Data For Good

Overview

Data For Good is an open-source browser extension and backend system that enables users to voluntarily and anonymously donate their social media interaction data from **Instagram** and **X (Twitter)**. The project aims to help researchers and data scientists understand user engagement patterns, content trends, and platform dynamics, while prioritizing user privacy and transparency.

Table of Contents

How It Works

- **User installs the browser extension** (Chrome).
- **Extension injects content scripts** into Instagram and X (Twitter) pages.
- **User interactions** (views, likes, comments, shares, time spent, etc.) are detected in real time using DOM observers.
- **Data is anonymized** in the background script, batched, and sent to a Python backend.
- **Backend classifies content topics** using AI models (Hugging Face, Google Gemini), aggregates data by session, and stores it in AWS S3.
- **Gamification**: Users earn points and tiers for their contributions, visible in the extension popup.

Features

- **Multi-Platform Support**: Tracks Instagram and X (Twitter) interactions.
- **Anonymized Data Collection**: No personal identifiers are stored.
- **Rich Interaction Tracking**: Views, likes, comments, shares, time spent, media engagement, etc.

- **Content Classification**: Uses AI to classify post topics (zero-shot, image classification).
- **Session-Based Aggregation**: Data is grouped by user session for richer analytics.
- **Gamification**: Points and tier system to encourage participation.
- **User Consent & Privacy**: Data collection is opt-in and transparent.
- **Robust Error Handling**: Handles SPA navigation, network errors, and browser quirks.
- **Scalable Backend**: Asynchronous processing, AWS S3 storage, and extensible architecture.

Tech Stack

Frontend (Extension)

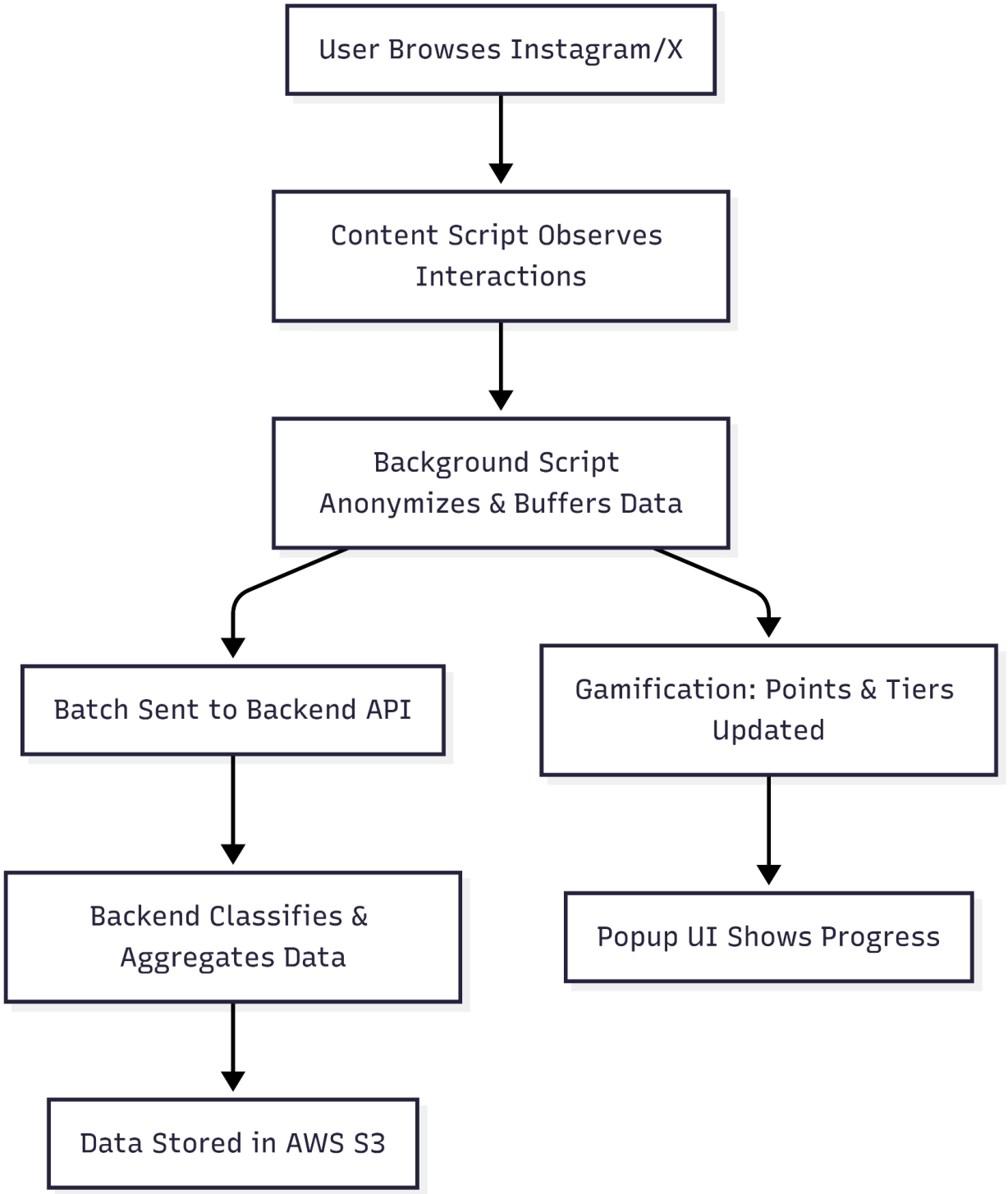
- JavaScript (ES6+)
- Chrome Extension APIs (Manifest V3)
- DOM APIs (MutationObserver, IntersectionObserver)
- HTML/CSS for popup UI

Backend

- Python 3
- Flask (REST API)
- ThreadPoolExecutor (async processing)
- Hugging Face Transformers (zero-shot classification)
- Google Gemini Vision API (image classification)
- AWS S3 (data storage)
- Docker (optional, for deployment)

Architecture & Workflow

High-Level Workflow



Detailed Steps

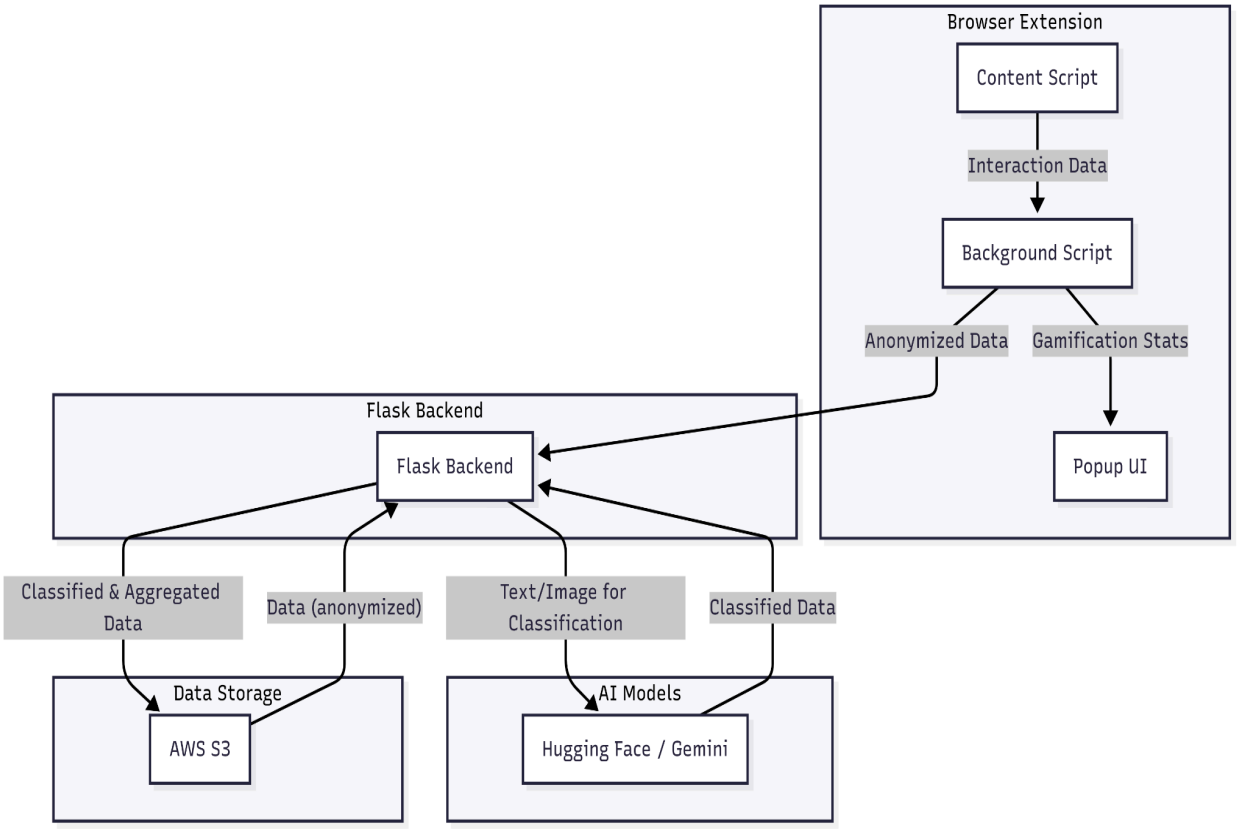
- **Content Script**
  - Detects posts, tracks views, likes, comments, shares, and time spent.
  - Handles SPA navigation and dynamic content loading.
- **Background Script**
  - Buffers and anonymizes data.
  - Adds session IDs, timestamps, and hashes user agent.
  - Implements points and tier logic.
  - Syncs data to backend in small batches.
- **Backend (Flask API)**
  - Receives data batches.
  - Classifies text with Hugging Face zero-shot models.
  - Classifies images with Google Gemini Vision API.
  - Aggregates data by session and stores in AWS S3.
- **Gamification**
  - Points awarded for each interaction.
  - Tier (level) increases every 100 points.
  - Stats are shown in the extension popup.

Key Facts

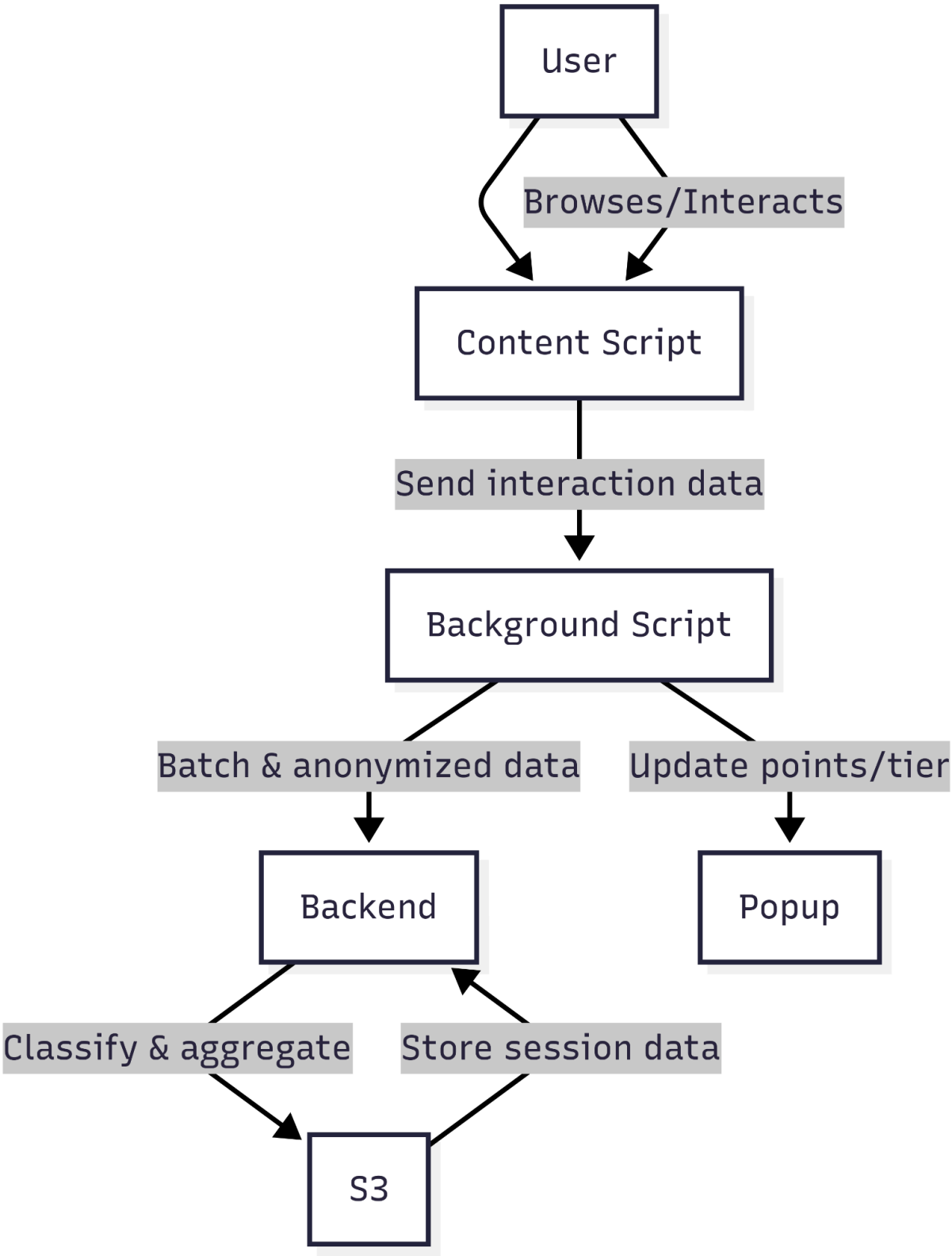
- **Privacy-First**: No usernames, emails, or direct identifiers are ever stored.
- **Session-Based**: Data is grouped by session, not by user.
- **Extensible**: Easy to add new platforms, interaction types, or analytics.
- **Open Source**: All code is available for review and contribution.

Diagrams

1. System Architecture



2. Extension Data Flow



Things That Can Be Improved

- **UI/UX**: Add more visual feedback, badge icons, and progress bars in the popup.
- **User Data Export**: Let users export their own anonymized data.
- **More AI Models**: Use additional models for sentiment, toxicity, or trend detection.
- **Performance**: Optimize for very large data volumes or slow networks.

Getting Started

Prerequisites

- Node.js & npm (for extension development)
- Python 3.8+ (for backend)
- AWS account (for S3 storage)
- Hugging Face and Google Gemini API keys (for classification)

Setup

1. Extension

- Load the extension/ folder as an unpacked extension in Chrome.
- Update manifest.json as needed.

2. Backend

- Install dependencies:
  pip install -r backend/requirements.txt

- Set environment variables for AWS and API keys. Create a .env file in the backend directory:
  AWS_ACCESS_KEY_ID=<Your AWS access key>
  AWS_SECRET_ACCESS_KEY=<Your AWS secret key>
  AWS_REGION=<AWS region, e.g., us-east-1>
  S3_BUCKET=<Name of your S3 bucket>
  GEMINI_API_KEY=<Your Gemini API Key>

- Run the backend:
  python backend/app.py

3. Connect Extension to Backend

- Ensure the backend URL in background.js matches your backend server (e.g., http://127.0.0.1:5000/collect).

Contributing

Contributions are welcome! Please open issues or pull requests for bug fixes, new features, or documentation improvements.

License

MIT License