Project Specification

Introduction

- **Problem Statement**: Many users fall victim to harmful or fraudulent links. This project aims to protect Android users by intercepting link clicks and analyzing them against a database of malicious URLs, opening them in a “sandbox” environment and running the url in a trained ai model to add another layer of security.

- **Project Goal**: Build an Android application that intercepts link clicks, checks them in real time, and blocks the link if it is identified as malicious at high likelihood. This will help protect people from phishing scams and other types of malicious websites.

User interaction

This app should work in the background and users shouldnt interact with it very often.

Once a malicious link is detected the user will be notified, the link click will be intercepted and the user will be able to proceed to the link if he trusts the source.

There might be some indication whenever a link is clicked to let the use know the app works in the background and approved the link (we are not sure about the way this will be implemented yet, if at all)

Features & Requirements

- **Link Interception**: The app should capture the event of clicking on any link.

- **Link Prevention**: The app will block the event from completing until the app approves the link.

- **Malicious Link Database**: The app should maintain or access a database of known malicious or phishing URLs.

- **User Notification**: If a link is found to be dangerous, the user should be informed with an alert or warning dialog.

- **Allow/Block Decision**: Give the user an option to proceed (in some cases) or block automatically if the link is confirmed as malicious.

- **Context reading**: read the context surrounding the link (i.e SMS message, email, web page), this also allows more information for detecting malicious links.

- **Offline Capabilities** (Optional): App can store a local copy of the malicious URL DB for faster response time and seamless interactions.

- **Sandbox** (optional): might be able to open the link in a sterile environment and check the content of the page once the link is clicked allowing even more information to decide if a link is safe.

- **Performance Constraints**: The interception and analysis should not significantly degrade user experience (i.e., minimal latency).

Admin Interface for System Monitoring and Decision Influence

The proposed Admin Interface serves as a centralized tool for system administrators to monitor and influence the behavior of the system in a structured and user-friendly way. The primary goals of this interface are:

1. **Monitoring System Responses:**

   - Enable administrators to review examples of inputs received by the system, the corresponding responses provided, and the rationale behind those responses (if available).

2. **Decision Influence:**

   - Allow administrators to make adjustments that influence the system's behavior in the future without modifying the underlying codebase. (like greenlighting a certain link or adding it to the database from the client itself)
   - Examples include:
     Marking links as valid/invalid to refine the system's validation mechanisms.
     Updating internal decision-making rules or response parameters directly through the interface.

3. **User Notification:**

   - In some cases, provide a mechanism for administrators to notify users about specific updates, such as correcting a misclassified link. Or a general notification for a known malicious source to avoid.

User Interface Description

The user interface (UI) is designed to ensure seamless integration of the app’s security features while maintaining a user-friendly experience. Key elements of the UI include:

1. Link Interception Feedback:

- When a link is clicked, a subtle, non-intrusive loading indicator will appear to inform the user that the link is being analyzed for safety.

2. Alert Dialog for Dangerous Links:

- If a link is identified as malicious, the user will receive a clear and concise warning dialog.
- The dialog will include a warning message explaining why the link is considered dangerous.
- Options:
    - Block Link : Prevents the link from opening.
    - Proceed Anyway : Lets the user open the link at their own risk, with a secondary confirmation step for added safety.

3. User Action History:

- A section in the app will display a history of intercepted links, categorized as “Safe,” “Blocked,” or “Allowed at Risk.”
- Each entry will include details such as the link, the context (e.g., SMS, email, web page), and the decision made.

4. Settings Menu:

- Link Handling Preferences:
- Choose between automatic blocking for all malicious links or manual decision prompts.
- Database Updates:
    - Option to enable offline mode for using a local malicious link database.
    - Sandbox Settings (TBD):

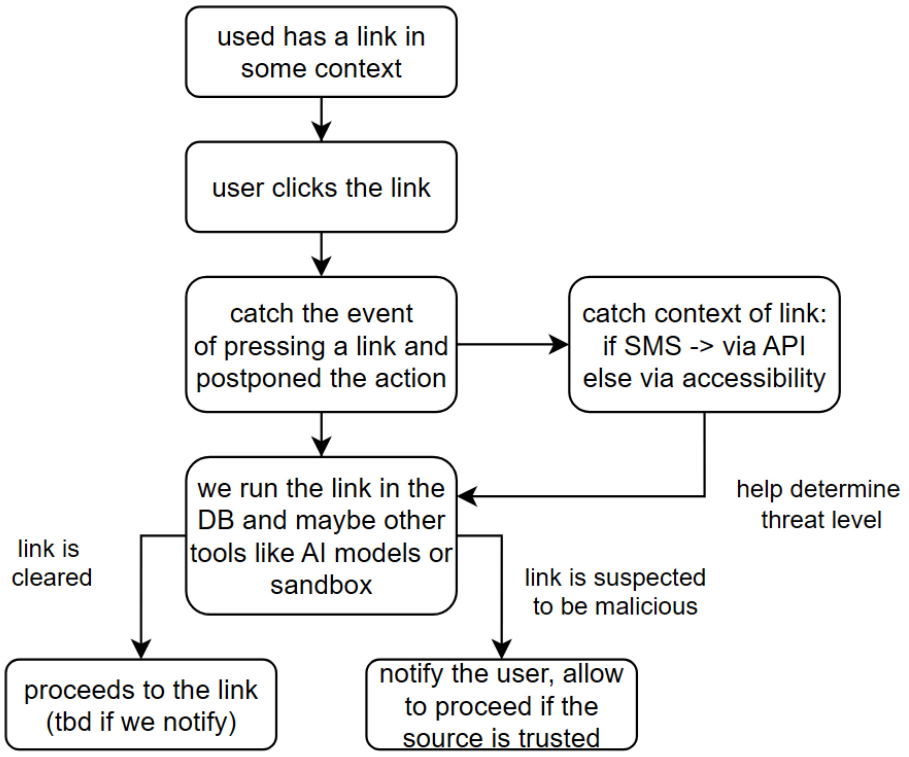
5. Performance Assurance:

- The UI will prioritize minimal interference, ensuring link interception and decisions feel fast and smooth.
- Notifications and dialogs will use modern, clean designs with clear actions and minimal clutter to avoid overwhelming the user.

The overall design focuses on clarity, simplicity, and empowering users to make informed decisions about their online safety.

User Flow & Use Cases

- User clicks on a link in any app or browser.
- The interception module activates and extracts the URL.
- The URL is checked against a local/remote malicious links database (possibly other tools as mentioned)
- Catch context of link and analyze threat level
- If malicious, display warning or block access. If safe, open the link as normal (with maybe a little indication that the link is safe).


used has a link in some context
user clicks the link
catch the event of pressing a link and postponed the action
catch context of link: if SMS -> via API else via accessibility
help determine threat level
we run the link in the DB and maybe other tools like AI models or sandbox
link is cleared
link is suspected to be malicious
proceeds to the link (tbd if we notify)
notify the user, allow to proceed if the source is trusted

High-Level Design

Division Between Client and Server

Client (Android App):

- Handles link interception and initial analysis.
- Reads context surrounding the link (SMS->api, else->Accessibility)
- Performs offline checks against a local database for malicious links.
- Sends requests to a server or external API for further analysis if local checks are inconclusive.
- Provides user feedback and allows actions (e.g., "Block" or "Proceed").

Server:

- Stores and manages a comprehensive database of malicious links.
- Provides a REST API for the client to query suspicious links.
- Could be hosted on:
  - **Cloud Platforms** (AWS, Google Cloud, Azure).
  - **Self-Hosted Server** (e.g., using VPS or on-premise infrastructure).

Division Between Different Components

Client Components:

1. **Interceptor Module:**
   - Captures link clicks and extracts URLs.
   - Implementation Options:
     - **Accessibility Service:** Monitors system-wide clicks, requires user consent.
     - **Custom URL Handler:** Set the app as the default browser for opening links.
2. **Analysis Engine:**
   - Performs security checks on URLs.
   - Implementation Options:
     - **Local Database Check:** Searches URLs in an offline database (SQLite/Room).
     - **Remote API Check:** Queries a server for updated threat assessments.
3. **UI Layer:**

User:

- Displays warnings or alerts when malicious URLs are detected.
- Allows user actions like "Cancel", "Proceed", or "Report."
- Small indication that a link we scanned and approved.

Admin:

- Reviewing users requests.
- Updating database.

4. **Database Manager:**
   - Maintains a local database of known malicious links.
   - Handles syncing with the server for periodic updates.

Server Components:

1. **Database Server:**
    - Stores a global repository of malicious links.
    - Could include additional metadata like threat levels and categories.
    - Implementation Options:
        - **Cloud Databases:** Firebase, AWS RDS, or MongoDB Atlas.
        - **Self-Hosted Database:** MySQL, PostgreSQL.
2. **API Server:**
    - Provides a REST API to clients for checking URLs or updating their local databases.
    - Implementation Options:
        - **Cloud Services:** AWS Lambda, Google Cloud Functions, or Firebase Functions.
        - **Custom Backend:** Node.js, Python (Flask/Django), or Java (Spring Boot).

DB Structure (High-Level Overview)

The database structure consists of tables designed to efficiently store and manage information about malicious links and their associated metadata. Here’s a high-level breakdown:

- **Malicious Links Table:**
    - Stores URLs flagged as malicious.
    - Includes details like the type of threat (e.g., phishing, malware, or spam) and the last time the entry was updated.
- **Threat Levels Table (Optional):**
    - Contains predefined threat categories or levels (e.g., low, medium, high).
    - Helps provide context or severity ratings for flagged URLs.
- **Update Tracking (Optional):**
    - Tracks when the local or remote database was last updated.
    - Useful for ensuring the client app has the latest information.

This flexible structure supports both local databases for offline checks and server-side databases for global management and synchronization. The design can be scaled to include additional metadata like source reliability or user-reported threats in the future.

Languages, Libraries, and External Tools

Languages:

- **Client:** Java/Kotlin (for Android development).
- **Server:**
  - **Backend:** Python, Node.js, or Java.
  - **Database:** SQL-based (MySQL, PostgreSQL).

Libraries:

- **Retrofit/:** For network requests from the client to the server.
- **SQLite:** For managing the local database on the client.
- **Gson:** For parsing JSON responses from the server.

External Tools:

- **Cloud Platforms:** AWS, Google Cloud, or Azure for hosting server components.
- **Malicious Link Detection APIs:**
  - **PhishTank:** For querying known phishing URLs.
  - **VirusTotal:** For a comprehensive threat analysis.

Optional Future Tools:

- **Machine Learning Libraries:** Pytorch. TensorFlow Lite (for on-device ML) or cloud-based ML APIs.
- **Security SDKs:** Integration with antivirus APIs or third-party security services.

System Structure with Frameworks and Libraries

Client-Side (Android App)

1. **Interceptor Module and context extraction**
    - **Functionality**: Captures link clicks and extracts URLs and context for processing.
    - **Libraries/Technologies**:
        - **Accessibility Service**:
            - Part of the **Android Accessibility Framework**, allowing the app to monitor system-wide user interactions, such as link clicks in apps or browsers.
            - Used when you don't have direct control over the app generating the link (e.g., SMS app or web browsers).
        - **Custom URL Handler (Android Intent Framework)**:
            - A core feature in Android that intercepts intents to open links.
            - By setting your app as the default handler for URLs, the app ensures that all links clicked by the user are passed through the interceptor module before opening.

---

2. **Analysis Engine**
    - **Functionality**: Analyzes intercepted URLs to determine if they are malicious.
    - **Libraries/Technologies**:
        - **Local Database (cache)**:
            - **SQLite**: A lightweight local database for offline storage of malicious links.
        - **HTTP Client**:
            - **OkHttp**: A lower-level library (used by Retrofit internally) for making HTTP requests.
        - **JSON Parsing**:
            - **Gson**: Converts JSON responses from the server (e.g., malicious link metadata) into Java/Kotlin objects for easy handling in the app.

---

3. **UI Layer**
    - **Functionality**: Displays user-facing elements like warnings, prompts, and a history of intercepted links.
    - **Libraries/Technologies**:
        - **Android Jetpack Libraries**:
            - **LiveData**: For observing and updating UI components when data changes (e.g., new intercepted links).
            - **ViewModel**: For managing UI-related data in a lifecycle-aware way.
            - **Navigation Component**: Simplifies navigation within the app.
        - **Material Design Components**:
            - Used for consistent, modern UI designs, such as warning dialogs, notifications, and history lists.

---

4. **Database Manager (client side)**
    - **Functionality**: Syncs and updates the local malicious link database periodically.
    - **Libraries/Technologies**:
        - **WorkManager**:
            - Used for background syncing tasks, ensuring updates happen even if the app is killed or the device restarts.
            - **Why WorkManager?** It offers backward compatibility (API 14+) and guarantees task execution.
            - **Alternatives**: JobScheduler (API 21+), AlarmManager, or Executors for simpler scenarios.

Backend-Side

1. **Database**
    - **Functionality**: Maintains a global repository of malicious links, their threat levels, and associated metadata.
    - **Technologies**:
        - **Database**:
            - **MySql**: A robust relational database suitable for structured data and complex queries.

---

2. **API Server**
    - **Functionality**: Exposes REST endpoints for querying and updating the malicious link database.
    - **Technologies**:
        - **Framework**:
            - **Node.js with Express.js**: A lightweight, scalable backend framework for handling API routes.
        - **Authentication**:
            - **JWT (JSON Web Tokens)**: Verifies the app's identity with the server, ensuring only authorized requests are processed.
            - **Why Authentication?** It prevents abuse of the API and ensures secure communication between the client and server.
            - **Firebase as an Alternative**: Firebase Authentication could replace JWT, managing authentication with less effort, especially for apps integrated with Firebase services.
        - **Request Validation**:
            - **Joi**: Validates the structure and format of incoming data (e.g., ensuring that the URL sent by the client is valid and formatted properly).
            - **Why Validation?** It ensures the integrity of data before processing, preventing errors or misuse.

---

3. **Integration with External APIs**
    - **Functionality**: Augments the server's database with verified malicious links and metadata.
    - **Technologies**:
        - **PhishTank API**: Queries a trusted phishing URL database.
        - **VirusTotal API**: Analyzes URLs in real-time for potential threats.

Connections

1. **Client ↔ Server Communication**
    - **Protocol**: HTTPS (to encrypt data in transit and protect against eavesdropping).
    - **Data Format**: JSON (lightweight and easy for both client and server to handle).
    - **Libraries**:
        - **Retrofit**:
            - Makes HTTP requests from the client to the server (e.g., for malicious URL checks or database updates).
            - **Alternative**: Volley (less popular but still viable).

---

2. **Server ↔ External API Communication**
    - **Protocol**: HTTPS (secure communication with external services).
    - **Data Format**: JSON/XML (depending on the external API specifications).
    - **Libraries**:
        - **Axios**:
            - A promise-based HTTP client in Node.js, used for making outgoing requests to APIs like VirusTotal.
            - **Why Axios?** It simplifies HTTP requests and has robust error handling.
            - **If Using Express:** Axios complements Express by handling external API calls, while Express handles the server's API endpoints.

---

3. Server ↔ Database Communication

- **Protocol**: Direct database connection using raw SQL queries.
- **Data Format**: SQL queries and responses in tabular format.
- **Libraries/Technologies**:
    - **PostgreSQL** or **MySQL**: A relational database system for structured data storage.
    - **Node.js Database Driver**:
        - For PostgreSQL: **pg** (node-postgres), a robust client for connecting to and interacting with PostgreSQL.
        - For MySQL: **mysql2**, a performant and modern driver for MySQL.

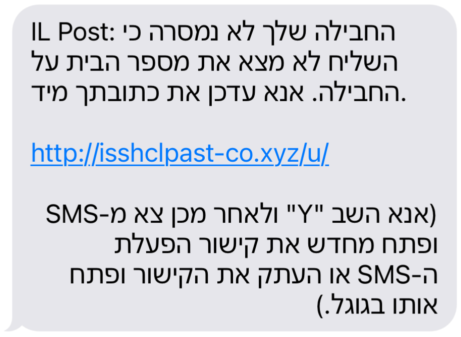
Summary of Elaborations:

- Explained **API Levels** and the reasoning behind choosing specific tools for backward compatibility (e.g., WorkManager vs. JobScheduler).
- Clarified the **role of Axios in a Node.js server**, distinguishing it from Express.
- Highlighted **authentication vs. validation** and their specific use cases in the app.
- Elaborated on **Retrofit** and how it simplifies HTTP requests on the Android client.

Examples of Analyzable Messages and Links:

1. Phishing Message Example

Message:



IL Post: החבילה שלך לא נמסרה כי השליח לא מצא את מספר הבית על החבילה. אנא עדכן את כתובתך מיד.

http://isshclpast-co.xyz/u/

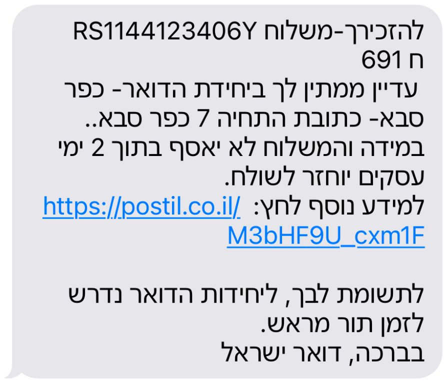
(אנא השב "Y" ולאחר מכן צא מ-SMS ופתח מחדש את קישור הפעלת ה-SMS או העתק את הקישור ופתח אותו בגוגל.)

Analysis:

- **URL:** `http://isshclpast-co.xyz/u/`
- **Context:** The message mentions a delivery problem and urges immediate action, which is a common phishing tactic to instill urgency.
- **Decision:**
  - User: "Warning! This link is flagged as phishing. It appears to be impersonating a delivery service. Proceed at your own risk."
  - Admin:
    - URL Analysis: "URL identified as suspicious. The domain `isshclpast-co.xyz` is not associated with any known delivery service."
    - Contextual Indicators: "Message includes urgency ('update immediately'), which is a strong phishing indicator."
    - Action: "Added to blacklist for phishing."

2. Legitimate Message Example (Potentially Misclassified)

Message:



להזכירך-משלוח RS1144123406Y
691 ח
עדיין ממתין לך ביחידת הדואר- כפר סבא- כתובת התחיה 7 כפר סבא..
במידה והמשלוח לא יאסף בתוך 2 ימי עסקים יוחזר לשולח.
למידע נוסף לחץ: https://postil.co.il/M3bHF9U_cxm1F

לתשומת לבך, ליחידות הדואר נדרש לזמן תור מראש.
בברכה, דואר ישראל

Analysis:

- **URL:** `https://postil.co.il/su8XJGl_cxm1F`
- **Context:** The message appears to come from Israel Post and includes specific information about a package and branch address.
- **Decision:**
  - **User:** "This link appears legitimate and matches the context of a delivery notification. Opening link..."
  - **Admin:**
    - **URL Analysis:** "Domain `postil.co.il` is associated with Israel Post services."
    - **Contextual Indicators:** "Includes specific details (package number, branch address) that validate its authenticity."
    - **Action:** "Marked as safe."

Clarification of Decision Factors:

Decision Based on URL:

1. If the domain is not recognized or is flagged in the database, it is immediately marked as suspicious.
   - Example: `http://isshclpast-co.xyz` → Unrecognized domain.
2. If the domain matches known safe domains, it is marked as safe unless other indicators suggest otherwise.
   - Example: `https://postil.co.il` → Recognized domain.

Decision Based on Context:

1. If the message creates urgency or asks for sensitive information, it raises the threat level.
   - Example: "Update your address immediately" → Common phishing tactic.
2. If the message provides verifiable details (e.g., package ID, branch address), it lowers the threat level.
   - Example: "RS1144123406Y still waiting at branch כפר סבא התחיה 7."

Sandbox Analysis (Optional):

1. For unrecognized or suspicious links, a sandbox environment analyzes:
   - Redirect chains.
   - Page content for phishing forms or malicious scripts.
   - Example: If `http://isshclpast-co.xyz` redirects to a fake Israel Post login page, it is flagged as phishing.

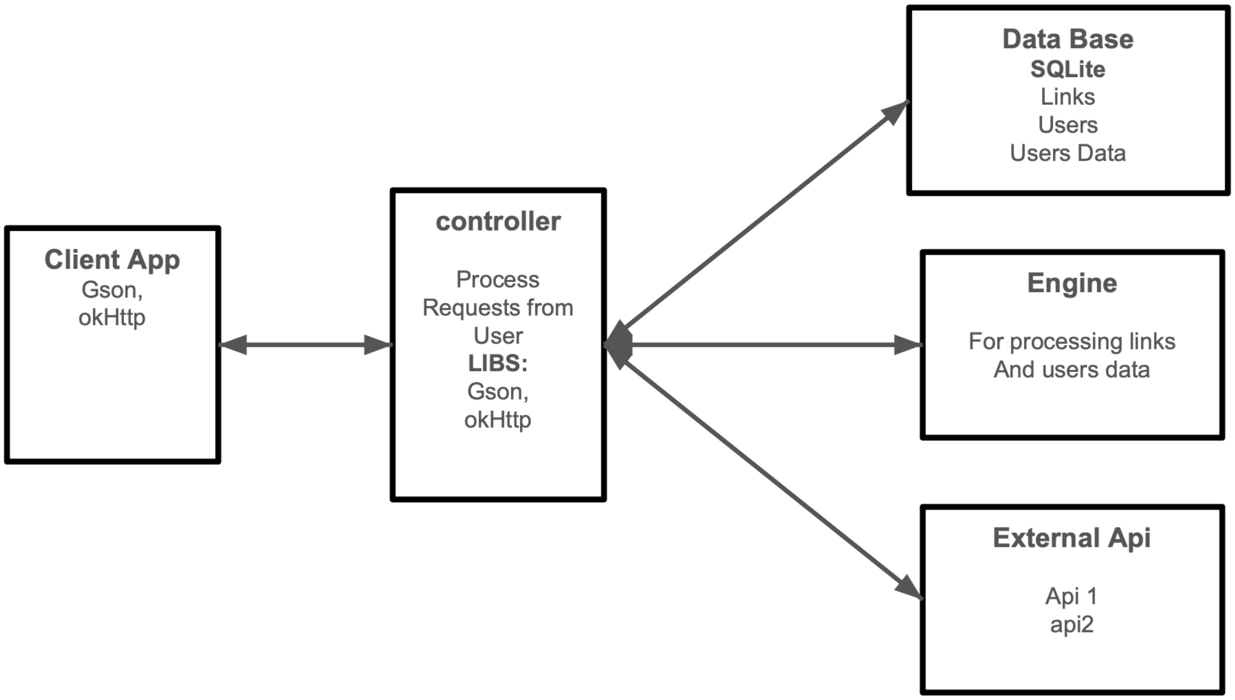
Outputs for User and Admin:

Phishing Message Example

- **User:**
  - "This link is flagged as phishing and may steal your information. Proceed with caution."
  - Options:
    - **Block Link**
    - **Proceed Anyway** (with confirmation).
- **Admin:**
  - **Link:** `http://isshclpast-co.xyz/u/`
  - **Context:** "Delivery notification claiming 'update address immediately.'"
  - **Threat Level:** High.
  - **Action:** "Added to blacklist for phishing."

Legitimate Message Example

- **User:**
  - "This link appears safe and matches the context of a delivery notification. Opening now..."
- **Admin:**
  - **Link:** `https://postil.co.il/su8XJGl_cxm1F`
  - **Context:** "Package delivery notification with branch details."
  - **Threat Level:** Low.
  - **Action:** "Marked as safe."


Client App
Gson,
okHttp
controller
Process
Requests from
User
LIBS:
Gson,
okHttp
Data Base
SQLite
Links
Users
Users Data
Engine
For processing links
And users data
External Api
Api 1
api2