Projectdocumentation

Höhere technische Bundeslehr- und Versuchsanstalt Villach

Informationstechnologie – Cyber Security

SEW - PhishGuard

Created by

Leon Walder
Fabian Huber

am 15.06.2023

Auftraggeber
Prof. Wolf Harald

Inhalt

1. Introduction

1.1. Description

PhishGuard is an Android Kotlin app designed to efficiently manage screenshots of login pages obtained from our backend. These screenshots are scraped from various websites and used for phishing landing page templates, primarily for testing purposes. The app allows users to view, organize, and comment on these screenshots, which are stored in a database.

1.2. Features

- **Screenshot Management**: PhishManager provides a user-friendly interface to view and manage the collection of screenshots obtained from the backend. Users can easily browse through the screenshots and access their details.

- **Database Integration**: The app integrates with a robust database system, ensuring secure storage and efficient retrieval of screenshots. The database management role is handled by Leon Walder.

- **Efficient User Interface**: PhishManager offers a seamless and intuitive user interface, designed with a focus on usability and performance. Users can navigate through the app effortlessly and quickly find the information they need.

1.3. Team

PhishManager is developed by a dedicated team with expertise in their respective roles:

Leon Walder - Database & App GUI: Leon is responsible for creating the user interface (UI) and the frontend of the PhishGuard app. He also focuses on developing the database.

Fabian Huber - Backend & App GUI: Fabian is responsible for connecting the backend system to the PhishGuard app. He is also working for the user interface (UI).

1.4. Motivation

PhishGuard was developed to efficiently provide landing page options from phishing emails. By streamlining the process, our app empowers users to proactively defend against phishing attacks. With PhishGuard, individuals and organizations can quickly and effectively gather the necessary resources to enhance their security measures and protect sensitive information. Choose PhishGuard for efficient phishing landing page management and stay ahead in the fight against cyber threats.

2. Technical Implementation

2.1 Backend & API Call (Fabian Huber)

The makeAPICall function takes three parameters:

url (string): The URL to which the API request should be sent.
inputValue (String): The value to send to the server as part of the request.
onSuccess (Function): A function that will be called when the API request completes successfully. This function expects a bitmap object as a parameter and has no return value.

MakeAPIcall.kt:

```
fun makeAPICall(url: String, inputValue: String, onSuccess: (Bitmap) -> Unit) {
    println("myinfo: apicall starting")
    val client = OkHttpClient.Builder()
        .connectTimeout(30, TimeUnit.SECONDS)
        .readTimeout(120, TimeUnit.SECONDS)// Increase the timeout to 30 seconds
        .build()
```

The function creates an instance of OkHttpClient from the OkHttp library to execute the HTTP request. Some configuration options for the client are set, such as the timeout values for connecting and reading the response.

Displaying the Bitmap/Image:

```
makeAPICall(
    "http://192.168.0.6:5000/newScreenshot",
    searchQuery
) { bitmap ->
    isLoading.value = false
    imageBitmapState.value = bitmap.asImageBitmap()
}
```

The resulting bitmap is assigned to the imageBitmapState.value, possibly after being converted to an ImageBitmap object.

```
Box(
    modifier = Modifier
        .fillMaxSize()
        .wrapContentSize(Alignment.Center)
) {
    if (isLoading.value) {
        Text("Loading...")
    } else {
        imageBitmapState.value?.let { imageBitmap ->
            Image(
                bitmap = imageBitmap,
                contentDescription = "Image",
                modifier = Modifier.fillMaxSize(),
                contentScale = ContentScale.Fit
            )
        }
    }
}
Spacer(modifier = Modifier.height(64.dp))
```

Inside the Box, there is a conditional rendering based on the value of isLoading.value. If it is true, the text "Loading..." is displayed. Otherwise, the imageBitmapState.value is checked for a non-null value.

2.2 User Interface & Navigation (Leon Walder)

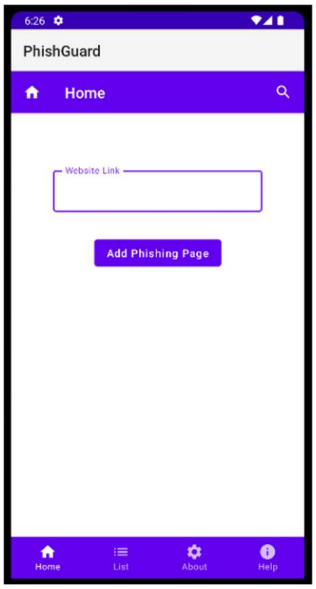
The navigation system in the PhishGuard Android app allows users to easily navigate between four screens: Home, List, About, and Help. The system utilizes NavRoutes, NavBarItems, BottomBar, and Screens to provide a smooth and intuitive navigation experience. Users can switch between screens using the bottom navigation bar, which displays icons and labels for each screen. This simple and efficient navigation system ensures seamless exploration and interaction within the app.
NavRoutes:

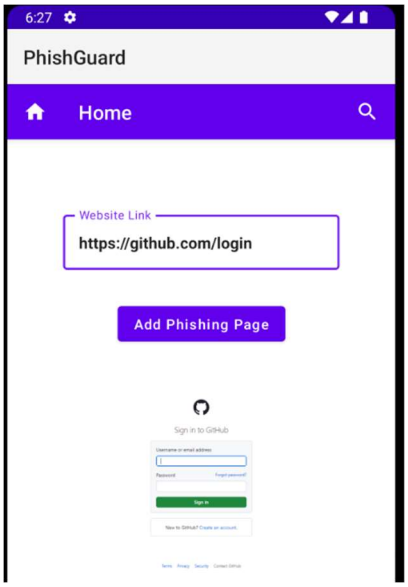
```
package com.example.phishguard.navigation

sealed class NavRoutes(val route: String) {
    object Home : NavRoutes("home")
    object List : NavRoutes("list")
    object About : NavRoutes("about")
    object Help : NavRoutes("help")
}
```

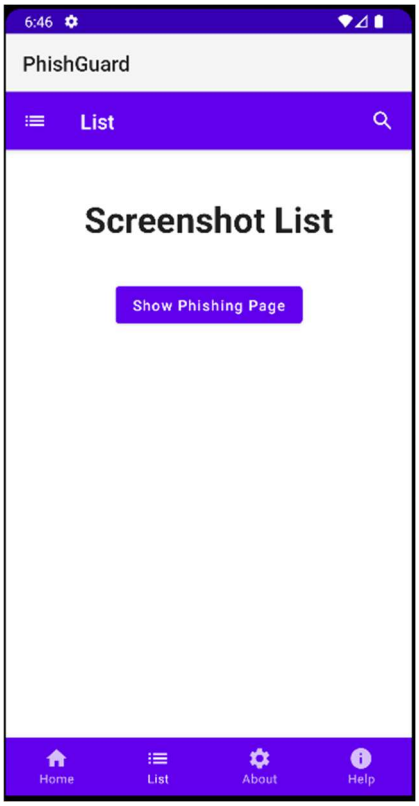
The navigation consists of 4 Screens.

1. **Home**





2. List



3. About

2.3 Database (Leon Walder)

Does not work!

The PhishGuard Android app incorporates the Room database framework for managing Screenshot objects. The implementation includes key components such as the RoomDatabase, ScreenshotDao, ScreenshotRepository, and MainViewModel. These components work together to facilitate the storage, retrieval, and manipulation of Screenshot data within the app. With RoomDatabase serving as the entry point to the database, ScreenshotDao provides the necessary methods for data access, while ScreenshotRepository acts as a bridge between the ViewModel and data operations. MainViewModel manages the UI state and interacts with the repository to retrieve and manipulate Screenshot objects. This Room database implementation ensures efficient and organized data management in the PhishGuard app.

At the end, the elements in the Database should be displayed on the List Screen when clicking the Button.

```
10    @Dao
11    interface ScreenshotDao {
12        @Insert
13        suspend fun insertScreenshot(screenshot: Screenshot)
14
15        @Query("SELECT * FROM screenshots")
16        fun getAllScreenshots(): LiveData<List<Screenshot>>
17    }
```