumbnail', thumbnailFile);

    try {
      const res = await axios.post('/videos', formDataToSend, {
        headers: {
          'Content-Type': 'multipart/form-data'
```

```
        },
        onUploadProgress: (progressEvent) => {
          const percentCompleted = Math.round(
            (progressEvent.loaded * 100) / progressEvent.total
          );
          setUploadProgress(percentCompleted);
        }
      });

      navigate(`/watch/${res.data.video.id}`);
    } catch (error) {
      console.error('Error uploading video', error);
      setError(error.response?.data?.message || 'Error uploading video. Please try again.
      setUploading(false);
    }
  };

  return (
    <UploadContainer>
      <Title>Upload Video</Title>

      <UploadForm onSubmit={handleSubmit}>
        <FormGroup>
          <label htmlFor="title">Title</label>
          <input
            type="text"
            id="title"
            name="title"
            value={formData.title}
            onChange={handleChange}
            required
            maxLength="100"
          />
        </FormGroup>

        <FormGroup>
          <label htmlFor="description">Description</label>
          <textarea
            id="description"
            name="description"
            value={formData.description}
            onChange={handleChange}
            maxLength="5000"
          />
        </FormGroup>

        <FormGroup>
          <label>Video</label>
          <FileInput onClick={() => document.getElementById('video-input').click()}>
            <input
              type="file"
              id="video-input"
              accept="video/*"
              onChange={handleVideoChange}
            />
            <FiUpload className="icon" />
```

```
            <p>Click to select a video file</p>
            {videoFile && (
              <p className="selected-file">{videoFile.name}</p>
            )}
          </FileInput>
        </FormGroup>

        <FormGroup>
          <label>Thumbnail</label>
          <FileInput onClick={() => document.getElementById('thumbnail-input').click()}>
            <input
              type="file"
              id="thumbnail-input"
              accept="image/*"
              onChange={handleThumbnailChange}
            />
            <FiUpload className="icon" />
            <p>Click to select a thumbnail image</p>
            {thumbnailFile && (
              <p className="selected-file">{thumbnailFile.name}</p>
            )}
          </FileInput>

          {thumbnailPreview && (
            <ThumbnailPreview>
              <img src={thumbnailPreview} alt="Thumbnail Preview" />
            </ThumbnailPreview>
          )}
        </FormGroup>

        {error && <ErrorMessage>{error}</ErrorMessage>}

        {uploading && (
          <ProgressBar progress={uploadProgress}>
            <div className="progress"></div>
          </ProgressBar>
        )}

        <Button type="submit" disabled={uploading}>
          {uploading ? 'Uploading...' : 'Upload Video'}
        </Button>
      </UploadForm>
    </UploadContainer>
  );
};

export default UploadVideo;
```

Conclusion

In this comprehensive guide, we've built a fully-functional YouTube clone using modern web technologies:

- A robust **backend with Node.js and Express.js** that handles authentication, video uploads, comments, likes, and more

- A **PostgreSQL database** with **Prisma ORM** for efficient data modeling and querying
- A dynamic **React frontend with Vite** for improved development experience
- Video streaming capabilities with **ReactPlayer**
- Styled components for a clean and responsive UI

We've implemented core features found in YouTube, including:

- User authentication and profile management
- Video uploads with custom thumbnails
- Video playback and streaming
- Comments and likes functionality
- Channel subscriptions
- Recommended and trending videos
- Search functionality

This project provides a solid foundation that you can extend with additional features such as:

- More advanced recommendation algorithms
- Video categories and playlists
- Notifications system
- Analytics for video creators
- Monetization options
- Mobile responsiveness enhancements

By building this YouTube clone, you've gained valuable experience with full-stack development, database design, and implementing complex features like video streaming and user interactions.

Remember that this is just the beginning - you can continue to improve and add features to make your YouTube clone even more robust and feature-rich!

⁂