Project Overview

This project is an app similarity surveillance system designed to assist companies in identifying apps on Google Play and the App Store that are similar to their own products. The system comprises multiple components, including a Node.js/Next.js web application, Firebase backend services, a Python API, and scheduled tasks, all working together to automate data surveillance and report generation. Here are the main components and functionalities of the project:

Key Components

1. ### Node.js/Next.js Web Application
   - **Role**: The Node.js/Next.js application serves as the project's front end, providing a user-friendly interface for users to view, add, delete, and edit the company's Products.
   - **Functions**: Users can access application information related to the company's products through this application and interact with Firebase to access surveillance Reports.

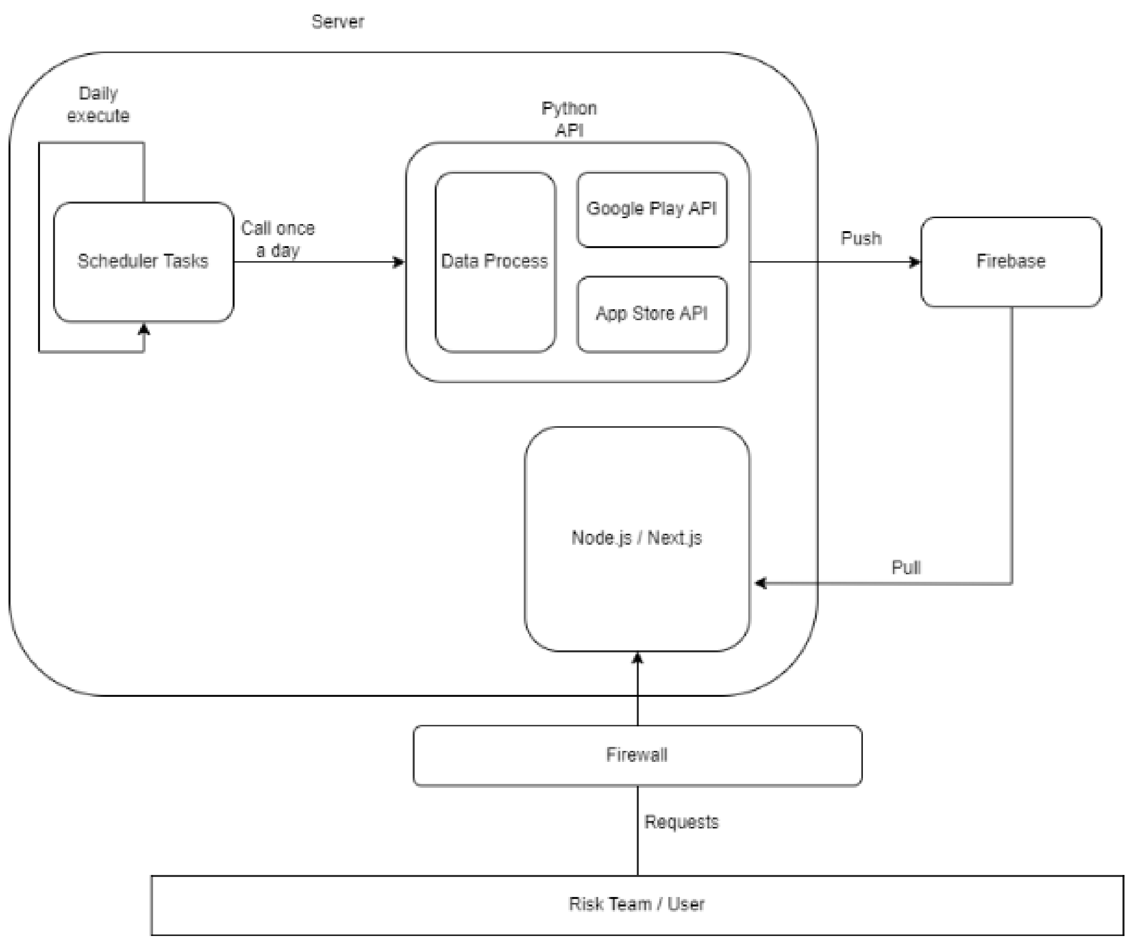
2. ### Firebase
   - **Role**: Firebase is the project's backend service responsible for storing app surveillance data and generated reports.
   - **Functions**: The Node.js/Next.js application and Python API interact with Firebase for data exchange, including storing surveillance results in the Firebase database.

3. ### Python API
   - **Role**: The Python API is a core component responsible for executing app similarity surveillance on Google Play and the App Store.
   - **Functions**: The Python API interacts with the Google Play API and the App Store API to search for apps similar to the company's products. Once similar apps are found, the API generates surveillance reports and uploads them to Firebase for user Access.

4. Google Play API and App Store API
    - **Role**: These APIs communicate with the Google Play Store and the App Store to retrieve app information and provide it to the Python API for analysis.
    - **Functions**: These APIs assist the Python API in finding relevant app information, including names, developer details, and more.

5. Scheduler Tasks
    - **Role**: Scheduled tasks are responsible for running the Log Generator as per the daily schedule to generate surveillance reports.
    - **Functions**: Daily execution of the Log Generator ensures the continuous surveillance of apps, generating up-to-date reports and passing them to Firebase for user access.

Project architecture



Project Workflow

1. Scheduled tasks (Scheduler Tasks) execute the Log Generator daily to create surveillance reports.

2. The Python API utilizes the Google Play API and the App Store API to search for and identify apps similar to the company's products.

3. The Python API generates surveillance reports and uploads them to the Firebase database.

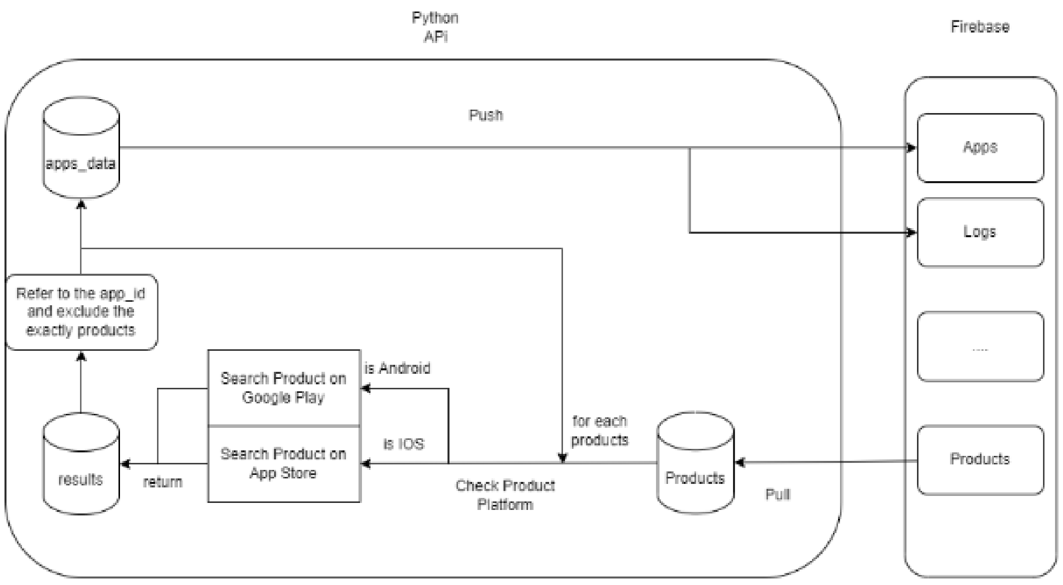
4. The Node.js/Next.js application offers a user-friendly interface, allowing users to access

and view these surveillance reports.

5. Users can view, add, delete, and edit the company's products in the Node.js/Next.js Application.

Python API

Python API architecture



Python API Workflow

1. **Retrieve Products Data (Firebase)** - The workflow begins by fetching data for the company's products from Firebase.

2. **Check Platform for Each Product** - For each product in the data, the workflow verifies the platform (Android or iOS).

3. **Search for Similar Apps** - The workflow uses the Google Play API and/or the App Store API to search for applications similar to the company's products.

4. **Exclude Company's Apps** - In the results, the workflow filters out applications developed by the company itself to avoid false positives.

5. **Generate Logs** - The workflow generates logs to record the scouting process and its Results.

6. **Upload to Firebase** - The final results, including the list of potentially similar applications are uploaded to Firebase.

This revised workflow ensures that the company's products data is initially retrieved, followed by a process of platform verification, application search, result filtering, logging, and ultimately uploading the results to Firebase. This workflow is designed to assist in identifying and monitoring potential competitor applications effectively.

Installation Document

Requirements

Ram: 1 GB
storage space: 10GB
Broadband: 50Mbps
Operating System: Windows
Node.js, Next.js, Python

Installation steps

This part provides step-by-step instructions for installing and setting up your project, which comprises two main components: a Python API and a Next.js web application.

Installing the Python API

Step 1: Install Required Python Packages

The first step is to install the required Python packages by running the following command. Ensure you have Python and pip installed on your system.
pip install -r requirement.txt

Step 2: Configure API Settings

Next, navigate to the `API` directory and locate the `run.bat` file. In this file, make the following modifications:

- Replace `<log_generator.py>` with the absolute path to the `log_generator.py` script.
- Replace `<m1project-d3fa3-firebase-adminsdk-nkgqq-e0d35363cf.json>` with the absolute path to the JSON file.

Your `run.bat` file should now be correctly configured.

Step 3: Task scheduling

To automate the daily execution of the Python API for market scouting, phishing attack mitigation, and copyright maintenance, you should set up task scheduling. Task scheduling allows you to run the API at specified intervals without manual intervention.

Follow these steps to schedule tasks:

Method 1: Using Windows Task Scheduler

1. Open Windows Task Scheduler:
    - Press `Win + S`, type "Task Scheduler," and press Enter to open Task Scheduler.

2. Create a Basic Task:
    - In the right-hand panel, click on "Create Basic Task..." to open the Basic Task Wizard.

3. Set Task Name and Description:
    - Provide a name and description for your task. Click "Next" to proceed.

4. Choose Trigger:
    - Select a trigger type. For daily execution, choose "Daily" and click "Next."

5. Set Daily Trigger Details:
    - Specify the time and recurrence pattern for your daily task. Click "Next" to proceed.

6. Select Action:
    - Choose "Start a Program" and click "Next."

7. Configure Action:
    - In the "Program/script" field, browse to the location of your Python API script (e.g.,`run.bat`).
    - Click "Next" to proceed.

8. Review and Confirm:
    - Review your task settings. If everything looks correct, click "Finish."

9. Your task is now scheduled and will run daily at the specified time.

Method 2: Using Command Prompt

You can also schedule tasks using the Windows Command Prompt. Here's how:

1. Open Command Prompt:
    - Press `Win + S`, type "Command Prompt," and press Enter to open Command Prompt.
    -
2. Schedule the Task:
    - Use the `schtasks` command to schedule your task. The basic syntax is as follows:
      ```
      schtasks /create /sc DAILY /tn "AppGuardianLogGenerator" /tr
      "C:\Path\to\Script.bat" /st 09:15
      ```

2. Replace the following:

        - `"C:\Path\to\Script.bat"`: Replace with the full path to your script or batch file (e.g.,`run.bat`).

    For example:
    schtasks /create /sc DAILY /tn "AppGuardianLogGenerator" /tr "C:\Path\to\run.bat" /st 09:15

3. Confirm the Task:
    ● The Command Prompt will confirm that the task was created successfully.

Your task is now scheduled and will run daily at the specified time.

Choose the method that best suits your preferences and needs for task scheduling in Windows.

Your Python API is now fully installed, configured, and scheduled for regular use.

Installing the Next.js Project

Step 1: Define Port Number

In the Next.js project, you may need to specify the port number you want the application to run on. This is typically done in the `server.js` file. Update the configuration as needed.

Step 2: Run Next.js as a Windows Service

To run your Next.js application as a Windows service, you'll need to use the `qckwinsvc` package.

Step 2.1: Install qckwinsvc

Run the following command to install the `qckwinsvc` package globally on your system:

npm install -g qckwinsvc

Step 2.2: Create the Service

Once `qckwinsvc` is installed, you can create a Windows service for your Next.js application. Run the following command and provide the requested information when prompted:

Qckwinsvc

- Service name: Enter a name for your service.

- Service description: Provide a brief description of the service.
- Node script path: Enter the absolute path to the `server.js` script in your Next.js project directory.
- Should the service get started immediately: Choose 'y' for yes if you want the service to start immediately.

Upon successful completion, you will see the following message:

```
Service installed.
Service started.
```

Your Next.js project is now set up as a Windows service and is ready to be used.