MANIPAL INSTITUTE OF TECHNOLOGY
BENGALURU
(A constituent unit of MAHE, Manipal)

WSMA MINI PROJECT REPORT

On

GitHub Profile Analyzer

Submitted By:

Varikuti Mahathi Reddy

235816142

Submitted To:

Ms. Charu Chauhan

Assistant Professor, School of Computer Engineering

School of Computer Engineering

Date of Submission: 20/10/2025

Abstract

The Personal GitHub Profile Analyzer is an interactive web dashboard designed to provide a comprehensive, data-driven overview of any public GitHub user's profile. In the modern software development landscape, a GitHub profile serves as a dynamic resume. However, manually assessing a user's skills, primary interests, and activity level is often time-consuming and subjective. This project addresses that gap by creating a tool that automates the analysis process.

Using Python and the Streamlit framework, the application fetches data directly from the official GitHub API. It processes this data to generate key insights, including a user's most frequently used programming languages, their most popular repositories by star count, and general account statistics. A key feature is the application of Natural Language Processing (NLP) through the TF-IDF algorithm to analyze the textual content of repository README files, extracting the most significant project keywords.

The final output is a clean, interactive, and easily digestible dashboard that visualizes these metrics, offering a quick yet powerful summary of a developer's technical footprint. The project successfully demonstrates the integration of external APIs, data analysis, NLP techniques, and web-based visualization.

1. Introduction

For software developers, recruiters, and technical managers, a GitHub profile has become a critical tool for showcasing technical abilities and project experience. It acts as a living portfolio, offering a transparent view into a developer's coding habits, interests, and contributions to the open-source community. While valuable, interpreting this information quickly and objectively presents a challenge. Assessing a profile often involves manually browsing through repositories, reading code, and trying to piece together a coherent picture of the user's expertise.

This project was initiated to streamline this process by creating an automated tool that generates a concise and insightful summary of any public GitHub profile. The "Personal GitHub Profile

Analyzer" is a web-based dashboard that transforms raw profile data into actionable insights, making it easier to understand a developer's technical strengths and areas of focus at a glance.

2. Problem Statement

The primary problem is the lack of a simple, accessible tool for the rapid, data-driven analysis of a GitHub user's public profile. Key stakeholders face the following challenges:

- **Developers**: Find it difficult to objectively assess the focus of their own profile or that of potential collaborators.
- **Recruiters**: Spend significant time manually reviewing profiles to gauge a candidate's primary skills and experience.
- **Managers**: Need a quick way to understand the technical expertise of team members or new hires.

Existing methods are manual, subjective, and do not efficiently aggregate key metrics like language proficiency, project popularity, or thematic focus. This project aims to solve this problem by providing an automated, visual, and quantitative summary.

3. Objectives

The primary objectives of this project are as follows:

- To develop a user-friendly, interactive web dashboard using the Streamlit framework.
- To ethically fetch public user and repository data from the official GitHub API.
- To analyze and aggregate the programming languages used across a user's repositories.
- To identify and rank a user's top repositories based on community engagement (star count).
- To apply Natural Language Processing (TF-IDF) to perform keyword extraction on repository README files, identifying a user's main project themes.
- To visualize all analyzed data in a clean, consolidated dashboard.
- To provide a feature for downloading a text-based summary of the generated report.

4. Scope

The scope of this project is defined by the following boundaries:

In Scope:

- Analysis of public GitHub user profiles and their public repositories only.
- Data fetching is limited to the information provided by the public GitHub REST API.
- Text analysis is confined to the content of repository README files.
- Visualization includes top 5 programming languages and top 5 repositories.
- Keyword extraction is limited to the top 15 most relevant terms.

Out of Scope:

- Analysis of private repositories or organizational data.
- Analysis of commit history, contribution frequency, or code complexity.
- User authentication (the application uses a developer's API token on the backend for rate-limiting purposes only).
- Sentiment analysis of issue comments or pull request discussions.

5. System Design / Methodology

The project follows a simple, modular architecture designed to separate data collection, analysis, and presentation.

5.1. Requirements

- **Functional:**
    - The system must provide an input field for a GitHub username.
    - It must display the user's avatar, name, and key statistics (follower count, etc.).
    - It must display a bar chart of the top programming languages.
    - It must list the top repositories by star count with links.
    - It must display a list of extracted project keywords.

    - It must provide a button to download the report as a .txt file.
- **Non-Functional:**
    - The application must be responsive and provide feedback (e.g., a loading spinner) during data fetching.
    - The dashboard must be clean, readable, and present information clearly.
    - API calls should be efficient to minimize load times.

5.2. Design (Architecture)

The application follows a simple data flow:

1. **User Interface (Streamlit Frontend):** The user enters a GitHub username and clicks "Analyze".
2. **Backend Logic (Python/Streamlit):**
    - The request triggers the main analysis function.
    - **Data Collection:** Helper functions make sequential GET requests to the GitHub API to fetch user data, repository lists, and the raw text of README files.
    - **Data Processing & Analysis:** The fetched JSON data is parsed. The code aggregates language data, sorts repositories, and performs text cleaning and TF-IDF analysis on the README content.
    - **Visualization:** A Matplotlib chart is generated from the language data.
3. **User Interface (Streamlit Frontend):** The Streamlit app dynamically renders the processed data, tables, and the Matplotlib chart on the dashboard.

5.3. Tools & Technologies

- **Python:** The core programming language for the entire application.
- **Streamlit:** The web framework used to build the interactive dashboard with Python-only code.

- **Requests:** The library used for making HTTP requests to the GitHub API.
- **Pandas:** Used for organizing repository data into a structured DataFrame for easy display.
- **Scikit-learn:** Used for its TfidfVectorizer to perform keyword extraction.
- **Matplotlib:** The library used to generate the language distribution bar chart.
- **GitHub API:** The external REST API used as the primary data source.

6. Implementation

The project is implemented in a single Python script (app.py) using the Streamlit framework. The code is organized into several key functional areas:

- **API Helper Functions (get_user_data, get_repos, get_readme_content):** Each function is responsible for a specific API call, handling authentication headers and response parsing. This modularity makes the code easy to debug.
- **Analysis Functions (analyze_languages, get_top_repos, extract_keywords):** These functions take the raw data from the API helpers and perform specific analytical tasks. The extract_keywords function includes a text cleaning pipeline using regular expressions to remove URLs, HTML tags, and other noise before analysis.
- **Visualization Function (create_language_chart):** This function takes the aggregated language data and uses Matplotlib to generate a styled bar chart, which is then passed to the Streamlit frontend.
- **Streamlit UI:** The final section of the code uses Streamlit's commands (st.title, st.text_input, st.columns, st.pyplot, st.dataframe) to construct the entire user interface and orchestrate the calls to the analysis functions.

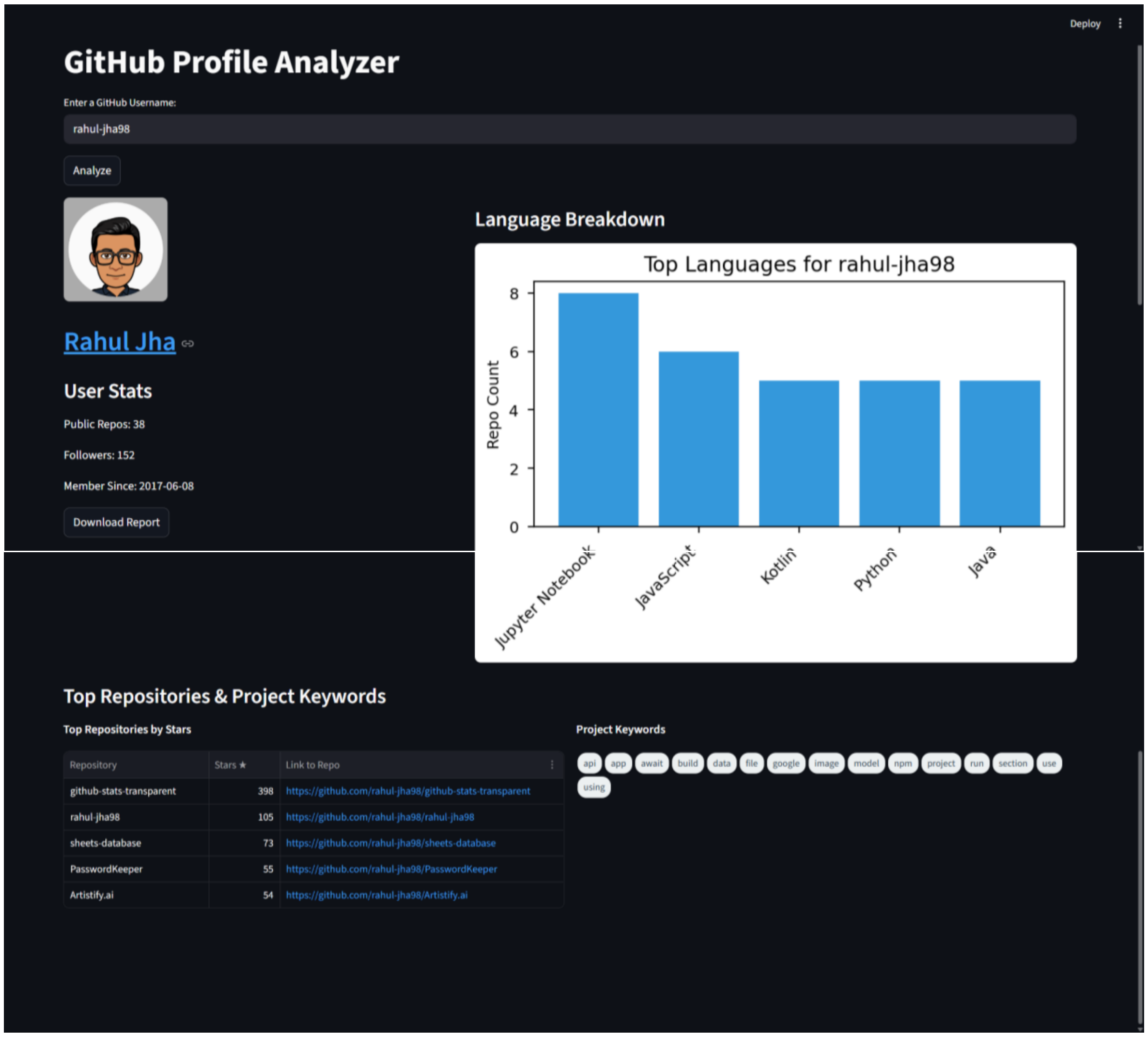
7. Testing and Results

The application was tested with various GitHub usernames to ensure robustness, including profiles with many repositories (e.g., google), profiles with few repositories, and profiles with non-English READMEs. The error handling for non-existent users was also verified.

Results:

The final application successfully generates a dashboard as intended. For a sample user, the dashboard provides a clear overview:

- The **User Stats** section gives an immediate sense of the user's scale and tenure.
- The **Language Breakdown** chart quickly identifies the user's primary tech stack.
- The **Top Repositories** table highlights their most impactful work.
- The **Project Keywords** section provides a deeper, thematic insight into their specific interests and expertise (e.g., AI, web development, data science).



GitHub Profile Analyzer

Enter a GitHub Username:

rahul-jha98

Rahul Jha

User Stats

Public Repos: 38

Followers: 152

Member Since: 2017-06-08

Language Breakdown

Top Repositories & Project Keywords

Top Repositories by Stars

Repository	Stars ★	Link to Repo
github-stats-transparent	398	https://github.com/rahul-jha98/github-stats-transparent
rahul-jha98	105	https://github.com/rahul-jha98/rahul-jha98
sheets-database	73	https://github.com/rahul-jha98/sheets-database
PasswordKeeper	55	https://github.com/rahul-jha98/PasswordKeeper
Artistify.ai	54	https://github.com/rahul-jha98/Artistify.ai

Project Keywords

api app await build data file google image model npm project run section use using

8. Discussion

The project successfully demonstrates that a great deal of insight can be derived from the GitHub API. The combination of quantitative data (like star counts and language usage) with qualitative data (keyword analysis from READMEs) provides a well-rounded view of a developer's profile.

One key finding during development was the necessity of a robust text-cleaning pipeline for the keyword extraction. Initial versions produced noisy keywords from URLs, image links, and code snippets. The implementation of a multi-step cleaning process using regular expressions significantly improved the quality and relevance of the extracted keywords.

A limitation of the project is its reliance on well-maintained README files for keyword analysis. Users with sparse or empty READMEs will yield less insightful keyword results.

9. Conclusion and Future Work

Conclusion:

This project successfully achieved its objective of creating a functional, interactive dashboard for analyzing GitHub profiles. It serves as a valuable tool for developers, recruiters, and managers by automating the process of data collection and presenting insights in a clear, visual format. The use of Streamlit allowed for rapid development and iteration, resulting in a polished and professional-looking application.

Future Work:

The project has several avenues for future enhancement:

- **Commit Activity Analysis:** Integrate analysis of commit history to visualize a user's contribution frequency and patterns over time.
- **Contribution Graph:** Recreate the iconic GitHub contribution graph with more detailed statistics.
- **User Comparison:** Add a feature to analyze two user profiles side-by-side.

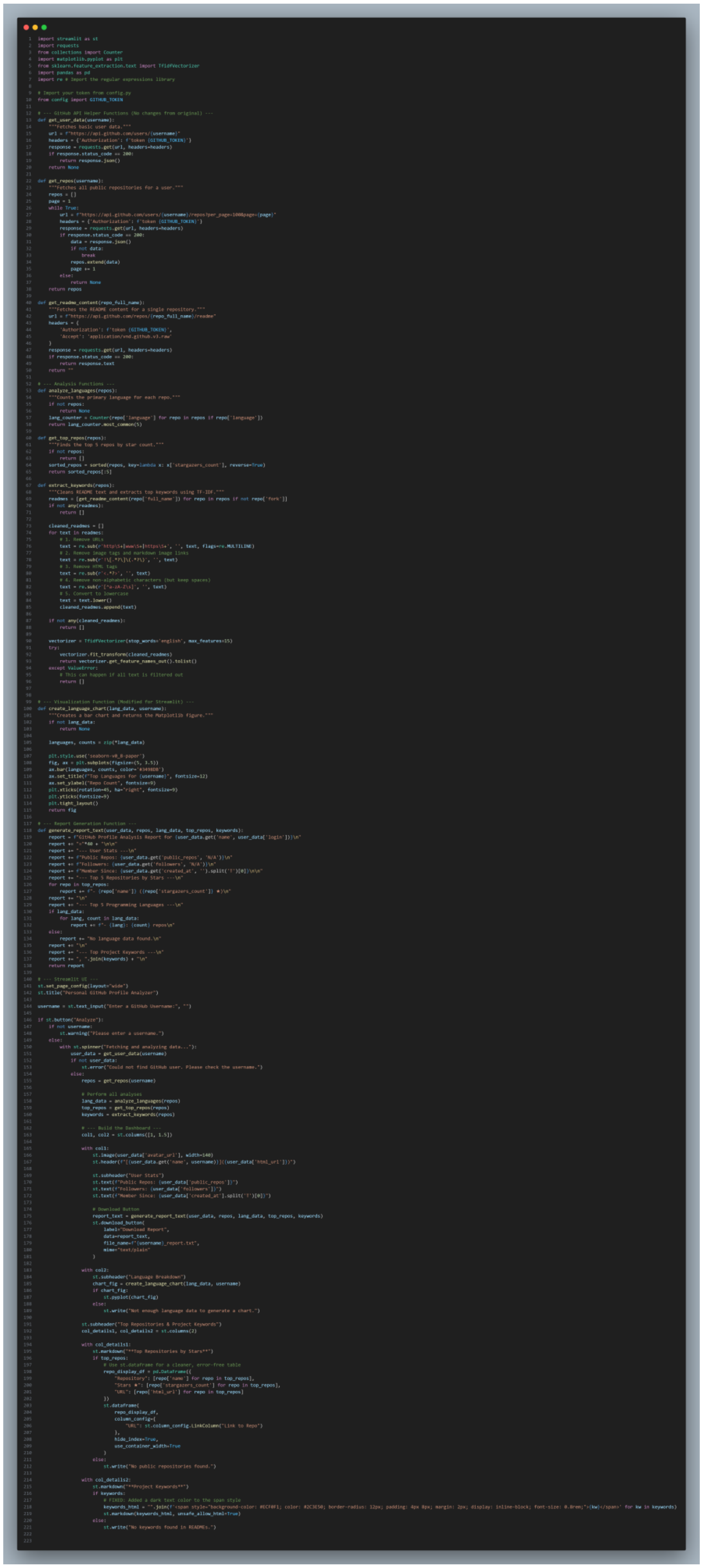
- **Deployment:** Deploy the application to a cloud service like Streamlit Community Cloud to make it publicly accessible.

10. References

- **Streamlit (Framework):** Streamlit Inc. (2023). *Streamlit Documentation*. Retrieved October 20, 2025, from https://docs.streamlit.io/
- **GitHub API (Data Source):** GitHub, Inc. (2023). *GitHub REST API Documentation*. Retrieved October 20, 2025, from https://docs.github.com/en/rest
- **Scikit-learn (NLP Library):** Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.
- **Matplotlib (Visualization Library):** Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90-95.
- **Pandas (Data Manipulation Library):** The pandas development team. (2023). *pandas-dev/pandas: Pandas*. Zenodo. https://doi.org/10.5281/zenodo.8282384

11. Appendices

Appendix A: Source Code



```python
import streamlit as st
import requests
from collections import Counter
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
import re # Import the regular expressions library

# Import your token from config.py
from config import GITHUB_TOKEN

# --- GitHub API Helper Functions (No changes from original) ---
def get_user_data(username):
    """Fetches basic user data."""
    url = f"https://api.github.com/users/{username}"
    headers = {'Authorization': f'token {GITHUB_TOKEN}'}
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        return response.json()
    return None

def get_repos(username):
    """Fetches all public repositories for a user."""
    repos = []
    page = 1
    while True:
        url = f"https://api.github.com/users/{username}/repos?per_page=100&page={page}"
        headers = {'Authorization': f'token {GITHUB_TOKEN}'}
        response = requests.get(url, headers=headers)
        if response.status_code == 200:
            data = response.json()
            if not data:
                break
            repos.extend(data)
            page += 1
        else:
            return None
    return repos

def get_readme_content(repo_full_name):
    """Fetches the README content for a single repository."""
    url = f"https://api.github.com/repos/{repo_full_name}/readme"
    headers = {
        'Authorization': f'token {GITHUB_TOKEN}',
        'Accept': 'application/vnd.github.v3.raw'
    }
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        return response.text
    return ""

# --- Analysis Functions ---
def analyze_languages(repos):
    """Counts the primary language for each repo."""
    if not repos:
        return None
    lang_counter = Counter(repo['language'] for repo in repos if repo['language'])
    return lang_counter.most_common(5)

def get_top_repos(repos):
    """Finds the top 5 repos by star count."""
    if not repos:
        return []
    sorted_repos = sorted(repos, key=lambda x: x['stargazers_count'], reverse=True)
    return sorted_repos[:5]

def extract_keywords(repos):
    """Cleans README text and extracts top keywords using TF-IDF."""
    readmes = [get_readme_content(repo['full_name']) for repo in repos if not repo['fork']]
    if not any(readmes):
        return []

    cleaned_readmes = []
    for text in readmes:
        # 1. Remove URLs
        text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
        # 2. Remove image tags and markdown image links
        text = re.sub(r'!\[.*?\]\(.*?\)', '', text)
        # 3. Remove HTML tags
        text = re.sub(r'<.*?>', '', text)
        # 4. Remove non-alphabetic characters (but keep spaces)
        text = re.sub(r'[^a-zA-Z\s]', '', text)
        # 5. Convert to lowercase
        text = text.lower()
        cleaned_readmes.append(text)

    if not any(cleaned_readmes):
        return []

    vectorizer = TfidfVectorizer(stop_words='english', max_features=15)
    try:
        vectorizer.fit_transform(cleaned_readmes)
        return vectorizer.get_feature_names_out().tolist()
    except ValueError:
        # This can happen if all text is filtered out
        return []


# --- Visualization Function (Modified for Streamlit) ---
def create_language_chart(lang_data, username):
    """Creates a bar chart and returns the Matplotlib figure."""
    if not lang_data:
        return None

    languages, counts = zip(*lang_data)

    plt.style.use('seaborn-v0_8-paper')
    fig, ax = plt.subplots(figsize=(5, 3.5))
    ax.bar(languages, counts, color='#3498DB')
    ax.set_title(f"Top Languages for {username}", fontsize=12)
    ax.set_ylabel("Repo Count", fontsize=9)
    plt.xticks(rotation=45, ha="right", fontsize=9)
    plt.yticks(fontsize=9)
    plt.tight_layout()
    return fig

# --- Report Generation Function ---
def generate_report_text(user_data, repos, lang_data, top_repos, keywords):
    report = f"GitHub Profile Analysis Report for {user_data.get('name', user_data['login'])}\n"
    report += "="*40 + "\n\n"
    report += "--- User Stats ---\n"
    report += f"Public Repos: {user_data.get('public_repos', 'N/A')}\n"
    report += f"Followers: {user_data.get('followers', 'N/A')}\n"
    report += f"Member Since: {user_data.get('created_at', '').split('T')[0]}\n\n"
    report += "--- Top 5 Repositories by Stars ---\n"
    for repo in top_repos:
        report += f"- {repo['name']} ({repo['stargazers_count']} ★)\n"
    report += "\n"
    report += "--- Top 5 Programming Languages ---\n"
    if lang_data:
        for lang, count in lang_data:
            report += f"- {lang}: {count} repos\n"
    else:
        report += "No language data found.\n"
    report += "\n"
    report += "--- Top Project Keywords ---\n"
    report += ", ".join(keywords) + "\n"
    return report

# --- Streamlit UI ---
st.set_page_config(layout="wide")
st.title("Personal GitHub Profile Analyzer")

username = st.text_input("Enter a GitHub Username:", "")

if st.button("Analyze"):
    if not username:
        st.warning("Please enter a username.")
    else:
        with st.spinner("Fetching and analyzing data..."):
            user_data = get_user_data(username)
            if not user_data:
                st.error("Could not find GitHub user. Please check the username.")
            else:
                repos = get_repos(username)

                # Perform all analyses
                lang_data = analyze_languages(repos)
                top_repos = get_top_repos(repos)
                keywords = extract_keywords(repos)

                # --- Build the Dashboard ---
                col1, col2 = st.columns([1, 1.5])

                with col1:
                    st.image(user_data['avatar_url'], width=140)
                    st.header(f"[{user_data.get('name', username)}]({user_data['html_url']})")

                    st.subheader("User Stats")
                    st.text(f"Public Repos: {user_data['public_repos']}")
                    st.text(f"Followers: {user_data['followers']}")
                    st.text(f"Member Since: {user_data['created_at'].split('T')[0]}")

                    # Download Button
                    report_text = generate_report_text(user_data, repos, lang_data, top_repos, keywords)
                    st.download_button(
                        label="Download Report",
                        data=report_text,
                        file_name=f"{username}_report.txt",
                        mime="text/plain"
                    )

                with col2:
                    st.subheader("Language Breakdown")
                    chart_fig = create_language_chart(lang_data, username)
                    if chart_fig:
                        st.pyplot(chart_fig)
                    else:
                        st.write("Not enough language data to generate a chart.")

                st.subheader("Top Repositories & Project Keywords")
                col_details1, col_details2 = st.columns(2)

                with col_details1:
                    st.markdown("**Top Repositories by Stars**")
                    if top_repos:
                        # Use st.dataframe for a cleaner, error-free table
                        repo_display_df = pd.DataFrame({
                            "Repository": [repo['name'] for repo in top_repos],
                            "Stars ★": [repo['stargazers_count'] for repo in top_repos],
                            "URL": [repo['html_url'] for repo in top_repos]
                        })
                        st.dataframe(
                            repo_display_df,
                            column_config={
                                "URL": st.column_config.LinkColumn("Link to Repo")
                            },
                            hide_index=True,
                            use_container_width=True
                        )
                    else:
                        st.write("No public repositories found.")

                with col_details2:
                    st.markdown("**Project Keywords**")
                    if keywords:
                        # FIXED: Added a dark text color to the span style
                        keywords_html = "".join(f'<span style="background-color: #ECF0F1; color: #2C3E50; border-radius: 12px; padding: 4px 8px; margin: 2px; display: inline-block; font-size: 0.8rem;">{kw}</span>' for kw in keywords)
                        st.markdown(keywords_html, unsafe_allow_html=True)
                    else:
                        st.write("No keywords found in READMEs.")
```