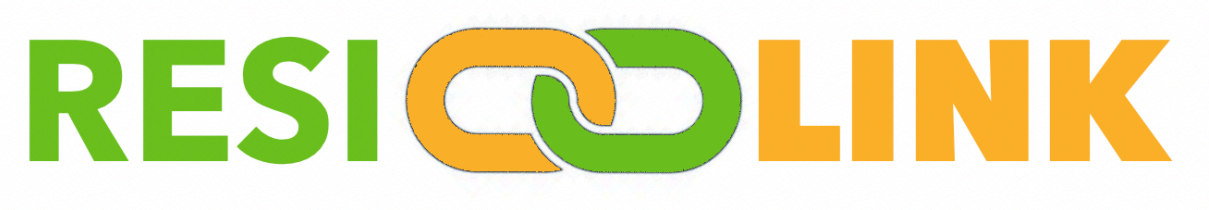


Increasing Resilience of Smallholders with Multi-Platforms Linking Localized Resource Sharing

Deliverable D2.4b

Final report on test and validation of the full RESILINK framework

Responsible Editor:	ACICT & UPPA
Contributors:	ALL
Document Reference:	RESILINK D2.4b
Distribution:	Public
Version:	1.1
Date:	Apr. 2026

CONTRIBUTORS TABLE

DOCUMENT SECTION	AUTHOR(S)
SECTION 1	C. Pham (UPPA)
SECTION 2	A. Cazaux (UPPA)
SECTION 3	A. Cazaux (UPPA)
SECTION 4	A. Cazaux (UPPA)
SECTION 5	A. Cazaux (UPPA)
SECTION 6	A. Cazaux (UPPA) & M. Despland (Orange)

DOCUMENT REVISION HISTORY

Version	Date	Changes
V1.1	April 21st, 2026	PUBLIC RELEASE
V1.0	April 17th, 2026	FIRST DRAFT VERSION FOR INTERNAL APPROVAL
V0.1	April 10th, 2026	FIRST RELEASE FOR REVIEW

Executive Summary

Deliverable D2.4b presents the comprehensive testing and validation of the RESILINK framework, covering both the RESILINK mobile application and RESILINK back-end server components. It especially details the procedures followed to ensure compliance with Google Play Store requirements, including internal testing, closed testing phases, and the final production release process.

In addition, this deliverable documents the setup and build process of the RESILINK mobile application and the testing of the open API and interoperability features. Particular attention is given to system behavior, data handling, and real-world use cases, ensuring the robustness and scalability of the RESILINK platform.

Table of Contents

1. INTRODUCTION

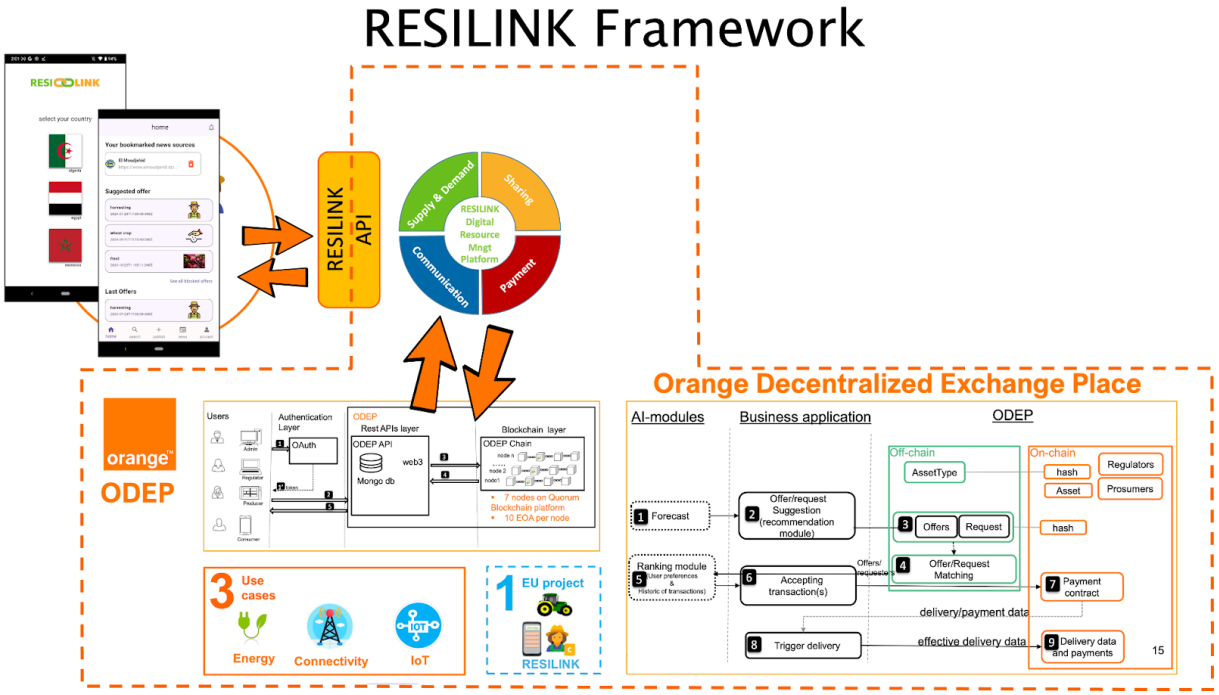
The lowest layer of the RESILINK digital resource management consists of the **Orange Decentralized Exchange Place (ODEP)** platform initially developed by Orange for energy and telecommunication ecosystems. RESILINK will actually completely hide the ODEP platform to both users and developers by offering a higher level of abstraction and functionalities to easily build resource sharing applications.

In addition to ODEP, and to maximize RESILINK's usage by the end users, RESILINK will develop a mobile application, referred to as **RESILINK mobile app**, that will be the main interface to simply, quickly and intuitively manage resources. An intermediate platform, referred to as the **RESILINK platform**, will serve as the back-end server for the RESILINK mobile application.

Due to ODEP's development constraints and internal architecture, it is difficult to add new functionalities in the time frame of the Living-Labs. Therefore, the RESILINK platform can run is 2 versions: (1) with ODEP at the lowest layer to implement smart contracts based on Blockchain technologies and (2) without ODEP and without smart contracts, but with many additional features that are useful for end-users that participate in the Living-Labs program. This 2 branches approach has been implemented since RESILINK platform – v2.

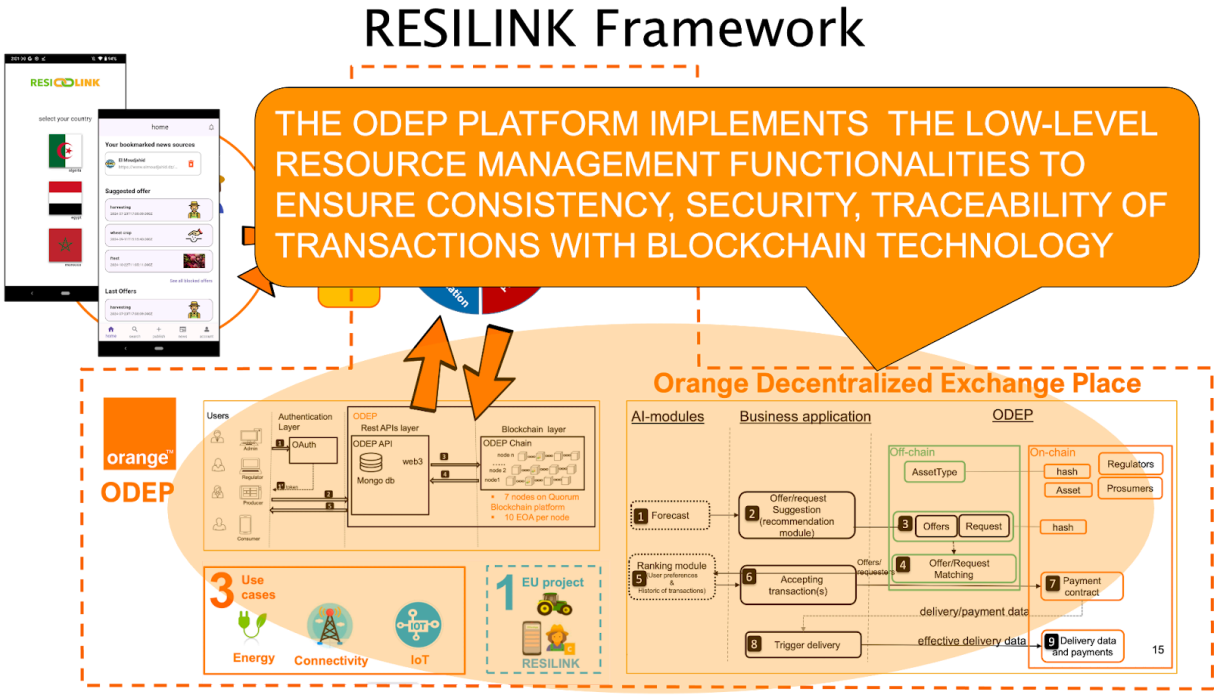
The overall architecture is illustrated in the figures below.

Big picture

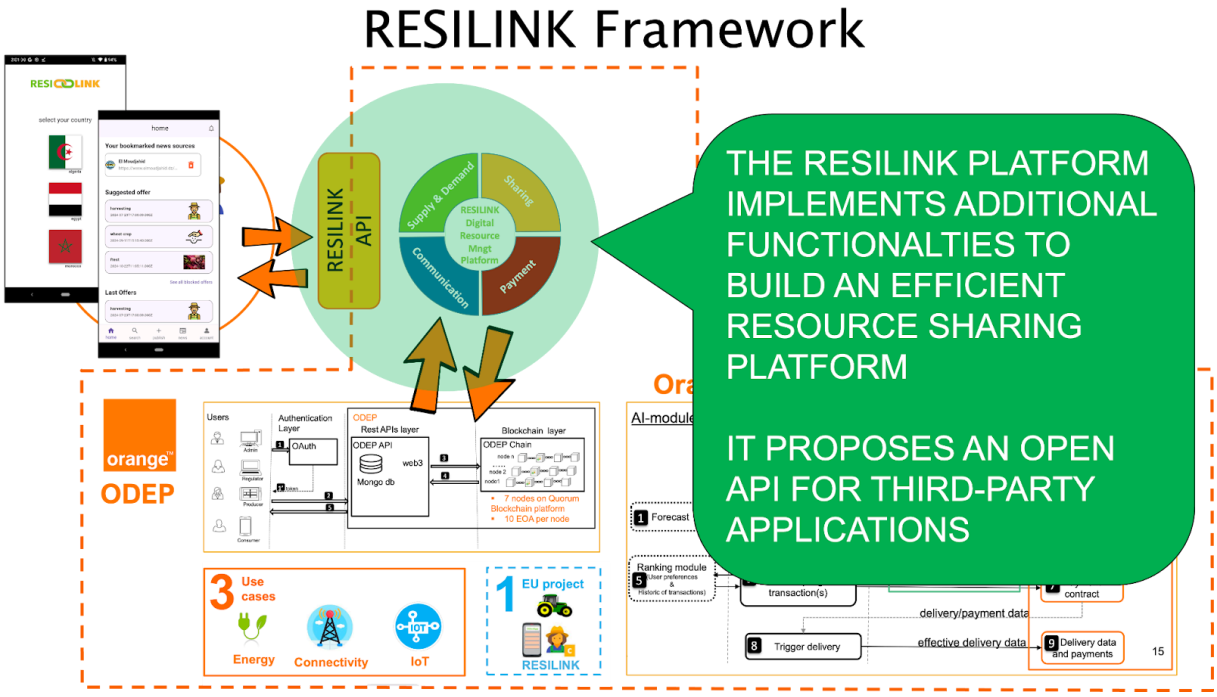


It must be noted that since the RESILINK Mobile Application is the main interface to the RESILINK Digital Platform, the evaluation is mainly conducted on the mobile application and the offered functionalities. However, taking into **account feedback requires updating both the RESILINK Mobile Application and the RESILINK Digital Platform as the offered functionalities are in reality provided by the underlying digital platform**.

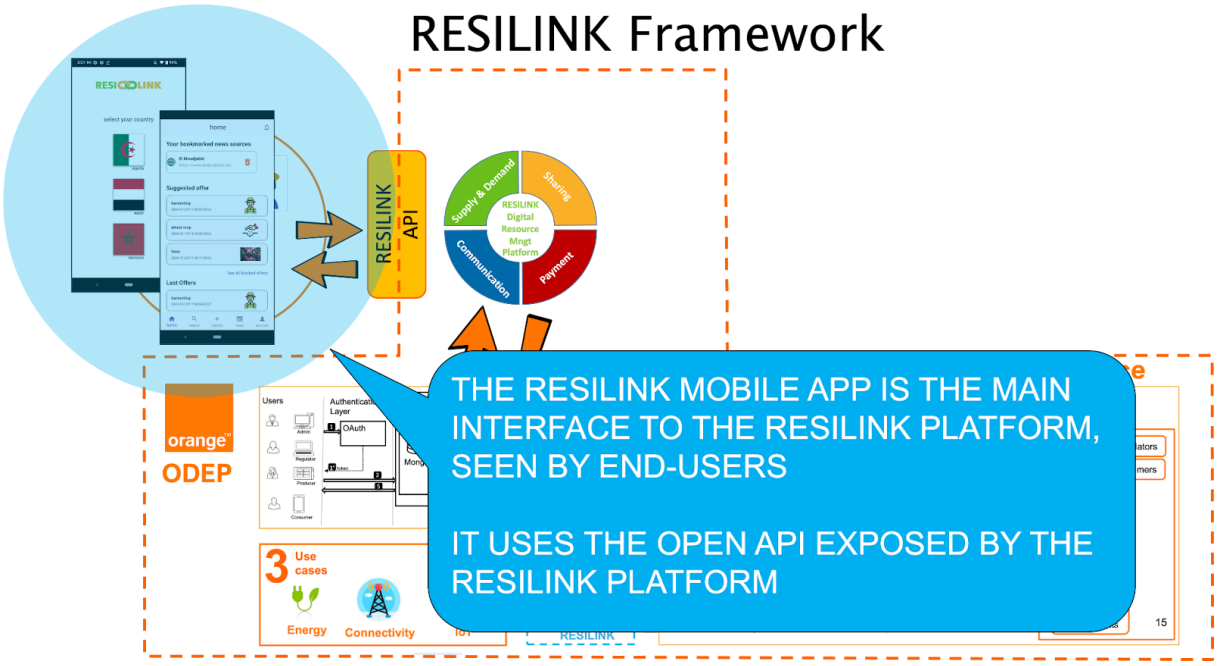
ODEP layer



RESILINK platform (the RESILINK back-end server)



RESILINK front-end mobile application



2. Internal tests for Google Play store

2.1. Review of Google Play Store procedure

The publication of the RESILINK mobile application on the Google Play Store follows the official distribution framework established by Google. The overall process can be divided into three main stages :

- Application setup in the Google Play Console
- Compliance and release preparation
- Testing and deployment

2.1.1. Application setup in the Google Play Console

Publishing an application begins with the creation of a Google Play Developer account, which grants access to the Google Play Console for application management. In most typical cases, this account can be registered as an individual; however, for applications that facilitate or enable exchange of services or monetary transactions (for instance, via redirection to external payment platforms such as PayPal), Google Play policies require that the developer account be registered as an organization rather than an individual entity.

In the context of RESILINK, this requirement introduced additional complexity. Registering an organization account requires providing verifiable legal identification, such as a company registration number (e.g., SIRET in France). Obtaining or using such identifiers was not immediately straightforward within the project's deployment scope. Consequently, an intermediate approach was adopted, using a developer account configuration with a mobile application without the possibility of monetary exchange within the application. In this case,

the application itself became a “platform for viewing services between individuals,” ensuring that policies were in place for subsequent larger scale deployment scenarios.

Once the account setup is complete, the application can be created within the Play Console. This includes defining its core store metadata:

- Application name and default language,
- Unique package identifier (application ID),
- Application category and type,
- Short and long descriptive texts.

To establish its visual presence, graphical resources such as the application icon, feature graphic, and screenshots for different device formats must be supplied. These elements shape user perception and are mandatory before submission.

The application is then prepared for distribution as an Android App Bundle (AAB), signed with a secure release key and uploaded to the Play Console, where Google optimizes the package for device-specific configurations.

2.1.2. Data Handling, permissions and policy requirements

Google Play applies strict requirements intended to ensure user safety, responsible data processing, and compliance with international regulations (including GDPR for European users). As part of the publication process, several mandatory declarations must be completed within the Google Play Console:

- **Data Safety Form**. The developer must explicitly state:
    - Which categories of data are collected (e.g., personal identifiers, contact info, location data),
    - How these data are processed (stored locally, transmitted to servers, encrypted in transit),
    - Whether the data is shared with third parties,
    - Whether the user can request deletion.
      The accuracy of this declaration is essential, as Google may suspend applications that misrepresent data usage.

- **Privacy Policy**. A publicly accessible Privacy Policy must be provided to:
    - Clearly describe the data collected and the purpose of collection,
    - Indicate how data are stored and protected,
    - Explain users’ rights regarding data deletion or consent withdrawal.
      This document must be hosted online (e.g., on the RESILINK project webpage or the webserver web page) and remain available for the lifetime of the application.

- **Target Audience and Content Rating**. The application must indicate its age group. For RESILINK, the target audience corresponds to **general users (13+)**. A

questionnaire is then used to determine the **official content rating** displayed in the Play Store.

- **Sensitive Permission Declarations**. If the application requests access to sensitive device functionality — such as notification permissions, SMS handling, contact lists, camera, microphone, or device identifiers — each of these must be justified. The developer must demonstrate:
    - The permission is required for a core feature,
    - Users are informed when the permission is requested,
    - The application does not misuse or collect unnecessary data.

Only once **all declarations are consistent and complete** can the application proceed to the closed testing or review phase. Any inconsistency (e.g., requesting a permission without declaring its use) may result in:

- Application rejection,
- A request for developer justification,
- Or restrictions on release eligibility until corrected.

2.1.3. Testing, Google review and production release

Before making the application publicly available, Google requires controlled testing phases:

- Internal testing for rapid iterative evaluation,
- Closed testing with selected testers — used in the case of the RESILINK Living-Lab deployments,
- Optional open testing for broader early access..

In the closed testing track, Google may require a minimum number of active testers over a continuous period to ensure the application demonstrates stable real-world usage before being eligible for full production release.

After testing feedback is integrated, the application is submitted for Google's review, which includes automated checks and manual assessment of:

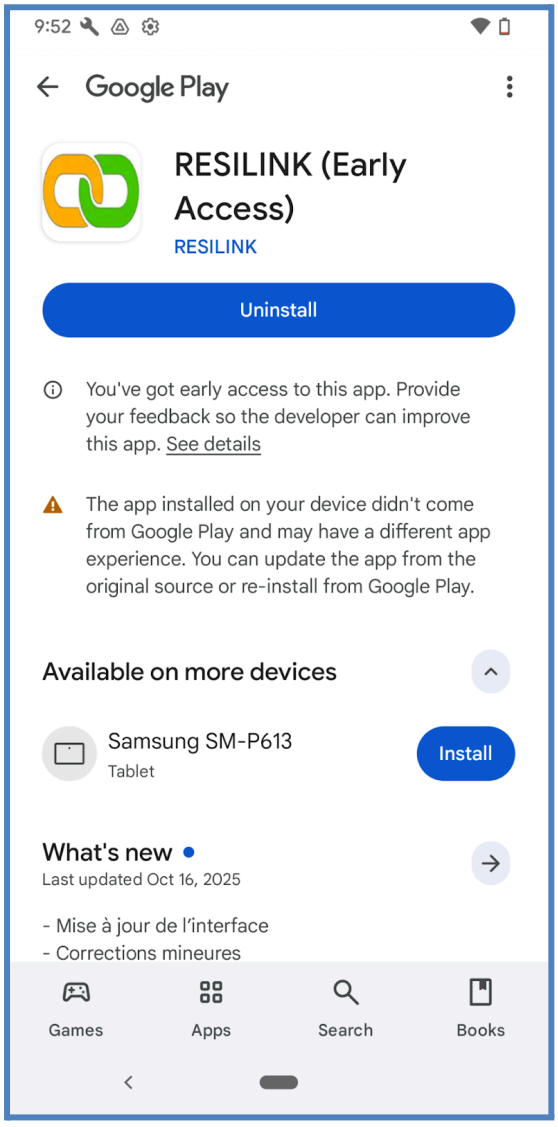
- Technical quality and stability,
- Policy and security compliance,
- Appropriateness of user experience and content.

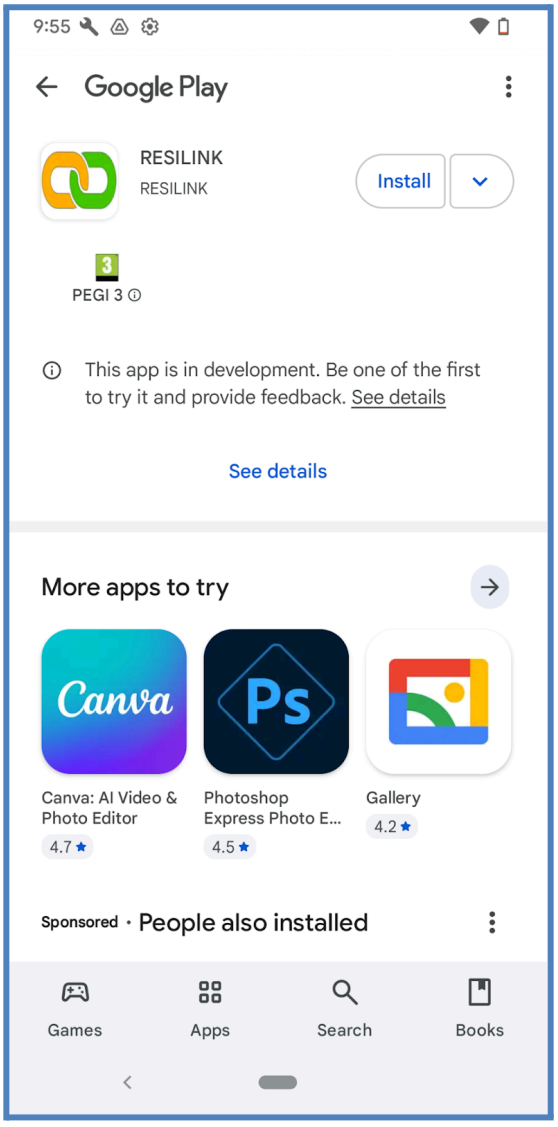
If approved, the application is promoted to Production, becoming available to the selected geographic distribution regions. Post-release monitoring and maintenance are then conducted through analytics tools integrated within the Play Console.

2.2. Running the closed tests for Google Play Store

2.2.1. Review & Feedback

The closed tests have been realized with about 18 testers, from RESILINK partners and from an external test service. Each tester is registered by RESILINK with a valid gmail email address to be allowed to get access and install the temporary RESILINK application from Google Play.





Google Play link: https://play.google.com/store/apps/details?id=com.uppa.resilinkMobileApplication

QR code to flash from an Android smartphone →

Each tester is asked to use the RESILINK mobile application daily and to provide feedback on Google Play. They report bugs or any difficulty directly to the UPPA developer team. They are also asked to check regularly (every 3 days for instance) whether a new version is available and, if so, update their mobile application.

From the UPPA developer team side, we correct the reported bugs and update every 3-4 days the application on Google Play to show that we are responsive.

2.2.2. Google Analytics for the closed test

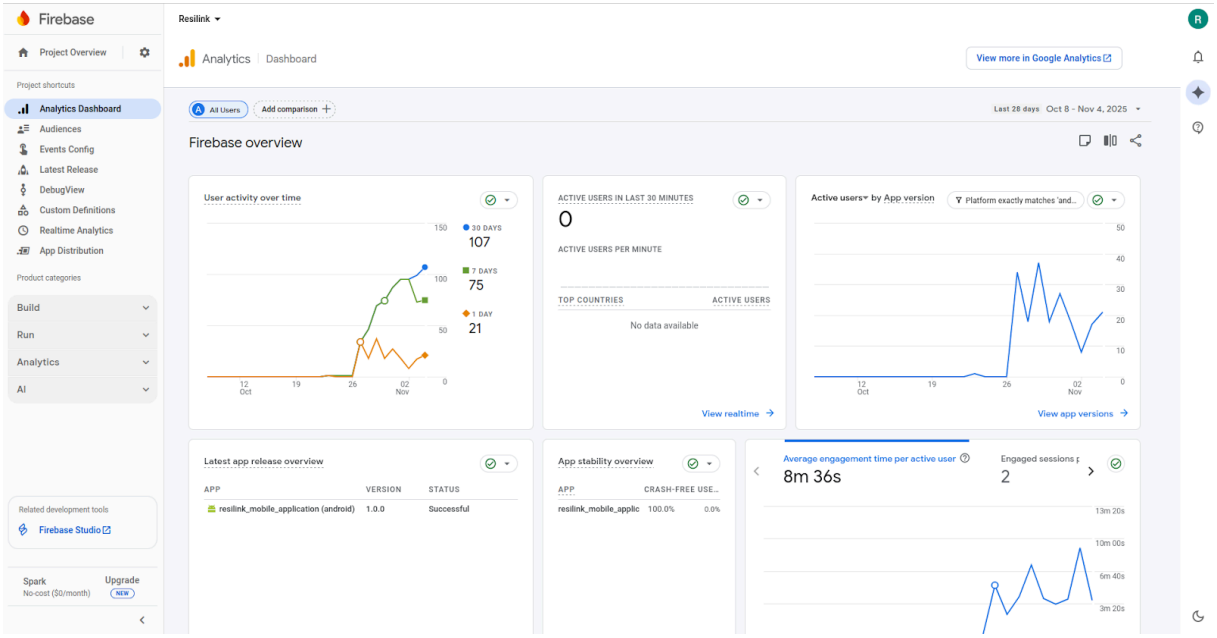
To monitor user engagement and application behavior during the closed test phase, the RESILINK mobile application is instrumented with Firebase Analytics. This integration is particularly relevant, as Firebase is also the system used internally by Google Play to verify the conditions required to progress toward public release. In particular, Google Play evaluates whether the application meets the threshold of sustained active testers (e.g., at least 12 active users over a period of approximately 14 consecutive days). By using Firebase, the development team can therefore observe the same activity metrics that Google itself verifies during the review process.

Firebase Analytics offers a non-intrusive and privacy-respectful method for monitoring application usage. It does not collect personal identity data but instead provides aggregate insights such as:

- User activity frequency (daily, weekly, monthly active users)
- Application version usage, which helps confirm that testers are consistently updating the app
- Device and platform characteristics (e.g., phone models, Android versions), useful for detecting device-specific issues
- Geographic distribution, allowing confirmation that testing aligns with deployment regions
- User interaction events, such as screen transitions and feature usage (e.g., posting an offer, refreshing the home screen, viewing onboarding help screens)

This level of observability ensures the platform behaves as expected during real usage conditions and supports stability evaluation during iterations.

As an example, below is a screenshot of the dashboard of data analyzed by Firebase.



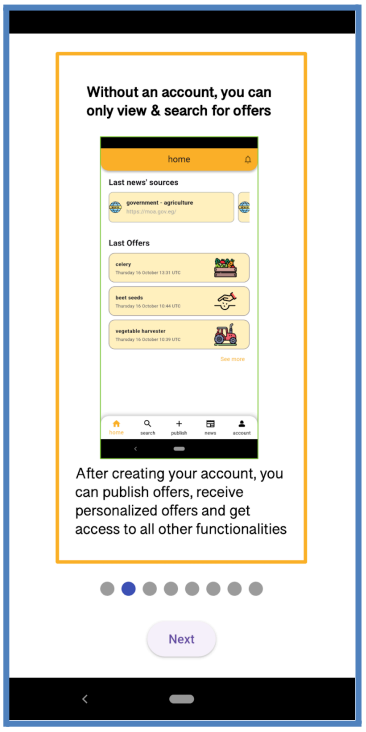
The figure above illustrates the Firebase Analytics dashboard for the RESILINK closed testing phase. Several key indicators can be observed:

- **User Activity Over Time** (top-left graph):
  Shows the evolution of active users across 1-day, 7-day, and 30-day ranges.
  For the current test cycle, usage shows a consistent upward trend, confirming regular engagement.
- **Active Users by App** Version (top-right graph):
  Indicates that testers are using the target release version (v1.0.0), and that updates have been adopted without fragmentation.
- **App Stability Overview** (bottom center):
  Reports 100% crash-free usage during the observation period, demonstrating the robustness of the application on tested devices.
- **Average Engagement Time Per Active User** (bottom right):
  Indicates that users spend several minutes per session in the application, suggesting meaningful interaction rather than disposable usage.

These analytics confirm both technical stability and successful user engagement, fulfilling the conditions required to validate the closed testing phase before progressing to broader deployment.

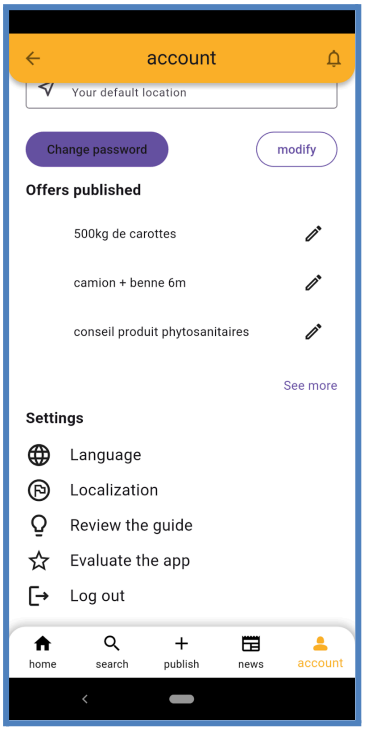
2.2.3. Correction and update of the application

- Addition of 8 help screens when the application is launched for the first time. See the 8 screens design here: https://resilink.eu/resilink-mobile-app-helper-screens

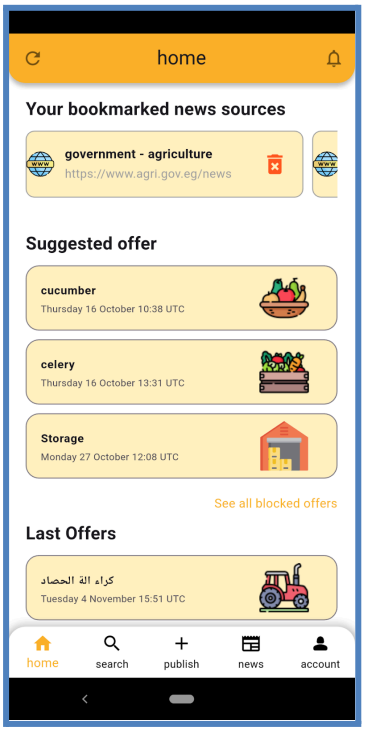


...

- The language for the application is determined automatically from the smartphone system's setting, this allows for help screens to be in the correct language
- Add a "Review the guide" icon in user's account to visualize the help screens



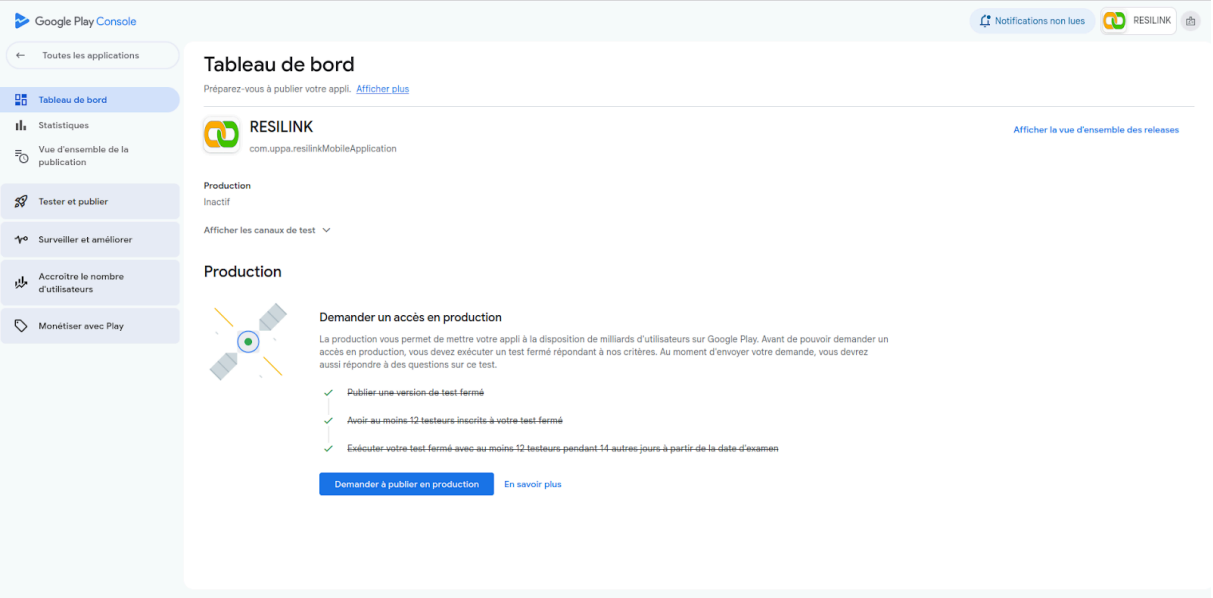
- Add a refresh control on the Home screen (top-left). With the refresh icon, new published offers can appear in the “Last Offers” section. The somehow standardized refresh icon is more intuitive than touching the “Home” icon in the bottom menu bar.



- Add a non-intrusive, well-integrated link to the RESILINK website
- Embed Rules & Privacy Policy pages inside the app
- Fix offer creation bug for the “Machinery” category (enumeration list)
- Fix initial-install language initialization bug (Language page)
- Fix offer description text-direction in offer’s description (RTL when bugging)
- Fix Arabic localization bug for activity domains and specializations in account creation
- Addition of a blocking pop-up in case of an older version than the one in the Play Store. On the server side, the body of the data can change quite often, which can completely break the mobile application if it is not in its latest version.

3. Production access on Google Play

3.1. Production Readiness and Validation Process



After the closed testing phase, the application becomes eligible for production release once Google Play verifies that it has been used by a sufficient number of real testers and shows stable behavior. In our case, these criteria were validated through Firebase Analytics, which confirmed continued usage over several days and the absence of critical crashes.

The production request was submitted twice, in September and October 2025. The first submission was rejected, as the app did not show enough real user activity. During that attempt, several testers were emulated using virtual devices, which Google Play does not consider as valid usage, and therefore the engagement was judged insufficient. Following this feedback, a second testing session was organized with real devices and active participants, which provide the necessary engagement metrics to meet Google Play's release requirement.

Once these conditions were met, the build could be submitted again for final review. This review ensures compliance with platform requirements such as privacy, data handling, and content policies. After approval, the application can be made publicly available, either through a progressive rollout or a full release in the selected regions.

3.2. Successful Approval

The production release request was ultimately approved by Google Play Store. To meet the eligibility criteria and ensure sufficient real-world engagement, we organized a coordinated testing effort over a two-week period. This activity combined three sources: internal collaborators who used the application daily, a group of paid testers recruited through Fiverr to guarantee the required testing volume, and a community of volunteer testers obtained through a Play Store developer support platform where developers exchange testing

services. These combined contributions provided consistent usage metrics across all key features, as well as stable crash-free sessions, which collectively satisfied Google Play's validation requirements.

Following this successful verification, the application was deployed and is now publicly available on the Play Store under the "Early Access" label and can be accessed using this QRCode.

Although the build has fully passed Google Play's compliance checks—the same review criteria applied to a production release—we elected to keep the app in open testing at this stage. This decision is due to the application's evolving nature: major backend changes are ongoing, including a transition from isolated server responses to a new multi-server communication model. This shift significantly alters the structure of the server's response payloads, which means that older app versions are expected to crash systematically if they receive the new response format.

Furthermore, the in-app upgrade library, which notifies users of the availability of a newer Play Store version, has only recently been integrated. As a result, users running outdated builds would not be alerted to update before encountering these breaking changes. Maintaining the application in "Early Access" therefore ensures a safer environment while these architectural updates are rolled out and stabilization continues. Once these elements are fully aligned and backward-compatibility risks are removed, the application can be transitioned to full production at any time.

4. Mobile application - setup and build

The RESILINK Mobile Application is built with Flutter and targets Android (primary) and Linux and it is available in this Git repository https://github.com/ZiQuwi/RESILINK-Mobile-App. This section covers all code-level configurations required before building the app, followed by instructions for generating a signed APK or Android App Bundle (AAB).

4.1. Prerequisites

Requirement	Details
Flutter SDK	Version ^3.8.1
Android SDK	Required for Android builds (included with Android Studio)
JDK	Java Development Kit (required by Gradle)
Firebase project	A configured Firebase project (see section 2.2)
RESILINK API server	A running instance of the backend (see section 1)

4.2. Firebase Configuration

The repository does not include Firebase configuration files as they contain sensitive API keys. You must provide the following files before the app will compile:

File	How to obtain
lib/firebase_options.dart	Run flutterfire configure in the project root, or copy from your Firebase console
android/app/google-services.json	Download from your Firebase project settings under the Android app section

⚠ **WARNING** Without both of these files the app will not compile. Never commit them to a public repository.

4.3. Connecting the App to the RESILINK Server

The mobile app communicates with the RESILINK REST API. The server URL must be configured at the code level before building. Open the following file:

```
lib/providers/main_provider.dart
```

Locate lines 187 and 188 and update the username and password of your “public” account from your RESILINK server. Then open the following file:

```
lib/constants/global_variables.dart
```

Then change the ipAddressMasterServer to the IP address or domain name of the main server in the server network and the ipDomain/protocol with your own. These three constants define the API endpoints. Failing to update them means the app will attempt to reach the wrong server and all API calls will fail.

4.4. Install Dependencies

From the project root, fetch all Flutter packages:

```
flutter pub get
```

4.5. Generate Localizations

The app supports English and Arabic. Regenerate the localization files whenever the ARB translation files are modified:

```
flutter gen-l10n
```

4.6. Keystore - Signing the application

A keystore is required to sign any release build of the app (both APK and AAB). The keystore is a cryptographic file that identifies the publisher of the application. It must be kept secret and backed up safely. Losing it makes it impossible to publish updates to the same app on the Play Store.

1. Generate a keystore

Run the following command from the project's root directory, replacing the alias and file path as appropriate:

```
keytool -genkey -v -keystore android/RESILINK.jks -alias upload -keyalg RSA
-keysize 2048 -validity 10000
```

You will be prompted to enter a keystore password and provide identity information (name, organisation, country). Remember both the keystore password and the key password.

> ⚠ **WARNING** Back up your .jks file and passwords in a secure location. If lost, you will not be able to push updates to the same Play Store listing.

2. Create the key.properties File

Create a file named key.properties inside the android/ directory of the project:

```
# android/key.properties
storePassword=<your_store_password>
keyPassword=<your_key_password>
keyAlias=upload
storeFile=./RESILINK.jks
```

The build.gradle file in android/app/ is already configured to use a key with the alias “upload” so no changes are necessary and all .keystore, .jks and key.properties files are ignored by github from .gitignore in the android folder.

4.7. Build a signed APK

To generate a standalone APK file (suitable for direct device installation or side-loading):

```
flutter build apk --release
```

The output file is located at:

```
build/app/outputs/flutter-apk/app-release.apk
```

NOTE An APK can be installed directly on a device without going through the Play Store. It is useful for internal distribution or manual testing.

4.8. Build a signed Android App Bundle (AAB)

To generate an AAB file (required for publishing on the Google Play Store):

```
flutter build appbundle --release
```

The output file is located at:

```
build/app/outputs/bundle/release/app-release.aab
```

NOTE The AAB format allows Google Play to optimize the app delivery for each device. It is smaller than an APK and is mandatory for new apps published on the Play Store.

4.9. Google Play Store - public release

Before public release, Google requires apps to complete an internal testing phase. For RESILINK, this phase involves 12 active testers over a period of two weeks.

The 12 testers install the app and actively use it for two weeks. During this period:

- Collect feedback via a shared form or communication channel
- Monitor crashes and ANRs in Play Console → Android vitals
- Upload new AAB releases as needed to fix reported issues (increment the versionCode in pubspec.yaml before each new build)
- Verify that all testers have opted in and confirmed the app works on their devices

> ℹ **NOTE** Google Play requires at least 12 testers to have had the app installed for at least 14 continuous days before allowing promotion to production (open testing or production track).

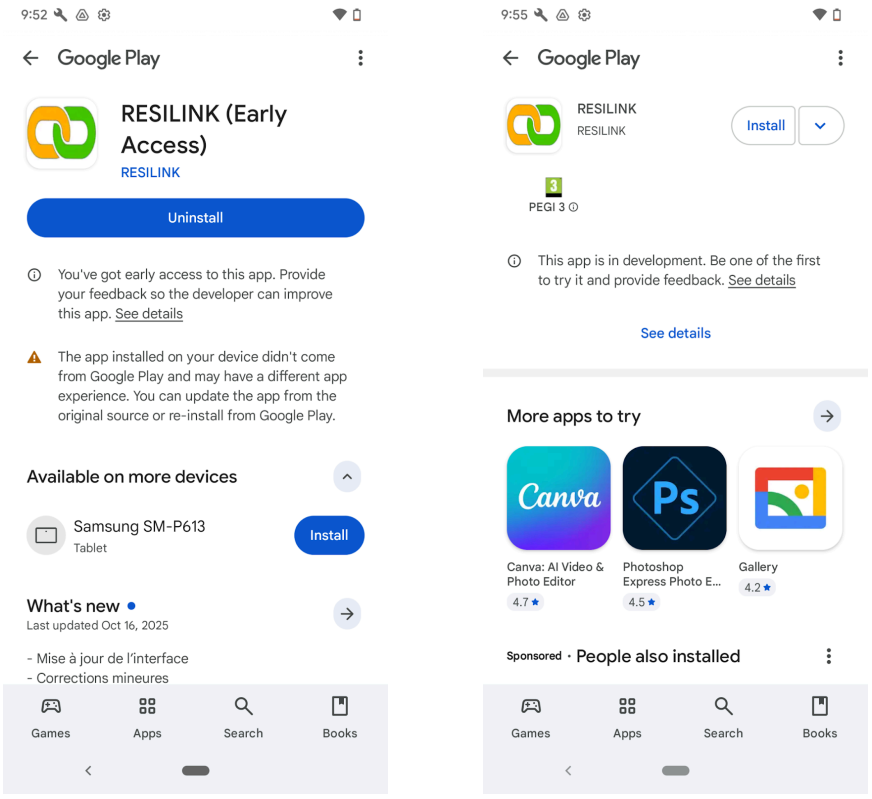
After the two-week internal testing period and once all critical issues are resolved, the app can be promoted to the production track for full public release.

1. In the Play Console, navigate to Testing → Internal testing and open the latest release.
2. Click Promote release → Production.
3. Set the rollout percentage (start with 20% for a staged rollout, or 100% for immediate full release).
4. Review and confirm all store listing information, pricing, and distribution settings.
5. Click Start rollout to Production.

The app will undergo a review by Google (typically 1–3 business days for the first submission). Once approved, it will be publicly available on the Play Store in all selected countries.

> ⚠ **WARNING** Ensure the RESILINK backend server is operational and accessible from the public internet before publishing the app. Users will not be able to use the app if the API server is unreachable.

Here is an example for the RESILINK mobile app in the play store.



Google Play link: https://play.google.com/store/apps/details?id=com.uppa.resilinkMobileApplication

5. Test of the RESILINK back-end server

5.1. Server Architecture and Current Capabilities

The RESILINK back-end server currently operates in an isolated configuration, where one server instance supports one client application without inter-server communication. In this mode, the server functions reliably and provides stable access to all required API endpoints. A Swagger interface is available for inspection and testing of the exposed routes at the following address: **https://resilink-dp.org/v1/api-docs/**

This standalone implementation is intended primarily for internal use, as it mirrors the behavior of ODEP in a self-contained environment. While it does not act as a bridge to ORANGE's ODEP API, it supports the same functional interactions and remains available alongside the ODEP-based version. The main codebase is hosted on GitHub at **https://github.com/ZiQuwi/RESILINK_Render_Server**

5.2. Recommendation Engine and New Data Model

A functional recommendation engine has been implemented to tailor suggestions both to individual users and to global application trends across asset types. To support this feature, two new collections were introduced into the database: **RecommendationStats** for user-specific metrics, and **GlobalRecommendation** for aggregated application-wide statistics.

Example of a RecommendationStats entry:


```
{
  "_id": { "$oid": "69149b87708b451d10810054" },
  "name": "acazaux",
  "totalOffersCreated": 30,
  "assetType": {
    "Fruit": 12, "Vegetable": 9, "Crop": 4, "Machinery": 4,
    "Transport": 3, "Inputs": 2, "Storage": 5, "Labor": 3,
    "Other services": 3, "Livestock": 2
  },
  "lastUpdated": "2025-11-25T14:08:10.374Z"
}
```


Example of a GlobalRecommendation entry:


```
{
  "_id": { "$oid": "6915f9e819bbf4d49698bcb0" },
  "assetType": {
    "Fruit": 45, "Vegetable": 30, "Crop": 30, "Machinery": 10,
    "Transport": 9, "Inputs": 0, "Storage": 18,
    "Labor": 23, "Other services": 5
  },
  "lastUpdated": "2025-11-13T09:09:53.163Z"
}
```


The separation into two collections allows for simpler updates and future refinements, particularly regarding global recommendations. User preferences are updated whenever a user creates an offer or views a specific offer within the mobile application. Each interaction increases the weight associated with the corresponding asset type, progressively modeling user interest.

Global preferences, on the other hand, are computed through a scheduled CRON task that sums asset-type weights across all users. These global weights are used when the user is not authenticated in the application. For authenticated users, the recommendation score is computed as a weighted combination: **70% user-specific preferences and 30% global preferences**, introducing diversity and encouraging discovery.

The full suggestion mechanism is implemented and functional in the development environment.

5.3. Ongoing Development: Inter-Server Communication and Data Retrieval

The next major development phase involves enabling communication between multiple RESILINK servers. To support this distributed architecture, we rely on the version of the server that operates without ODEP, allowing full control and flexibility in data handling and routing logic.

Two new collections have been added to support this multi-server model:

RegisteredServers:


```
{
  "_id": { "$oid": "6925d035c7fc344e01adfd1e" },
  "name": "ResiLink globale",
  "url": "https://resilink-dp.org",
  "createdAt": "2025-11-25T12:30:26.000Z",
  "updatedAt": "2025-11-25T12:30:26.000Z"
}
```


FavoriteServers:


```
{
  "_id": { "$oid": "6927299e6bbc21eee66cde51" },
  "id": "acazaux",
  "lastUpdated": "2025-11-26T16:24:58.467Z",
  "createdAt": "2025-11-26T16:23:58.606Z",
  "servers": [
    "https://resilink-dp.org",
    "https://resilink-dp"
  ]
}
```


RegisteredServers maintains a registry of all deployed RESILINK servers and allows each instance to identify which peers it can communicate with.

FavoriteServers enables each user to bookmark servers from the global registry, granting permission for cross-server retrieval of public offers. This design deliberately keeps user profiles lightweight by isolating cross-server preferences in a dedicated collection.

The current development effort focuses on fetching offers from these favorite servers. A local server contacts each selected remote server and retrieves its valid offers. This introduces several new challenges—most notably the issue of non-unique offer and asset identifiers. In isolated servers, offers use incrementing numeric IDs, which collide when external data is imported.

The resolution involves restructuring the server response format so that each block of offers is grouped under its origin server, with the server's registered name acting as the unique identifier. This approach naturally restores both offer uniqueness and asset-offer relationships without altering the identity model of individual servers.

5.4. Current Challenges and Open Design Questions

Two additional topics remain under discussion, though they were temporarily addressed manually for testing:

1. **Global Server Authentication for Offer Retrieval**
    - *Option 1:* Remove token requirements for read-only offer retrieval (simplest approach).
    - *Option 2:* Create an automatic global user profile for each local server on first launch (avoids embedding real credentials in code).
2. **Automatic Registration of Local Servers in the Global Registry**
    - Should servers self-register automatically on deployment, or should registration require user confirmation to allow manual naming and configuration?

These elements will need to be finalized prior to deploying a fully automated multi-server environment, as they affect how new instances announce themselves and how remote data is securely accessed.

6. Test of open-API & Interoperability

6.1. Orange Use Case

6.1.1. General view

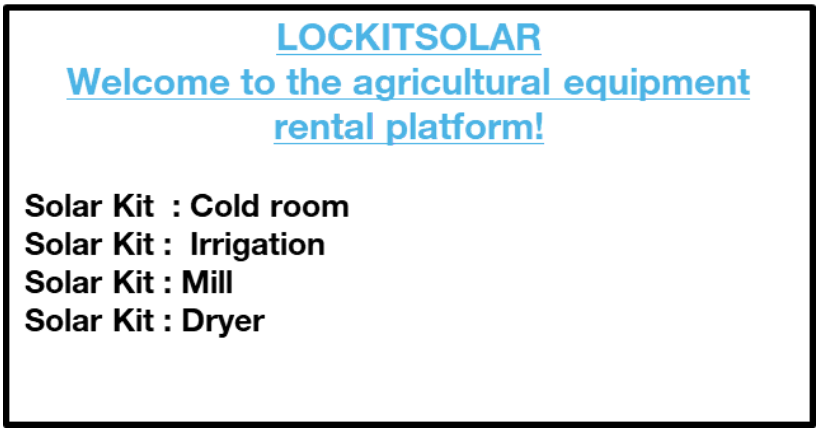
The objective of Orange's demonstrator is to create a small application to show how to use RESILINK Open API.

Linked to Orange Research projects in the MEA Area (Middle East and Africa), the suggested use case is a rental service of different solar powered tools for agriculture activities.

Today the offer of Orange Energies is a solar kit complete with different kinds of equipment, and in the context of professional activities, you can find some tools like solar Dryer, Solar Mill or a Chicken farm Kit.

6.1.2. The story of LocKitSolar

Let's imagine a startup named LKS for LocKitSolar. LKS offers to rent solar powered machines for agriculture.



To increase its visibility, LKS decided to become RESILNK partner. The first idea is to expose some promotion offers to Resilink users. But there seems to have other opportunities: publication of news on Resilink portal and display of targeted ads on LKS portal side.

As the **administrator** of LKS platform,

1) I want to publish a promotion offer on RESILINK platform

« **One-month promotional offer period on irrigation kits, rental at 1 euro/month**»

I can fill the form on LKS website, RESILINK API allows me to publish this offer on RESILINK platform.

2) I search for people to install irrigation kits for a particular region. I want to publish this information in RESILINK News.

« **6 months engagement, payment after each installation using Orange Money**»

I describe the content of the news, RESILINK API allows to add this news on RESILINK platform.

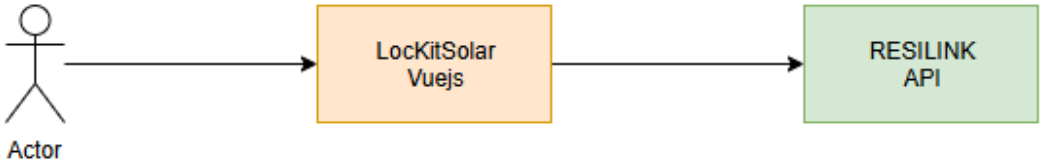
3) I configure the current Resilink offers to be displayed on the LOCKITSOLAR platform

« **Selection of active offers in a geographical area (Morocco for example) linked to solar energy** »

I mention the parameters for the search, RESILINK API allows searching in the current ads on RESILINK platform.

6.1.3. Technical view

For a normal service, we should have created a specific backend for the application that will communicate with the RESILINK API. However, since the idea is to create a sample application as simple as possible to demonstrate the use of the RESILINK API for creating a sharing resources service, we decided to create a backendless application that directly interacts with the RESILINK API.



The LocKitSolar is developed as a simple web app using the JavaScript framework Vue.js. It will be hosted on Google Cloud Platform with an Nginx component using Cloud Run to reduce hosting costs. Since we decided not to implement a backend, users must log in before using the service to avoid putting the password inside the JavaScript code exposed to the client.

6.1.4. Latest version of the LocKitSolar demonstrator

See D4.2b “Final report software templates demonstrating RESILINK's open API”.

Acronyms List

Acronym	Explanation
AAB	Android App Bundle
API	Application Programming Interface
APK	Android Package Kit
CRON	Command Run ON
MEA	Middle East and Africa
ODEP	Orange Decentralized Exchange Place
QR code	Quick Response code
SIRET	Système d'identification du répertoire des établissements
SMS	Short Message Service
URL	Universal Resource Locator

Project Co-ordinator Contact

Pr. Congduc Pham

University of Pau

Avenue de l'Université

64000 PAU

FRANCE

Email: Congduc.Pham@univ-pau.fr

ACKNOWLEDGEMENT

This document has been produced in the context of the PRIMA RESILINK project. The RESILINK project consortium would like to acknowledge that the research leading to these results has received funding from the European Union through the PRIMA program.