Final Project Proposal

Project Title and Description

Application Name:

OffTube Record

Description:

OffTube Record is a wrapper around YouTube, enabling users to access YouTube's content while providing additional features not supported or discouraged by the platform. These features include independent comment boards and video watch histories, all hosted outside of YouTube's ecosystem. The core problem solved is creating a separate space for interaction and community around YouTube videos, such as open, persistent comment boards and a history of user activity.

Feature Breakdown & Design Considerations

YouTube Search + Watch:

- **Search:** Users can search for YouTube videos through a search bar.
- **Watch:** Users can select and watch videos from the search results.
- **Proxy Setup**: Use Express to call the YouTube Data API to keep the API key hidden.
- **Video Player**: Use YouTube’s embed iframe; avoid hosting or streaming video content.

Independent Comment Boards:

- **Comments:** Users can read and post comments for specific videos.
- **Persistence:** Comments are persisted between sessions.
- **Separation from YouTube**: Comments are stored in MongoDB, retrieved based on video ID.

- **Anonymous Usernames**: Allow anonymous commenting for simplicity.

User History:

- **History**: Users can view their interaction history, including videos watched and comments made

- **Minimal Persistence**: Track video IDs and timestamps.

Technical Implementation

1. Routing & Component Architecture (Frontend - React)

Planned Components:

Component	Purpose
SearchBar	Input for triggering YouTube search
VideoPlayer	Embedded YouTube player via iframe
VideoResultList	Display search results
CommentBoard	Display and input custom comments
HistoryPanel	Display user comment/video history
Header/Navbar	Basic navigation

Routing (React Routes):

Route	Component	Purpose
`/`	SearchBar	Home page with search bar
`/watch/:videoId`	VideoPlayer + CommentBoard	Video player with associated comment board
`/history`	HistoryPanel	View user's interaction history

---

2. API Communication (Axios + Express)

Frontend Axios Requests:

- **GET** `/api/search?q=some-query` → Proxy to YouTube API
- **GET** `/api/comments/:videoId` → Load custom comments
- **POST** `/api/comments/:videoId` → Submit comment
- **GET** `/api/history` → Load user history
- **POST** `/api/history` → Log new history item

Backend Express Routes:

- **GET** `/api/search`: Accept query parameter, call YouTube API server-side, and return filtered results (title, id, thumbnail). API key remains hidden.
- **GET/POST** `/api/comments/:videoId`: Handle MongoDB comments collection, filter by video ID.

- **GET/POST `/api/history`**: Access `user_history` collection to log or retrieve user interactions based on session ID or token.

---

3. Database Modeling (MongoDB via Mongoose)

Collections:

Comment Schema:

```
{
  videoId: String,
  username: String,
  content: String,
  timestamp: Date
}
```

History Schema:

```
{
  userId: String,  // anonymous or named
  actions: [
    { type: 'watch' | 'comment', videoId: String, timestamp: Date }
  ]
}
```

Design Considerations:

- Avoid complex authentication; rely on simple session management via `localStorage`.

- Implement soft-limits for comments to mitigate spam.
- Optional: Use anonymous usernames for simplicity in comments.

---

4. CRUD Operations

For the minimal viable product only create and read operations will be fully implemented. As Update and delete are not necessary at this time.

Operation	Action
Create	Post comment, log watch action
Read	Retrieve comments and user history
Update	Optional: Edit comments (future?)
Delete	Optional: Delete comments (future?)

All CRUD operations are handled via standard Express REST routes.

Team Member Roles

- **Dillon Jackson** will manage all project tasks, including:
    - React frontend development
    - Node/Express backend development
    - MongoDB database management

- Documentation and presentation

Timeline & Milestones

Date	Milestone
April 29	Basic structure and components ready. Frontend and backend parts implemented separately with initial testing.
May 6	Integration phase: Assemble frontend, backend, and database, followed by integration testing.
May 9	Final submission: Complete the project and submit for review.
