YouTube Companion Web App: Technical Requirements

Date: May 26, 2025

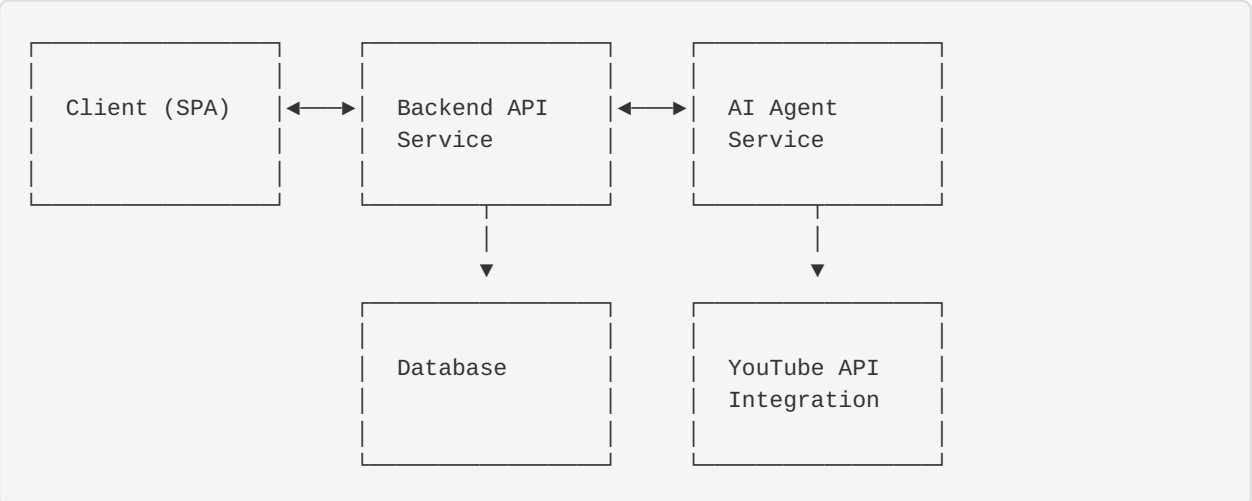
This document outlines the technical requirements for building a YouTube companion web app powered by AI agents with social features. It covers the architecture, AI agent frameworks, database schema, authentication, and implementation details necessary for development.

Table of Contents

1. System Architecture

The YouTube companion web app will follow a modern, scalable architecture with the following components:

1.1 High-Level Architecture



1.2 Component Descriptions

1. **Client (SPA)**: A React-based Single Page Application that provides the user interface for interacting with YouTube content, AI agents, and social features.

2. **Backend API Service**: A Node.js/Express or Python/FastAPI service that handles authentication, data processing, and communication between the client, database, and AI agent service.
3. **AI Agent Service**: A dedicated service built on a multi-agent framework that manages AI agents, their interactions, and learning processes.
4. **Database**: A combination of relational and vector databases to store user data, agent data, and embeddings for content analysis.
5. **YouTube API Integration**: A service component that handles communication with the YouTube Data API, managing quotas and caching responses.

2. AI Agent Framework Selection

Based on our research, we recommend using **CrewAI** as the primary framework for implementing AI agents in the YouTube companion app, with **LangGraph** for specific complex workflows.

2.1 Framework Comparison

Framework	Strengths	Limitations	Fit for Our Use Case
CrewAI	Role-based agents, team collaboration, rapid development	Early-stage development	Excellent for YouTube companion with multiple specialized agents
LangGraph	Dynamic workflows, decision-making, visual debugging	Setup complexity	Good for complex recommendation and content analysis flows
AutoGen	Multi-agent collaboration, enterprise reliability	Coordination complexity	Strong alternative, especially for scaling
LangChain	Flexibility, extensive tooling, community support	Debugging challenges	Good foundation but less specialized for our multi-agent needs

2.2 Recommended Agent Architecture

We recommend implementing a team of specialized AI agents using CrewAI:

1. **Content Analyzer Agent**: Processes video metadata, captions, and comments to extract key information.
2. **Recommendation Agent**: Generates personalized video recommendations based on user preferences and viewing history.
3. **Summarization Agent**: Creates concise summaries of video content for quick consumption.
4. **Social Facilitator Agent**: Manages sharing and collaborative features between users and their agents.
5. **Learning Coordinator Agent**: Orchestrates knowledge sharing between agents and implements collaborative learning.

2.3 Agent Communication Protocol

Agents will communicate using a standardized JSON-based protocol with the following structure:

```
{
  "messageId": "unique-message-id",
  "fromAgent": "agent-identifier",
  "toAgent": "agent-identifier",
  "messageType": "request|response|notification",
  "content": {
    "action": "action-name",
    "parameters": {},
    "data": {}
  },
  "timestamp": "ISO-timestamp"
}
```

3. Database Schema

The database schema will support user profiles, agent data, content interactions, and social features.

3.1 Core Entities

Users Table

```
CREATE TABLE Users (
    UserID VARCHAR(36) PRIMARY KEY,
    Username VARCHAR(50) UNIQUE NOT NULL,
    Email VARCHAR(255) UNIQUE NOT NULL,
    PasswordHash VARCHAR(255) NOT NULL,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    LastLogin TIMESTAMP,
    ProfilePicture VARCHAR(255),
    IsActive BOOLEAN DEFAULT TRUE
);
```

UserPreferences Table

```
CREATE TABLE UserPreferences (
    PreferenceID VARCHAR(36) PRIMARY KEY,
    UserID VARCHAR(36) NOT NULL,
    PreferenceKey VARCHAR(50) NOT NULL,
    PreferenceValue TEXT,
    LastUpdated TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    UNIQUE (UserID, PreferenceKey)
);
```

Agents Table

```
CREATE TABLE Agents (
    AgentID VARCHAR(36) PRIMARY KEY,
    UserID VARCHAR(36) NOT NULL,
    AgentType VARCHAR(50) NOT NULL,
    AgentName VARCHAR(100) NOT NULL,
    Configuration JSONB,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    LastActive TIMESTAMP,
    VisualAssets JSONB,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

AgentMemories Table

```
CREATE TABLE AgentMemories (
    MemoryID VARCHAR(36) PRIMARY KEY,
    AgentID VARCHAR(36) NOT NULL,
    MemoryType VARCHAR(50) NOT NULL,
    Content TEXT NOT NULL,
    Metadata JSONB,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Importance FLOAT DEFAULT 0.5,
    FOREIGN KEY (AgentID) REFERENCES Agents(AgentID)
);
```

VideoInteractions Table

```
CREATE TABLE VideoInteractions (
    InteractionID VARCHAR(36) PRIMARY KEY,
    UserID VARCHAR(36) NOT NULL,
    VideoID VARCHAR(20) NOT NULL,
    InteractionType VARCHAR(50) NOT NULL,
    Timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Data JSONB,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

AgentInteractions Table

```
CREATE TABLE AgentInteractions (
    InteractionID VARCHAR(36) PRIMARY KEY,
    FromAgentID VARCHAR(36) NOT NULL,
    ToAgentID VARCHAR(36) NOT NULL,
    InteractionType VARCHAR(50) NOT NULL,
    Content TEXT,
    Metadata JSONB,
    Timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (FromAgentID) REFERENCES Agents(AgentID),
    FOREIGN KEY (ToAgentID) REFERENCES Agents(AgentID)
);
```

SocialConnections Table

```
CREATE TABLE SocialConnections (
    ConnectionID VARCHAR(36) PRIMARY KEY,
    UserID1 VARCHAR(36) NOT NULL,
    UserID2 VARCHAR(36) NOT NULL,
    ConnectionType VARCHAR(50) NOT NULL,
    Status VARCHAR(20) NOT NULL,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    LastInteraction TIMESTAMP,
    FOREIGN KEY (UserID1) REFERENCES Users(UserID),
    FOREIGN KEY (UserID2) REFERENCES Users(UserID),
    UNIQUE (UserID1, UserID2, ConnectionType)
);
```

SharedDiscoveries Table

```
CREATE TABLE SharedDiscoveries (
    DiscoveryID VARCHAR(36) PRIMARY KEY,
    AgentID VARCHAR(36) NOT NULL,
    DiscoveryType VARCHAR(50) NOT NULL,
    Content TEXT NOT NULL,
    Metadata JSONB,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    IsPublic BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (AgentID) REFERENCES Agents(AgentID)
);
```

ShopItems Table

```
CREATE TABLE ShopItems (
    ItemID VARCHAR(36) PRIMARY KEY,
    ItemType VARCHAR(50) NOT NULL,
    Name VARCHAR(100) NOT NULL,
    Description TEXT,
    Price DECIMAL(10, 2) NOT NULL,
    AssetURL VARCHAR(255),
    Metadata JSONB,
    IsActive BOOLEAN DEFAULT TRUE
);
```

UserInventory Table

```
CREATE TABLE UserInventory (
    InventoryID VARCHAR(36) PRIMARY KEY,
    UserID VARCHAR(36) NOT NULL,
    ItemID VARCHAR(36) NOT NULL,
    AcquiredAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    IsEquipped BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (ItemID) REFERENCES ShopItems(ItemID),
    UNIQUE (UserID, ItemID)
);
```

3.2 Vector Database for Content

For storing and retrieving video content embeddings, we recommend using PgVector (PostgreSQL extension) or a dedicated vector database like Pinecone:

```
CREATE EXTENSION IF NOT EXISTS vector;

CREATE TABLE VideoEmbeddings (
    VideoID VARCHAR(20) PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    Description TEXT,
    Embedding vector(1536),
    Metadata JSONB,
    LastUpdated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX video_embedding_idx ON VideoEmbeddings USING ivfflat (Embedding vec-
tor_cosine_ops);
```

4. Authentication System

The YouTube companion web app will implement a secure password-based authentication system with the following features:

4.1 Authentication Flow

1. **Registration**: Users provide username, email, and password
2. **Login**: Username/email and password verification
3. **Session Management**: JWT (JSON Web Tokens) for maintaining authenticated sessions
4. **Password Reset**: Secure email-based password reset functionality

4.2 Security Implementation

• **Password Storage**: Passwords will be hashed using bcrypt with appropriate salt rounds
• **HTTPS**: All communications will be encrypted using HTTPS
• **JWT Configuration**:
• Short-lived access tokens (15-30 minutes)
• Refresh tokens stored in HttpOnly, Secure cookies
• Token rotation on refresh
• **Protection Against Common Attacks**:
• XSS: Content Security Policy (CSP) headers
• CSRF: Anti-CSRF tokens and SameSite cookies
• Brute Force: Rate limiting and account lockouts

4.3 Authentication Code Example

```
// Example Node.js authentication middleware
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');

// Password hashing
async function hashPassword(password) {
  const saltRounds = 12;
  return await bcrypt.hash(password, saltRounds);
}

// Password verification
async function verifyPassword(password, hashedPassword) {
  return await bcrypt.compare(password, hashedPassword);
}

// JWT token generation
function generateTokens(userId) {
  const accessToken = jwt.sign(
    { userId },
    process.env.JWT_ACCESS_SECRET,
    { expiresIn: '15m' }
  );

  const refreshToken = jwt.sign(
    { userId },
    process.env.JWT_REFRESH_SECRET,
    { expiresIn: '7d' }
  );

  return { accessToken, refreshToken };
}

// Authentication middleware
function authenticateToken(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) return res.sendStatus(401);

  jwt.verify(token, process.env.JWT_ACCESS_SECRET, (err, user) => {
    if (err) return res.sendStatus(403);
    req.user = user;
    next();
  });
}
```

5. Frontend Requirements

5.1 Technology Stack

- **Framework**: React.js with TypeScript
- **State Management**: Redux Toolkit or Context API
- **Styling**: Tailwind CSS or styled-components
- **Component Library**: Material UI or Chakra UI

- **Routing**: React Router

5.2 Key Features

1. **Responsive Design**: Mobile-first approach with responsive layouts
2. **Video Player Integration**: Custom YouTube video player with enhanced features
3. **Agent Interaction Interface**: Chat-like interface for communicating with AI agents
4. **Social Features UI**: Components for sharing discoveries and connecting with other users
5. **Agent Customization**: Interface for customizing agent preferences and visual appearance
6. **Shop Interface**: UI for browsing and purchasing visual customizations

5.3 Performance Considerations

- Implement code splitting for faster initial load times
- Use React.lazy and Suspense for component lazy loading
- Implement virtualized lists for handling large datasets
- Optimize image loading with lazy loading and WebP format
- Implement service workers for offline capabilities and caching

6. Backend Requirements

6.1 Technology Stack

- **Primary Language**: Node.js with TypeScript or Python
- **API Framework**: Express.js or FastAPI
- **Database ORM**: Prisma, Sequelize, or SQLAlchemy
- **Task Queue**: Redis with Bull or Celery for background processing
- **WebSockets**: Socket.io or native WebSockets for real-time features

6.2 Key Services

1. **Authentication Service**: Handles user registration, login, and session management
2. **User Service**: Manages user profiles and preferences
3. **Agent Service**: Coordinates AI agent creation, configuration, and interactions
4. **YouTube API Service**: Handles communication with YouTube Data API
5. **Recommendation Service**: Processes user data to generate personalized recommendations
6. **Social Service**: Manages connections between users and shared discoveries
7. **Shop Service**: Handles virtual item purchases and inventory management

6.3 API Endpoints

The backend will expose RESTful API endpoints for the following resources:

- `/api/auth`: Authentication endpoints
- `/api/users`: User management endpoints
- `/api/agents`: AI agent management endpoints
- `/api/videos`: YouTube video data endpoints
- `/api/recommendations`: Video recommendation endpoints
- `/api/social`: Social connection endpoints
- `/api/discoveries`: Shared discoveries endpoints
- `/api/shop`: Virtual shop endpoints

7. API Integration

7.1 YouTube Data API Integration

The application will integrate with the YouTube Data API to access video metadata, captions, and comments.

Key Integration Points:

1. **Authentication**: Implement OAuth 2.0 for server-side web applications
2. **Data Retrieval**:
   - Video metadata via `videos.list` endpoint
   - Captions via `captions.list` and `captions.download` endpoints
   - Comments via `commentThreads.list` endpoint
3. **Quota Management**:
   - Implement caching to reduce API calls
   - Monitor quota usage and implement fallback mechanisms
   - Batch requests when possible

7.2 AI Model Integration

The AI agent service will integrate with language models through their respective APIs:

1. **OpenAI GPT-4**: For sophisticated reasoning and natural language understanding
2. **Anthropic Claude**: As an alternative for certain agent roles
3. **Open-source Models**: For specific tasks where appropriate

8. Social Features Implementation

8.1 Agent-to-Agent Knowledge Sharing

Implement Google's Social Learning framework for agent-to-agent knowledge sharing:

1. **Synthetic Examples**: Agents generate new examples to teach other agents
2. **Synthetic Instructions**: Agents create instructions for specific tasks
3. **Privacy Preservation**: Implement Secret Sharer metric to quantify and prevent data leakage

8.2 User-to-User Social Interactions

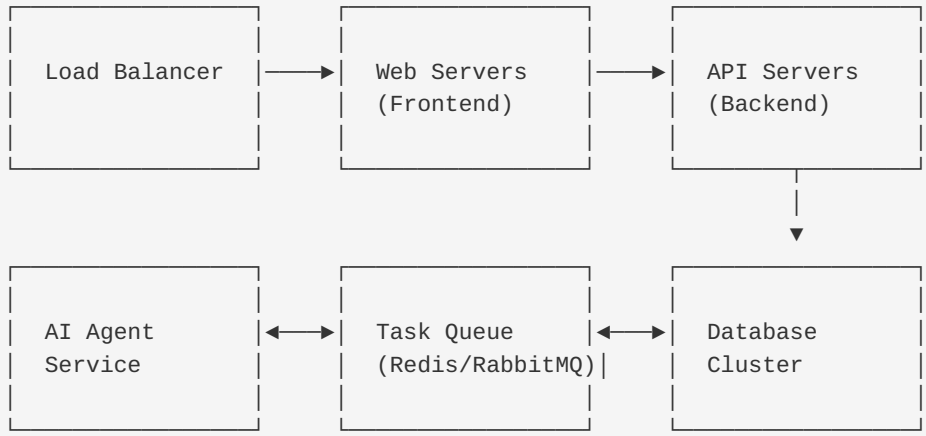
1. **Connection System**: Allow users to connect with others with similar interests
2. **Discovery Sharing**: Enable sharing of agent discoveries with connected users
3. **Collaborative Viewing**: Implement features for synchronized video watching
4. **Activity Feed**: Display recent activities from connected users and their agents

8.3 Collaborative Learning Implementation

```python
# Example implementation of agent-to-agent teaching
def teach_agent(teacher_agent, student_agent, task_description):
    # Generate synthetic examples based on teacher's knowledge
    examples = teacher_agent.generate_synthetic_examples(
        task=task_description,
        num_examples=8,
        diversity_factor=0.7
    )

    # Generate task instructions
    instructions = teacher_agent.generate_instructions(
        task=task_description,
        complexity_level="adaptive",
        format="step_by_step"
    )

    # Transfer knowledge to student agent
    learning_result = student_agent.learn_from_teaching(
        examples=examples,
        instructions=instructions,
        task=task_description
    )

    # Evaluate learning effectiveness
    evaluation_score = evaluate_agent_performance(
        agent=student_agent,
        task=task_description,
        test_cases=generate_test_cases(task_description)
    )

    return {
        "learning_result": learning_result,
        "evaluation_score": evaluation_score
    }
```

9. Deployment and Infrastructure

9.1 Deployment Architecture



9.2 Infrastructure Requirements

1. **Compute Resources**:
   - Web/API Servers: 2-4 vCPUs, 8GB RAM per instance
   - AI Agent Service: 4-8 vCPUs, 16GB RAM per instance
   - Database: 4 vCPUs, 16GB RAM minimum

2. **Storage Requirements**:
   - Database: 100GB+ SSD storage initially, scalable
   - Object Storage: For user uploads and assets

3. **Networking**:
   - Load balancer with SSL termination
   - Internal network for service-to-service communication
   - CDN for static assets

9.3 Scaling Strategy

1. **Horizontal Scaling**: Add more instances of web/API servers based on load
2. **Vertical Scaling**: Increase resources for database and AI services as needed
3. **Database Sharding**: Implement if user data grows significantly
4. **Caching Layer**: Redis for caching frequently accessed data

10. Security Considerations

10.1 Data Protection

1. **Encryption**:
   - Data at rest: Database encryption
   - Data in transit: HTTPS/TLS 1.3
   - Sensitive data: Field-level encryption for PII

2. **Access Control**:
   - Role-based access control (RBAC)
   - Principle of least privilege
   - Regular permission audits

10.2 AI Agent Security

1. **Input Validation**: Sanitize all user inputs to prevent prompt injection
2. **Output Filtering**: Implement content moderation for agent outputs
3. **Rate Limiting**: Prevent abuse of AI agent capabilities
4. **Monitoring**: Track agent behavior for anomalies or misuse

10.3 Compliance Considerations

1. **Privacy Policy**: Clear disclosure of data usage and AI capabilities
2. **Terms of Service**: Define acceptable use of AI agents and social features
3. **GDPR Compliance**: Implement data subject rights (access, deletion, etc.)
4. **COPPA Compliance**: Age verification if targeting users under 13

This technical requirements document provides a comprehensive foundation for building the YouTube companion web app with AI agents and social features. The architecture, frameworks, and implementation details outlined here are designed to create a scalable, secure, and feature-rich application that enhances the YouTube viewing experience through AI assistance and social interaction.