**Software Requirements Document (SRD)**

**Project: Mobile Usage and YouTube Watch History Analyzer**

**Version**: 1.0  
**Date**: June 20, 2025  
**Prepared by**: LK consultancy services  
**Project Owner**: LK group

**1. Introduction**

**1.1 Purpose**

This document outlines the functional and non-functional requirements for a web-based application that monitors mobile app usage (via Digital Wellbeing/Screen Time), analyzes YouTube watch history, and provides actionable insights to optimize user behavior. The system aims to help users reduce excessive or unproductive screen time (e.g., on social media or entertainment videos) through real-time notifications, data visualizations, and optional social media feed control. The project leverages Django, AI, and APIs (e.g., YouTube Data API, Perplexity AI) to deliver a robust, user-friendly experience.

**1.2 Scope**

The system, tentatively named **ScreenWise**, will:

* Collect and analyze mobile app usage data (e.g., time spent on YouTube, X, Instagram).
* Access YouTube watch history to categorize videos (e.g., productive vs. unproductive) and estimate “wasted” time.
* Provide a user dashboard with usage statistics, visualizations, and personalized suggestions.
* Deliver pop-up notifications (web and mobile) to alert users when exceeding usage limits.
* Optionally integrate a browser extension to filter social media feeds (e.g., X, YouTube) and enforce time restrictions.
* Use AI (local models or Perplexity API) for content analysis and recommendations.
* Ensure privacy, scalability, and compliance with platform policies (e.g., Google, Apple, X).

The system will be accessible via a website, with a companion mobile app for data syncing and notifications. It targets productivity-conscious users (e.g., students, professionals) and aims to stand out as a portfolio project.

**1.3 Definitions, Acronyms, and Abbreviations**

* **Digital Wellbeing**: Android’s feature for tracking app usage.
* **Screen Time**: iOS’s feature for monitoring device usage.
* **YouTube Data API v3**: Google’s API for accessing YouTube data, including watch history.
* **Perplexity API**: AI-powered API for content analysis and recommendations (if available).
* **DOM**: Document Object Model, used for browser extension feed manipulation.
* **FCM**: Firebase Cloud Messaging, for push notifications.
* **OAuth 2.0**: Authentication protocol for secure API access.
* **Wasted Time**: User-defined or AI-classified unproductive usage (e.g., excessive gaming videos).

**1.4 References**

* YouTube Data API: <https://developers.google.com/youtube/v3>
* Android UsageStatsManager: <https://developer.android.com/reference/android/app/usage/UsageStatsManager>
* Django Documentation: <https://docs.djangoproject.com>
* Perplexity AI: <https://www.perplexity.ai> (for API details, if available)
* Firebase Cloud Messaging: <https://firebase.google.com/docs/cloud-messaging>

**2. Overall Description**

**2.1 User Needs**

* **Productivity Tracking**: Users want to monitor and reduce time spent on distracting apps (e.g., YouTube, X).
* **YouTube Insights**: Users need insights into their watch history (e.g., video categories, time spent) to identify unproductive habits.
* **Actionable Feedback**: Users expect real-time alerts and suggestions to manage screen time effectively.
* **Customization**: Users want to set usage goals (e.g., “Limit YouTube to 1 hour/day”) and customize “productive” vs. “unproductive” categories.
* **Social Media Control**: Users may want to filter social media feeds to prioritize relevant content and restrict access after limits.
* **Privacy**: Users require secure data handling with minimal collection and clear consent mechanisms.

**2.2 Assumptions and Dependencies**

* **Assumptions**:
  + Users have Android (with Digital Wellbeing) or iOS (with Screen Time) devices.
  + Users are willing to grant YouTube API access via Google OAuth.
  + Perplexity API (if used) supports content analysis and recommendations.
* **Dependencies**:
  + Google Cloud Console for YouTube API credentials and quota management.
  + Firebase for push notifications.
  + Mobile app for syncing usage data (especially on iOS, where direct access is limited).
  + Compliance with Google, Apple, and X policies to avoid API or app store rejections.

**3. System Features**

**3.1 Mobile Usage Data Collection**

* **Description**: Collect app usage data (e.g., time spent on YouTube, X) from Digital Wellbeing (Android) or Screen Time (iOS).
* **Functional Requirements**:
  + **F1.1**: Interface with Android’s UsageStatsManager API to fetch daily/weekly app usage stats (e.g., time spent, app categories).
  + **F1.2**: For iOS, prompt users to manually input Screen Time data or develop a companion app to sync limited data via Family Sharing APIs.
  + **F1.3**: Sync data to the Django backend via REST APIs for analysis and storage.
  + **F1.4**: Allow users to authenticate via Google/Apple accounts for secure data access.
* **Priority**: High
* **Input**: User permission, app usage data.
* **Output**: JSON data with app names and time spent (e.g., {"YouTube": 7200, "X": 3600}).

**3.2 YouTube Watch History Analysis**

* **Description**: Access and analyze YouTube watch history to categorize videos and estimate “wasted” time.
* **Functional Requirements**:
  + **F2.1**: Use YouTube Data API (playlistItems.list, videos.list) to fetch watch history (video IDs, titles, categories, durations).
  + **F2.2**: Implement an AI model (e.g., scikit-learn Naive Bayes) to classify videos as “productive” (e.g., education) or “unproductive” (e.g., gaming) based on titles, descriptions, or categories.
  + **F2.3**: Calculate total time spent per category using video durations and watch timestamps.
  + **F2.4**: Allow users to customize “productive” vs. “unproductive” categories (e.g., mark gaming as productive if desired).
  + **F2.5**: Optionally use Perplexity API to analyze video content or suggest alternative videos (e.g., “Watch coding tutorials instead”).
* **Priority**: High
* **Input**: YouTube API credentials, user watch history.
* **Output**: Categorized usage data (e.g., “Gaming: 1.5 hours, Education: 0.5 hours”).

**3.3 Usage Insights and Suggestions**

* **Description**: Provide personalized insights and suggestions based on mobile and YouTube usage.
* **Functional Requirements**:
  + **F3.1**: Generate insights (e.g., “You spent 2 hours on YouTube gaming videos, exceeding your 1-hour limit”).
  + **F3.2**: Suggest actions (e.g., “Reduce gaming videos to 30 minutes/day” or “Try these productivity videos”).
  + **F3.3**: Use Perplexity API (if available) to fetch external content (e.g., articles on screen time management).
  + **F3.4**: Allow users to set usage goals (e.g., “Limit YouTube to 1 hour/day”).
* **Priority**: High
* **Input**: Usage data, user-defined goals.
* **Output**: Text-based insights and recommendations.

**3.4 Pop-Up Notifications**

* **Description**: Deliver real-time alerts to warn users of excessive usage.
* **Functional Requirements**:
  + **F4.1**: Display pop-up notifications on the website (using React-Toastify) when usage limits are reached (e.g., “You’ve watched 1.5 hours of gaming videos”).
  + **F4.2**: Send push notifications via Firebase Cloud Messaging to mobile devices for real-time alerts.
  + **F4.3**: Allow users to customize notification triggers (e.g., time thresholds, specific apps).
* **Priority**: High
* **Input**: Real-time usage data, user preferences.
* **Output**: Pop-up alerts (web) and push notifications (mobile).

**3.5 User Dashboard**

* **Description**: Provide a visual interface for usage statistics and settings.
* **Functional Requirements**:
  + **F5.1**: Display charts (e.g., pie chart for YouTube category breakdown, line graph for weekly trends) using Chart.js.
  + **F5.2**: Show insights (e.g., “75% of YouTube time is gaming”) and suggestions.
  + **F5.3**: Allow users to set goals, customize categories, and manage notification settings.
  + **F5.4**: Ensure responsive design for web and mobile browsers.
* **Priority**: High
* **Input**: Usage data, user settings.
* **Output**: Interactive dashboard with charts and controls.

**3.6 Social Media Feed Control (Optional)**

* **Description**: Integrate a browser extension to filter social media feeds (e.g., X, YouTube) and restrict access based on usage limits.
* **Functional Requirements**:
  + **F6.1**: Develop a Chrome extension to manipulate DOM, hiding “unproductive” content (e.g., gaming videos, entertainment posts).
  + **F6.2**: Use YouTube API or Perplexity API to classify content for filtering.
  + **F6.3**: Block access to social media sites after user-defined limits (e.g., redirect to a blank page).
  + **F6.4**: Sync with website data to enforce consistent limits across platforms.
* **Priority**: Medium (optional feature)
* **Input**: Social media content, usage data.
* **Output**: Filtered feeds, restricted access.

**4. External Interface Requirements**

**4.1 User Interfaces**

* **Website**: Responsive React-based UI with:
  + Dashboard: Charts, insights, and settings.
  + Pop-Ups: Alerts for excessive usage.
  + Authentication: Google/Apple login for API access.
* **Companion Mobile App**: Flutter/React Native app for:
  + Syncing Digital Wellbeing/Screen Time data.
  + Displaying push notifications.
  + Optional manual data input (iOS).

**4.2 Hardware Interfaces**

* **Mobile Devices**: Android (API 21+) or iOS (15+) for usage data and notifications.
* **Browsers**: Chrome, Firefox, Safari for website and extension.

**4.3 Software Interfaces**

* **YouTube Data API v3**: For watch history and video metadata.
* **Android UsageStatsManager**: For app usage data.
* **iOS Screen Time**: Limited access; may require manual input.
* **Perplexity API**: For content analysis and recommendations (if available).
* **Firebase Cloud Messaging**: For push notifications.
* **Django REST Framework**: For API endpoints to sync data between app and website.

**4.4 Communication Interfaces**

* **HTTPS**: Secure API calls for data transfer.
* **WebSocket**: Real-time updates for dashboard and notifications.
* **OAuth 2.0**: Authentication for YouTube API and user login.

**5. Non-Functional Requirements**

**5.1 Performance Requirements**

* **Response Time**: API calls and dashboard updates < 2 seconds.
* **Data Sync**: Usage data synced every 10 minutes (configurable).
* **Scalability**: Support up to 10,000 users with minimal latency (AWS/Heroku).

**5.2 Security Requirements**

* **Data Privacy**: Encrypt usage data in transit (HTTPS) and at rest (PostgreSQL encryption).
* **User Consent**: Require explicit permission for YouTube history and app usage access.
* **Compliance**: Adhere to GDPR, CCPA, and Google/Apple policies.
* **Authentication**: Use OAuth 2.0 for secure Google/Apple login.

**5.3 Quality Attributes**

* **Usability**: Intuitive UI with clear instructions for granting permissions.
* **Reliability**: 99% uptime for website and API.
* **Maintainability**: Modular Django code with documentation for easy updates.

**5.4 Constraints**

* **API Quotas**: YouTube API limited to 10,000 units/day (free tier).
* **iOS Restrictions**: Limited Screen Time access; may require manual input.
* **Platform Policies**: Must comply with X, YouTube, and app store ToS to avoid bans.

**6. Other Requirements**

**6.1 Database Requirements**

* **PostgreSQL**:
  + Tables: Users, AppUsage (app name, time spent, date), YouTubeHistory (video ID, title, category, duration), UserSettings (goals, notification preferences).
  + Ensure indexing for fast queries on usage data.

**6.2 Regulatory Requirements**

* Comply with GDPR/CCPA for data collection and user consent.
* Adhere to Google’s YouTube API terms and Apple’s App Store guidelines.

**6.3 Documentation**

* User Manual: Guide for setting up permissions and using the dashboard.
* Developer Docs: API endpoints, AI model details, and deployment instructions.
* GitHub README: Showcase project for CV/portfolio.

**6.4 Testing Requirements**

* **Unit Tests**: For Django models, API endpoints, and AI classification.
* **Integration Tests**: For YouTube API, Firebase, and mobile app syncing.
* **User Testing**: Validate UI usability and notification effectiveness.

**7. Development Plan**

**7.1 Milestones**

1. **Month 1**: Set up Django backend, YouTube API integration, and basic dashboard.
2. **Month 2**: Implement mobile app for data syncing and Firebase notifications.
3. **Month 3**: Develop AI model for video categorization and usage analysis.
4. **Month 4**: Add Perplexity API integration (if available) and social media extension.
5. **Month 5**: Test, optimize, and deploy on Heroku; submit to app stores (if applicable).
6. **Month 6**: Gather user feedback and iterate.

**7.2 Tools and Technologies**

* **Backend**: Django, Django REST Framework, PostgreSQL.
* **Frontend**: React, Chart.js, React-Toastify.
* **Mobile App**: Flutter or React Native.
* **AI**: scikit-learn, TensorFlow, Perplexity API.
* **APIs**: YouTube Data API, Android UsageStatsManager, Firebase.
* **Deployment**: Heroku/AWS, GitHub for version control.

**7.3 Team Roles (for Portfolio Project)**

* **Developer**: You (Django, AI, frontend, and extension development).
* **Optional Collaborators**: UI/UX designer, tester (if scaling beyond solo project).

**8. Risks and Mitigation**

* **Risk**: YouTube API quota limits restrict frequent data pulls.
  + **Mitigation**: Cache data locally, optimize queries, or upgrade to paid quota.
* **Risk**: iOS Screen Time data access is limited.
  + **Mitigation**: Use manual input or companion app with Family Sharing APIs.
* **Risk**: Users distrust data collection.
  + **Mitigation**: Implement local processing, transparent privacy policies, and minimal data storage.
* **Risk**: Social media platforms block feed manipulation.
  + **Mitigation**: Use client-side DOM manipulation and monitor platform updates.

**9. Deliverables**

* **Website**: Deployed on Heroku with dashboard, notifications, and API endpoints.
* **Mobile App**: Optional Flutter/React Native app for data syncing and notifications.
* **Browser Extension**: Chrome extension for social media feed control (optional).
* **Documentation**: User manual, developer docs, and GitHub README.
* **Portfolio Entry**: Live demo link, GitHub repo, and project presentation.

**10. Appendix**

**10.1 Sample User Workflow**

1. User logs into ScreenWise via Google, grants YouTube API and Digital Wellbeing access.
2. Mobile app syncs usage data (e.g., 2 hours on YouTube, 1 hour on X).
3. Website analyzes YouTube history, showing “1.5 hours gaming, 0.5 hours tutorials.”
4. Pop-up alerts: “You’ve exceeded your 1-hour gaming limit; try these productivity videos.”
5. Dashboard displays charts and allows goal-setting (e.g., “Limit YouTube to 1 hour/day”).
6. Optional extension filters X/YouTube feeds, blocking access after limits.

**10.2 Sample Code Snippet (Django + YouTube API)**

python

CollapseWrapRun

Copy

from django.http import JsonResponse

from rest\_framework.views import APIView

from googleapiclient.discovery import build

from .models import YouTubeHistory

class YouTubeHistoryView(APIView):

def get(self, request):

youtube = build('youtube', 'v3', credentials=request.user.youtube\_credentials)

request = youtube.playlistItems().list(part='snippet', playlistId='HL', maxResults=50)

response = request.execute()

for item in response['items']:

video = item['snippet']

YouTubeHistory.objects.create(

user\_id=request.user.id,

video\_id=video['resourceId']['videoId'],

title=video['title'],

category=video.get('videoCategoryId', 'unknown'),

timestamp=video['publishedAt']

)

*# Analyze time spent*

total\_time = sum(video.duration for video in YouTubeHistory.objects.filter(user\_id=request.user.id))

if total\_time > 3600: *# 1 hour*

return JsonResponse({'message': 'Excessive YouTube use detected!', 'alert': True})

return JsonResponse({'message': 'Usage within limits', 'alert': False})