

MOSIP Docs 1.2.0

307 M

MOSIP

The Future of Digital Identity: Secure, Scalable and Open-Source.

Welcome ! 

What is MOSIP?

Modular Open-Source Identity Platform (MOSIP) is an open-source, open standards-based [foundational identity platform](#) designed to help countries build and manage their national ID systems. Anchored at the [International Institute of Information Technology, Bangalore \(IIIT-B\)](#), MOSIP enables governments to conceive, develop, implement, and own secure and scalable digital identity solutions.

Built with an API-first approach, MOSIP provides [ID Lifecycle Management](#) features, covering [Identity Issuance](#), [Identity Verification](#), and [Identity Management](#). The platform is designed to foster global Digital Public Goods (DPGs) in digital identification and governance. By leveraging open-source technologies, MOSIP ensures scalability, security, and privacy, adhering to best industry practices.

MOSIP's modular and adaptable architecture gives adopting countries full ownership and flexibility to customize the system to their needs. As a transparent and human-centric initiative, MOSIP encourages global collaboration, welcoming contributions from developers, technology partners, and research institutions worldwide.

Our global team of experts and advisors supports countries throughout the adoption and implementation of their digital ID systems, ensuring that they function as a common governance infrastructure for inclusive and secure digital transformation.

👉 [Learn more](#)

Our Mission & Objectives

At MOSIP, our mission is to empower governments worldwide to build and own secure, inclusive, and scalable digital identity systems. As an open-source, modular platform, MOSIP provides a strong foundation for nations to establish their digital public infrastructure, ensuring privacy, security, and interoperability.

Our key objectives include:

- **Enabling countries to develop secure, citizen-centric ID systems** by providing a robust and fully functional **framework**.
- **Offering flexibility and customization** so that nations can tailor the platform to their specific needs.
- **Ensuring privacy, security, and confidentiality** of individuals' data, following global best practices.
- **Upholding transparency, security, and human-centric technology** to foster trust and reliability.
- **Supporting global collaboration** through open-source contributions, partnerships, and an ecosystem of technology providers, researchers, and developers.
- **Providing a scalable and accessible solution** that serves populations from a few thousand to several hundred million.

By fostering an open and collaborative ecosystem of technology partners, researchers, and developers, MOSIP enables nations to build robust, adaptable, and future-ready identity solutions tailored to their unique needs.

 [Start Exploring](#)

Why MOSIP?

1. Modular

MOSIP's mutually exclusive technology bundles allow adopting nations to build custom workflows instead of relying on a fixed system. Each functionality is designed as an independent microservice, ensuring flexibility and control over digital ID implementation.

2. Open Source

We are committed to developing trusted and transparent technology while contributing to global standards. Our open-source code repository is available entirely on [GitHub](#), encouraging contributions from user communities and providing governments with full control over their ID systems. MOSIP actively collaborates with IEEE, governments, and industry partners to establish common Open Standards and Protocols, making MOSIP easy to integrate, interoperable, cost-effective, and time-efficient.

3. Vendor-Neutral

We offer governments the ability to integrate with a wide range of compliant technology partners. A foundational digital ID system requires smooth integration of platforms, biometric devices, and system integrators. MOSIP ensures vendor neutrality through the MOSIP Partner Programme and [Marketplace](#), allowing countries to choose and update their technology solutions freely, saving both time and financial resources.

4. Secure and Private Design

We believe every individual should be in control of their own data. MOSIP ensures data security and privacy, with cryptographic encryption and zero-knowledge architecture safeguarding information both in transit and at rest. Governments retain sovereign control over their ID systems, and data sharing occurs only with the individual's consent.

Learn more about our security and privacy principles [here](#).

5. Cost-Effective

MOSIP provides a zero-cost platform for adoption and licensing, enabling governments and organizations to build a foundational digital ID system under Mozilla's Public License 2.0. The automation of processes within MOSIP minimizes implementation costs, allowing countries to test and establish effective digital infrastructure efficiently.

(i) Note: For more details on licensing, click [here](#).

6. Human-Centric

We aim to develop technology that can cater to diverse and varying requirements around the world. The expert team at MOSIP constantly strives to learn from on-ground experiences in adopting nations, to understand their context-specific requirements and provide unique and adaptable solutions.

7. Inclusive

MOSIP is committed to inclusivity, ensuring technology remains accessible to all, regardless of gender, race, or economic status. Our ongoing collaborations with universities, research institutions, and field teams help refine our technology for unrestricted accessibility. Innovations like Inji, a digital wallet mobile app, empower residents to access their digital identities even in remote, low-connectivity areas.

To know more, click [here](#).

Explore Key Areas

Principles

Explore MOSIP's core principles.

Modules

Explore different modules.

Releases

Explore latest releases.

Roadmap

Explore MOSIP's key milestones and objectives.

Resources

Dive into interactive workshops, webinars and more.

Community

Join the MOSIP community.

- ⓘ **Note:** To read through the previous version of the documentation, please refer to our [Older Documentation](#).

Overview

Understanding MOSIP's Role in Foundational ID Systems.

What is a Foundational Identity?

A **Foundational Identity System** provides individuals with a unique identifier issued by the government, enabling identity assertion and verification across multiple services. Unlike functional IDs, which are designed for specific use cases such as healthcare, finance, and social services, foundational IDs serve as a universal framework that various sectors can leverage.

MOSIP empowers governments to build **secure, interoperable, and inclusive** foundational ID systems. With robust privacy protections and modular architecture, MOSIP ensures individuals have control over their personal information while enabling seamless access to public and private services.

Below is a diagram illustrating the relationship between **Foundational IDs** and **Functional IDs**.

- **Foundational ID systems** provide individuals with a **unique, government-recognized identifier** for identity verification. They help **de-duplicate** records, **authenticate** individuals, and **verify** identity attributes.
- **Functional IDs** are sector-specific and designed for specific use cases such as **healthcare, finance, social protection, education, and voting**. These IDs can leverage the foundational ID system to ensure accuracy, efficiency, and seamless service access.

This structure helps create a **secure and interoperable identity ecosystem** for both public and private services.



What is MOSIP?

MOSIP ([Modular Open-Source Identity Platform](#)) is a secure, scalable, and open-source framework designed to help governments and organizations build foundational identity systems. It offers a flexible and configurable approach, allowing countries to tailor their national ID systems to meet specific requirements while ensuring privacy, security, and interoperability.

The below image illustrates MOSIP as a modular, open-source identity platform designed for secure and scalable digital identity systems. It highlights key features such as interoperability, open standards, security, and privacy, ensuring seamless integration without vendor lock-in.

Additionally, it showcases MOSIP's core functionalities, including ID issuance, identity verification, and lifecycle management, making it a flexible solution for national ID implementations.



MODULAR OPEN SOURCE IDENTITY PLATFORM



Open APIs and Standards

Allows interoperability and seamless interaction with external systems and avoids lock-in



Open Source

Source code and documentation available under Mozilla Public License 2.0



Platform Approach

Unbundles monolithic ID system complexity and enables building advanced use cases



Security by Design & Privacy by Intent

Resident data privacy and safety is the highest priority

MOSIP provides a robust and secure platform for

- ID issuance
- Identity verification
- ID life cycle management

Interoperability

No Vendor Lock-in

Privacy and Security

The image below highlights MOSIP's security and privacy principles, emphasizing its "Security by Design" and "Privacy by Intent" approach.

These principles align with MOSIP's Privacy and Security framework.

For more details, please refer to [Privacy and Security](#) section. To learn more about MOSIP's Principles, [click here](#).



MOSIP Modules

The image below illustrates the various MOSIP Modules:

- Pre-Registration – Enables users to provide basic demographic data and book appointments for registration.
- Registration – Facilitates registration of an individual, for availing a Unique Identification Number (UIN) by providing demographic and biometric data (fingerprint, iris, and face photograph) in online/offline mode.
- Registration Processor – Generates a Unique Identification Number (UIN) post a regulated process and enriches data.
- ID Authentication – Provides demographic and biometric authentication services, including Yes/No and E-KYC services, enabling access to a myriad of rights and services.
- Resident Services – Allows residents to update and monitor usage of their IDs, giving individuals control over their own identity.
- Partner Management – For onboarding, managing, and integrating external partners (relying parties) within the MOSIP ecosystem.

To learn more about the MOSIP modules, please refer [here](#).

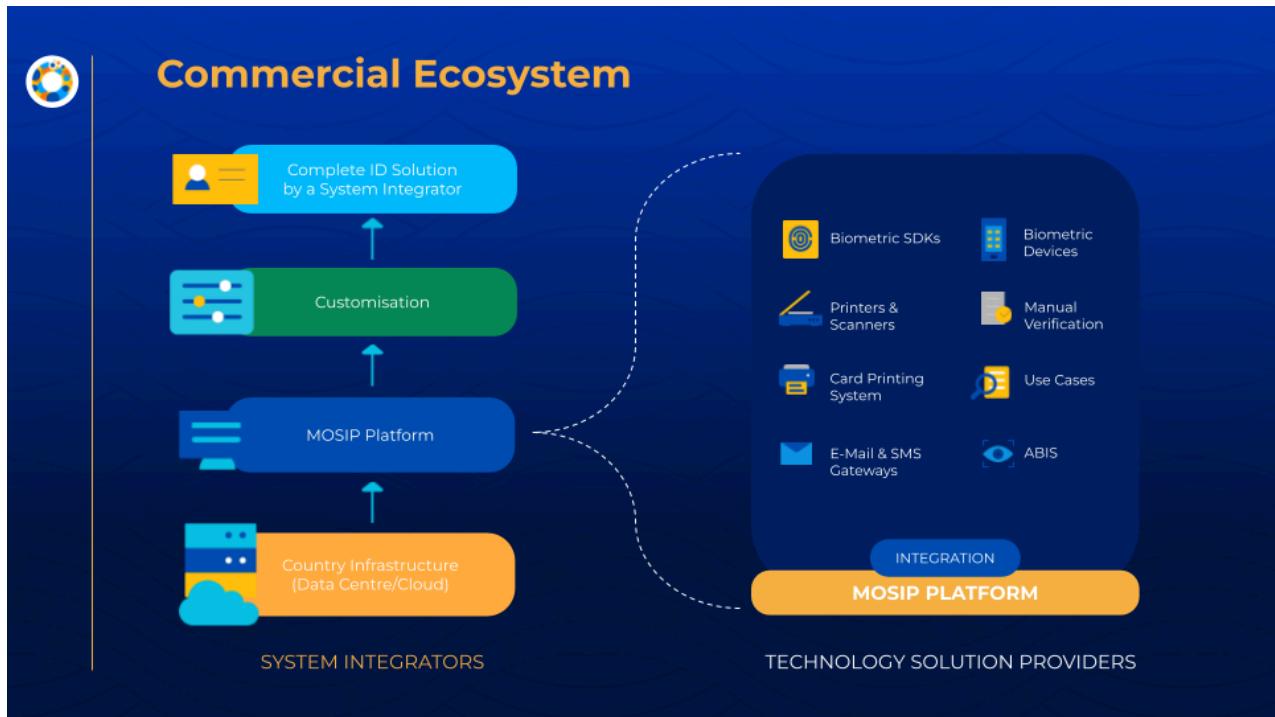


MOSIP Ecosystem

MOSIP collaborates with ecosystem partners to develop tailored identity solutions for each country.

The diagram below illustrates the **MOSIP Ecosystem**, highlighting how various components integrate with the **MOSIP Platform** to deliver a comprehensive ID solution.

To learn more about the MOSIP Ecosystem, please refer [here](#).



MOSIP's Offerings

The diagram illustrates **MOSIP's Key Offerings** in **ID Lifecycle Management** and **ID Authentication**, highlighting two main processes:

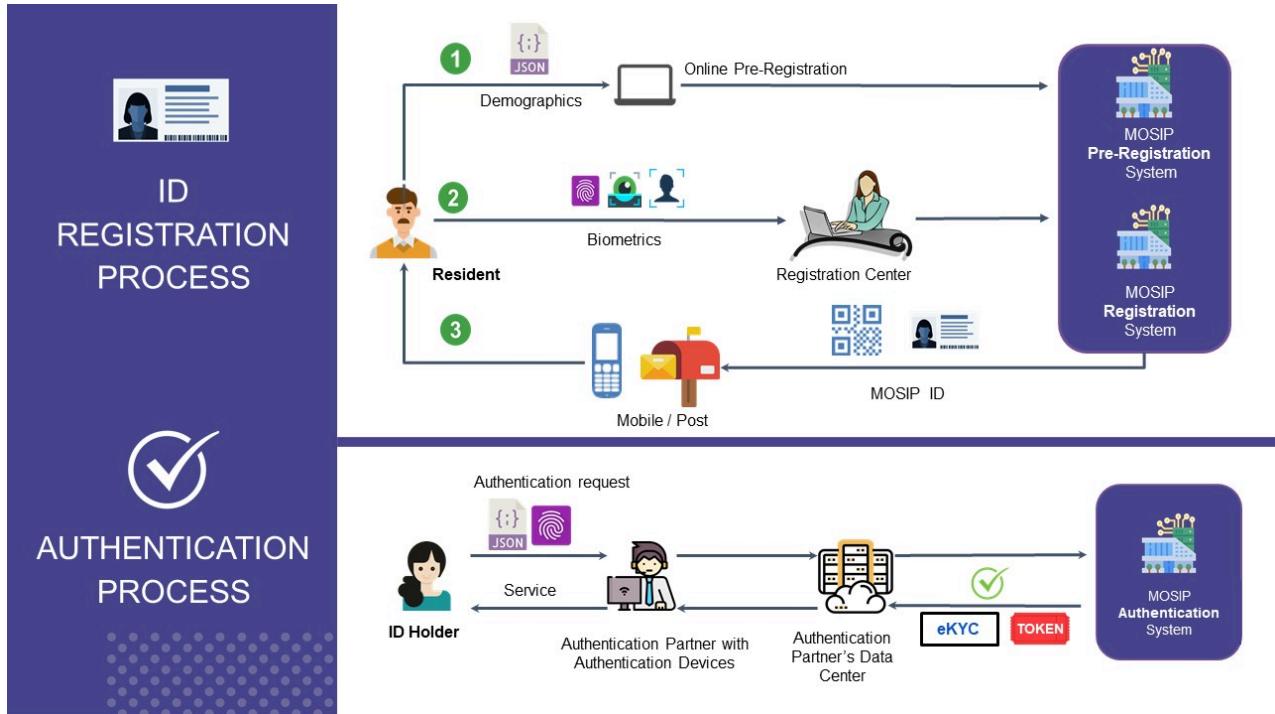
1. ID Registration Process

- Step 1: Online Pre-Registration – A resident submits demographic details online.
- Step 2: Biometric Enrollment – The resident visits a registration center for biometric data collection.
- Step 3: ID Issuance – After successful validations and processing, the resident receives a Unique Identification Number (UIN).

2. ID Authentication Process

- An ID holder requests authentication to access services.
- Authentication is performed via an authentication partner equipped with biometric or digital verification tools.
- The MOSIP Authentication System validates the identity, providing eKYC or token-based responses for service access.

These offerings enable secure, scalable, and modular identity management and authentication solutions.



Building a National ID System Using MOSIP

Countries can leverage MOSIP as the base identity platform and configure, customize, and extend it to build systems just the way needed.

The image below depicts how MOSIP provides the base layer to build a national ID platform.



Use Cases Layer

Country-specific ID-linked services



Implementation Layer

Country Solutions with customisations
by Systems Integrators



Core Technology Layer

MOSIP Platform

License

Empowering users through transparent licensing.

The documentation is licensed under a Creative Commons Attribution 4.0 International License.



CC license Image

All MOSIP's [core](#) repositories are licensed under the terms of [Mozilla Public License 2.0](#).

All trademarks are the property of their respective holders. Other products and company names mentioned [herein](#) may be trademarks and/or service marks of their respective owners.

Principles

The guiding foundation.

MOSIP is designed, developed, and implemented based on **core principles** that drive its effectiveness and adaptability. These principles ensure that MOSIP remains open, secure, interoperable, scalable, and inclusive, allowing it to meet the needs of diverse users and support the creation of accessible, digital identity systems globally.

- **Modular:** Identity systems that serve unique needs with mutually exclusive technology bundles.
Rather than a stand-alone solution offered to every adopting nation, MOSIP technology allows user countries to build custom workflows. Each functionality is built with independent microservices to provide flexibility and control.
- **Open Source:** Trusted & transparent technology, adhering to global standards.
An open-source code repository, available on GitHub, encourages contributions from user communities and offers governments control over their ID systems. MOSIP also works with the **Institute of Electrical and Electronics Engineers (IEEE)**, governments, and a commercial ecosystem to arrive at common **Open Standards and Protocols**. This makes MOSIP easy to integrate, interoperable, cost- and time-efficient.
- **Vendor-Neutral:** Offering the ability to integrate with a wide range of compliant technologies.
In the setting up of a foundational digital ID system, it is critical that the platform, biometric devices, and system integrators, are able to integrate and function smoothly. Vendor-neutrality, maintained through the **MOSIP Partner Programme** and the establishment of the **MOSIP Marketplace** offers countries the ability to choose and change their technology solutions at any time and save precious time and financial resources.
- **Private and Secure:** Ensuring every individual is in control of their data.
MOSIP is designed to keep **data security and privacy** in mind, ensuring that data is protected in flight and rest. **Cryptographic encryption** and **zero knowledge architecture** ensure that no sensitive data is stored on the MOSIP system. Governments of adopting nations have sovereign control over their ID systems, and data sharing only happens with the individual's consent.
- **Scalable:** Designed to accommodate growth and changing needs.
The platform is designed to be scalable, accommodating the growth of users

and services. It can cater to large populations, support various regions, and integrate with multiple systems securely, without compromising performance. Its **modular architecture** allows it to easily expand and adapt to new requirements, making it suitable for both **small-scale deployments** and **national level rollouts**. This scalability ensures that MOSIP can **serve diverse countries** and populations with varying technological and infrastructural capabilities.

- **Human-Centric:** Technology for all.

We aim to develop technology that can cater to **diverse** and varying requirements around the world, yet being human-centric. The team at MOSIP constantly strives to learn from on-ground experiences in adopting nations, to understand their context-specific requirements and provide **adaptable** solutions.

- **Inclusive:** Technology that leaves no one behind.

With the mission of empowering lives all over the world, MOSIP continues to take steps toward being an inclusive platform. Ongoing collaborations with global universities, research organizations, and strong on-ground teams have sharpened our focus on developing technology that is **unrestricted by gender, race, and economic status**. Additionally, technology features allow residents to access their digital identities even in remote areas with **low connectivity**.

Please refer [here](#) for more details.

Inclusion

Technology that leaves no one behind.

With the mission of empowering lives all over the world, MOSIP continues to take steps towards being an inclusive platform. Ongoing collaborations with global universities, research organizations, and strong on-ground teams have sharpened our focus on developing technology that is unrestricted by gender, race, and economic status. Additionally, technology features allow individuals to access services with digital identities through multiple channels, and even in remote areas with low connectivity, ensuring no one is left behind.

Some mechanisms through which the MOSIP platform supports inclusivity is illustrated below:

- **Introducer Support For The Undocumented**
 - The **Introducer concept** in MOSIP allows individuals **without formal identity documents** to be enrolled into the digital identity system. A trusted and verified **Introducer**, such as a community leader or an authorized individual, vouches for the person's identity. This ensures that **marginalized populations**, including refugees, nomadic groups, and those lacking paperwork, can still obtain a digital identity. By enabling **inclusion through trust-based verification**, the Introducer mechanism helps governments provide **identity access to all**, promoting **social and financial inclusion**.
- **Multi-Language Support for Linguistically Diverse Communities**
 - Multi-language support in MOSIP enhances **inclusivity** by ensuring individuals can interact with digital identity systems in their **native or preferred languages**. This helps bridge **linguistic barriers**, making identity services accessible to diverse populations, including those with limited literacy in dominant languages. By supporting **localization**, MOSIP enables governments to **customize** the platform to suit national needs, fostering **greater adoption, trust, and usability**. This approach aligns with the goal of creating **inclusive and equitable digital identity ecosystems** worldwide.
- **Biometric Exception Handling for Individuals with Difficult-to-Capture Biometrics**
 - MOSIP's **biometric exception handling** ensures that individuals with **difficult-to-capture biometrics**, such as worn fingerprints or missing

biometrics like finger or eyes, are not excluded from digital identity systems. When biometrics cannot be collected, MOSIP provides **alternative methods**, such as marking the biometric modality as an exception and capturing a photograph as evidence, thereby allowing inclusive identity registration for all. This approach ensures that **elderly individuals, manual laborers, and persons with disabilities** can still obtain a secure and verifiable identity, reinforcing **inclusivity and accessibility** in digital identity.

- **Enabling In-Home registration with Android Registration Client**

- The [Android Registration Client](#) (ARC) enhances inclusivity by enabling **in-home registration** for individuals who are unable to visit physical registration centers. This portable solution ensures that residents, especially in **remote locations**, can access identity registration services at their convenience. By providing mobility for registration agents and allowing remote registrations, ARC significantly expands coverage and accessibility, ensuring that even those in underserved or hard-to-reach areas are included in national identity systems.

- **Ensuring Gender Inclusivity with MOSIP's GenderMag Collaboration**

- MOSIP's collaboration with [GenderMag](#) ensures inclusivity by integrating **gender-sensitive design principles** into the **User Interface (UI)** and **Design**. This approach helps identify and address barriers that may disproportionately affect users based on gender, ensuring that the digital identity platform is accessible and effective for **all users**, regardless of gender. This approach emphasizes on creating interfaces that are intuitive and equitable, thus ensuring an **inclusive** user experience for diverse populations.

- **Multi-Modal Verification with Inji and eSignet**

- MOSIP ensures **inclusivity** by offering multiple **verification modalities** through [Inji](#) and [eSignet](#). These solutions provide **secure and convenient** citizen verification across various channels, including **online/offline, self-service/assisted modes**, and accessible even with **smartphones, feature phones, or no phones**. This flexibility allows individuals from diverse backgrounds, including those with limited access to technology or support, to easily access and benefit from the services.

Through its inclusive mechanisms, MOSIP continues to push boundaries in providing accessible and equitable **digital identity solutions**. By focusing on **universal access** and **adaptability**, MOSIP ensures that everyone—regardless of

their background, location, or physical limitations—can benefit from the services enabled by the platform. These efforts reflect MOSIP's commitment to creating a **more inclusive and sustainable digital future** for all.

Privacy and Security

MOSIP's fundamental architecture and design incorporate the highest levels of privacy and security.

Security by design

Key security features:

- Encryption of data in-flight or rest. (See [Data Protection](#))
- Integration with trusted applications only.
- Fraud avoidance - association of authentication only with specific transactions.
- Misuse prevention - user can lock or unlock their authentication.
- Virtual ID and Tokens to prevent identity theft.
- All data sent out of MOSIP will be digitally signed.
- All incoming data will be signed by the respective entity.
- Any data sent to a relying party will be encrypted.
- Protection against internal attacks with every record in DB protected with integrity.
- Centralized key management.
- All API's are protected with OAUTH 2.0.

Privacy by intent

Key privacy features:

- Minimal data with selective disclosure on a need-to-know basis.
- Sensitive data protected (not stored or logged in clear form).
- Consent support – the user decides who can receive what credentials & what attributes.
- No search on the database (You can find a record only if you know the ID).

- Clear segregation of Biometric & Demographic data.
- De-centralised ID usage and data (cannot profile based on usage).
- Users are not limited to one permanent ID - Virtual ID.
- All relying party gets a privacy enabled tokens to prevent profiling across transactions. Permanent ID is never shared.
- Supports Wallet based decentralized ID issuance and usage.
- Face data is not sent to ABIS for deduplication.

Security

MOSIP Security Practices

This document provides a comprehensive analysis of security implementations across multiple levels. Additionally, it clearly delineates the boundaries of responsibility between MOSIP and the countries implementing the system.

Within this document, we have categorized security practices into 'Internal Practices' and 'Operational Protection'.

Internal security practices are integrated into the MOSIP development lifecycle to build security within the system from the ground up. These include rigorous threat modeling, secure coding practices, comprehensive code reviews, and continuous vulnerability assessments to ensure that potential risks are identified and mitigated early. By embedding these security measures during development MOSIP fosters a proactive security culture that not only minimizes vulnerabilities but also supports a robust defense strategy throughout the system's lifecycle.

On the other hand, **operational security practices** include firewall rules, intrusion detection systems, continuous monitoring, and incident response strategies. These measures focus on maintaining the security and integrity of the system during its operational phase, addressing runtime threats and ensuring compliance with best practices. Operational practices are outside of MOSIP development stage and to be taken up by the implementing countries.

Internal Practices

Internal security practices encompass measures such as security requirement elicitation, design, adherence to the MOSIP Principles, Platform development, static and dynamic code analysis, dependency scanning, code signing, and vulnerability management. These practices ensure that potential threats are identified and mitigated early in the development lifecycle.

Platform Design

MOSIP's fundamental architecture and design incorporate high levels of privacy and security.

(Table to be updated soon)

Development and Release Practices

This section details the measures taken during the development, testing, and release phases to ensure maximum security. Multiple checks are enforced at each stage through the use of various tools, tests, and scans. Key practices include:

Static Application Security Testing (SAST)

Static Application Security Testing (SAST) is a cornerstone of our security strategy. Tools like SonarCloud are used to perform in-depth code analysis during the development phase, identifying vulnerabilities such as SQL injection, cross-site scripting (XSS), and insecure coding practices. SAST provides developers with real-time feedback, enabling them to address security flaws early, thereby reducing the cost and effort of remediation later in the software lifecycle. These tools integrate seamlessly into our CI/CD pipelines, ensuring that security is addressed continuously and early. Dependency scanning tools like Dependabot, CodeQL, and others further enhance this layer of protection by monitoring and updating vulnerable dependencies.

- **Sonar Cloud** - Development Phase - SonarCloud is integrated with Github actions, offering developers actionable insights directly within the workflow. By highlighting security hotspots and technical debt, it enables teams to prioritize and address critical issues efficiently.

MOSIP Sonar cloud Link : <https://sonarcloud.io/organizations/mosip/projects>

- **CodeQL** (Java and Python) - Development Phase - CodeQL performs semantic code analysis, enabling the detection of complex vulnerabilities

- **Github Dependabot** (Vulnerability assessment and Version upgrade suggestions) - Development Phase - Dependabot simplifies the process of updating dependencies by creating pull requests with the necessary upgrades reducing manual effort and ensuring the codebase remains secure against known vulnerabilities. Its integration into GitHub workflows ensures timely updates and fosters a proactive approach to dependency management.
- **Open source compliance scanning** - Ensures that all open-source components in use comply with licensing requirements and security best practices. This scanning helps in identifying potential legal or security risks associated with third-party libraries. Automated tools are used to track, analyze, and flag issues related to incompatible or outdated licenses, ensuring smooth and compliant project operations.
- **Github scan** - Provides robust scanning capabilities integrated directly into GitHub repositories. It includes features such as secret scanning, dependency graph analysis, and vulnerability alerts, helping developers proactively detect and fix security issues within their workflows.
- **Data Breach Detector** - It is a production grade tool/script which goes through the DB and utilizes Deduce library to find out anomalies in various places such as names, address or numbers in plaintext etc.

Dynamic Application Security Testing (DAST)

DAST focuses on identifying security vulnerabilities in a running application by simulating real-world attack scenarios. Unlike SAST, which examines static code, DAST tests live applications, analyzing responses to detect flaws such as authentication issues, session management vulnerabilities, and exposure of sensitive data. Tools like Burp Suite Professional and ZED Attack Proxy (ZAP) are leveraged to conduct automated and manual penetration tests. These tools allow testers to evaluate application behavior under various conditions, ensuring robust protection against runtime threats. By integrating DAST into the release process, vulnerabilities can be identified and mitigated before applications are deployed into production.

- **Burp Suite Professional** : This tool is used for automated and manual penetration testing . It provides features such as intercepting proxy, web vulnerability scanner, and advanced debugging capabilities. Burp Suite enables testers to identify vulnerabilities like SQL injection, cross-site scripting (XSS),

and insecure session management. It also supports extensions for customized scanning and integrates seamlessly into security workflows.

- **ZED Attack Proxy:** This tool is used for finding vulnerabilities in web applications during the development and testing phases.

Release Practices

Release practices are essential for ensuring the security, authenticity, and traceability of software releases. Here is an overview of the components used in MOSIP releases.

- **Image Signing:** ASC (ASCII-armored PGP) signing is typically used to ensure the authenticity and integrity of software artifacts, including Docker images, by attaching a digital signature. When signing software images, MOSIP uses the private key to sign the image, and users can verify the signature using the corresponding public key.
- **JAR (Java Archive) Signing** is the practice of signing Java archive files to ensure that the contents of the JAR haven't been tampered with and to provide a way to verify the source of the file. This is currently not implemented in MOSIP.

Operational Practices

These recommendations provide a robust framework to ensure the security and integrity of production systems for MOSIP implementing countries, helping to mitigate risks and enhance overall cybersecurity posture.

- **SBI Compliant Devices:** Ensure that all devices used in the production environment are compliant with the latest Secure Biometric Interface (SBI) standards to ensure a highest level of security.
- **Trusted Platform Module:** A Trusted Platform Module (TPM) is a specialized chip on a local machines that stores cryptographic keys specific to the host system for hardware authentication. The private key is maintained inside the chip and can't be extracted out. By leveraging this security feature every individual machine would be uniquely registered and identified by the MOSIP server component with its TPM public key.

- **Compliance Tool Kit:** MOSIP Provides Compliance Tool Kit (CTK) to help the device vendors to check if their products comply with SBI specifications.
- **Access and Audit Logs:** Enables detailed access and audit logging for all critical systems and services in the production environment.
- **Patch Management (Host/Machines):** Implement a robust patch management policy to ensure that all production systems are up-to-date with the latest security patches.
- **Safe Data Centers:** Ensure that data centers housing production systems are designed and operated with a focus on security, availability and operational safety.
- **International standards:** Stay compliant on international standards such as ISO/IEC 27001, NIST Cybersecurity Framework, and relevant national regulations. Better to validate the compliance using third party assessments.
- **Ensuring Data Protection:** Enforce robust data protection measures to safeguard sensitive information at rest and in transit.
- **Consent-Based Data Handling:** Ensure that data is only collected, processed, and stored with the explicit consent of the individuals it pertains to, in accordance with privacy laws and regulations.
- **Regular Security Audits:** Perform regular security audits to assess the effectiveness of security measures and identify potential vulnerabilities in production systems.
- **Principle of Least Privilege:** Ensure that users and systems are granted only the minimum level of access necessary to perform their tasks, reducing the risk of accidental or malicious misuse.
- **Rate Limiting:** Implement rate limiting to protect services from abuse, such as brute force attacks or denial-of-service (DoS) attempts.

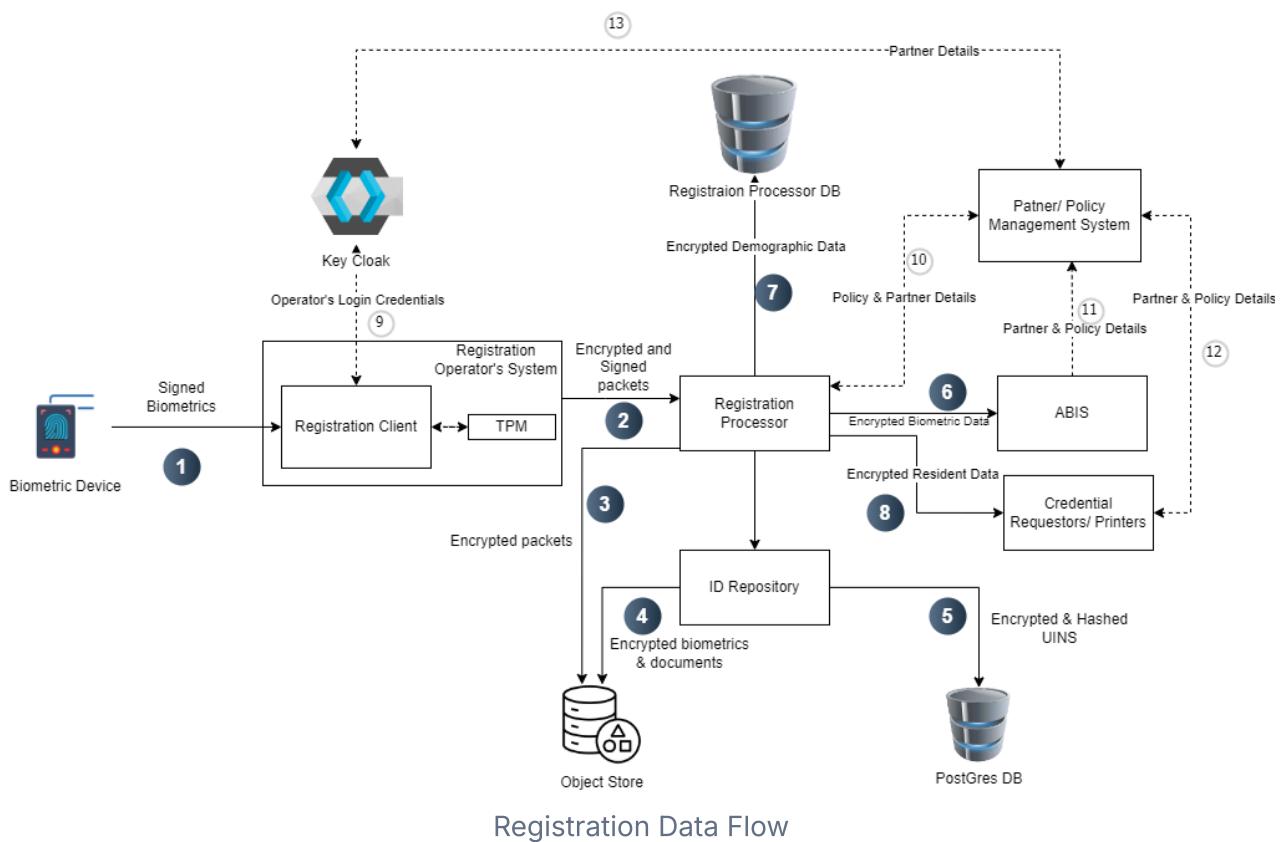
Data Protection

The security of user data is given the highest priority in MOSIP. Data is protected in flight and rest using strong cryptographic techniques. All operations on decrypted data are done in memory.

Various flows with encryption are illustrated below. Refer to [Keys](#) for all references of the type 'Kx' and 'KPx'.

Registration data flow

The below diagram represents a registration data flow system for biometric authentication and identity management.



1. Biometric Capture:

- A biometric device captures and signs biometric data before sending it to the **Registration Client (PK2)**. Then registration client verifies the signature

2. Registration & Encryption:

- The **Registration Client**, running on the operator's system, receives biometric data and securely encrypts it into packets.
- The client always refers to **Keycloak** for authentication, ensuring that only authorized operators can access the system.
- [Registration Client](#) signs the packet using the TPM key of the machine (K10) and encrypts the packet using MOSIP public key specific to (the registration centre, machine id) combination (K11).

3. Data Processing & Storage:

- The encrypted packets are transmitted to the **Registration Processor**, which processes and signs the data.
- The processed data is then stored in the **Object Store** and **ID Repository** for further use.

4. Secure Storage of Biometric Data:

- [ID Repository](#) encrypts biometrics, demographics, and documents and stores them in the Object Store. (K7.1,K7.2,K7.3)

5. Hashed UIN Storage:

- The UINs are hashed, encrypted, and stored in `uin` the table of `mosip_idrepo` DB. (K7.4)

6. Data Sharing & Policy Enforcement:

- When encrypted biometric data needs to be shared, it is sent to **ABIS** for authentication.
- The system consults the **Partner/Policy Management System** to verify partner details and enforce data-sharing policies.
- Only partners who have registered and authenticated via **Keycloak** can access the **Partner Management System**, where they must subscribe to specific policies to receive data.

7. Demographic Data Storage:

- Encrypted demographic data is stored in the **Registration Processor Database**. (K11)

8. Credential Issuance:

- Encrypted resident data is shared with **credential requestors and printers** based on the subscribed policies. (K12)

9. Operator Authentication:

- The **Registration Client** checks **Keycloak** to ensure that only authenticated operators can perform registrations.

10. Policy Validation for Data Transfer:

- Before transferring encrypted data to **ABIS, partners, or credential requestors**, the system refers to the **Partner/Policy Management System** to validate policies.

11. Partner & Policy Control:

- The **Partner Management System** is controlled by **Keycloak**, ensuring that only registered partners with valid credentials can subscribe to and enforce policies for data access. (11,12,13)

Datashare



Data shared with all partners like ABIS, Print, Adjudication, IDA, etc. is encrypted using partners' public key. Note that IDA is also a partner, however, a special partner in the sense that data is additionally zero-knowledge encrypted before sending to IDA (see the section below).

Zero-knowledge encryption

The [ID Authentication](#) module (IDA) is independent and may be hosted by several providers. IDA hosts all the biometric templates and demographic data. Unique additional protection is provided here to make sure that mass decryption of user

data is very difficult to achieve. The data can only be decrypted if the user's UIN is provided. Here is the encryption scheme:

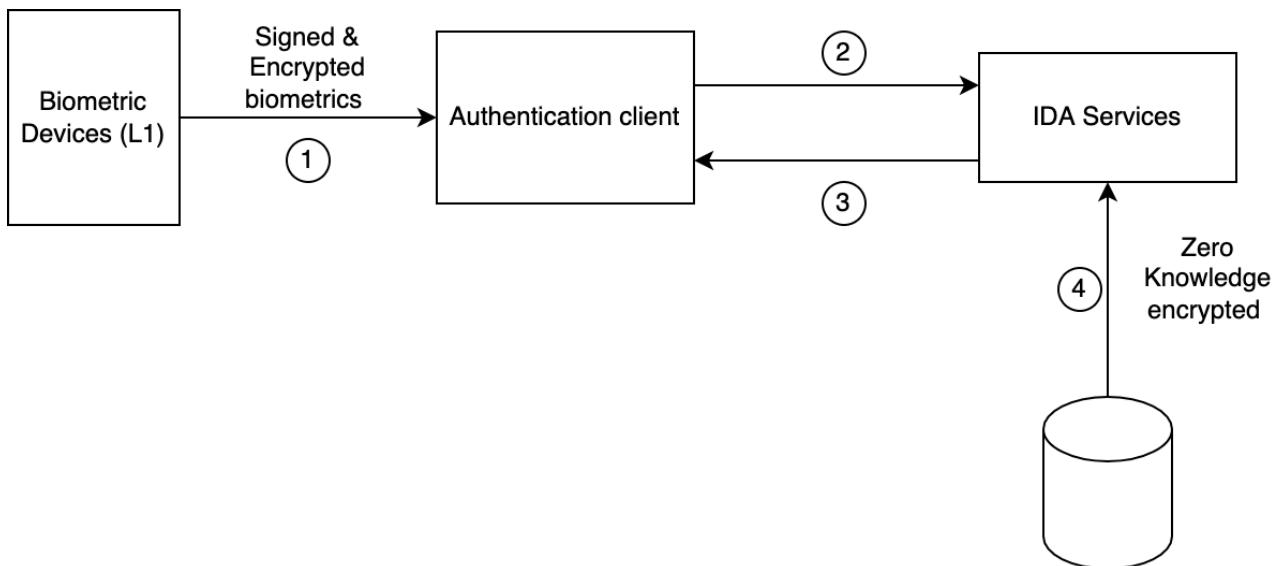
Encryption and sharing by Credential Service

1. Generate master symmetric encryption key K9.
2. Generate a 10,000 symmetric keys pool (ZKn). Encrypt each ZKn with K9 and store it in DB. (K12)
3. Randomly select one key from ZKn, and decrypt using K9.
4. Derive new key $ZKn' = ZKn + UIN/VID/APPID$.
5. Encrypt biometric templates and demographics.
 - $BIO = \text{encrypt}(\text{bio/demo with } ZKn')$.
6. Encrypt ZKn (this is done to share ZKn with IDA).
 - $ZKn\text{-IDA} = \text{encrypt}(ZKn \text{ with K22})$
7. Share the following with IDA:
 - a. ZKn-IDA
 - b. BIO
 - c. Random index (0 - 9999)
 - d. SHA-256 hash of UIN/VID/APPID
8. Share data in step 7 via standard [Datashare encryption](#) (which encrypts entire data with PK8).

Decryption by IDA

1. Generate master symmetric encryption key K18.
2. Decrypt data in Step 8 above using PK8.
3. Decrypt ZKn-IDA with K22 to get ZKn.
4. Encrypt ZKn with K18 and store it at a random index.
5. Bio-data is stored as is.

ID authentication flow



1. L1 devices contain [FTM](#) to encrypt (DE1, K21) and sign (FK1) biometrics at the source and send them to the authentication client.
2. The authentication client further encrypts the auth request with the IDA-PARTNER public key.
3. IDA decrypts zero-knowledge data as given in [Step 4](#) and then performs a demographic and/or biometric authentication.
4. The match result is returned to the Auth client. In the case of KYC, the KYC attributes are encrypted with the Partner's public key (as in [Datashare](#)).

Privacy

Overview

The right to privacy is a fundamental right in many contexts. Privacy protection or preservation can be ensured in an application by adopting a privacy friendly design stance.

What is privacy and privacy protection?

Privacy takes many forms. From an identity system perspective, the confidentiality of identity information and anonymity when using the identity offers privacy.

MOSIP views the identity system as a custodian of the individual's data. This data has to be protected in order to protect the individual from privacy and security risks. Privacy protection measures include [data protection](#), transparency, user control, confidentiality, selective disclosure, user anonymity and intrusion protection.

Privacy design elements

MOSIP addresses privacy design at four levels.

1. Functional privacy

- Selective disclosure
- Anonymization
- Need to know
- Encryption
- Tokenization

2. Security

- [Data security](#)
- Trusted applications
- Access control

3. User centricity

- User control
- Consent
- Usability
- Inclusion

4. Transparency

- Openness
- Verifiability
- Governance

These design principles have resulted in features as well as development practices in MOSIP that enhance privacy protection. A typical example for a practice is how PII (Personally Identifiable Information) is dealt with when creating application or audit logs. An example of a feature is how our [Datashare policies](#) allow selective sharing of information.

Technology

Discover how MOSIP's tools, components, and architecture come together.

Architecture

Modular, Secure, and Scalable: The Architectural Foundation of MOSIP.

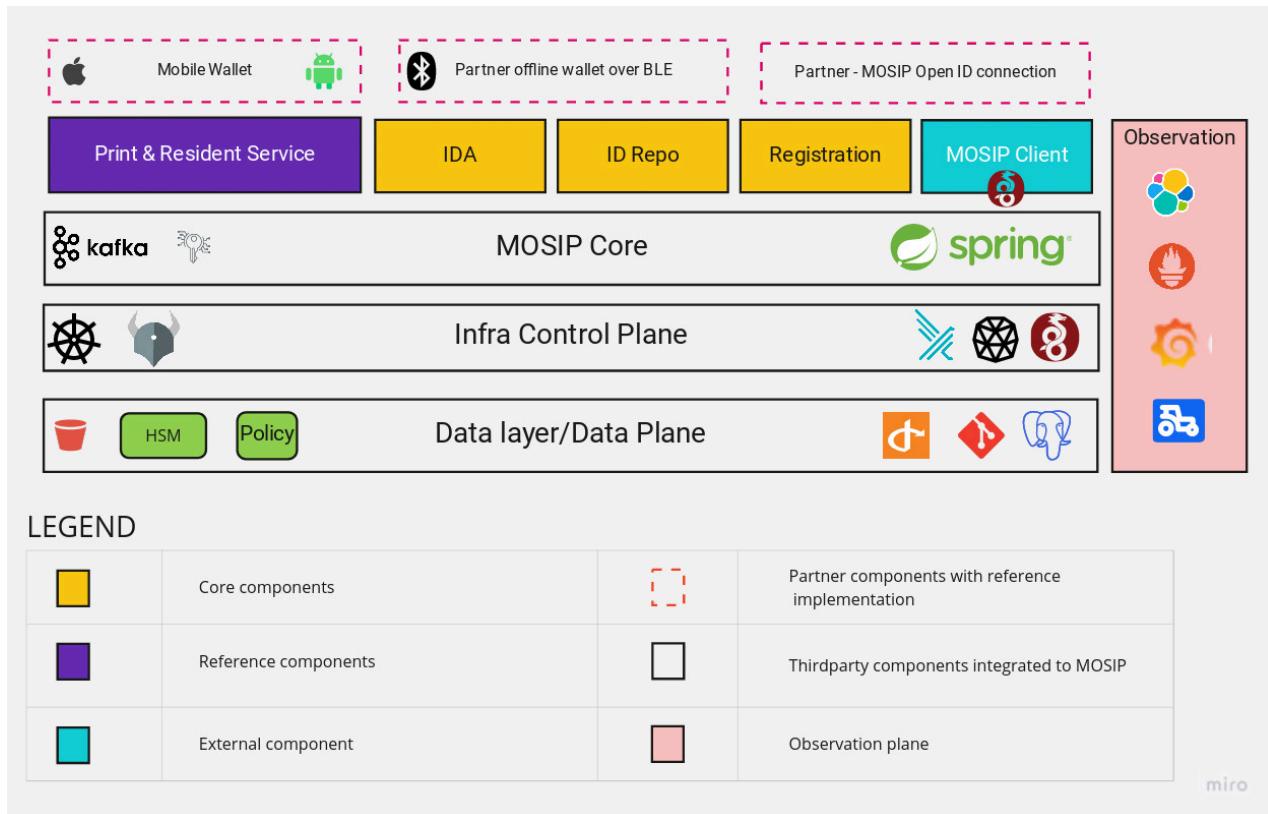
MOSIP is built on a modular, microservices-based architecture. Its modularity enables seamless adoption even in complex scenarios. Most [MOSIP modules](#) are designed as robust foundational infrastructure components, making them suitable for integration into various projects.

MOSIP is designed with the following architectural principles. These architecture principles are core to the development of the system's features and have a great influence on how and why specific software design patterns are used within.

- Data Privacy
- No Vendor Lock-in
- Open Standards
- Async/ Offline First
- Commodity Computing
- Fault tolerant
- Manageable
- Secure By Default

Architecture Overview

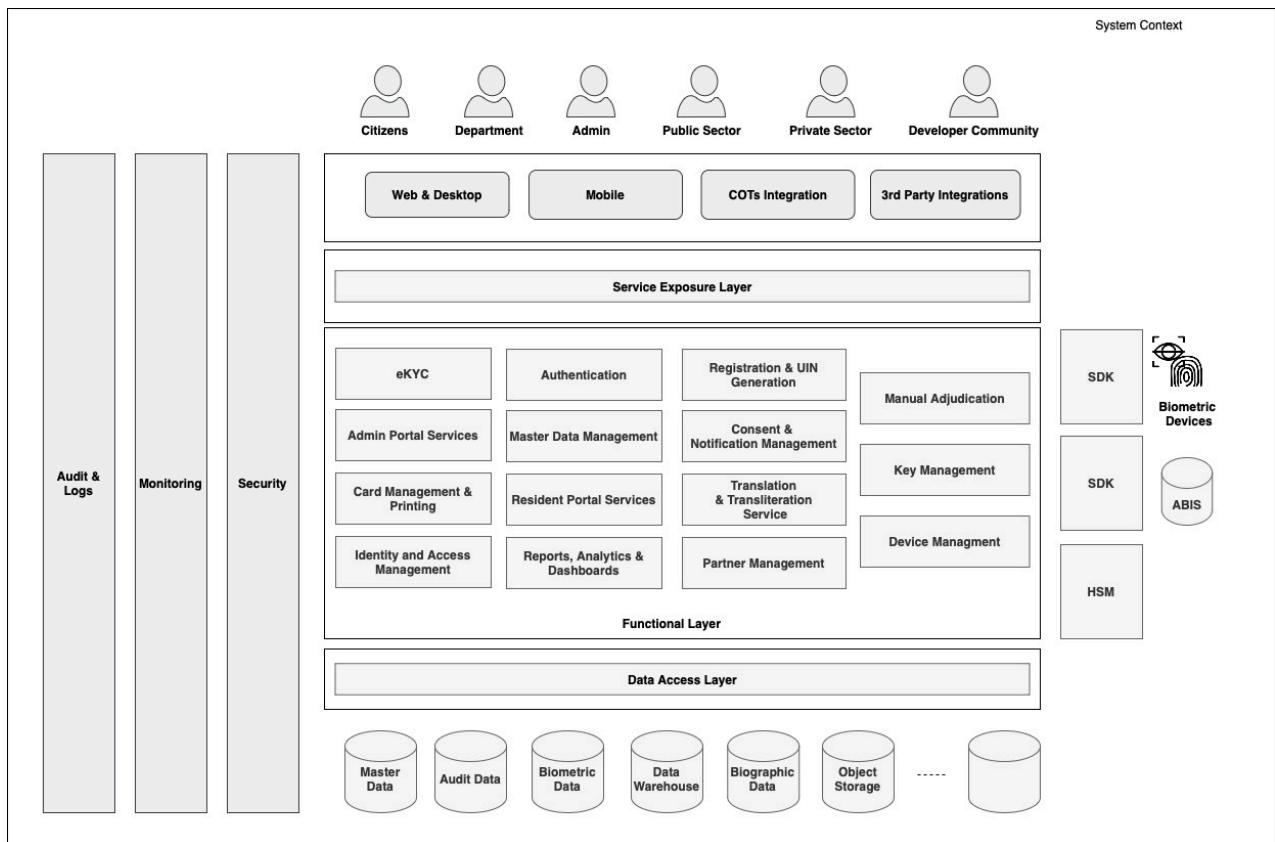
The diagram below provides an architectural overview, visually representing the components of the MOSIP Identity framework and its associated technology stack.



Architecture Overview

High Level Functional Architecture

The High Level Reference Functional Architecture serves as a blueprint outlining the system's high-level functioning and interactions, providing a structured framework.



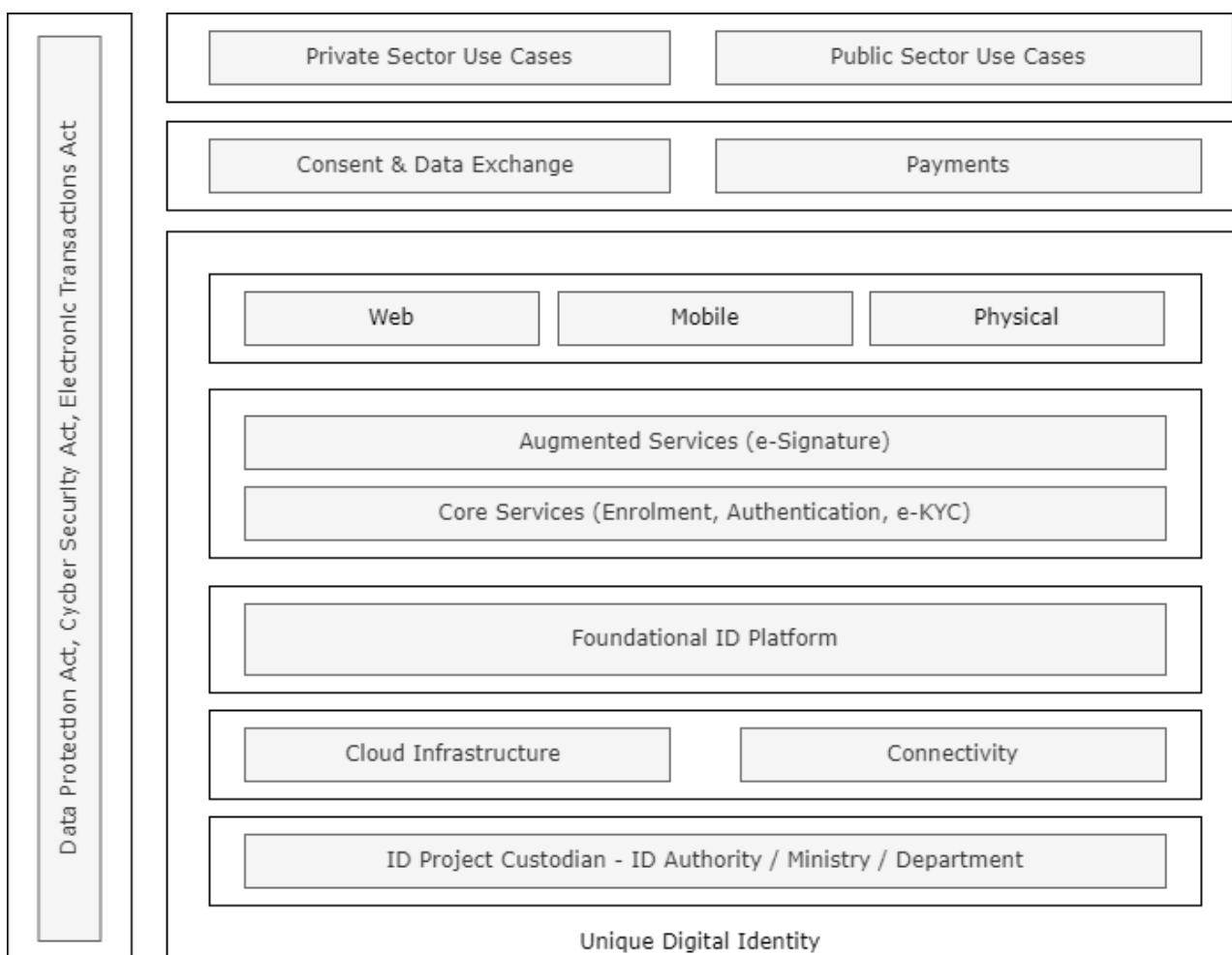
High Level Reference Functional Architecture

To know how MOSIP can be deployed, refer to [Getting Started](#). The different installation models are detailed in the [Deployment](#) section.

Digital ID DPI Framework

Blueprint for Scalable and Interoperable Identity Systems.

This reference blueprint provides a comprehensive vision for designing and implementing a Digital ID-led DPI infrastructure. It outlines how foundational identity systems can be leveraged alongside key DPI components, enabling various use cases across both the public and private sectors. The blueprint is structured in layers of technology, governance, and service delivery, ensuring scalability, inclusivity, and compliance with legal frameworks.



Blueprint of Digital ID led Development

Below is an explanation of its core components and their roles within the ecosystem.

1. Unique Digital Identity

At the core of the system is the Unique Digital Identity, which establishes a singular, definitive identity for every individual. This digital identity forms the backbone of all operations, ensuring secure identification and verification across diverse domains.

2. ID Project Custodian

The ID Project Custodian, whether a designated authority, ministry, or department, oversees this infrastructure. This custodian is crucial for maintaining governance, regulatory compliance, and operational efficiency, ensuring that all identity services align with national policies and frameworks.

3. Infrastructure Layer

Supporting the ecosystem is a robust Infrastructure Layer, consisting of cloud infrastructure and connectivity. The cloud ensures scalability, resilience, and the ability to manage large volumes of identity data. The connectivity layer ensures the system's accessibility, making it functional in both urban and remote areas.

4. Foundational ID Platform

At the heart of the blueprint is the Foundational ID Platform, a centralized framework for issuing and managing foundational IDs. This platform underpins all identity-related services, ensuring secure identity handling and enabling core functions like registration, lifecycle management, verification, authentication, and e-KYC (Electronic Know Your Customer). These services are essential for building trusted interactions across all sectors.

5. Core & Augmented Services

Now that we have established the core layers to anchor trust, we can expand functionality through augmented services, such as **eSignature** capabilities. These enable secure digital transactions and help governments transition from traditional in-person signatures. These added features enhance the system's utility, which is especially crucial for enabling paperless workflows.

6. Accessibility and Inclusivity

To ensure widespread adoption and accessibility, the system features a Multi-Channel Access Layer that enables interaction through **web, mobile (Apps, USSD), and physical interfaces(QR)**. This design ensures inclusivity by accommodating individuals with diverse backgrounds and varying levels of technological proficiency, to interact with the system effortlessly.

7. Consent Management, Data Exchange and Payments

In addition to its core and augmented services, consent management and data exchange capabilities are crucial for advancing digitization and digital development. These capabilities facilitate the regulated sharing of information across both public and private platforms. Moreover, the integration of payment systems adds significant value to Government-to-People (G2P) use cases. Additionally, data exchange can be further enhanced through the use of Verifiable Credentials.

8. Use Cases and Benefits Delivery

This blueprint is designed to act as a catalyst for seamless service delivery and impactful use cases across both the private and public sectors. It is essential to involve the private sector in the digitization process, as it plays a key role in enabling services such as banking, e-commerce, and telecommunications, thereby streamlining operations. In the public sector, this blueprint enhances service delivery in areas like social welfare, healthcare, taxation, and education, contributing to better governance and increased citizen engagement. By adopting an overall ID-led approach, it not only accelerates the adoption of digital identity but also ensures improved governance, streamlined operations, and enhanced citizen participation.

9. Legal and Regulatory Framework

Finally, the framework operates within a robust legal and regulatory environment, complying with key legislation such as the Data Protection Act, the Cybersecurity Act, and the Electronic Transactions Act. These laws safeguard data privacy and security, ensuring that the system operates transparently, trustworthy, and in full legal compliance.

Technology Stack

MOSIP is built using the below tools and technologies.

Technologies and Tools Used

This page lists all the technologies used in building MOSIP. Free and open-source software with clear long-term support availability has been chosen. For a deployment, certain choices can be replaced with other free or commercial options.

Domain	Tools/Tech nologies	Version	Licence Type	Commerc ial	Productio n	Cc
Operating System	CentOS	7.7	MIT License	Yes	Yes	NA of .
Operating System	Ubuntu Server	20.04	Free	No	No	NA
Infrastructure	Cloud - Azure/AWS	NA - Cloud tool	Commercial	Yes	Depends on Deploymen t Arch.	De on Det A
Development - Language Runtime	Java SE 11	OpenJDK 11	Oracle Binary Code License	No	Yes	NA
Development - Expression language	mvel2	2.4.7.Final				
Development - Scheduling	quartz	2.2.1				
Development - File Server	tus-java-client	0.4.3				

Development - Internalization	nv-i18n	1.29				
Development - Cryptography	TPM	2.0				
Development - UI Application framework	JavaFx	OpenJFX 11	GPL v2 + Classpath	No	Yes	NA
Development - Application Framework	Vert.x	3.9.1	Apache License 2.0	No	Yes	NA
Development - Application Framework	Spring	5	Apache License 2.0	No	Yes	NA
Development - Utilities	Apache commons(60+ to be considered)	Latest version	Apache License 2.0	No	Yes	NA
Development - Data Grid	Apache Ignite	2.4.0	Apache License 2.0	No	Yes	NA
Development - Object Mapper	Orika	1.5.2	Apache License 2.0	No	Yes	NA
Development - validator	Hibernate validator	5.4.2	Apache Software License 2.0	No	Yes	NA

Development - Encryption	BouncyCastle	1.59	Adaptation of MIT X11 License	No	Yes	NA
Development - JSON marshal/unmarshal	Jackson	2.9.5	Apache License 2.0	No	Yes	NA
Development - Device Driver	RXTX	RXTX-2-2-20081207	LGPL v 2.1	No	Yes	NA
Development - Unit Testing	Junit	5.x and above	Common Public License - v 1.0	No	No	NA
Development - Unit Testing	Junit	4.x and above	Common Public License - v 1.0	No	No	NA
Development - Log	logback	1.2.3	GNU Lesser GPL Version 2.1	No	Yes	NA
Development - Templating	velocity	1.7	Apache License 2.0	No	Yes	NA
Development - Tools	Open street view	NA - Cloud tool	Open Database License (ODbL)	No	Yes	NA
Development - IDE	Eclipse Oxygen	4.7.3	Eclipse Public License Version 2.0	No	No	NA
Development - Unit Testing	Karma	2.0.x	MIT License	No	No	NA

Development - Unit Testing	Jasmine	2.6.1	MIT License	No	No	NA
Development - API Documentation	Swagger	3.13.2	Apache License 2.0	No	No	NA
Development - Application Server	Tomcat server	8	Apache License 2.0	No	Yes	NA
Development - Orchestration	Apache Camel	2.19.3	Apache License 2.0	No	Yes	NA
Development - WebSub	Ballerina Websub	slbeta2	Apache License 2.0	No	Yes	NA
Development - Database	H2 DB	1.4.197	No	Yes	NA	
Development - Database	PostgreSQL	Server: 10	Postgres License BSD 2-clause "Simplified License"	Yes	No	NA
Development - Database	Derby DB	10.13.1.1				
Development - Database Modeling tool	PG Data Modeler	0.9.2	Commercial	No	Yes	No
Development -	Morena scanner	7	Commercial			

Scanner	library					
library Development - Code quality	Sonar	7.2	Open Source License	No	No	NA
Development - UI Designs	Pencil Project	3.0.4	GNU Public License version 2	No	No	NA
Development - TPM Java client	TSS.Java	0.3.0				
Testing tools	Rest-assured	3.0.0 and above	Apache License 2.0			
Testing tools	WireMock or Citrus framework	2.16.0 or respectively	Apache License 2.0	No	No	NA
Testing tools	JMeter	5.3.x	Apache License 2.0	No	No	NA
Testing tools	Burp suite Professional +	9.0.3.7	PortSwigger - Burp suite Professional + / V1.7.33	No	No	NA
Testing tools	TestNG	6.11	Apache License 2.0	No	No	NA
Testing tools	awaitility	4.0.3	Apache License 2.0	No	No	NA
Testing tools	testfx	4.0.16-alpha	EUPL1.1	No	No	NA

Testing tools	extentreports	3.1.5	Apache License 2.0	No	No	NA
Testing tools	selenium-java	3.141.59	Apache License 2.0	No	No	NA
DevOps tools	Jira	6.4 and above	Not Open source			
Testing tools	Java profiler	12.x	No	NA	12.0.3	Op Solu Lic
DevOps tools	SonarLint	v3.5	GNU GPL			
DevOps tools	GitHub	2.7.x	Commercial - Github			
DevOps tools	SonarQube	6.7.3 LTS	GNU GPL			
DevOps tools	Maven	3.53.x	Apache License 2.0			
DevOps tools	Docker	18.03.x CE	Apache 2.0			
DevOps tools	Ansible	2.2	GNU GPL v3.0			
DevOps tools	Github actions	NA - Cloud tool				
DevOps tools	Travis	NA - Cloud tool	MIT License			
DevOps tools	Glowroot	Apache License 2.0				
DevOps tools	Prometheus	Apache License 2.0				

DevOps tools	Grafana	Apache License 2.0				
DevOps tools	Python	3.x	PSF License			
Messaging	ActiveMQ	Apache License 2.0				
Messaging	Apache Kafka	Apache License 2.0				
Caching	Hazelcast	Apache License 2.0				
Object Store	MinIO	GNU AGPL v3				

Sandbox Details

Seamless Integration with MOSIP: Explore Our Sandbox Environments.

Are you interested in integrating with MOSIP as a partner? We invite you to connect with us by completing the [form](#). This will assist us in facilitating a seamless integration with our designated sandbox environments.

Please find below the two sandbox environments available for your use.

Collab

Collab is our development integration environment that has QA-tested dockers deployed. Our partners and contributors can use this to build on the platform or integrate with the QA-certified version of the latest platform code.

Regular nightly builds from engineering are deployed here and this environment is used for continuous development.

The link to access the **Collab environment** is available [here](#).

Looking to collaborate with us? Click [here](#) to get started with the Collab environment!

Synergy

Synergy is our stable environment where the latest released version of the MOSIP platform and applications are deployed for partners to integrate and test.

The link to access the **Synergy environment** is available [here](#).

Standards & Specifications

Ensuring secure and interoperable digital identity through global standards.

At MOSIP (Modular Open Source Identity Platform), we are committed to building a secure, interoperable, and privacy-centric identity system for nations worldwide. MOSIP adheres to internationally recognized standards in biometric authentication, security, cryptography, privacy, and interoperability to ensure the highest levels of security, efficiency, and compliance.

1. Why Standards Matters?

By following these global standards, MOSIP ensures that our identity platform is:

- Secure** – Protecting citizens' data from cyber threats.
- Privacy-First** – Upholding the highest standards of data protection.
- Scalable & Future-Ready** – Enabling nations to build robust digital identity programs.
- Interoperable** – Seamlessly integrating with digital identity solutions worldwide.

2. Our Adherence to Global Standards

1. Biometric Standards for Secure Identity Verification

To ensure seamless biometric interoperability and security, MOSIP follows:

- **ISO/IEC 19794** – Standardized biometric data formats for fingerprints, iris, and facial recognition.
- **CBEFF (Common Biometric Exchange Formats Framework)** – Facilitates interoperability and efficient biometric data exchange.
- **IEEE P3167 (DRAFT)** – Strengthening the trustworthiness of biometric devices and their captured data while ensuring overall data security.

2. Security & Cryptography Standards for Data Protection

To guarantee robust data security and encryption, MOSIP aligns with:

- **NIST Cybersecurity Framework** – Provides guidelines for IT security evaluation and risk management.
- **RSA, EC, JSE** – Implements industry-standard cryptographic algorithms for secure encryption and data integrity.

3. Open Standards for Seamless Interoperability

To enable effortless integration with national and global identity ecosystems, MOSIP adopts:

- **OAuth 2.0 / OpenID Connect** – Providing secure and scalable authentication mechanisms.
- **REST, OpenAPI Standards** – Ensuring standardized communication across different platforms.
- **JMS (Java Message Service) & WebSub** – Facilitating real-time messaging and event-driven architecture.

MOSIP Standards

This page lists the standards and specifications published by MOSIP which are mentioned below:

169 - QR Code Specifications 1.0.0

CBOR Identity Data in QR Code

Tag: 169 (identity-data)

Data Item: JSON Object

Semantics: Identity Data of a Person in QR-Code

Point of Contact: Resham Chugani (resham@mosip.io)

IANA Registration: [IANA CWT Registry](#) (Search Key: 169)

Version: 1.0.0

1. Introduction

This document specifies a generic data structure and encoding mechanism for storing the Identity Data of a registered person using any ID platform. It also provides a transport encoding mechanism in a machine-readable optical format (QR).

2. Rationale

Once a person is registered in an identity system, their data serves as the foundation for identification, granting them access to social benefits and government services. The level of assurance in this identification process varies depending on the authentication methods employed. Low assurance is achieved through basic identifiers like ID numbers, demographic data, passwords, or PINs. Conversely, higher assurance levels are attained through one-time passwords (OTP) and biometrics.

Among these methods, biometric-based authentication, such as facial authentication, offers the highest level of assurance as it assures the presence of the individual. While this is effective for online systems & personal phones where verification is conducted on a server or a personal device; offline authentication presents challenges in maintaining a similarly high level of assurance. The offline authentication mechanism should work for people with no phone.

For instance, in a cross-border scenario, remote areas often face significant internet connectivity issues. Even when internet access is available, server reliability may be inconsistent. In such circumstances, scanning a standard QR code containing the person's facial photograph and identity information, alongside assurance that the data is country-signed, provides an additional layer of security and affirmation for the countries involved.

Please note: The trust layers required to sync the country's key are beyond the scope of this document. We assume the app scanning the QR code already has the country's key to verify.

To tackle the aforementioned challenge, we propose a **standard CBOR-based QR Code** that involves embedding a low-resolution **image** of the person with a minimal demographic dataset within the QR code. This QR code would be **digitally signed** by the ID authorities (Issuer) and then printed on a physical card. Subsequently, the signed data within the QR code can be utilized for **facial authentication**. This approach also helps enhance **interoperability**.

Note: It is essential to recognize that QR codes have limitations regarding size. We suggest leveraging CBOR Web Token (CWT) with ED25519/ECC keys to generate a smaller signature and more condensed data.

3. Semantics

Claim 169 represents a JSON Object that includes the following ID attributes, as outlined in the table. You can find an illustration of the ID structure contained within Claim 169, as stated below:

3.1 CBOR Map Structure Overview

Note: All the fields here are optional.

Attribute #	Type	Attribute Name	Description
1	tstr	ID	Unique ID to indicate the PII data
2	tstr	Version	Version of the ID data
3	tstr	Language	Language used in other attributes
4	tstr	Full Name	Full name of the person
5	tstr	First Name	First name of the person
6	tstr	Middle Name	Middle name of the person
7	tstr	Last Name	Last name of the person
8	tstr	Date of Birth	Date of birth in YYYYMMDD format
9	tstr	Gender	Gender with the following values: 1 - Male, 2 - Female, 3 - Others
10	tstr	Address	Address of the person - Separator character \n
11	tstr	Email ID	Email id of the person
12	tstr	Phone Number	Contact number of the person
13	tstr	Nationality	Nationality of the person
14	int	Marital Status	Marital status - Can contain the following values: 1 - Unmarried, 2 - Married, 3 - Divorced
15	tstr	Guardian	Name/id of the entity playing the role of a guardian, such as a

16	tstr	Binary Image	mother, father, spouse, Binary image of the person's photograph etc.
17	int	Binary Image Format	Binary image format. Can contain the following values 1 - JPEG, 2 - JPEG2, 3 - AVIF, 4- WEBP
18	[int]	Best Quality Fingers	An unsigned 8-bit number encoding the hand position of the finger. It must be in the range 0-10, where 0 represents "Unknown", 1-5 represents right thumb to little finger, and 6-10 represents left thumb to little finger in sequence
19.. 99	tstr	Reserved	Reserved for future attributes

3.2 CBOR Map Structure Example

```
{  
    "1": "COUN",  
    "6": 1665980929,  
    "8": {  
        "3": "dfd1aa976d8d4575a0fe34b96de2bfad"  
    },  
    "169": {  
        "1": "11110000324013",  
        "2": "1.0",  
        "3": "EN",  
        "4": "Peter M Jhon",  
        "5": "Peter",  
        "6": "M",  
        "7": "Jhon",  
        "8": "19880102",  
        "9": 1,  
        "10": "New City, METRO LINE, PA",  
        "11": "peter@example.com",  
        "12": "+1 234-567",  
        "13": "US",  
        "14": 2,  
        "15": "Jhon Honai",  
        "16": "03CBABDF83D068ACB5DE65B3CDF25E0036F2C546CB90378C587A076E7A"  
        "17": 2,  
        "18": [1, 2],  
    }  
}
```

4. IANA Considerations

4.1 Registry Contents

Claim Name: identity-data

Claim Description: Registering the claim for storing identity data of a person, which could be personally identifiable data (PII) mostly used in Foundational/National ID for cross-border interoperability.

Claim Key: 169

Claim Value Type(s): map

Change Controller: MOSIP

Specification Document(s): [Section 3](#), [Section 4](#)

5. References

[1] C. Bormann, and P. Hoffman. "Concise Binary Object Representation (CBOR)". [RFC 7049](#), October 2013.

[2] Mike Jones, Hannes Tschofenig, Ludwig Seitz "CBOR Web Token (CWT)". [RFC 8392](#), March 2018.

6. Author(s)

Mahammed Taheer (mohd.taheer@gmail.com)

Resham Chugani (resham@mosip.io)

Rounak Nayak (rounak@ooru.io)

Sasikumar G (sasi@duck.com)

Sreenadh S (sreeavtar@gmail.com)

169 - QR Code Specifications

CBOR Identity Data in QR Code

Tag: 169 (identity-data)

Data Item: JSON Object

Semantics: Identity Data of a Person in QR-Code

Point of Contact: Resham Chugani (resham@mosip.io)

IANA Registration: [IANA CWT Registry](#) (Search Key: 169)

Version: 1.1.0

1. Introduction

This document specifies an updated version of the generic data structure and encoding mechanism for storing the Identity Data of a registered person using any ID platform. It also provides a transport encoding mechanism in a machine-readable optical format (QR).

2. Rationale

Once a person is registered in an identity system, their data serves as the foundation for identification, granting them access to social benefits and government services. The level of assurance in this identification process varies depending on the authentication methods employed. Low assurance is achieved through basic identifiers like ID numbers, demographic data, passwords, or PINs. Conversely, higher assurance levels are attained through one-time passwords (OTP) and biometrics.

Among these methods, biometric-based authentication, such as facial authentication, offers the highest level of assurance as it assures the presence of the individual. While this is effective for online systems & personal phones where verification is conducted on a server or a personal device; offline authentication presents challenges in maintaining a similarly high level of assurance. The offline authentication mechanism should work for people with no phone.

For instance, in a cross-border scenario remote areas often face significant internet connectivity issues. Even when internet access is available, server reliability may be inconsistent. In such circumstances, scanning a QR code containing the person's facial photograph and identity information, alongside assurance that the data is country-signed, provides an additional layer of security and affirmation for the countries involved.

Please note: The trust layers required to sync the country's key are beyond the scope of this document. We assume the app scanning the QR code already has the country's key to verify.

To tackle the challenge above, we propose a standard CBOR-based QR Code that involves embedding a low-resolution image of the person with a minimal demographic dataset within the QR code. This QR code would be digitally signed by the ID authorities (Issuer) and then printed on a physical card. Subsequently, the signed data within the QR code can be utilized for facial authentication. However, it's essential to recognize that QR codes have limitations regarding size. We suggest leveraging CBOR Web Token (CWT) with ED25519/ECC keys to generate a smaller signature and more condensed data.

Claim 169 represents a JSON Object that includes the below table as ID attributes. You can find an illustration of the ID structure contained within Claim 169, where:

3. Semantics

3.1 CBOR Map Structure Overview

All the fields here are OPTIONAL.

Attribute	Type	Attribute Name	Description
1	tstr	ID	Unique ID to indicate the PII data
2	tstr	Version	Version of the ID data
3	tstr	Language	Language used in other attributes
4	tstr	Full Name	Full name of the person
5	tstr	First Name	First name of the person
6	tstr	Middle Name	Middle name of the person
7	tstr	Last Name	Last name of the person
8	tstr	Date of Birth	Date of birth in YYYYMMDD format
9	int	Gender	Gender with the following values 1 - Male, 2 - Female, 3 - Others
10	tstr	Address	Address of the person, separator character \n
11	tstr	Email ID	Email id of the person
12	tstr	Phone Number	Contact number of the

			person
13	tstr	Nationality	Nationality of the person
14	int	Marital Status	Marital status - Can contain the following values 1 - Unmarried, 2 - Married, 3 - Divorced
15	tstr	Guardian	Name/id of the entity playing the role of a guardian, such as a mother, father, spouse, sister, legal guardian etc.
16	tstr	Binary Image	Binary image of the person's photograph
17	int	Binary Image Format	Binary image format. Can contain the following values 1 - JPEG, 2 - JPEG2, 3 - AVIF, 4 - WEBP
			An unsigned 8-bit number encoding the hand position of the finger. It must be in the range 0-

18	[int]	Best Quality Fingers	10, where 0 represents "Unknown", 1-5 represents right thumb to little finger, and 6-10 represents left thumb to little finger in sequence
19.. 49		Reserved	Reserved for future attributes
50.. 74		Reserved	Reserved for Person's Biometrics Data attributes
50	[Biometrics]	Right Thumb	Person's Right Thumb biometrics
51	[Biometrics]	Right Pointer Finger	Person's Right Pointer Finger biometrics
52	[Biometrics]	Right Middle Finger	Person's Right Middle Finger biometrics
53	[Biometrics]	Right Ring Finger	Person's Right Ring Finger biometrics
54	[Biometrics]	Right Little Finger	Person's Right Little Finger biometrics

55	[Biometrics]	Left Thumb	Person's Left Thumb biometrics
56	[Biometrics]	Left Pointer Finger	Person's Left Pointer Finger biometrics
57	[Biometrics]	Left Middle Finger	Person's Left Middle Finger biometrics
58	[Biometrics]	Left Ring Finger	Person's Left Ring Finger biometrics
59	[Biometrics]	Left Little Finger	Person's Left Little Finger biometrics
60	[Biometrics]	Right Iris	Person's Right Iris biometrics
61	[Biometrics]	Left Iris	Person's Left Iris biometrics
62	[Biometrics]	Face	Person's Face biometrics
63	[Biometrics]	Right Palm Print	Person's Right Palm Print biometrics
64	[Biometrics]	Left Palm Print	Person's Left Palm Print biometrics
65	[Biometrics]	Voice	Person's Voice biometrics
66.. 74		Reserved	Reserved for future for Person's Biometrics

			Data attributes
75 .. 99		Reserved	Reserved for future

Biometrics

Attribute	Type	Attribute Name	Description
0	bstr	Data	Biometrics binary data
1	int	Data format	Optional biometrics data format
2	int	Data sub format	Optional biometrics data sub format
3	tstr	Data issuer	Optional biometric data issuer

Data formats

Data format	Description
0	Image
1	Template
2	Sound
3	Bio hash

Data sub formats

Image

Subformat	Description

0	PNG
1	JPEG
2	JPEG2000
3	AVIF
4	WEBP
5	TIFF
6	WSQ
...	

Template

Subformat	Description
0	Fingerprint Template ANSI 378
1	Fingerprint Template ISO 19794-2
2	Fingerprint Template NIST
100..200	Vendor specific

Sound

Subformat	Description
0	WAV
1	MP3

3.2 CBOR Map Structure Example

```
1: COUN # iss
6: 1665980929 # iat
8: # cnf
3: dfd1aa976d8d4575a0fe34b96de2bfad # kid
169: # identity-data
1: "11110000324013" # ID
2: "1.0" # Version
3: EN # Language
4: Peter M Jhon # Full name
5: Peter # First name
6: M # Middle name
7: Jhon # Last name
8: "19880102" # Date of birth
9: 1 # Gender: Male
10: New City, METRO LINE, PA # Address
11: peter@example.com # Email ID
12: "+1 234-567" # Phone number
13: US # Nationality
14: 2 # Marital status: Married
15: Jhon Honai # Guardian
16: 03CBABDF83D068ACB5DE65B3CDF25E0036F2C54(...)E54D23D8EC7DC9BB9F69FD7I
17: 2 # Binary image format: JPEG
18: [1, 2] # Best quality fingers
50: # Right Thumb Biometrics
    # Right Thumb image
    - 0: 03CBA(...)0378C58 # Data
        1: 0 # Image
        2: 1 # JPEG
    # Right Thumb template
    - 0: 03CBA(...)0378C58 # Data
        1: 1 # Template
        2: 100 # Vendor specific
        3: VendorA # Biometric data issuer
51: # Right Pointer Finger Biometrics
    # Right Pointer Finger image
    - 0: 36F2C546(...)CB90378C58 # Data
        1: 1 # Image
        2: 6 # WSQ
        3: VendorA # Biometric data issuer
    # Right Pointer Finger template
    - 0: 36F2C546(...)CB90378C58 # Data
        1: 1 # Template
        2: 1 # Fingerprint Template ISO 19794-2
        3: VendorA # Biometric data issuer
58: # Left Ring Finger Biometrics
    # Left Ring Finger image
    - 0: 36F2C546(...)CB90378C58 # Data
        1: 1 # Image
```

```
2: 6 # WSQ
3: VendorA # Biometric data issuer
# Left Ring Finger template
- 0: 36F2C546(...)CB90378C58 # Data
  1: 1 # Template
  2: 1 # Fingerprint Template ISO 19794-2
  3: VendorA # Biometric data issuer
60: # Right Iris Biometrics
# Right Iris image
- 0: 36F2C546(...)CB90378C58 # Data
  1: 1 # Image
  2: 6 # WSQ
  3: VendorX # Biometric data issuer
# Right Iris image
- 0: 36F2C546(...)CB90378C58 # Data
  1: 1 # Image
  2: 6 # WSQ
  3: VendorY # Biometric data issuer
61: # Left Iris Biometrics
# Left Iris template
- 0: 36F2C546(...)CB90378C58 # Data
  1: 1 # Template
  2: 100 # Vendor specific
  3: VendorX # Biometric data issuer
# Left Iris image
- 0: 36F2C546(...)CB90378C58 # Data
  1: 1 # Template
  2: 100 # Vendor specific
  3: VendorY # Biometric data issuer
65: # Voice Biometrics
# Voice sound
- 0: 03CBA(...)0378C58 # Data
  1: 2 # Sound
  2: 1 # MP3
# Voice template
- 0: 03CBA(...)0378C58 # Data
  1: 1 # Template
  2: 100 # Vendor specific
  3: VendorZ # Biometric data issuer
```

4. Security Considerations

TODO:

1. Current map structure is in plain text and its not the recommended way to handle privacy. Adoption of SD-JWT or equivalent can be considered.
2. CWT MUST be signed, create a COSE_Sign/COSE_Sign1 object using the Message as the COSE_Sign/COSE_Sign1 Payload; all steps specified in [RFC8152](#) for creating a COSE_Sign/COSE_Sign1 object MUST be followed.
3. If the CWT is a COSE_Encrypt/COSE_Encrypt0 object, create a COSE_Encrypt/COSE_Encrypt0 using the Message as the plaintext for the COSE_Encrypt/COSE_Encrypt0 object; all steps specified in [RFC8152](#) for creating a COSE_Encrypt/COSE_Encrypt0 object MUST be followed.
4. To verify the claims the CWT is a COSE_Sign/COSE_Sign1, follow the steps specified in Section 4 of [RFC8152](#) ("Signing Objects") for validating a COSE_Sign/COSE_Sign1 object. Let the Message be the COSE_Sign/COSE_Sign1 payload. Once signature is valid we SHOULD validate the public key against a

preconfigured key. In case encrypted Else, if the CWT is a COSE_Encrypt/COSE_Encrypt0 object, follow the steps specified in Section 5 of [RFC8152] ("Encryption Objects") for validating a COSE_Encrypt/COSE_Encrypt0 object. Let the Message be the resulting plaintext.

The security of the CWT relies upon on the protections offered by COSE. Unless the claims in a CWT are protected, an adversary can modify, add, or remove claims.

Since the claims conveyed in a CWT is used to make identity claim decisions, it is not only important to protect the CWT but also to ensure that the recipient can authenticate the party that assembled the claims and created the CWT. Without trust of the recipient in the party that created the CWT, no sensible identity verification can be made. Furthermore, the creator of the CWT needs to carefully evaluate each claim value prior to including it in the CWT so that the recipient can be assured of the validity of the information provided.

Syntactically, the signing and encryption operations for Nested CWTs may be applied in any order; however, if encryption is necessary, producers normally should sign the message and then encrypt the result (thus encrypting the signature). This prevents attacks in which the signature is stripped, leaving just an encrypted message, as well as providing privacy for the signer. Furthermore, signatures over encrypted text are not considered valid in many jurisdictions.

5. IANA Considerations:

IANA is requested to register the revised specifications of claim 169 in "CBOR Web Token (CWT) Claims" registry [IANA CWT Claims](#).

5.1 Registry Content

Claim Name: identity-data

Claim Description: Registering the claim for storing identity data of a person, which could be Personally Identifiable Data (PII) mostly used in Foundational/National ID for cross-border interoperability.

Claim Key: 169

Claim Value Type(s): map

Change Controller: MOSIP

Specification Document(s): [Section 3](#), [Section 4](#)

6. Acknowledgments

This work is the result of the dedicated efforts of contributors who recognize the critical importance of interoperability and a consistent QR code specification. The revised version has been shaped significantly by the input of our working group committee, comprising members from the following organizations: GetGroup, PWC and Tech 5.

We extend our gratitude to the committee members for their invaluable time and insights throughout the evaluation phase.

6.1 Working Group Committee Members:

GetGroup: Aiman Tarek

PWC: Chaitanya Giri

Tech 5: Bejoy Ak, Nelson Branco, Rahul Parthe

MOSIP: Harini Sampathkumar, Janardhan BS, Mahammed Taheer, Ramesh Narayanan, Resham Chugani, Reeba Thomas, Sanchi Singh, Sasikumar Ganesan, Sreenadh S, Swati Goel, Vishwanath V

7. Authors

Mahammed Taheer (mohd.taheer@gmail.com)

Resham Chugani (resham@mosip.io)

Rounak Nayak (rounak@ooru.io)

Sasikumar G (sasi@duck.com)

Sreenadh S (sreeavtar@gmail.com)



Empowering Trusted Digital Identity with Secure and Seamless Credential Verification.

About Inji

In today's fast-paced, interconnected world, ensuring seamless access to essential services—such as healthcare, financial inclusion, global mobility, and social support has never been more critical. The need for **trusted identity authentication** and **secure data exchange** is at the heart of accessing these services.

To address this, Inji offers a transformative solution – enabling the secure issuance, digitalization, storage, exchange, and seamless verification of trusted data as verifiable credentials, through a comprehensive set of tools.

Inji, meaning "**knowing**" or "**recognizance**" in Korean, is evolving into a comprehensive digital credential stack with a strong focus on user empowerment. Inji simplifies the management and verification of credentials by providing secure solutions that work across multiple interfaces. It aims to streamline creating, sharing, and verifying all types of digital and physical credentials.



[Explore more about Inji.](#)

eSignet

Enabling Secure, Inclusive, and Seamless Digital Identity Verification.

In today's digital-first world, most services are transitioning online, making a secure and trusted digital identity more important than ever. A secure and trusted digital identity is crucial to facilitate personalized access to these online services.

eSignet strives to provide a user-friendly and effective method for individuals to authenticate themselves and utilize online services while also having the option to share their profile information. Moreover, eSignet supports multiple modes of identity verification to ensure inclusivity and broaden access, thereby reducing potential digital barriers.

Additionally, eSignet offers a seamless and straightforward solution for incorporating an existing trusted identity database into the digital realm. By enabling digital identities and providing identity verification and service access, eSignet delivers a sophisticated and user-friendly experience.



[Explore more about eSignet.](#)

ID Lifecycle Management

Managing identities seamlessly.

Overview

Identity Lifecycle Management refers to the **process of issuing and managing user identities** in a given system. An individual can visit a registration centre to get a new ID or update their ID details. A registration officer would typically capture the individual's demographic (name, date of birth, gender, etc.) and biometric (face, iris, and fingerprint image) details.

The lifecycle of an identity involves multiple events, categorized into different stages, each managed within specific system **modules**. The following sections outline these **key lifecycle events**, detailing their roles and functions categorized under relevant modules.

Head below to **explore the various applications and services** that support Identity Lifecycle Management.

Identity
Issuance

Identity
Verification

Identity
Management

Support
Systems

Identity Issuance

Pre-registration

Overview

Introduction

Pre-registration module enables a resident to:

- enter demographic data and upload supporting documents,
- book an appointment for one or many users for registration by choosing a suitable registration center and a convenient time slot,
- receive appointment notifications,
- reschedule and cancel appointments.

Once the resident completes the above process, their data is downloaded at the respective registration centers before their appointment, thus, saving enrollment time. The module supports multiple languages.

MOSIP provides backend APIs for this module along with a reference implementation of [pre-registration portal](#).

Pre-registration process



Create an application

- User provides consent
- The user provides demographic information
- User uploads supporting documents (Proof of Address, Birth certificate, etc.)
- A pre-registration request ID known as [AID](#) is generated and provided to the user.

Note: The AID was formerly called PRID in the pre-registration context.

Book an appointment

- The user selects a registration center based on postal code, geo-location, etc.
- The available slots are displayed
- An option to cancel and re-book the appointment is made available

Receiving acknowledgement notifications

- An acknowledgment is sent via email/SMS
- The user can print the acknowledgment containing AID and QR code.
- This QR code can be scanned at the in-person registration centers.

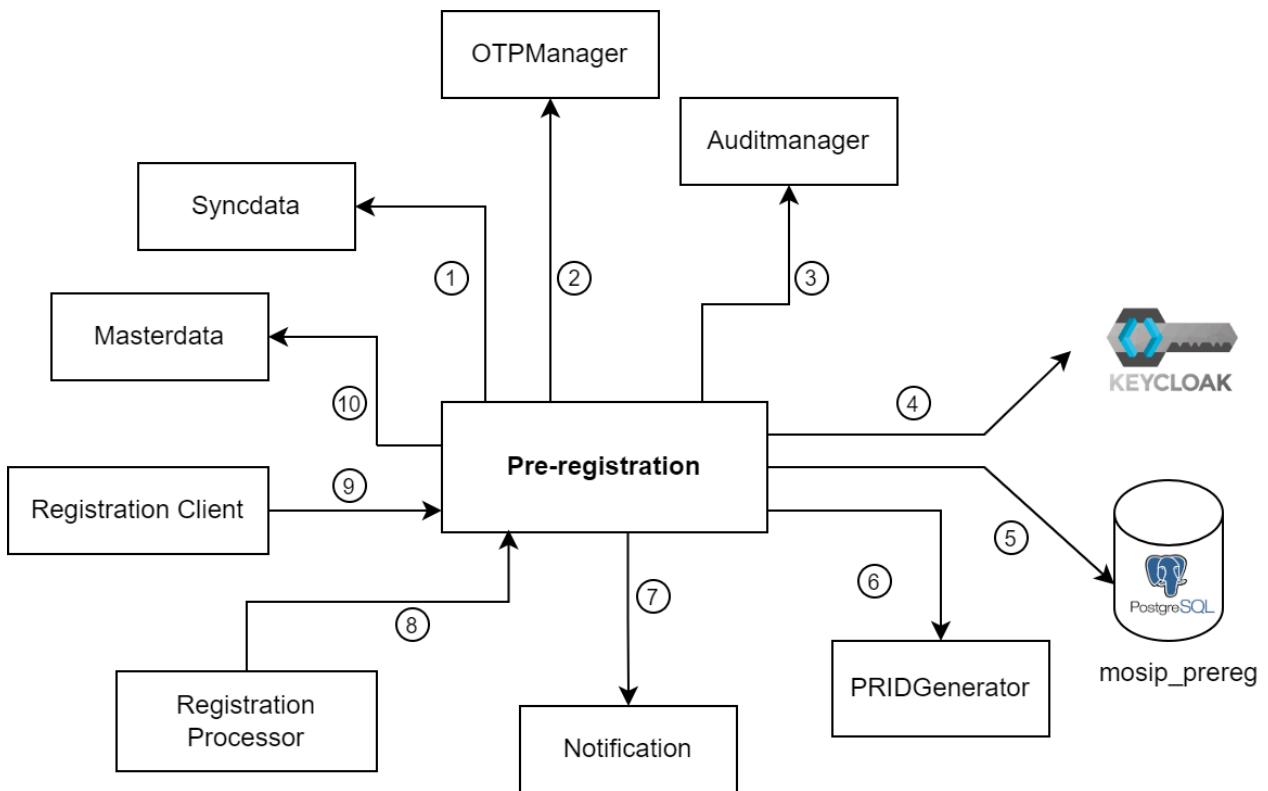
Download of pre-registration data at registration centers

- The user provides the AID/ QR code at the registration center.
- The registration form gets pre-filled with the pre-registration data.

Pre-registration module

The relationship of the pre-registration module with other services is explained here.

(i) NOTE: The numbers do not signify a sequence of operations or control flow.



1. Fetch [ID Schema](#) details with the help of Syncdata service.
2. Fetch a new OTP for the user on the login page.
3. Log all events.
4. Pre-Registration interacts with Keycloak via [kernel-auth-adapater](#). The Pre-Reg module communicates with endpoints of other MOSIP modules. However, to access these endpoints, a token is required. This token is obtained from Keycloak.
5. The database used by pre-reg.
6. Generate a new AID for the application.
7. Send OTP in the email/SMS to the user.
8. Registration Processor uses reverse sync to mark the pre-reg application as consumed.

9. Registration clients use the [Datasync service](#) to get the pre-reg application details for a given registration center, booking date, and AID.
10. Request data from the MasterData service to get data for dropdowns, locations, consent forms etc.

Services

Pre-registration module consists of the following services:

- [Application](#)
- [Booking](#)
- [Batchjob](#)
- [Datasync](#)
- [Captcha](#)

For more details, refer to the [Pre-registration repo](#).

Pre-registration portal

MOSIP provides a **reference** implementation of the pre-registration portal that may be customized as per country needs. The sample implementation is available at the [reference implementation repository](#). For getting started with the pre-registration, refer to the [Pre-registration user guide](#)

Build and deploy

To access the build and read through the deployment instructions, refer to the [Pre-registration repo](#).

Configurations

For details related to Pre-registration configurations, refer to [Pre-registration configuration](#).

Developer Guide

To know more about the developer setup, read the [Pre-registration Developers Guide](#).

API

Refer to [API Documentation](#).

Source code

[Github repo.](#)

Develop

Build, integrate, and enhance solutions.

Developers Guide

Overview

The documentation here will guide you through the pre-requisites required for [Pre-registration](#) developer setup.

Software setup

Below are the list of tools required in Pre-registration

1. JDK 11
2. Any IDE (Eclipse, IntelliJ IDEA)
3. Apache Maven (zip folder)
4. pgAdmin
5. Postman
6. Git
7. [lombok.jar](#) (file)
8. MOSIP Pre-registration specific JARs: The version will depend on which Pre Registration branch you have cloned. If it is "release-1.2.0.1" then you can download 1.2.0.1.B1 or 1.2.0.1.B2 version of below jars whichever is available.
 - [kernel-auth-adapter](#)
 - [kernel-transliteration-icu4j](#)
 - [kernel-ref-idobjectvalidator](#)
 - [kernel-virusscanner-clamav](#)
9. [settings.xml](#)
10. Notepad++ (optional)

Follow the steps below to setup Pre-registration on your local system

1. Fork the MOSIP [Pre-registration repository](#) from Github MOSIP repository to your GitHub account.
2. Clone the forked repository into your local machine.

- `git clone https://github.com/${urgithubaccname}/pre-registration.git`
- `git remote add upstream https://github.com/mosip/pre-registration.git`
- `git remote set-url --push upstream no_push`
- `git remote -v`
- `git checkout -b my-release-1.2.0.1`
- `git fetch upstream`
- `git rebase upstream/release-1.2.0.1`

3. Inside `settings.xml` change local repository directory to your directory name where `.m2 folder` is located. E.g.:

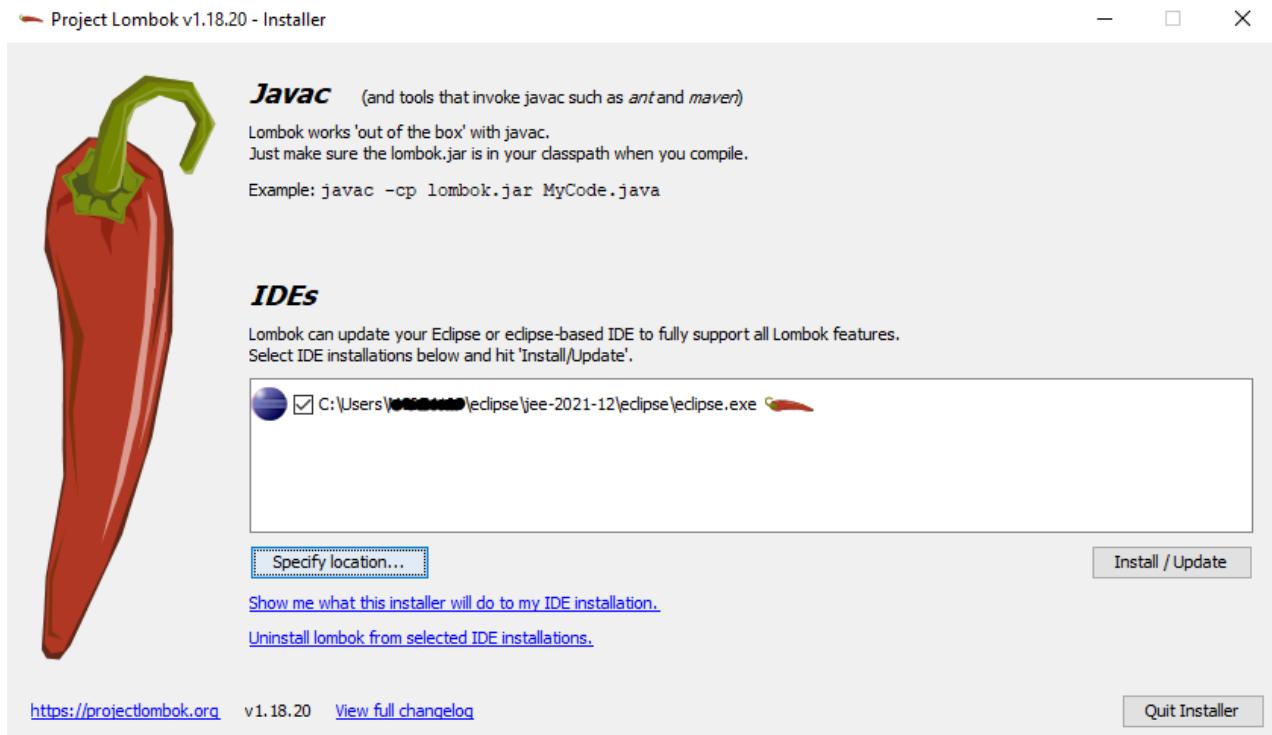
```
<localRepository>C:/Users/username/.m2/repository</localRepository>
```

4. Add `settings.xml` inside `.m2 folder` (Maven Folder). E.g.:

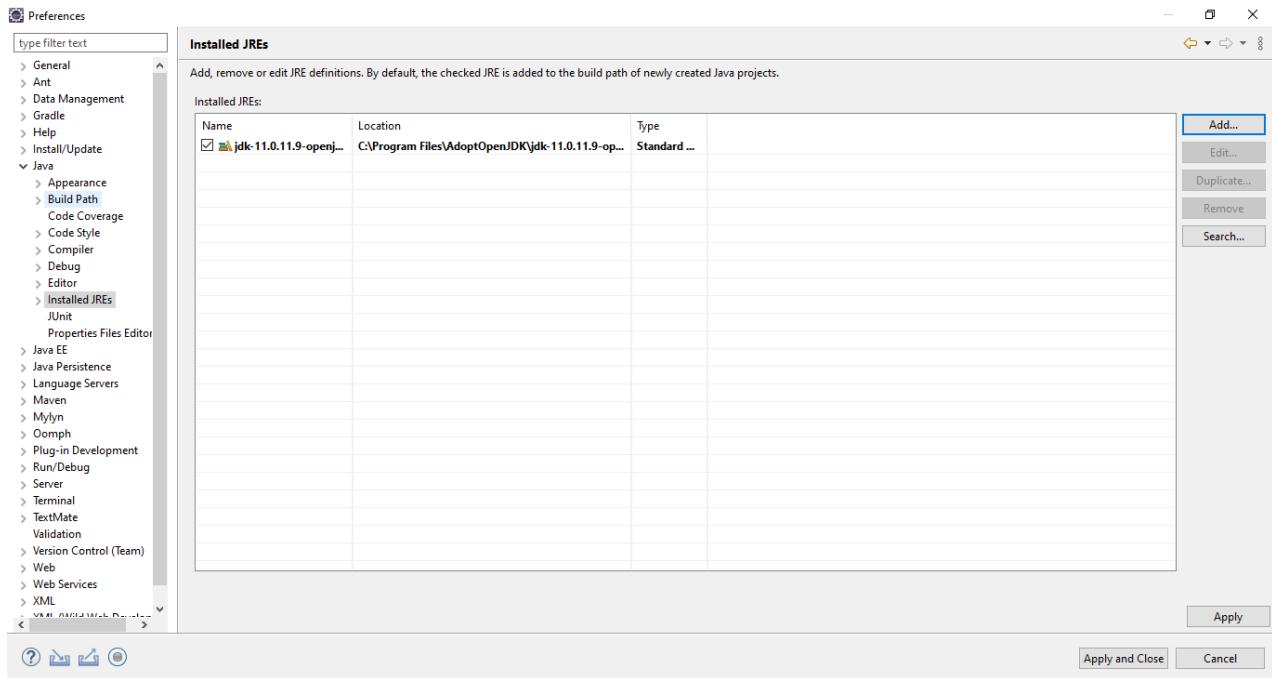
```
C:\Users\username\.m2
```

5. Import the project in Eclipse IDE and it starts updating Maven projects configuration, refreshing workspaces, project starts building (downloading sources, javadoc).

6. Add downloaded `lombok.jar` to project, click on downloaded JAR and install specifying Eclipse IDE(eclipse.exe) location.

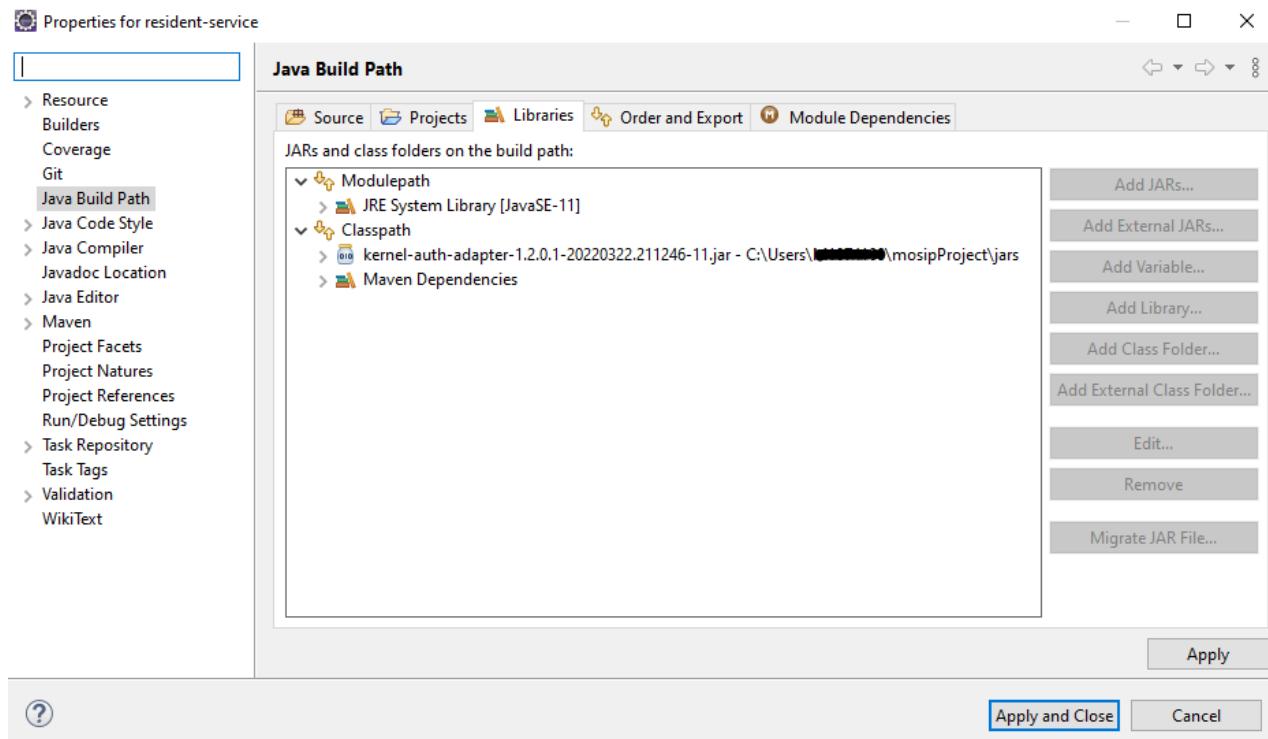


7. Configure the JDK (Standard VM) with your Eclipse by traversing through [Preferences → Java → Installed JREs](#).



8. Add MOSIP Pre-registration specific JARs from [Maven central](#):

- Adding JARs to Build Path: Right click on service → Build Path → Configure Build Path → click on Classpath → Add External JARs → Add required JARs → Apply and close.



9. Add `auth-adapter`, `transliteration`, `ref-idobjectvalidator`, `virusscanner`, `lombok` JARs to `pre-registration-application-service`, `pre-registration-datasync-service` classpath.
10. Add `auth-adapter`, `lombok` JARs to `pre-registration-core`, `pre-registration-batchjob`, `pre-registration-captcha-service`, `pre-registration-booking-service` classpath.
11. Run `mvn clean install -Dgpg.skip=true` command to build locally and to execute test cases.
12. Update Maven dependencies: Maven syncs the Eclipse project settings with that of the pom. It downloads dependencies required for the project.
13. Build and run the Project.
14. To run the pre-registration-application-service locally ***without config server***, update values in application.properties and bootstrap.properties:

- `spring.cloud.config.uri=https://localhost:8080`
- `spring.cloud.config.label=develop`
- `spring.cloud.config.name=pre-registration`
- `spring.application.name=pre-registration-application-service`
- `spring.profiles.active=default` ***Point below urls to a valid env which has MOSIP setup:***

- `mosip.base.url=https://yourenvurl`
- `auth-token-`
`generator.rest.issuerUrl:https://iam.yourenvurl/auth/realms/mosip`
- `javax.persistence.jdbc.password: XXXXXX`
- `javax.persistence.jdbc.url=jdbc:postgresql://yourenvurl:5432/mosip_prer`
`eg`
- `mosip.batch.token.authmanager.password: XXXXXX`
- `mosip.iam.adapter.appid=prereg`
- `mosip.iam.adapter.clientsecret=XXXXXX`
- `auth.server.admin.issuer.uri=https://iam.yourenvurl/auth/realms/`

Developer setup for MOSIP Pre-registration UI

1. Fork the [Pre-registration UI repo](#) to your GitHub account.
2. Clone the forked repository into your local machine.

- `git clone https://github.com/${urgithubaccname}/pre-registration-ui.git`
- `git remote add upstream https://github.com/mosip/pre-registration-ui.git`
- `git remote set-url --push upstream no_push`
- `git remote -v`
- `git checkout -b my-release-1.2.0.1`
- `git fetch upstream`
- `git rebase upstream/release-1.2.0.1`

3. Install all dependencies with `npm install`.
4. Install Angular JS `npm install -g @angular/cli`.
5. Start the Angular JS server `ng serve`.
6. Open `http://localhost:4200` to access the application.

- You will face CORS issue since API Services are hosted on `https://{{env}}`.

Using the Angular CLI proxy solution to get around CORS issue

- Update the API services `BASE_URL` in `config.json`:

- `config.json` is found inside assets directory.
- E.g.: `C:\MOSIP\pre-registration-ui\pre-registration-ui\src\assets\config.json`

```
{
  "BASE_URL": "https://localhost:4200/proxyapi/",
  "PRE_REG_URL": "preregistration/v1/"
}
```

- Create a new file named `proxy.conf.json`:

Location should be in `C:\MOSIP\pre-registration-ui\pre-registration-ui\proxy.conf.json` project folder.

```
{
  "/proxyapi": {
    "target": "https://{{env}}/",
    "secure": true,
    "changeOrigin": true,
    "pathRewrite": {
      "^/proxyapi": ""
    }
  }
}
```

- Start the server by executing `ng serve --proxy-config proxy.conf.json --ssl true`.
- Open the browser, load the app with `https://localhost:4200`.

Pre-registration API

- For API documentation, refer [here](#).
- The APIs can be tested with the help of **Swagger-UI** and **Postman**.
- Swagger is an interface description language for describing restful APIs expressed using JSON. You can access Swagger-UI of pre-registration here:

- Pre-registration Application service :

```
https://{{env}}/preregistration/v1/application-service/swagger-ui.html
```

- Pre-registration Datasync Service :

```
https://{{env}}/preregistration/v1/sync/datasync-service/swagger-ui.html
```

- Pre-registration Captcha service :

```
https://{{env}}/preregistration/v1/captcha/swagger-ui.html
```

- Pre-registration Booking service :

```
https://{{env}}/preregistration/v1/appointment/booking-service/swagger-ui.html
```

UI Specifications

Overview

UI specs of Pre-Registration module are used to configure the form fields in the Demographic Details and Document Upload functionality. UI specs are saved as a JSON file with a list of fields. Each field has a set of attributes/ properties that can be configured which affects the look and feel along with the functionality of the field. Below is the list of all the properties available for each field in the UI specs:

Property Name	Details	Sample Value
<code>id</code>	The id property is the unique id provided to a field to uniquely identify it. The id can be alpha-numeric without any spaces between them.	<code>"id": "zone"</code>
<code>description</code>	This is a non-mandatory property used to describe the field.	<code>"description": "zone"</code>
<code>labelName</code>	This property defines label name for the field. This property has sub-attributes as the language code (eng, fra, ara) to store data in different languages based on the country's configuration.	<code>"labelName": { "eng": "Zone", "ara": "منطقة", "fra": "Zone"} }</code>
	This property defines the kind of UI component to be used to capture data in UI. Currently the values that can be used are: <ul style="list-style-type: none">• textbox (creates multiple textboxes for each field to capture input in all the languages configured for	

<code>controlType</code>	<p>the setup)</p> <ul style="list-style-type: none"> • dropdown • fileupload • date (creates a date picker) • ageDate (creates a date picker along with number toggle to provide age directly) • checkbox (creates a toggle checkbox for the field which can be checked or unchecked) • button (creates dropdown options as buttons, which user can select easily)
<code>inputRequired</code>	<p>This property decides if the field is to be displayed in the UI form or not. It is useful for some internal fields which do not need any input from the user.</p>
<code>required</code>	<p>This is a mandatory property which decides if the field is a required form field or not. If true, user must provide some value for the field.</p>
<code>type</code>	<p>This property defines the data type of the value corresponding to this field. The data types supported are "number", "string" and "simpleType". The type "simpleType" means that language specific value will be saved along with the language code.</p>
	<p>This property is relevant when control type is "dropdown" or "button". It defines if the field is of type</p>

fieldType	<p>"default" or "dynamic". If it is "dynamic" then all the options for the dropdown are populated from the "master.dynamic_field" table otherwise they are populated from corresponding table example "master.gender"</p>
subType	<p>This is relevant for 2 cases:</p> <ol style="list-style-type: none"> When control type is "dropdown"/ "button" and the type is "dynamic" then "subtype" can be used to populate the options for the field with the data available in "master.dynamic_field" table. When the control type is "fileupload", then the property "subtype" is used to map the field to a "code" in the "master.doc_category" table.
	<ul style="list-style-type: none"> * This property enables us to add the list of language specific validators for the field. * Each validator can have the below fields, "langCode", "type", "validator", "arguments", "errorMessageCode" * The "type" defines the validation engine type. * The "validator" gives the pattern/ methodName/ scriptName/ expression * The "arguments" array to is used to hold parameter or <pre>"validators": ["langCode": "eng", "type": "regex", "validator": "^(?=.=.{0,50}\$).*", "arguments": [], "errorMessageCode": "</pre>

<code>validators</code>	<p>dependent field ids required for validation</p> <p>* The "errorMessageCode" can be given to add custom error message which will be shown to the user when the validation fails. The error message corresponding to this code will be picked from language specific i18n translation files. In case "errorMessageCode" is not given then generic error message will be displayed to the user when validation fails.</p> <p>Currently, regex is supported by MOSIP. If "langCode" is not added then same "validator" is used for all languages of the field.</p>	<pre>"UI_1000" },{ "langCode": "ara", "type": "regex", "validator": "^[A-Z]+\$", "arguments": [] },{ "langCode": "fra", "type": "regex", "validator": "^[A-Z]+\$", "arguments": [] }]</pre>
<code>transliteration</code>	If enabled, then value entered by user in one language is transliterated into other.	
<code>locationHierarchyLevel</code>	<p>This attribute is mandatory for the location dropdown fields.</p> <p>The value will be as per corresponding location hierarchy level from the master.loc_hierarchy_list table.</p>	<pre>{ "id":"region", "controlType":"dropdown", "fieldType":"default", "type":"simpleType", "parentLocCode":"MOR", "locationHierarchyLevel":1 ..}</pre>
	<p>This attribute is to be used only for location dropdown fields and it is optional.</p> <p>The corresponding location dropdown will be pre populated in UI based on the value in "parentLocCode".</p>	<pre>{ "id":"region",</pre>

<p><code>parentLocCode</code></p>	<p>If this attribute is NOT mentioned in UI specs, then the dropdown will be populated based on selection in its parent dropdown, as before. For the first dropdown, in case this attribute is not mentioned in UI specs then the value from "mosip.country.code" configuration will be used for backward compatibility.</p>	<pre>"controlType": "dropdown", "fieldType": "default", "type": "simpleType", "parentLocCode": "MOR", "locationHierarchyLevel": 1 ..}</pre>
<p><code>visibleCondition</code></p>	<p>The property is used to define an expression using json-rules-engine which will decide the visibility condition for the form field. Refer https://www.npmjs.com/package/json-rules-engine to get the syntax for the conditions.</p>	<pre>"visibleCondition": { "all": [{"fact": "identity", "operator": "lessThanInclusive", "value": 10, "path": "\$.age" }]} This condition will make the field visible only when the "age" field value <= 10.</pre>
<p><code>requiredCondition</code></p>	<p>The property is used to define an expression using json-rules-engine which will decide the required condition for the form field. Refer https://www.npmjs.com/package/json-rules-engine to get the syntax for the conditions.</p>	<pre>"requiredCondition": { "all": [{"fact": "identity", "operator": "equal", "value": "MLE", "path": "\$.gender.0.value" },{ "fact": "identity", "operator": "equal", "value": "102", "path": "\$.maritalStatus.0.value" }]} This condition will make the field required only when the "gender" field value = 'MLE' and "maritalStatus" field value is 102" .</pre>
<p>* This property is used to group the fields on the</p>		

<p><code>alignmentGroup</code></p>	<p>screen.</p> <ul style="list-style-type: none"> * If it is skipped, then all the fields will appear in same sequence (horizontally layout) as they appear in UI specs. * If you want the first and fifth field to be in same row in the screen, you can add this attribute with same group name. * The UI is responsive so it will accommodate as many fields in one row as they will fit comfortably. 	
<p><code>containerStyle</code></p>	<p>This is used to optionally apply some CSS styles to the UI field container.</p>	<pre>"containerStyle": { "width": "600px", "margin-right": "10px" }</pre>
<p><code>headerStyle</code></p>	<p>This is used to optionally apply some CSS styles to the UI header of the field container.</p>	<pre>"headerStyle": { "width": "600px", "margin-right": "10px" }</pre>
<p><code>changeAction</code></p>	<p>This property can be added to define interaction between the 2 or more form fields when user changes value in one of them. Currently the "changeAction" supported are "copy&disable" and "copyto" ONLY.</p>	<p>1. "changeAction": "copyto:permanentZipcode, presentZipcode,addressCopy"</p> <p>When the checkbox "addressCopy" is checked the value of the field "permanentZipcode" will be copied to "presentZipcode".</p>
		<p>2.</p> <p>"changeAction":"copy&disable": permanentCountry=present Country, permanentAddressLine1=pr esentAddressLine1"</p> <p>When the checkbox "addressCopy" is checked</p>

the value of the field
“permanentCountry” will be
copied to “presentCountry”
and the field
“presentCountry” will be
disabled.
Same with
“permanentAddressLine1”
and “presentAddressLine1”.

Test

End User Guide

Overview

This guide helps in understanding the pre-registration sample UI implementation. The pre-registration portal can be used in **self-service** as well as in **assisted** mode.

Self-service mode

In this mode, residents can pre-register themselves by accessing the pre-registration portal. They can login with their email address or phone number and fill up the demographic form, upload relevant documents to book an appointment for themselves and their family/friends. Finally, they would receive an acknowledgment along with a pre-registration ID that can be used at the registration center.

Assisted mode

When used in an assisted mode, the operator could be handling the portal and helping other residents in filling up the details, and creating an application on their behalf. The languages that the operator and the resident understand, may or may not be the same. If we consider a country with linguistic diversity, the possibilities increase. In such cases, the operator might log in with a language that they are familiar with, and also select a language (data capture language) familiar to the resident for filling up the demographic form and other details.

1 2 LTS Pre Registration UI Demo -07-02-2022



Pre-registration process

The key steps in this process are:

- Login/create a user account
- Create an application
- Book an appointment
- Receive appointment acknowledgement

To create an application, the resident/operator can follow the steps below:

Login/create a user account



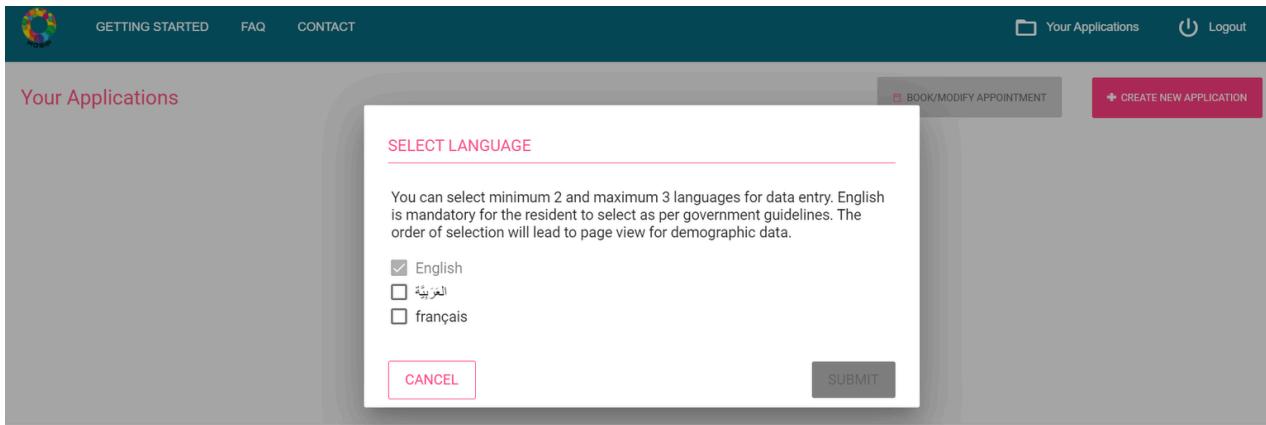
Pre-registration login page

1. Open the browser and visit the pre-registration portal.
2. Select the **language** of your preference from the dropdown.
3. Enter your valid email address or phone number in the text box.
4. Select the Captcha field.
5. Click **Send OTP** to receive a One Time Password (OTP) on your provided email address or mobile number.
6. Enter the OTP and click **Verify**.

(i) Note: If you have not received the One-Time Password (OTP), please click on **Send** to request another OTP. Enter the newly received OTP to proceed. Once your OTP has been successfully verified, you will be able to create, view, or update your pre-registration application.

Create an application

Step 1: Select the language for providing data



1. Once the OTP is verified, you will see a pop up for selecting the languages for data entry.
2. Select the languages and click **Submit**.

(i) **Note:**

- This choice will be available only if the ID issuer has configured the usage of optional languages.
- Countries will have multiple languages some of which will be *mandatory* while others could be *optional*(*regional languages*). MOSIP provides a configuration using which a country can mandate the capture of demographic data in the required number of languages (a combination of mandatory and optional).

Step 2: Provide consent

Demographic Data

Title : *

Title

Date Of Birth : *

DD/MM/YYYY

TERMS AND CONDITIONS

Terms and Conditions

Your Agreement

I understand that the data collected about me during pre-registration by the said authority includes my - • Name • Date of birth • Gender • Address • Contact details • Documents I also understand that this information will be stored and processed for the purpose of verifying my identity in order to access various services, or to comply with a legal obligation. I give my consent for the collection of this data for this purpose.

الأحكام والشروط

موافق
وأنا أفهم أن البيانات التي تم جمعها عن خلال التسجيل المسبق من قبل السلطة المذكورة تشمل بدني - • الاسم • تاريخ الميلاد • نوع الجنس • العنوان • تفاصيل الاتصال • الوثائق كما أفهم أنه سيتم تخزين هذه المعلومات ومعالجتها بغرض التحقق من هويتي من أجل الوصول إلى خدمات مختلفة، أو الامتناع لالتزام قانوني، وأوافق على جمع هذه البيانات لهذا الغرض.

Click to accept the terms and conditions / انقر لقبول الشروط والأحكام

CANCEL

ACCEPT

1. On the Demographic details page, read the **Terms and Conditions** and select the check box to agree. This agreement is to provide consent for the storage and processing of your personal information.
2. Click **Accept** and proceed.

(i) Note: User consent is mandatory for creating/updating applications. The contents on this page will be displayed in all data capture languages selected.

Step 3: Enter Demographic details

Demographic Details (English)

Title : *	Full Name : *	Date Of Birth : *
Mr.	Chaitanya	1/1/1989 OR 33 Years
Gender : *	Nationality : *	I was born in Morocco : *
Male	French	Oui
Place of birth : *	Birth Pay : *	Region : *
Rabat-Agdal	France	Rabat Sale Keritra

CONTINUE

1. Enter your demographic details, which include Name, Age/DOB, Gender, Residential Status, Address, Mobile Number, Email ID, etc.
2. You can also change or verify your demographic details in the other selected language.
3. After you have filled in and verified your demographic details, click **Continue**.

Note: The mandatory fields/labels have a ***** mark. Field and button labels, error, and information messages will be displayed in the user-preferred language selected on the login screen. The fields displayed on this screen are configurable based on the [ID schema](#) defined by the country.

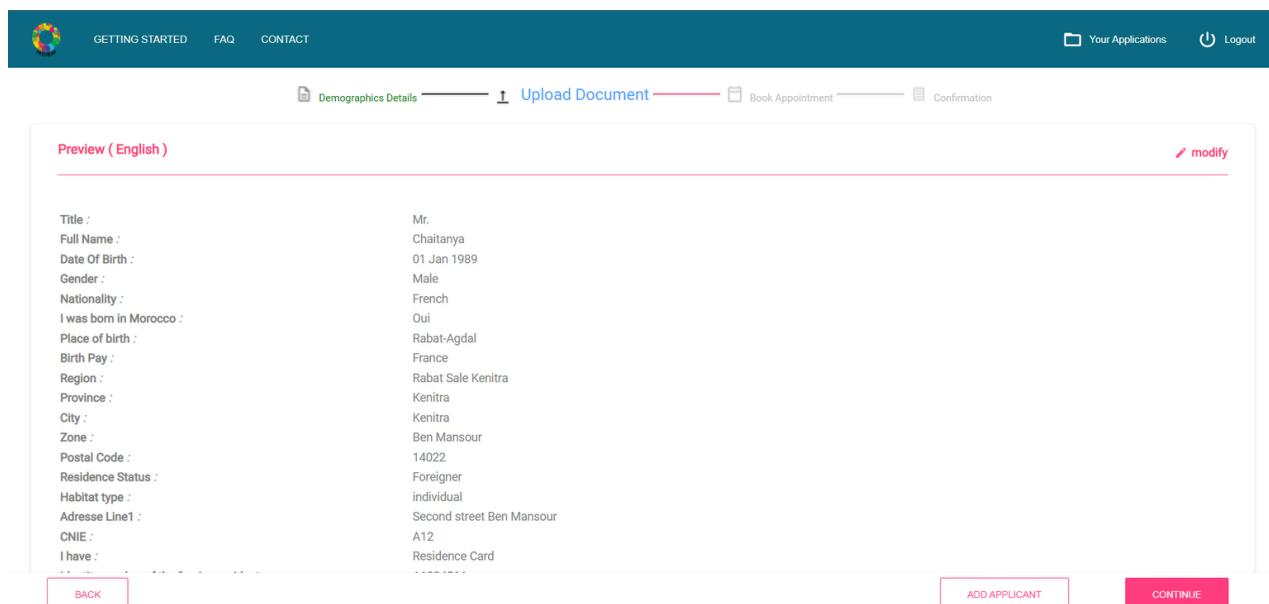
[UI specs](#) of the Pre-registration module are used to configure the form fields in the Demographic Details and Document Upload functionality pages. These specs are saved as a JSON file with a list of fields.

Step 4: Upload documents

1. Select the document (e.g. Passport, Reference Identity Number, etc.) from the document drop-down list.
2. Click **Browse** to locate the scanned document on your machine.
3. Select the file that you want to upload.

4. When the file is uploaded successfully, the document will appear on the right side. Verify that you have uploaded the correct document.
5. Repeat the steps above to upload the document(s) for each applicable document category.
6. When adding an applicant, if a newly added applicant's Proof of Address (POA) document is the same as that of the existing user's POA, which has been already uploaded, click the **Same As** option and select the name of the applicant.
7. Click **Continue** to preview your application.

Step 5: Preview data



The screenshot shows a web-based application interface for previewing application data. At the top, there is a navigation bar with links for 'GETTING STARTED', 'FAQ', 'CONTACT', 'Your Applications', and 'Logout'. Below the navigation bar, a horizontal navigation bar indicates the current step: 'Demographics Details' (highlighted), 'Upload Document', 'Book Appointment', and 'Confirmation'. The main content area displays a table of demographic information:

Title :	Mr.
Full Name :	Chaitanya
Date Of Birth :	01 Jan 1989
Gender :	Male
Nationality :	French
I was born in Morocco :	Oui
Place of birth :	Rabat-Agdal
Birth Place :	France
Region :	Rabat Sale Kenitra
Province :	Kenitra
City :	Kenitra
Zone :	Ben Mansour
Postal Code :	14022
Residence Status :	Foreigner
Habitat type :	individual
Adresse Line1 :	Second street Ben Mansour
CNIE :	A12
I have :	Residence Card

At the bottom of the preview section, there are three buttons: 'BACK', 'ADD APPLICANT', and 'CONTINUE'. To the right of the preview table, there is a 'modify' link with a pencil icon.

1. To change the demographic details (Name, Age, etc.), click **modify** at the top-right corner adjacent to the Demographic details section.
2. To modify the uploaded documents, click **modify** at the bottom-right corner adjacent to the Documents Uploaded section and make changes.
3. To add a new applicant, click **Add Applicant**. On clicking the **Add Applicant** option, you will be navigated to the Demographic details page to provide Consent and proceed with providing the required demographic data and uploading documents.
4. Click **Continue**.

Add new application

On Your Applications page, click **Create New Application** to generate a new application.

Viewing applications

The screenshot shows a dashboard titled 'Your Applications' with five application cards. Each card contains the following information:

- Application ID 71450694613204**: Name: Raju, Appointment Date: --, Status: Application Incomplete, Data Capture Languages: English. Buttons: Modify Information, View Acknowledgement.
- Application ID 36537935645738**: Name: Kestraju, Appointment Date: --, Status: Pending Appointment, Data Capture Languages: English. Buttons: Modify Information, View Acknowledgement.
- Application ID 63197507532871**: Name: Chaitanya, Appointment Date: (09:45 - 10:00) 14 Jan 2022, Status: Booked, Data Capture Languages: English. Buttons: View Information, View Acknowledgement.
- Application ID 51027904659758**: Name: asdf, Appointment Date: (09:15 - 09:30) 11 Jan 2022, Status: Booked, Data Capture Languages: English. Buttons: View Information, View Acknowledgement.
- Application ID 26973013824720**: Name: asdfgh, Appointment Date: (09:00 - 09:15) 10 Jan 2022, Status: Booked, Data Capture Languages: English. Buttons: View Information, View Acknowledgement.
- Application ID 37624631465275**: Name: Kestraju, Appointment Date: (09:00 - 09:15) 07 Jan 2022, Status: Expired, Data Capture Languages: English. Buttons: View Information, View Acknowledgement.
- Application ID 47810791403879**: Name: zxccb, Appointment Date: --, Status: Cancelled, Data Capture Languages: English. Buttons: View Information, View Acknowledgement.

Dashboard

Once the application is created, there could be multiple statuses depending on the data filled by the user/resident or the actions performed by them. The user can view all the pre-registration applications created by them in the Dashboard. The different statuses with a brief explanation are mentioned below:

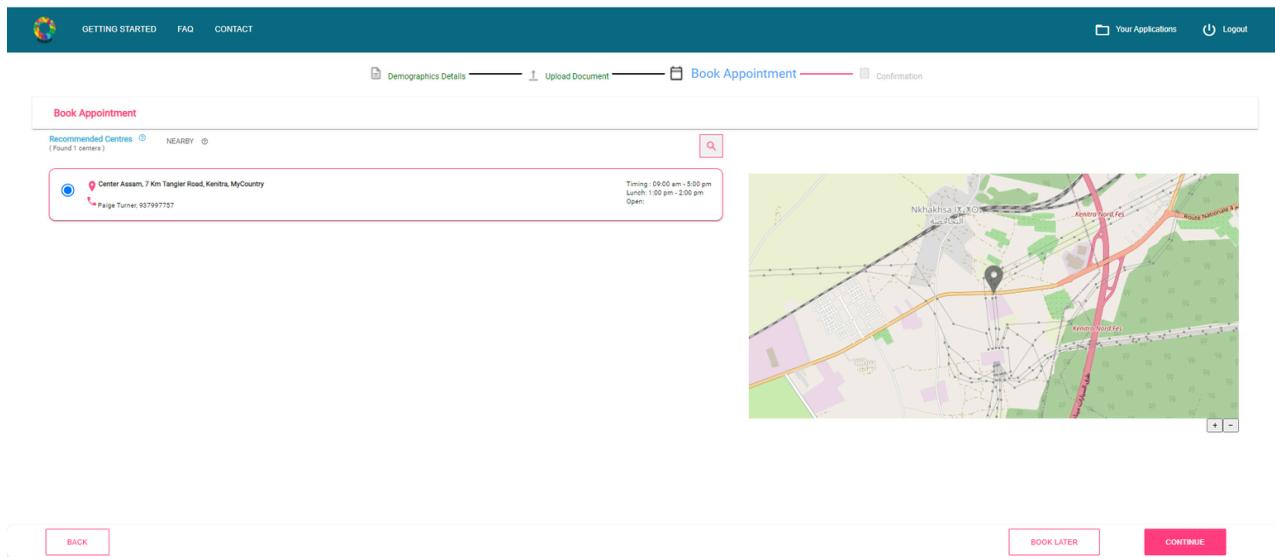
Status	Description	User Action
Incomplete	Filled only demographic details	Upload documents and book an appointment
Pending appointment	Filled demographic details and uploaded documents	Book an appointment
Booked	Filled demographic details, uploaded documents, and booked appointment	Visit the registration center on the appointment date and time

Expired	Appointment date has passed	Re-book an appointment
Cancelled	Appointment has been	Re-book an appointment

- The applications are sorted and displayed by the order of creation of the application. The last application created appears first in the list.
- If the user visits the registration center and consumes the appointment, then the application will be removed from the list.
- If the appointment date has passed, the status changes to "Expired" and is retained on the dashboard for further rebooking/modification as required.

Book an appointment

Choose a registration center



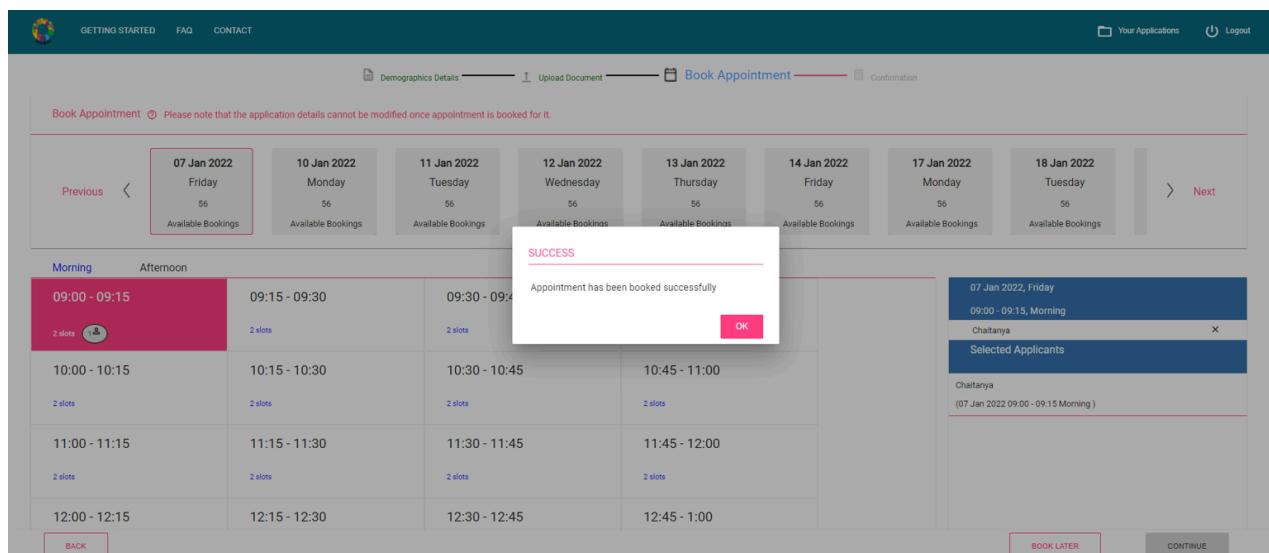
- The recommended registration centers are automatically displayed based on your demographic details (Postal Code)
- On the Book Appointment page, you can find a registration center through the three options as follows:
 - Click **Nearby** centers to view the registration centers based on your geographical location.
 - Use the search box to find the registration center based on your search criteria.

- Click **Recommended Centers** to view registration centers based on your demographic details. (Postal Code)
- Click **Continue.**

Note: The default display of registration centers will be based on the Postal Code of the user. To modify this setting, please update the location hierarchy in the `pre-registration-default.properties` file using the property: `preregistration.recommended.centers.locCode`.

Select an appointment time slot

- Select your preferred date from the list of available calendar days and the number of available bookings.
- The list of available time slots for your selected date is categorized between *Morning* and *Afternoon*.
- Select your preferred time slot from the list.
- Select the particular applicant's name to book an appointment (click + to add the applicant). Note: On clicking the **Add Applicant** option, you will be navigated to the Demographic Details page to provide Consent and proceed with providing the required demographic data/documents.
- Verify the time slot(s) as selected against the applicant's name(s).
- Click **Continue.**
- On the confirmation pop-up, click **Confirm**.
- Click **OK.**



Receive appointment acknowledgement

The screenshot shows a confirmation page from the MOSIP platform. At the top, there's a navigation bar with links for 'GETTING STARTED', 'FAQ', 'CONTACT', 'Your Applications', and 'Logout'. Below the navigation is a breadcrumb trail: 'Demographics Details' → 'Upload Document' → 'Book Appointment' → 'Confirmation'. A green success message box at the top states: '37624631465275 : The Pre-Registration id and Appointment details have been sent to the registered email id and phone number'. The main content area contains the following information:

- Pre Registration ID :** 37624631465275
- Appointment Date & Time :** 9:00 AM, 07 Jan 2022
- Center Contact Number :** 937997757

A QR code is displayed below this information.

Below the QR code, there are two rows of text:

- Name : Chaltanya
- Center Name : Center Assam

At the bottom of the page, a note says: '1. Guideline 1 2. Guideline 2 3. Guideline 3 4. Guideline 4 5. Guideline 5 6. Guideline 6 7. Guideline 7 8. Guideline 8 9. Guideline 9 10. Guideline 10'

At the bottom right, there are three buttons: 'SEND EMAIL/SMS', 'DOWNLOAD PDF', and 'PRINT'.

- After successful completion of the Pre-registration application, you will receive an acknowledgment on the registered phone number (SMS) or email address as per details provided in the demographic form.
- The acknowledgment contains the following information: name, pre-registration ID, age/DOB, mobile number, email ID and registration center details, appointment date, and appointment time)
- A QR code containing the pre-registration ID is generated. This QR code can be scanned at the registration center to fetch the details to be used during the registration process.
- You can print, download, email, or SMS your acknowledgment.
 - To print your acknowledgment, click **Print**.
 - To download your acknowledgement, click **Download PDF**.
- To add the additional recipient(s) to receive the acknowledgment of your application, follow these steps:
 - Click **Send Email/SMS**.
 - Enter the mobile number and/or enter the email ID.
 - Click **Send** to receive the acknowledgment on your provided e-mail address or mobile number.

Re-book appointment

1. On **Your Applications** page, select the check box for the applicable applicant.
2. Click **Book/Modify Appointment** to re-book an appointment (on the top right corner)..
3. The user can select any appointment date available and the appointment slot available
4. A user cannot re-book the appointment if the appointment booking is less than 48 hours (configurable) from the time of booking

Discard application

1. On the Your Applications page, click on the **delete** icon against the pre-registration application of an applicant, and a pop-up window appears on the screen.
2. Select the **Discard entire application** option in the pop-up window.
3. Click **SUBMIT** to discard your application.

Cancel appointment

1. On Your Applications page, click on the **delete** icon against the pre-registration application of an applicant, and a pop-up window appears on the screen.
2. Select **Cancel appointment and save the details** option in the pop-up window.
3. Click **SUBMIT** to cancel an appointment.

Following a successful appointment cancellation, the system unlocks the time slot of the registration center to ensure that someone else can book it.

Registration Client

Overview

The Registration Client is a thick Java-based client where the resident's demographic and biometric details are captured along with the supporting documents in online or offline mode. Data is captured in the form of registration packets and is cryptographically secured to ensure that there is no tampering. The captured information is packaged and sent to the server for further processing.

MOSIP provides a reference implementation of a Java-based Registration Client. The code, build files for the Registration Client are available in the [Registration Client repo](#).

Multiple language support

- Registration Client is featured to allow an operator to choose the operation language. Option to select their preferred language is provided on the login screen.
- Data collection during registration client supports more than one language at a time.
- Before starting any registration process, the operator can choose the languages amongst the configured ones.

To know more about setting up the reference registration client, refer to [Registration Client Installation Guide](#).

To know more about the features present in the Registration Client, refer to [Registration Client User Guide](#).

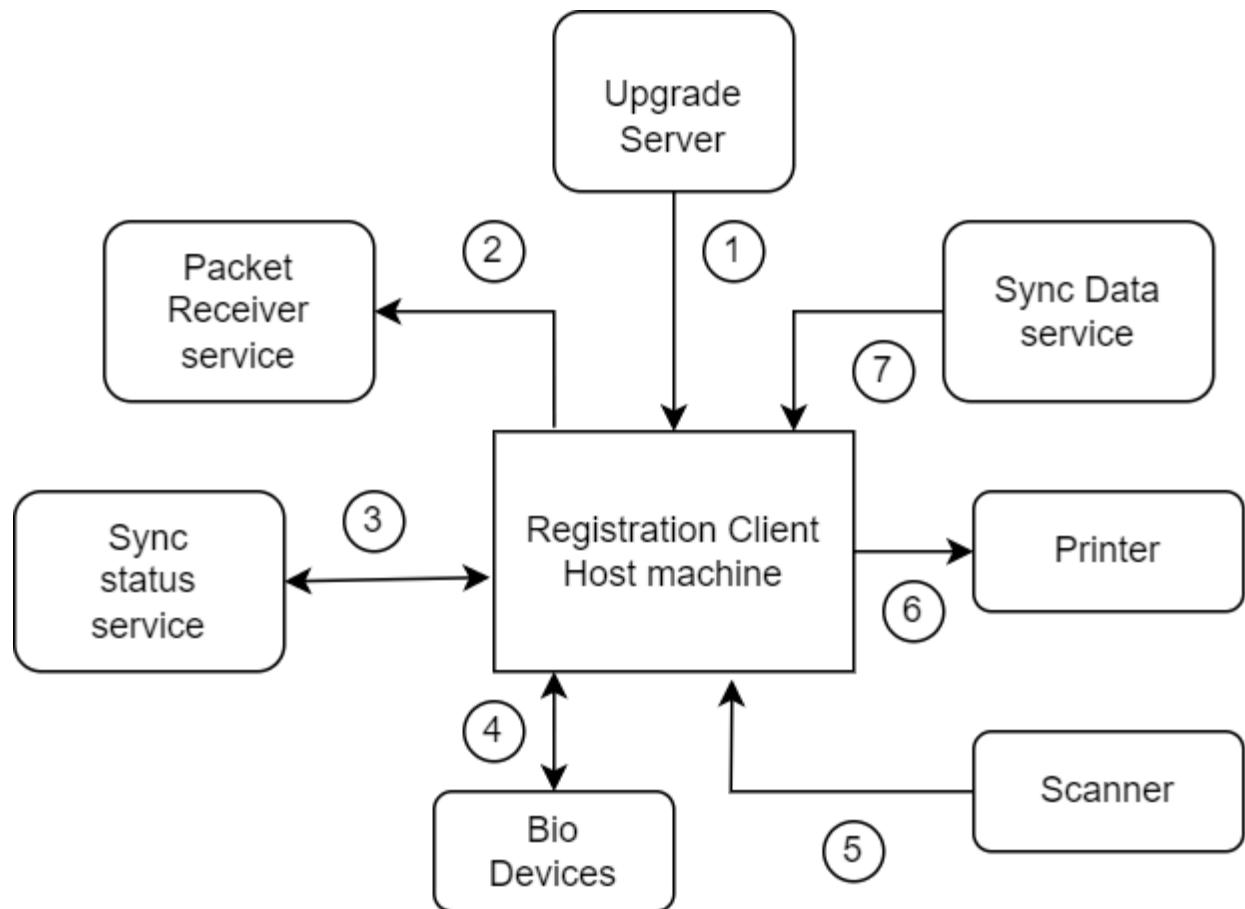
Who operates the Registration Client?

The Registration Client can be operated by an operator who can be either a **Supervisor** or an **Officer**. They can login to the client application and perform various activities. The Supervisor and the Officer can perform tasks like

Onboarding, Synchronize Data, Upgrade software, Export packet, Upload packets, View Re-registration packets, Correction process, Exception authentication, etc. In addition to this, the Supervisor has exclusive authority to Approve/reject registrations.

To know more about the onboarding process of an operator, refer to [Operator onboarding](#).

Registration Client entity diagram

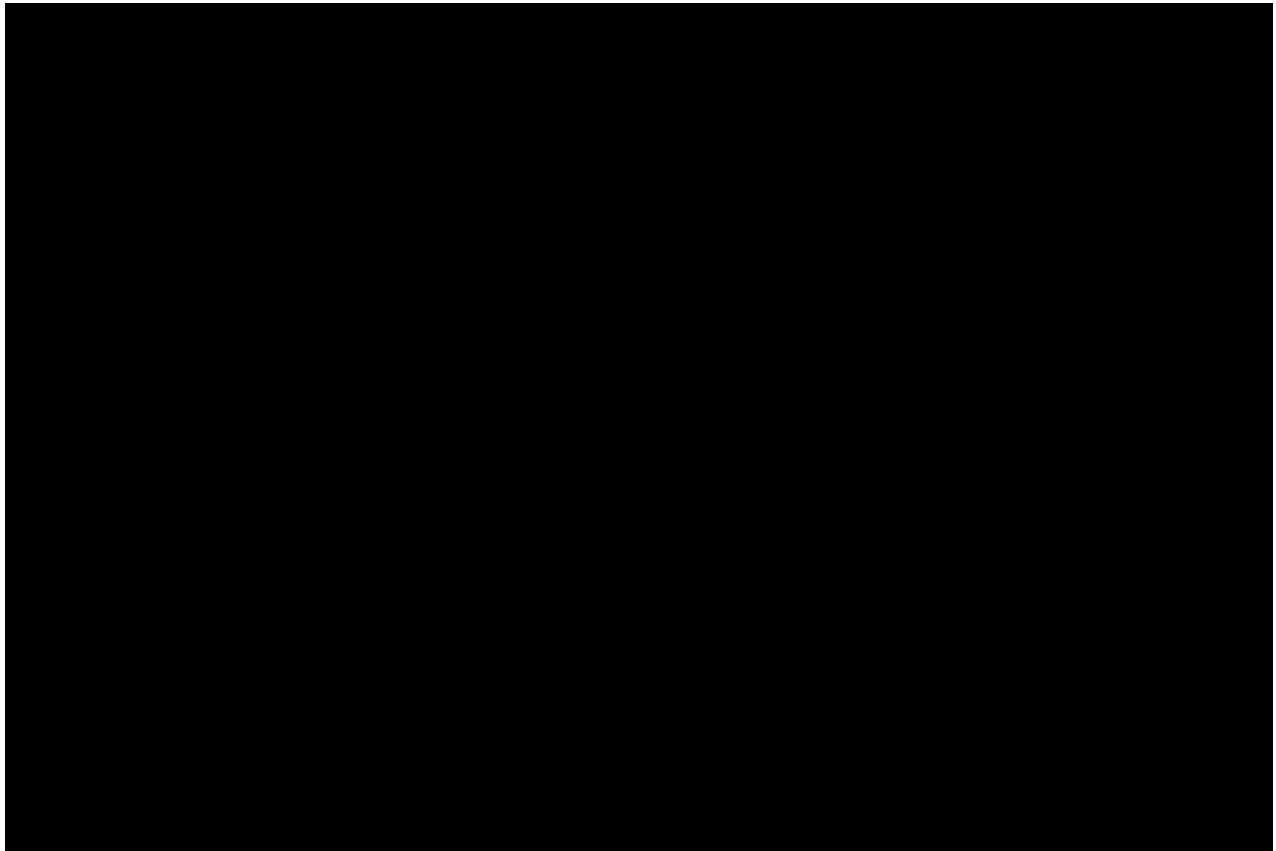


Note: Arrows indicate the direction of data flow.

The relationship of Registration Client with other services is explained here. **NOTE:** The numbers do not signify sequence of operations or control flow.

1. Registration Client connects to the Upgrade Server to check on upgrades and patch downloads.
2. All the masterdata and configurations are downloaded from SyncData-service.
3. Registration Client always connects to external biometric devices through SBI.
4. Registration Client scans the document proofs from any document scanner.
5. Acknowledgement receipt print request is raised to any connected printers.
6. Packets ready to be uploaded meta-info are synced to Sync Status service. Also, the status of already uploaded packets are synced back to Registration Client.
7. All the synced packets are uploaded to Packet Receiver service one by one.

The image below shows the setup of Registration Client Host machine.



1. Registration Client comprises of JavaFX UI, Registration-services libaries and any third party biometric-SDK.
2. SBI is allowed to run on loopback IP and should listen on any port within 4501-4600 range. More than one SBI can run on the host machine. Registration Client scans the allowed port range to identify the available SBI.

3. Registration Client connects to local Derby database. This is used to store all the data synced.
4. All the completed registration packets are stored under packetstore directory.
5. `.mosipkeys` directory is used to store sensitive files. `db.conf` under this directory stores encrypted DB password. This is created on the start of registration client for the first time.
6. TPM - is the hardware security module used to get machine identifier, to secure DB password, prove the client authenticity on auth requests and packets created in the host machine.

Data protection

- The registration packets and synced data are stored in the client machine.
- Most of the synced data are stored in the Derby DB. Derby DB is encrypted with the bootpassword.
- Derby DB boot password is encrypted with machine TPM key and stored under `.mosipkeys/db.conf`.
- Synced UI-SPEC/script files are saved in plain text under registration client working directory. During sync, SPEC/script file hash is stored in derby and then the files are saved in the current working directory. Everytime the file is accessed by the client performs the hash check.
- Synced pre-registration packets are encrypted with TPM key and stored under configured directory.
- Directory to store the registration packets and related registration acknowledgments is configurable.
- Registration packet is an signed and encrypted ZIP.
- Registration acknowledgment is also signed and encrypted with TPM key.

Background Tasks

The Registration Client runs background tasks to keep data synchronized with the Registration Processor. It continuously updates the server with newly created

packets and syncs additional metadata to improve packet recovery in case of a client failure.

Another background task handles packet uploads. If supervisor approval is required (`'y'`), approved packets are uploaded in batches. If approval is not required (`'n'`), packets are uploaded during the next scheduled job. With this feature, the registration client has fully capable auto upload.

You can configure these settings in the [Scheduled Jobs](#) and [Batch Configuration](#) sections.

Configurations

Registration Client can be customized as per a country's requirements. For details related to Registration Client configurations, refer to [Registration Client configuration](#).

UI Specifications for Registration Tasks

- Blueprint of registration forms to be displayed in registration client are created as json called as UI-SPEC.
- Every process (NEW / LOST / UPDATE UIN / CORRECTION) has its own UI-SPEC json.
- Kernel-masterdata-service exposes API's to create and publish UI-SPEC.
- Published UI-SPEC json are versioned.
- Only published UI-SPEC are synced into registration-client.
- UI-SPEC json files are tamper proof, client checks the stored file hash everytime it tries to load registration UI.
- UI-SPEC json will fail to load if tampered.

Default UI Specifications loaded with Sandbox installation is available [here](#)

Developer Guide

To know more about the developer setup, read [Registration Client Developers Guide](#).

Develop

Developers Guide

Overview

[Registration Client](#) is a thick Java-based client where the resident's demographic and biometric details are captured along with the supporting documents in online or offline mode.

The documentation here will guide you through the prerequisites required for the developer' setup.

Software setup

Below are a list of tools required in Registration Client:

1. JDK 11
2. Any IDE (like Eclipse, IntelliJ IDEA)
3. Apache Maven (zip folder)
4. Git

Follow the steps below to set up Registration Client on your local system:

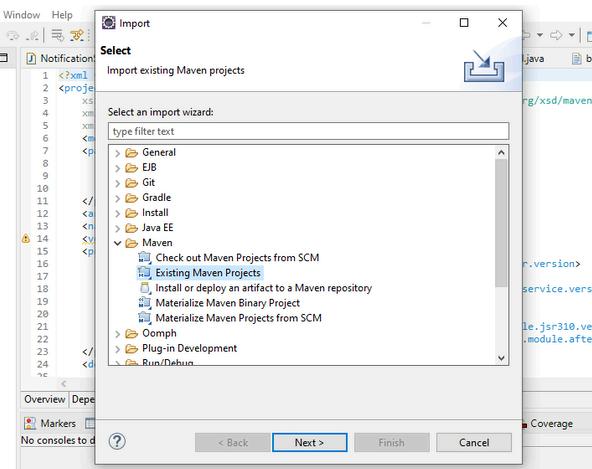
Code setup

For the code setup, clone the [Registration Client](#) repository and follow the guidelines mentioned in the [Code Contributions](#).

Importing and building of a project

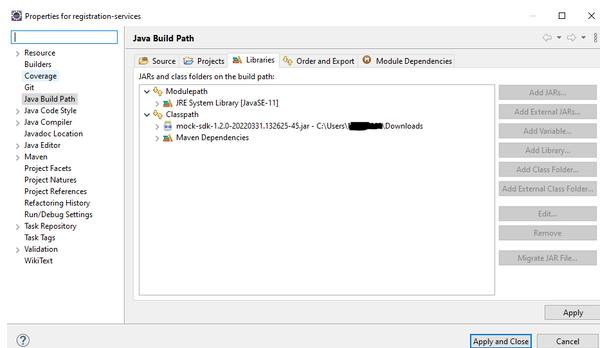
1. Open the project folder where `pom.xml` is present.

2. Open command prompt from the same folder.
3. Run the command `mvn clean install -Dgpg.skip=true -DskipTests=true` to build the project and wait for the build to complete successfully.
4. After building of a project, open Eclipse and select `Import Projects → Maven → Existing Maven Projects → Next → Browse to project directory → Finish`.
5. After successful importing of project, update the project by right-click on `Project → Maven → Update Project`.



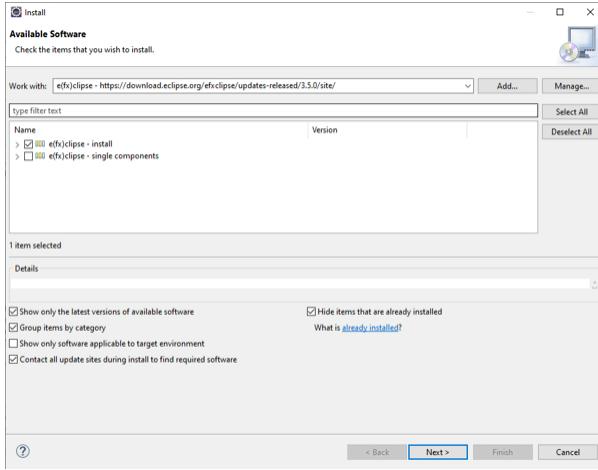
Environment setup

1. For the environment setup, you need an external dependency that is available [here](#) with different versions. (E.g.: You can download `mock-sdk.jar` and add to registration-services project `Libraries → Classpath → Add External JARs → Select Downloaded JAR → Add → Apply and Close`).



2. Registration Client UI is developed using JavaFX and the UI pages are fxml files which can be opened using a tool called [Scene Builder](#). The JavaFX integration

with the Eclipse IDE is provided with the e(fx)clipse tool. Go to `Help → Install New Software → Work with → Add`. Give Name and Location as mentioned in below image.



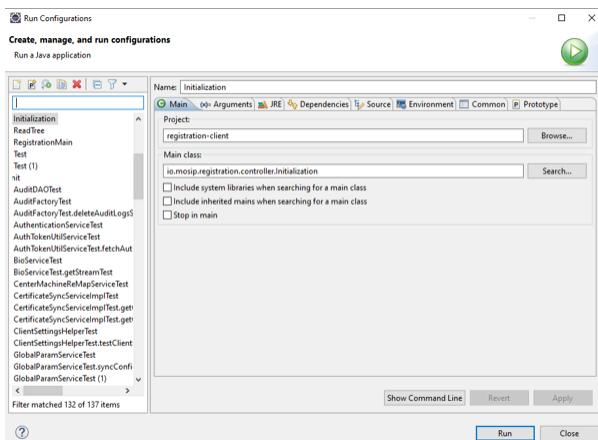
Once the software is installed, you will be prompted to restart your IDE.

3. Download `openjfx-11.0.2_windows-x64_bin-sdk.zip` from [here](#), unzip and place it in your local file system. This folder contains list of javafx related jars that are necessary for running Registration Client UI.

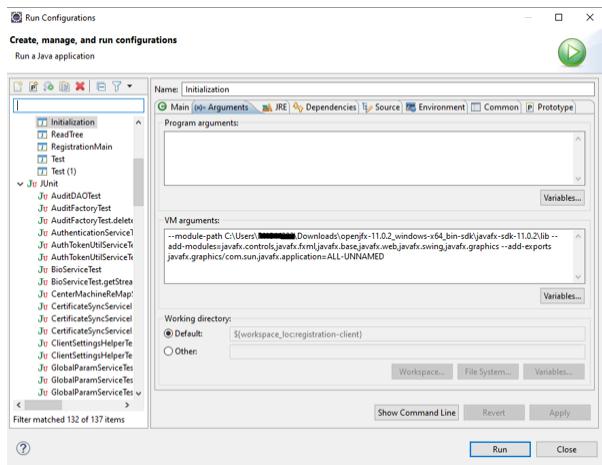
4. We can change the application environment in the file `registration-services\src\main\resources\props\mosip-application.properties` by modifying the property `mosip.hostname`

Below are the configurations to be done in Eclipse:

1. Open Eclipse and run the project for one time as `Java application`, so that it creates a Java application which you can see in debug configurations.



2. Open the arguments and pass this `--module-path C:\Users\<USER_NAME>\Downloads\openjfx-11.0.2_windows-x64_bin-sdk\javafx-sdk-11.0.2\lib --add-modules javafx.controls,javafx.fxml,javafx.base,javafx.web,javafx.swing,javafx.graphics --add-exports javafx.graphics/com.sun.javafx.application=ALL-UNNAMED` in VM arguments.



3. Click Apply and then debug it (starts running). You can see a popup which shows informative messages of what is happening in background while Registration Client UI is loading and the application will be launched.

Running the Registration Client Downloader Docker image

- Dockerfile is available under [registration-client repository](#).
- The concept here is to run nginx in the docker container from which registration-client.zip is hosted and also registration-client on the field will connect to this nginx to check for new updates or changes.

Note: We refer this nginx server as registration-client download and upgrade server.

While running the registration-client docker container we need to pass below environment variables:

Required

`client_version_env` : pom version of the registration client build

`client_upgrade_server_env` : public URL of this nginx server

`reg_client_sdk_url_env` : URL to download sdk zip. Make sure to have SDK jar and its dependent jars in the zip.

`artifactory_url_env` : artifactory URL to download below mentioned runtime dependencies

`"${artifactory_url}/artifactory/libs-release-local/icu4j/icu4j.jar"`

`"${artifactory_url}/artifactory/libs-release-local/icu4j/kernel-transliteration-icu4j.jar"`

`"${artifactory_url}/artifactory/libs-release-local/clamav/clamav.jar"`

`"${artifactory_url}/artifactory/libs-release-local/clamav/kernel-virusscanner-clamav.jar"`

`keystore_secret_env` : password of the keystore.p12 file

`host_name_env` : syncdata-service public domain name

`signer_timestamp_url_env` : URL of timestamp server to timestamp registration-client jar files.

Need to mount a volume to “/home/mosip/build_files” with below files

- `logback.xml`
- `Client.crt` : Signer certificate to be added in the registration-client build for jar sign verification.
- `keystore.p12` : Store private key used to sign the registration-client and registration-services jar files with “CodeSigning” alias.

- `maven.metadata.xml` : Holds the current version of registration-client hosted in the upgrade server.

Optional

`reg_client_custom_impls_url_env` : URL to download scanner custom implementation jars(runtime dependency).

Finally, you can download the registration client zip from below URL.

{registrationclient download server domain name/ip}/registration-client/{client_version}/reg-client.zip}

References

Run (<https://github.com/mosip/mosip-infra/blob/develop/deployment/v3/mosip/regclient/create-signing-certs.sh>) to generate `Client.crt` and `keystore.p12`.

To get the content of `maven-metadata.xml` and `logback.xml` check (<https://github.com/mosip/mosip-helm/blob/develop/charts/regclient/templates/configmap.yaml>)

UI Specifications

Overview

The registration UI forms are rendered using respective UI specification JSON. This is derived from the [ID Schema](#) defined by a country. Here, we would be discussing about the properties used in the UI specification of Registration Client.

In the Registration Client, currently, Registration Tasks(process) forms are configurable using the UI specifications.

Each process has multiple screens and each screen is rendered with one or more fields.

Process/ Task spec JSON template

```
{  
    "id": "<NEW/UPDATE/LOST/BIOMETRIC_CORRECTION process name as passed in  
    //order in which the process is displayed on the registration client |  
    "order": 1,  
    "flow": "<NEW/UPDATE/LOST/CORRECTION>,"  
    //Multi-lingual labels displayed based on the logged in language  
    "label": {  
        "eng": "Registration",  
        "ara": "تسجيل",  
        "fra": "Inscription"  
    },  
    //screen details - follows screen spec structure  
    "screens": [  
        {}  
    ],  
    //caption displays on-hover content  
    "caption": {  
        "eng": "Registration",  
        "ara": "تسجيل",  
        "fra": "Inscription"  
    },  
    //icon is the symbol that appears before the process label  
    "icon": "registration.png",  
    "isActive": true,  
    //group names that should be by default selected during UPDATE UIN pro  
    "autoSelectedGroups": [  
        ""  
    ]  
}
```

Screen spec JSON template

```
//Order of the screen in registration page
"order": 1,
"name": "<unique identifier for the screen>",
"label": {
    "ara": "شاشة عينة",
    "fra": "Exemple d'écran",
    "eng": "Sample screen"
},
"caption": {
    "ara": "شاشة عينة",
    "fra": "Exemple d'écran",
    "eng": "Sample screen"
},
//field details - follows field spec structure
"fields": [
    {}
],
"layoutTemplate": null,
//displays field to provide pre-reg application ID, data fetched from
"preRegFetchRequired": true,
//enable below flag to capture additionalInfo request ID , applicable
"additionalInfoRequestIdRequired": false,
//show or hide screens
"active": true
}
```

Field spec JSON template

```
{  
    "id": "<Unique identifier for the field, must be same as that described in the UI>",  
    //inputRequired is used to identify if UI input is needed or not  
    "inputRequired": true,  
    //type defines the datatype of the field, must be same as that is defined in the UI  
    "type": "<string/simpleType/documentType/biometricsType>",  
    "controlType": "textbox/fileupload/dropdown/checkbox/button/date/ageDate/  
    //minimum- applicable only for date controlType(defined in days)  
    "minimum": 0,  
    //maximum- applicable only for date controlType(defined in days)  
    "maximum": 365,  
    "description": "<Field description> ",  
    "label": {  
        "ara": "حقل العينة",  
        "fra": "Exemple de champ",  
        "eng": "Sample Field"  
    },  
    //fieldType is used to identify if it is a dynamic field  
    "fieldType": "<default/dynamic>",  
    //to validate the format should be in upper or lower case  
    "format": "<lowercase/uppercase/none> ",  
    //list of validators for the field  
    "validators": [  
        {  
            //type of validation engine (currently, only regex is supported)  
            "type": "regex",  
            //expression for the validation  
            "validator": "^(\\d{10,30})$ ",  
            //list of arguments needed for the validator  
            "arguments": [],  
            //if null, its applicable for all languages, else validator is language specific  
            "langCode": null,  
            /*error code to be used to display specific error message, if validation fails  
             * There error codes must be configured in registration  
             */  
            "errorCode": "UI_100001"  
        }  
    ],  
    //determines sharing and longevity policies applicable as defined in the UI  
    "fieldCategory": "<pvt/evidence/kyc> ",  
    "alignmentGroup": "<fields belonging to same alignment group are placed together> ",  
    //determines when to display and hide the field(set null if the field is always visible)  
    "visible": {  
        "engine": "MVEL",  
        "expr": "identity.get('ageGroup') == 'INFANT' && (identity.get('isChild') == true || identity.get('isTeen') == true)"  
    },  
    "contactType": null,  
    //used to group together the list of fields(only applicable in the Update UIN process)  
    "group": "<grouping used in update UIN process> ",  
}
```

```
"groupLabel": {  
    "eng": "Sample group",  
    "ara": "مجموعة العينة",  
    "fra": "Groupe d'échantillons"  
},  
/*on change of the field value, configured Action will be triggered on  
     Change action handlers should be implemented in registration  
"changeAction": null,  
//enable or disable auto-transliteration  
"transliterate": false,  
/*provide the templateName(applicable only for html controlType field)  
     These templates should be configured in templates table*/  
"templateName": null,  
"fieldLayout": null,  
"locationHierarchy": null,  
//On any biometric exception, Need to capture exception photo as proof  
"exceptionPhotoRequired": true,  
/*applicable only for BiometricsType field, defines the list of attributes  
     All the supported biometric attributes are listed down for reference  
"bioAttributes": [  
    "leftEye",  
    "rightEye",  
    "rightIndex",  
    "rightLittle",  
    "rightRing",  
    "rightMiddle",  
    "leftIndex",  
    "leftLittle",  
    "leftRing",  
    "leftMiddle",  
    "leftThumb",  
    "rightThumb",  
    "face"  
],  
//capture of above mentioned bioAttributes can be conditionally mandatory  
"conditionalBioAttributes": [  
    {  
        "ageGroup": "INFANT",  

```

```
"required": true,  
//if requiredOn is defined, the evaluation result of requiredOn.expr  
"requiredOn": [  
    {  
        "engine": "MVEL",  
        "expr": "identity.get('ageGroup') == 'INFANT' && (identity.get('ageGroup') < 12 || identity.get('ageGroup') > 18)"  
    }  
,  
    //used to identify the type of field  
    "subType": "<document types / applicant / heirarchy level names>"  
}
```

Sample correction process SPEC: Biometric Correction

```
{  
    "id": "BIOMETRIC_CORRECTION",  
    "order": 4,  
    "flow": "CORRECTION"  
    "label": {  
        "eng": "Biometric correction",  
        "ara": "التصحيح البيومترى",  
        "fra": "Correction biométrique"  
    },  
    "screens": [  
        {  
            "order": 1,  
            "name": "consentdet",  
            "label": {  
                "ara": "موافقة",  
                "fra": "Consentement",  
                "eng": "Consent"  
            },  
            "caption": {  
                "ara": "موافقة",  
                "fra": "Consentement",  
                "eng": "Consent"  
            },  
            "fields": [  
                {  
                    "id": "consentText",  
                    "inputRequired": true,  
                    "type": "simpleType",  
                    "minimum": 0,  
                    "maximum": 0,  
                    "description": "Consent",  
                    "label": {},  
                    "controlType": "html",  
                    "fieldType": "default",  
                    "format": "none",  
                    "validators": [],  
                    "fieldCategory": "evidence",  
                    "alignmentGroup": null,  
                    "visible": null,  
                    "contactType": null,  
                    "group": "consentText",  
                    "groupLabel": null,  
                    "changeAction": null,  
                    "transliterate": false,  
                    "templateName": "Registration Consent",  
                    "fieldLayout": null,  
                    "locationHierarchy": null,  
                    "conditionalBioAttributes": null,  
                }  
            ]  
        }  
    ]  
}
```

```
        "required": true,
        "bioAttributes": null,
        "requiredOn": [],
        "subType": "consentText"
    },
{
    "id": "consent",
    "inputRequired": true,
    "type": "string",
    "minimum": 0,
    "maximum": 0,
    "description": "consent accepted",
    "label": {
        "ara": "الاسم الكامل الكامل الكامل",
        "fra": "J'ai lu et j'accepte les termes et conditions",
        "eng": "I have read and accept terms and conditions"
    },
    "controlType": "checkbox",
    "fieldType": "default",
    "format": "none",
    "validators": [],
    "fieldCategory": "evidence",
    "alignmentGroup": null,
    "visible": null,
    "contactType": null,
    "group": "consent",
    "groupLabel": null,
    "changeAction": null,
    "transliterate": false,
    "templateName": null,
    "fieldLayout": null,
    "locationHierarchy": null,
    "conditionalBioAttributes": null,
    "required": true,
    "bioAttributes": null,
    "requiredOn": [],
    "subType": "consent"
},
{
    "id": "preferredLang",
    "inputRequired": true,
    "type": "string",
    "minimum": 0,
    "maximum": 0,
    "description": "user preferred Language",
    "label": {
        "ara": "لغة الإخطار",
        "fra": "Langue de notification",
        "eng": "User Preferred Language"
    }
}
```

```
        "eng": "Notification Langauge"
    },
    "controlType": "button",
    "fieldType": "dynamic",
    "format": "none",
    "validators": [],
    "fieldCategory": "pvt",
    "alignmentGroup": "group1",
    "visible": null,
    "contactType": null,
    "group": "PreferredLanguage",
    "groupLabel": null,
    "changeAction": null,
    "transliterate": false,
    "templateName": null,
    "fieldLayout": null,
    "locationHierarchy": null,
    "conditionalBioAttributes": null,
    "required": true,
    "bioAttributes": null,
    "requiredOn": [],
    "subType": "preferredLang"
}
],
"layoutTemplate": null,
"preRegFetchRequired": false,
"additionalInfoRequestIdRequired": false,
"active": false
},
{
    "order": 2,
    "name": "BiometricDetails",
    "label": {
        "ara": "التفاصيل البيومترية",
        "fra": "Détails biométriques",
        "eng": "Biometric Details"
    },
    "caption": {
        "ara": "التفاصيل البيومترية",
        "fra": "Détails biométriques",
        "eng": "Biometric Details"
    },
    "fields": [
        {
            "id": "individualBiometrics",
            "inputRequired": true,
            "type": "biometricsType",
            "minimum": 0,
            "maximum": 1
        }
    ]
}
```

```
        "maximum": 0,
        "description": "",
        "label": {
            "ara": "القياسات الحيوية الفردية",
            "fra": "Applicant Biometrics",
            "eng": "Applicant Biometrics"
        },
        "controlType": "biometrics",
        "fieldType": "default",
        "format": "none",
        "validators": [],
        "fieldCategory": "pvt",
        "alignmentGroup": null,
        "visible": null,
        "contactType": null,
        "group": "Biometrics",
        "groupLabel": null,
        "changeAction": null,
        "transliterate": false,
        "templateName": null,
        "fieldLayout": null,
        "locationHierarchy": null,
        "conditionalBioAttributes": [
            {
                "ageGroup": "INFANT",
                "process": "ALL",
                "validationExpr": "face",
                "bioAttributes": [
                    "face"
                ]
            }
        ],
        "required": true,
        "bioAttributes": [
            "leftEye",
            "rightEye",
            "rightIndex",
            "rightLittle",
            "rightRing",
            "rightMiddle",
            "leftIndex",
            "leftLittle",
            "leftRing",
            "leftMiddle",
            "leftThumb",
            "rightThumb",
            "face"
        ],
        "requiredOn": []
    }
}
```

```
        "required": false,
        "subType": "applicant"
    },
{
    "id": "proofOfException",
    "inputRequired": false,
    "type": "documentType",
    "minimum": 0,
    "maximum": 0,
    "description": "proofOfException",
    "label": {
        "ara": "إثبات الاستثناء",
        "fra": "Exception Proof",
        "eng": "Exception Proof"
    },
    "controlType": "fileupload",
    "fieldType": "default",
    "format": "none",
    "validators": [],
    "fieldCategory": "evidence",
    "alignmentGroup": null,
    "visible": null,
    "contactType": null,
    "group": "Documents",
    "groupLabel": null,
    "changeAction": null,
    "transliterate": false,
    "templateName": null,
    "fieldLayout": null,
    "locationHierarchy": null,
    "conditionalBioAttributes": null,
    "required": false,
    "bioAttributes": null,
    "requiredOn": [],
    "subType": "POE"
}
],
"layoutTemplate": null,
"preRegFetchRequired": false,
"additionalInfoRequestIdRequired": true,
"active": false
}
],
"caption": {
    "eng": "Biometric correction",
    "ara": "التصحيح البيومترى",
    "fra": "Correction biométrique"
},
"icon": "UINUpdate.png",
```

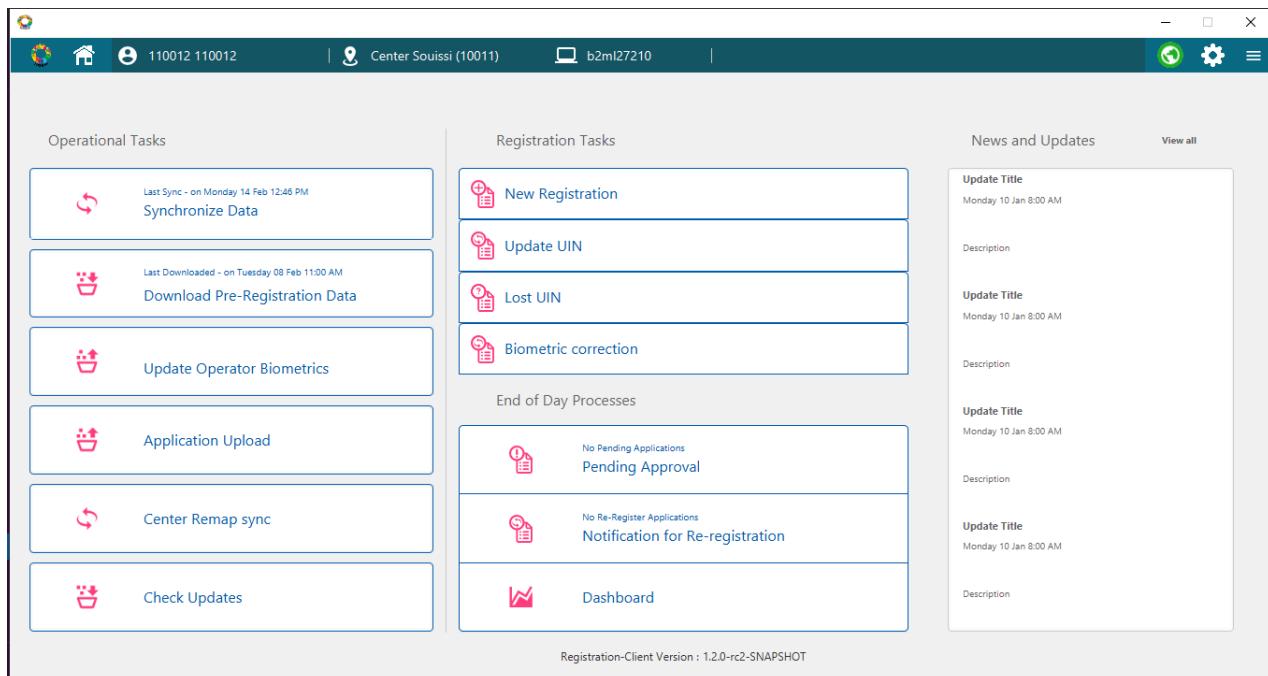
```
    "isActive": true,  
    "autoSelectedGroups": null  
}
```

Test

End User Guide

Overview

This guide describes the various functions provided in the Home page of the reference UI implementation of the registration client.



Menu bar

The Registration Client menu bar displays the following:

- MOSIP logo
- Home button
- Logged in username
- Center name
- Machine name
- Server connection status symbol (shows if the client is online or offline)
- [Settings icon](#)

- Breadcrumbs (User Guide/Reset Password/Logout)



Operational tasks

Synchronize Data

Synchronize data is the data required to make the Registration Client(RC) functional. Full sync is performed initially during the launch of the RC for the first time. Post this, the RC syncs only the changes from sever and this is called as the delta sync. Synchronize data is automated and can be triggered manually. This happens automatically while launching the RC and is also manually initiated by the operator.

1. **Configuration sync:** Sync of properties which drives in deciding the RC UI functionality. For example: Invalid login attempts, idle timeout, thresholds, etc.
2. **Masterdata sync :** As a part of this sync, supporting information like Dynamic fields data, templates, locations, screen authorization, blocklisted words, etc. are pulled in.
3. **UserDetails sync:** userID, along with their status is synced. Only the user details belonging to machine mapped center will be synced.
4. **Certificate sync:** Certificates used to validate the server signatures, device CA certificates, public key (specific to a center and machine, also called as policy key) used to encrypt the registration packet will be synced.
5. **Packet sync:**
 - All the approved/rejected Registration IDs(RIDs) will be synced to the server.
 - All the synced RID packets will be uploaded to the server.
 - All the uploaded packet status will be synced from server.

Download Pre-Registration Data

An operator can download the pre-registration data of a machine mapped center while being online and store it locally in the registration machine for offline use. Now, when the system is offline and the pre-registration data is already downloaded, the operator can enter the pre-registration ID to auto-populate the registration form. Provided the machine is online, any pre-registration application can be downloaded irrespective of the center booked by the resident.

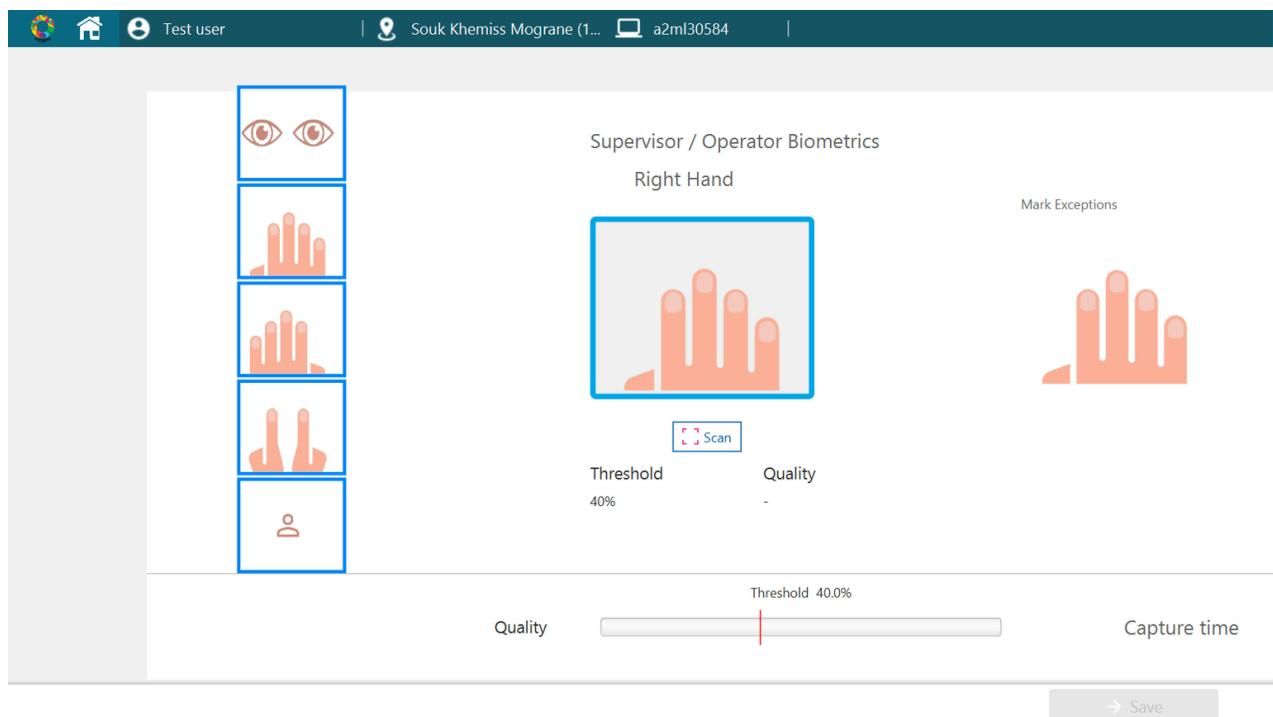
This pre-registration data is downloaded from pre-registration API as an encrypted packet and stored in the local storage. There is a batch job running in background for deleting the pre-registration packets after configurable number of days.

Note- Date Range of pre-registration data to be downloaded and storage location of pre-registration data in the registration machine is configurable. Also, this is synced as a part of configuration sync.

Update Operator Biometrics

Using this option, the operators can onboard themselves anytime.

For more details, refer [Operator Onboarding Guide](#).



Application upload

- Application upload refers to upload of supervisor reviewed registration packets (approved and rejected). From this page, the operator can export the packets to any location on their machine on clicking **Save to Device** button.
- Upload of registration packet from this page internally performs two operations in sequence:
 - Sync registration metadata to server
 - On successful sync of metadata, actual registration packets are uploaded to the server.

Client Status: This column displays the status of a registration packet based on the above mentioned operation.

1. APPROVED
2. REJECTED
3. SYNCED
4. EXPORTED

Server Status:

On success,

1. PUSHED
2. PROCESSED
3. ACCEPTED

On failure,

1. REREGISTER
2. REJECTED
3. RESEND

Registration Type

This column displays the type of registration packet (New packet, Lost packet, Update packet, Correction packet)

Center Remap Sync

- When a machine is re-mapped from one center to another center, all the pending activities in the machine related to the former center needs to be completed.
- On successful completion of pending tasks, the former center's details will be deleted from the local Derby DB and a full sync will be initiated to pull in the new center details.

What are the pending tasks related to a center?

- Packet Approvals/rejections
- Packet upload
- Server confirmation on receiving a packet
- Deletion of packets after receiving a confirmation
- Deletion of pre-registration packets
- Deletion of center specific data like the public/policy key

Note- After completing the above tasks, a *restart* will be prompted to initiate the full sync with new center details.

Check Updates

- Clicking on this button, triggers a check for any new client version availability in the upgrade server.
- The machine must be online to be able to check updates.
- If there is any new version available, a confirmation pop-up is displayed to the operator for starting the upgrade or a reminder is displayed.

Registration Tasks

The operator can initiate any task from amongst- New registrations, Update UIN, Lost UIN, Correction flow. To get started, the operator needs to select a language for data entry. The number of languages displayed in the UI is configurable and depends on the country.

New registration

An operator can initiate the process of registering a new applicant in the MOSIP ecosystem by filling the new registration form with the resident.

The screenshot shows a web-based registration interface. At the top, it displays 'Test user' and 'Souk Khemiss Mograne (1... azm30584)'. Below this, there are tabs for 'Consent', 'Demographic Details', 'Document Upload', and 'Biometric Details'. The 'Consent' tab is active. The main content area is titled 'English' and contains a text box with the following text:

I understand that the data collected about me during registration by the said authority includes

- Name
- Date of birth
- Gender
- Address
- Contact details
- Documents
- Biometrics

I also understand that this information will be stored and processed for the purpose of verifying my identity in order to access various services, or to comply with a legal obligation. I give my consent for the collection of this data for this purpose

I have read and accept terms and conditions to share my PII *

CONTINUE

Below are few of the processes that needs to be completed for a new registration.

1. **Capture consent**- For every registration, the registration client provides an option for the operator to mark an individual's consent for data storage and utilization.
2. **Enter demographic data and upload documents** If the resident has a pre-registration application ID, the operator can auto-populate the demographic data and the documents by entering the pre-registration ID. If the resident does not have a pre-registration ID, the operator can enter the resident's demographic details (such as Name, Gender, DOB, Residential Address, etc.) and upload the documents (such as Proof of Address, Proof of Identity, Proof of Birth) based on the [ID Schema](#) defined by the country.

3. **Capture biometrics of a resident** The capture of biometrics is governed by the country, i.e. capture of each modality (fingerprint, iris or face) can be controlled by the country using the global configuration. When the operator clicks on the **Capture** button and tries to capture the biometrics of the resident, the device needs to make the capture when the quality of the biometrics is more than the threshold configured by the country. The device will try to capture the biometrics until the quality threshold has crossed or the device capture timeout has crossed which is also configurable.

After the timeout has occurred and the captured quality of biometrics is less than the threshold, registration client provides an option to re-try capture of biometrics but for a particular number of times which is also configurable. If the resident has a biometric exception (resident is missing a finger/iris or quality of finger/iris is very poor) the operator can mark that particular biometrics as **exception** but the operator has to capture the resident's exception photo.

What is the difference between an adult' and an infant' biometric capture?

- For an adult, all the configured biometrics can be captured.
- For an infant, by default, only the face biometrics is allowed to be captured.

1. Concept of biometric exception

- Permanent or temporary missing / defective fingers or irises can be marked as exception during registration process.
- Marked exception finger / iris names are sent as part of `rcapture` request to SBI.
- A photo of resident is captured highlighting his/her biometric exceptions called as Proof of exception (POE).
- Biometric exception photo is captured by the biometric face camera device.
- Until 1.2.0, POE was collected only as `documentType` field. From 1.2.0.1, Captured biometric exception photo is stored in the biometric data file (CBEFF xml file).

Update UIN

When a resident visits the registration center to update their demographic or biometric details, the operator captures the updated data as provided by the resident in the registration client. The resident needs to give their UIN and also select the field(s) that needs an update. The image below gives an idea of the update UIN process Flow in the registration client.

The screenshot shows a web-based registration interface. At the top, it says "Test user" and "Souk Khemiss Mograne (1... a2m30584)". Below this is a form with a text input field labeled "Enter the UIN number". Underneath, there's a section titled "Select the details that need to be updated". A grid of checkboxes lists various personal details. The checked boxes are "DataOfBirth" and "AddressInfo". Other options include "CivilianName", "ResidenceStatus", "IntroducerDetails", "PersonalInfo", "Phone", "CivilFirstName", "Notification Language", and "Location". Unchecked boxes include "Biometrics", "Email", "Gender", "monthOfBirth", "Consent", "dayOfBirth", "Full Name", "Documents", and "yearOfBirth". At the bottom right of the form is a blue "CONTINUE" button.

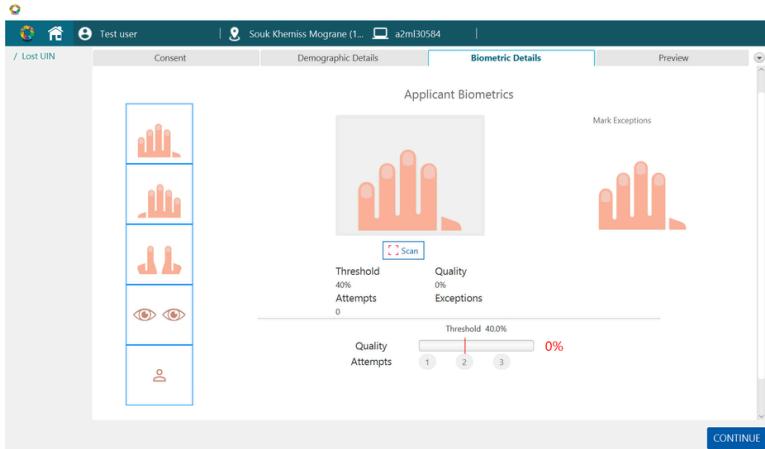
- (i) *The UIN update feature is configurable by the adopters. The Admin can turn ON or OFF, the UIN update feature using the configuration.*

Lost UIN

There are two situations where a Lost UIN flow is used. I am listing here the situations.

- The registrant has lost their Identity details.
- The registrant never received his identity due to a wrong address/email/phone

The registrant visits the registration centre to retrieve the same. The operator then captures the biometrics and the demographic details of the individual and processes a request to retrieve the lost UIN. As biometrics play a crucial role in identifying a person's identity, it is mandated to provide the biometrics as a part of the Lost UIN flow. Other details like demographic data, and uploading documents are optional.

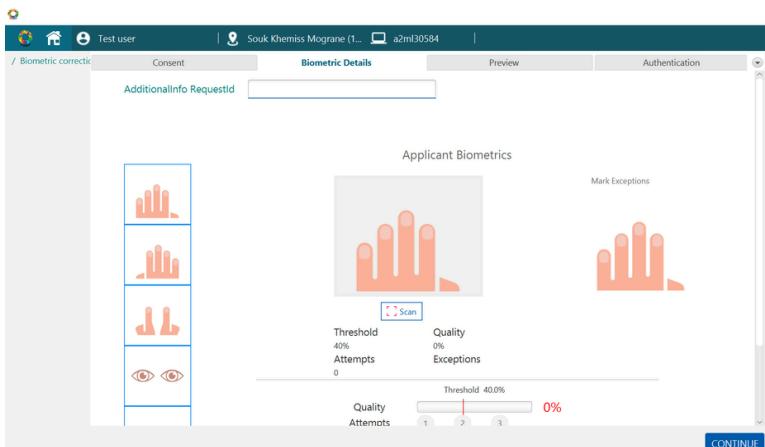


- As a part of Lost UIN flow, the contact details like the phone number/ e-mail address of the UIN holder can also be **updated** and stored in the ID Repository based on the value provided for the property `uingenerator.lost.packet.allowed.update.fields` which is specified in the Registration Processor properties.
- If the value already exists for the above mentioned property and once the lost UIN is found, details of the identity of the individual are sent to the newly provided phone number/ e-mail address.

Example: `uingenerator.lost.packet.allowed.update.fields= phone,email`

Correction process

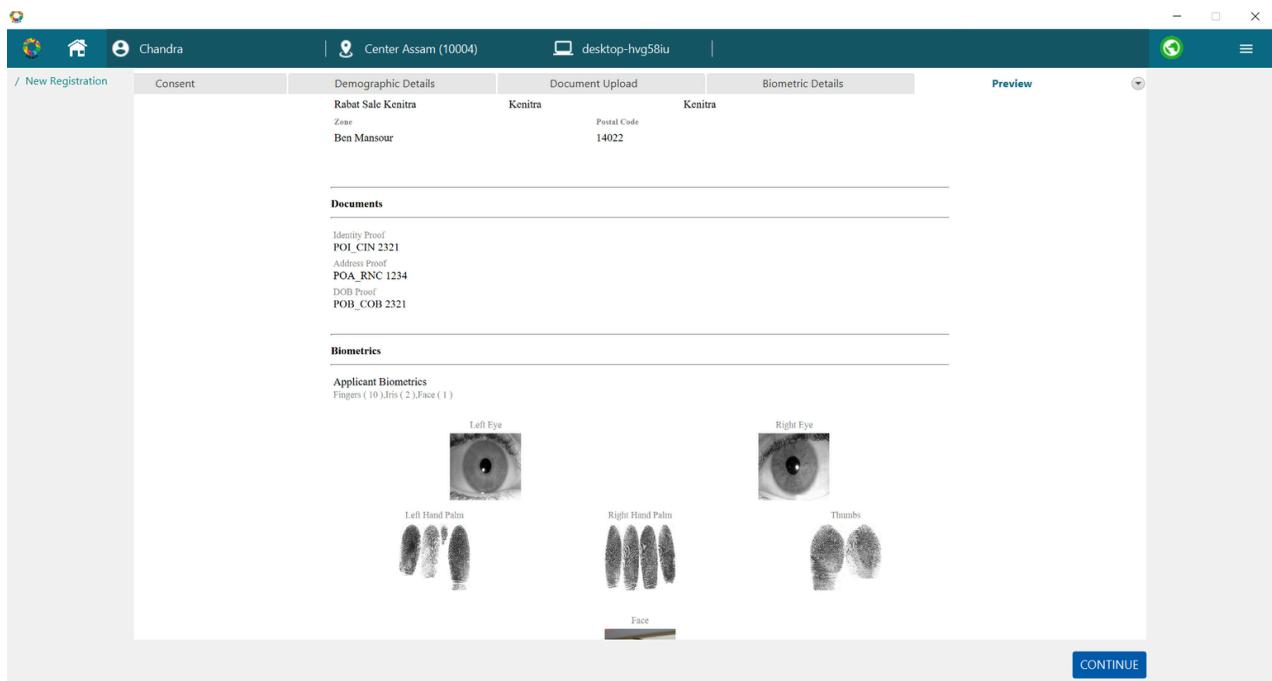
For a resident whose UIN is yet not generated, they can get a request intimation to re-provide their details with a RequestId. The same *AdditionalInfo RequestId* must be provided to the operator during the correction flow.



Note- The above mentioned Registration tasks are completely configurable through UI Specs<>

Preview and Packet authentication

- The operator can preview the data filled and navigate back to the respective tabs in case of corrections.
- Once the resident and the operator are satisfied with the data being captured, the operator can proceed to the Authenticate tab and provide his valid credentials to mark the complete of the registration task.



Acknowledgement receipt and printing

Once the registration process (new registration, UIN update or lost UIN, correction) is completed, the registration client generates an acknowledgement receipt. This receipt contains a QR code of the Registration application ID, captured demographic data in the selected language, photograph of the resident and ranking of each finger from 1 to 10 (1 being the finger with the best quality). This receipt is print friendly and can be used for printing using any printer.

The image below is that of a sample acknowledgement receipt.

The screenshot shows a registration acknowledgement page with the following details:

- Demographic Information:**
 - Full Name: Sekharl
 - Date Of Birth: 1992/01/01
 - Gender: Male
 - Phone: 0099998888
 - Email: chaitu636@gmail.com
- Address:**

Address Line 1 Addr	Province Kenitra	City Kenitra
Region Rabat Sale Kenitra		
Zone Ben Mansour	Postal Code 14022	
- Documents:**
 - Identity Proof
POL_CIN
 - Address Proof

New Registration button is visible at the bottom right.

End-of-day processes

Pending Approval

The Supervisor has the exclusive authority to approve/reject packets. The supervisor is supposed to manually re-verify the registrations before uploading them to the server. This page enables them to perform this activity.

Steps to approve/reject packets

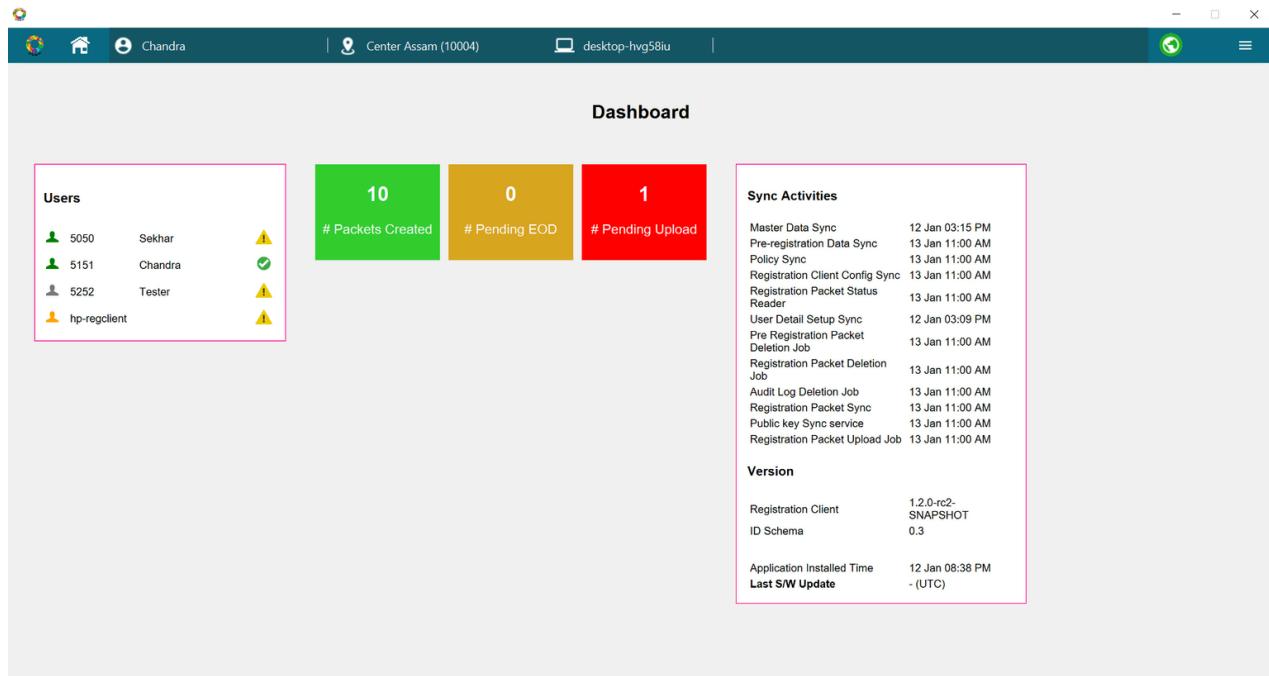
1. Click on any of the registrations listed in the left pane. The registration details are displayed on the right pane.
2. Supervisor needs to manually verify all the details in the right pane.
3. Supervisor can click **Approve/Reject** button based on their verification.
4. To mark the completion of this approval process, they need to click on **Authenticate** and provide their credentials.
5. On successful authentication, approved/rejected packets will be removed from here and be seen on the **Application Upload** page.

Re-registrations

All the registrations which are being marked with the RE-REGISTER status is listed here.

Dashboard

On clicking Dashboard, the Registration client dashboard HTML template is rendered. Default dashboard displays information about the operator, Packets and the Sync Activities.



News and updates

This section has been reserved for the countries to be able to display their live news and updates. This can be implemented as per a country's requirements.

Consent

During the training of the Operators, It must be ensured that consent is obtained directly from the resident whose personal information is being collected and processed. There is no technical way to handle this scenario, so the operator training must include this activity as a manual process and emphasize upon strictly following the same.

Deploy

Installation Guide

Overview

This guide helps the operator in setting up the registration client.

1.2 LTS Registration client UI Demo 08-02-2022



Know your machine TPM keys

A Trusted Platform Module (TPM) is a specialized chip on a local machines that stores RSA encryption keys specific to the host system for hardware authentication. The pair is maintained inside the chip and cannot be accessed by software. By leveraging this security feature every individual machine would be uniquely registered and identified by the MOSIP server component with it's TPM public key.

To extract the TPM public key, use the [TPM key extractor utility](#).

Prerequisites

To onboard the machine and the operator, the Admin needs to:

1. Create and activate the registration client machine using Admin portal.
2. Create a user/operator account in Keycloak
3. Assign the operator a role of either the Supervisor or Officer using the Admin portal.
4. Finally, perform the **User Zone mapping** and **User Center mapping** in the Admin portal.

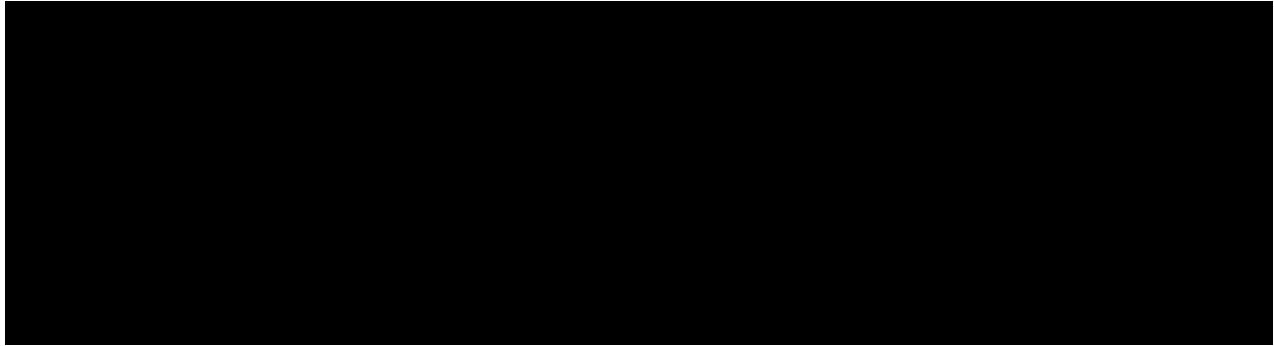
System prerequisites:

- CPU - Dual Core Processor - 2GHZ
- RAM - 16 GB
- Local Storage Disk Space - 500 GB
- USB 2.0 ports or equivalent hub.
- Physical machine with TPM 2.0 facility.
- Windows OS [10 v]

Registration client setup

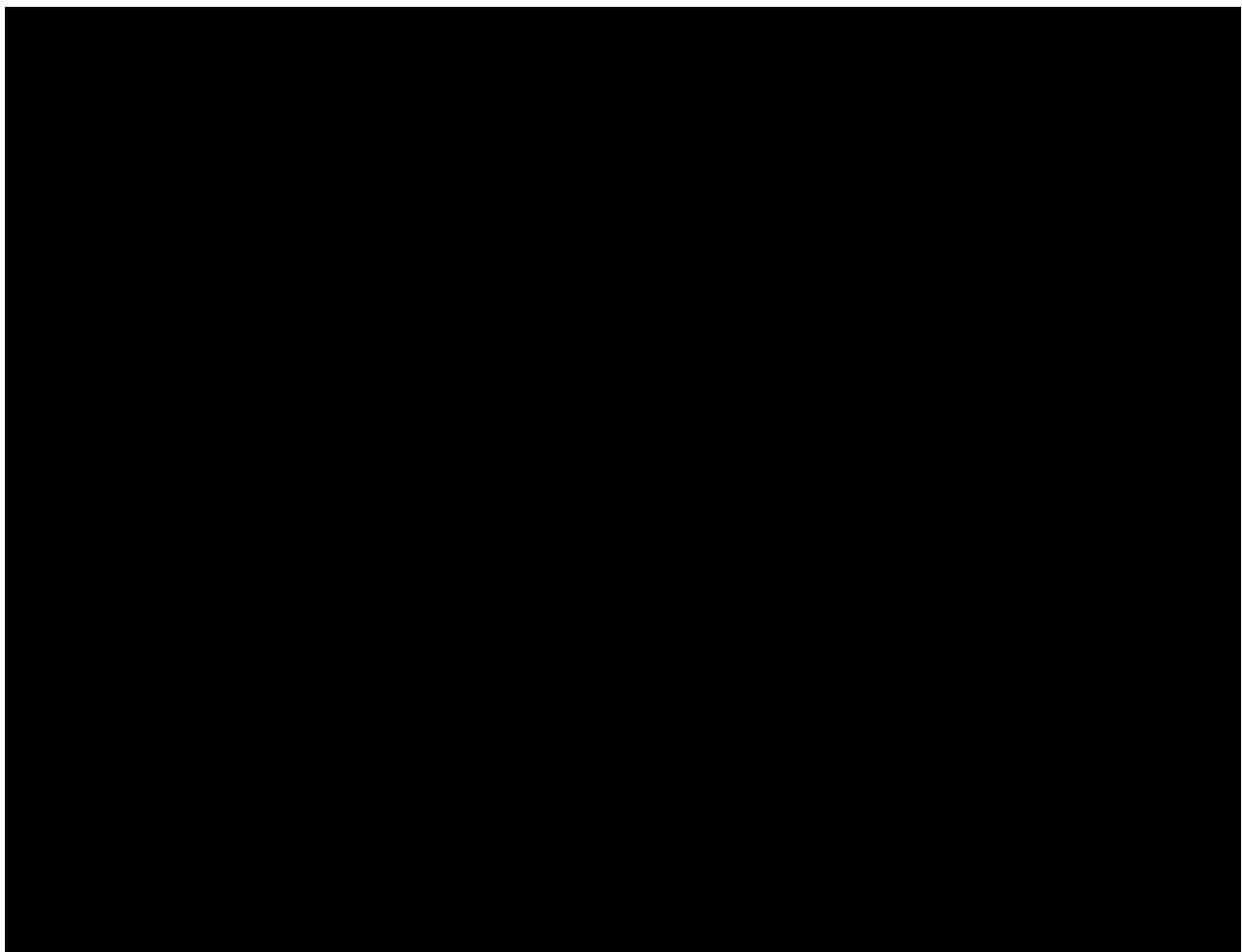
To setup the registration client:

1. Download the `reg-client.zip` from the hosted server.
2. Unzip the client. You can see the directory structure below.
3. Click `run.bat` to launch registration client.



Launching the registration client

The client always launches with the pre-loader screen which displays the information about the network status, build status verification, online status, etc.



1. First time launch

- After the pre-loader, the login screen is displayed.

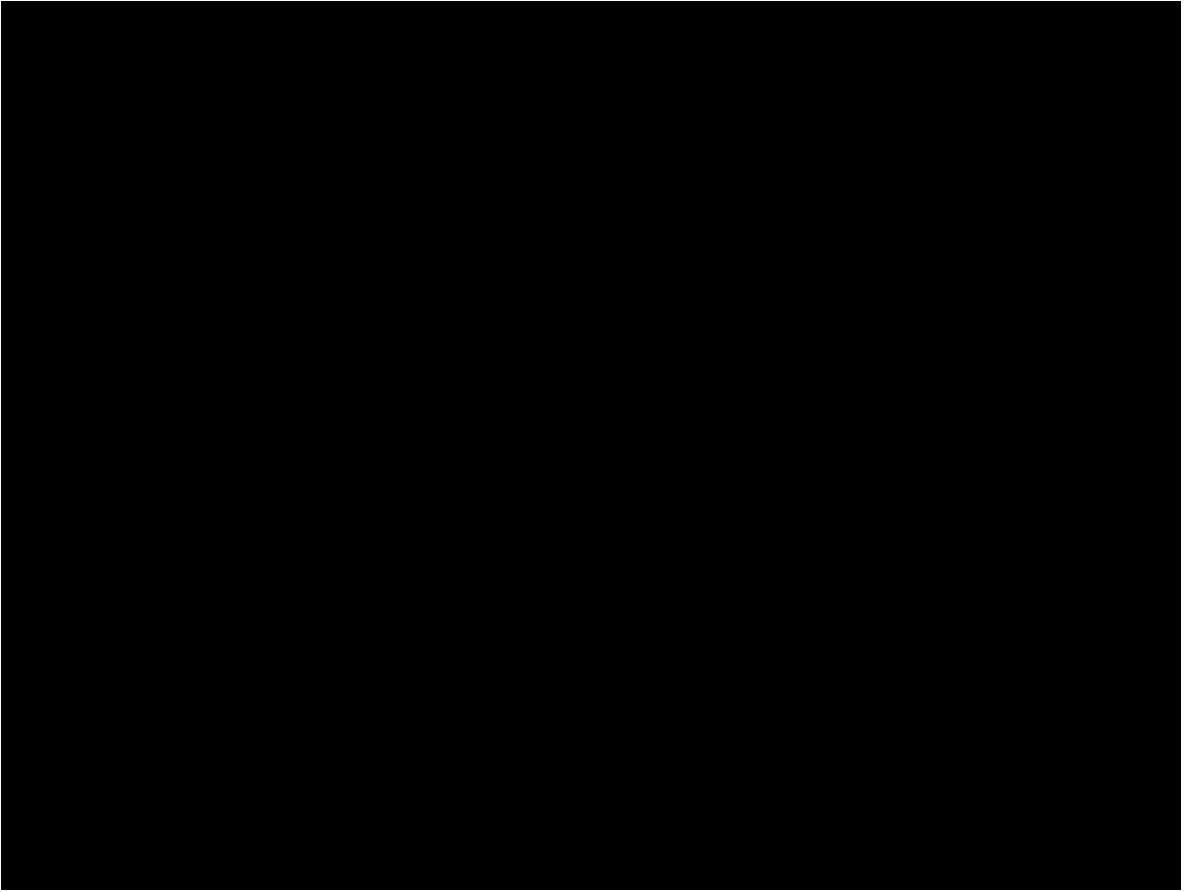
- Any valid operator credentials can be provided to authenticate and start the initial sync.
- On successful intial sync, the operator will be prompted to **restart** the application.
- After the first launch, the operator can notice **.mosipkeys** and **db** folders created under the registration client setup folder.

Note: Deletion of either the **.mosipkeys** or the **db** folder makes the application get into an invalid state and hence will fail to launch. To be able to launch the client again, the operator should make sure that both the folders are removed and then re-launch the client.

Name	Date modified	Type	Size
.mosipkeys	1/7/2022 5:07 PM	File folder	
db	1/7/2022 5:07 PM	File folder	
jre	1/5/2022 4:45 PM	File folder	
lib	1/5/2022 4:46 PM	File folder	
logs	1/10/2022 12:51 PM	File folder	
applicanttype.mvel	1/10/2022 12:38 PM	MVEL File	8 KB
bioCorrectionProcess	1/10/2022 12:38 PM	JSON File	5 KB
logback	1/5/2022 4:46 PM	XML Document	2 KB
lostProcess	1/10/2022 12:38 PM	JSON File	33 KB
MANIFEST.MF	1/10/2022 12:41 PM	MF File	51 KB
newProcess	1/10/2022 12:38 PM	JSON File	51 KB
run	1/5/2022 4:46 PM	Windows Batch File	1 KB
SCHEMA_0.3	1/10/2022 12:38 PM	JSON File	9 KB
startup	1/10/2022 12:41 PM	Text Document	0 KB
updateProcess	1/10/2022 12:38 PM	JSON File	51 KB

1. On the next launch after the initial sync,

- The registration client login page provides the operator an option to select the language for viewing the registration client UI.
- After successful login, the operator either lands into the operator onboard page or the home page.



For more details on operator onboarding, refer to [Operator onboarding guide](#).

For more details on Home page, refer to [Registration client home page](#).

Modes of operation

- **Offline**- Operator can use the registration client in offline mode to only do the registrations and EOD process. During offline mode, the operator authentication will be based on locally saved password hash. An operator can work in offline mode only if they have logged into to the registration client being online atleast once.
- **Online**- Machine must be online for the registration client first launch. For any server-client sync or vice-versa, the registration client must be online. In the online mode, the client reaches out to the server for password authentication.

Note: On successful onboard of the operator, biometric templates of the operator are stored locally. Biometric authentication does not reach out to the server

everytime, instead it is validated based on the locally stored templates on the registration client machine.

Setting up MOCK SBI (MDS)

In the development environment, Registration client can be tested using mock SBI. Find the instructions to build and run the mock SBI, click [here](#).

Troubleshooting

1. Incorrect username/password

- > Cross-check the machine keys mapping in server ('Machine not found' error)
- > Cross-check machine status
- > 'Invalid Request' error in log - Check your machine time, it shouldnt be off by more than 10 minutes
- > check logs/registration.log for more details

2. Configuration / masterdata Sync failed

- > check if kernel-syncdata-service is up.

Operator Onboarding

This guide contains all the details you may want to know about the operator onboarding.

Creating the first operator in MOSIP

To generate the first operator in MOSIP eco-system, refer to the steps below.

The Admin needs to:

1. Create the role **Default** in KeyCloak with all the other roles.
2. Create the operator' user account in KeyCloak.
3. Assign the operator user account with the **Default** role.
4. Perform Zone and Center mapping for the operator using the Admin Portal.
5. Onboard the operator machine using the Admin Portal. Machine' details can be extracted using the [TPM utility](#)

The operator will need to:

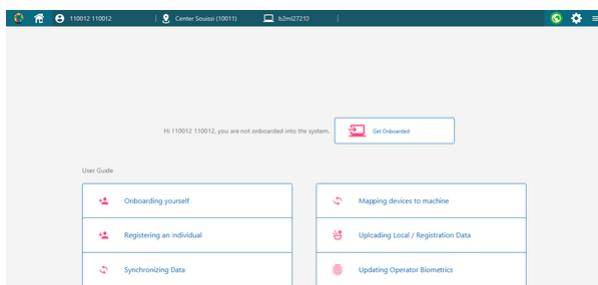
1. Download the latest registration client and login with the credentials set in KeyCloak. The operator will automatically skip Operator/Supervisor onboarding and reaches the home page of the registration client.
2. Register themselves in MOSIP and get a RID and UIN.

Once the operator is registered:

- The Admin changes the role of the operator to either **REGISTRATION_OFFICER** or **REGISTRATION_SUPERVISOR**.
- Deletes the role **Default** from KeyCloak so that no other user has the role Default.
- This operator can now register and onboard other Supervisors and Officers.

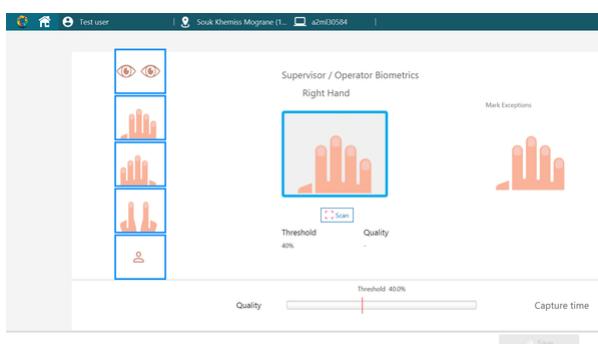
On-boarding an operator

- Admin needs to map the operator' UIN in Keycloak under Attributes with attribute name as `individualId`.
- Admin needs to remove the "Default" role mapping for the operator' user account if it exists.
- The operator needs to login (password based) to the Registration Client using Keycloak credentials.
- The operator needs to ensure that the Registration Client machine is online.
- The operator will land into the below page and needs to click on **Get Onboarded**



- The operator needs to provide their biometrics and click **Save**.
- All the biometric modalities displayed in the Operator biometrics page must be captured before clicking on Save.
- Captured biometrics quality must be greater than or equal to the threshold displayed in the UI.

Note- The threshold values are configurable and can be set as per the ID issuer.



- After successful onboarding, the operator is automatically re-directed to the [registration client home page](#).

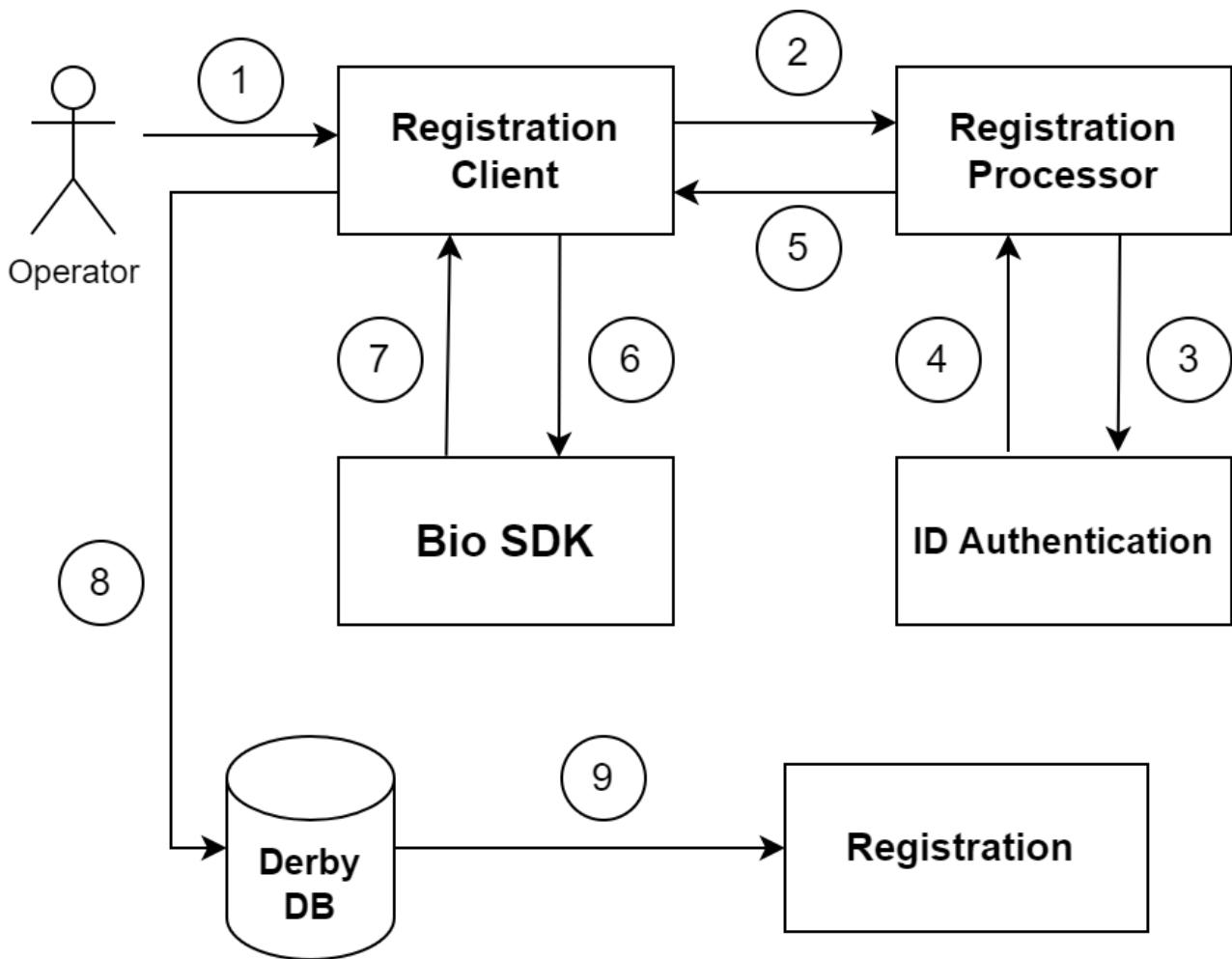
Note:

- After successful onboarding of the operator, the templates are extracted from the captured biometrics using configured Bio-SDK. The extracted templates are stored in Derby DB. This can be used later for operator' biometric-authentication and also for local de-duplication checks during registration.
- After the first login and successful on-boarding, the registration client would mandate the operator to login with the configured authentication mode decided by the administrator.
- Any number of operators can login to a registration client machine but they need to be mapped to the same center where the machine is onboarded.
- Login operator' user ID is case-insensitive.

Summarizing, on-boarding of an operator is successful only if,

- The operator is active and not block listed.
- The operator and the machine belongs to the same center.
- The operator's User ID is mapped to their UIN.
- The operator's biometric authentication is successful during on-boarding.
- The system is online during on-boarding.

Operator onboarding workflow



1. Operator logs into Registration Client for the first time and is redirected to Onboarding screen. Here, they need to capture all their biometrics and then click SAVE button.
2. Request from Registration Client goes to [Registration Processor](#) for operator authentication.
3. Registration Processor passes this request to [ID Authentication](#) where it checks whether the user is mapped to a valid UIN and then matches the biometrics sent in the request with the biometrics of the mapped UIN.
4. Success/Failure response sent back to Registration Processor based on the authentication result.
5. Registration Processor sends back this response to Registration Client.
6. After successful authentication, the captured biometrics are sent to configured Bio-SDK to extract templates.
7. Extracted templates are sent back from [Bio-SDK](#).
8. The extracted templates are stored in local Derby DB.

9. These templates stored in local DB can be used later for operator's biometric-authentication and also for local de-duplication checks during registration.

Modes of login

- MOSIP supports single factor and multi factor login including password, iris, fingerprint, and face authentication for registration client. An administrative configuration setting determines the mode of authentication login.
- The registration client can authenticate an operator in offline mode using the locally stored biometrics templates (face/finger/iris) and password hash.

Temporarily lock the operator

The registration client temporarily locks the operator's account in case they provides an invalid password/fingerprint/iris/face for X times continuously to login (X is configurable). The temporary account lock lasts for X minutes (X is again configurable).

Logout

An Operator can logout of the registration client by:

- Clicking on the *Logout* button,
- Closing the registration client,
- Being in-active on the registration client for configured amount of time after which they are automatically logged out.
- Upon logout, any unsaved data will be lost.
- Data will not be automatically saved in the database and will not be retained in memory though transaction details which is used for auditing will be captured and stored (except for PII data).

Note- Registration client provides an alerts to the operator 'X' minutes before reaching the auto logout time limit. Registration client displays a countdown timer in

the alert. The operator can choose to dismiss the alert and continue working. This will also reset the timer to zero.

Configuration Guide

Overview

The guide here lists down some of the important properties that may be customized for a given installation. Note that the listing here is not exhaustive, but a checklist to review properties that are likely to be different from default. If you would like to see all the properties, then refer to the files listed below.

 **IMPORTANT:**

From the LTS version, All the properties are synced to the registration-client only from `registration-default.properties` file.

Configuration files

```
application-default.properties  
registration-default.properties
```

See [Module Configuration](#) for the location of these files.

SBI related configurations

Registration Client reaches SBI on 127.0.0.1 within the below configured port range. As per SBI spec, the allowed port range is from 4501 to 4600.

```
mosip.registration.mdm.portRangeFrom=4501  
mosip.registration.mdm.portRangeTo=4600
```

Timeouts in milliseconds are set during any http calls to SBI.

```
mosip.registration.mdm.connection.timeout=10000  
mosip.registration.mdm.RCAPTURE.connection.timeout=40000  
mosip.registration.mdm.MOSIPDINFO.connection.timeout=5000  
mosip.registration.mdm.MOSIPDISC.connection.timeout=5000
```

Quality score threshold based on modality, Possible values 1 to 100

```
mosip.registration.leftslap_fingerprint_threshold=40  
mosip.registration.rightslap_fingerprint_threshold=40  
mosip.registration.thumbs_fingerprint_threshold=40  
mosip.registration.iris_threshold=60  
mosip.registration.face_threshold=9
```

Retry attempts, Possible values 1 to 10

```
mosip.registration.num_of_fingerprint_retries=3  
mosip.registration.num_of_iris_retries=3  
mosip.registration.num_of_face_retries=3
```

Quality score threshold based on the modality for operator authentication, Possible values 1 to 100

```
mosip.fingerprint_authentication.quality_score=30  
mosip.iris_authentication.quality_score=30  
mosip.face_authentication.quality_score=30
```

SDK

Registration clients can be integrated with more than one bio-sdks. Possible values for "modality-name" are "finger", "iris" or "face".

- SDK implementation class full name

```
mosip.biometric.sdk.providers.<modality-name>.<vendor-name>.classname
```

- SDK API version

```
mosip.biometric.sdk.providers.<modality-name>.<vendor-name>.version
```

- SDK implementation class constructor args - comma separated

```
mosip.biometric.sdk.providers.<modality-name>.<vendor-name>.args
```

- SDK initialization args, this will be passed as initparams

```
mosip.biometric.sdk.provider.<modality-name>.<vendor-name>.<key1>=<value1>
```

- Quality threshold used by SDK to match modality

```
mosip.biometric.sdk.providers.<modality-name>.<vendor-name>.threshold
```

Example configurations are shown below for MOCK SDK named as mockvendor:

```
mosip.biometric.sdk.providers.finger.mockvendor.classname=io.mosip.mock.sdk
mosip.biometric.sdk.providers.finger.mockvendor.version=0.9
mosip.biometric.sdk.providers.finger.mockvendor.args=
mosip.biometric.sdk.providers.finger.mockvendor.threshold=60
mosip.biometric.sdk.providers.iris.mockvendor.classname=io.mosip.mock.sdk
mosip.biometric.sdk.providers.iris.mockvendor.version=0.9
mosip.biometric.sdk.providers.iris.mockvendor.args=
mosip.biometric.sdk.providers.iris.mockvendor.threshold=60
mosip.biometric.sdk.providers.face.mockvendor.classname=io.mosip.mock.sdk
mosip.biometric.sdk.providers.face.mockvendor.version=0.9
mosip.biometric.sdk.providers.face.mockvendor.args=
mosip.biometric.sdk.providers.face.mockvendor.threshold=60
```

On every successful biometric capture during registration, the Quality of the biometrics is computed by bio-sdk if below config is enabled. Possible values are Y/N.

```
mosip.registration.quality_check_with_sdk=Y
```

Batch size

Jobs like RID sync, packet upload, and status sync are carried out in batches, number of registration records to be executed in a batch on every trigger.

```
mosip.registration.rid_sync_batch_size=5  
mosip.registration.packet_upload_batch_size=5  
mosip.registration.status_sync_batch_size=5
```

Operator onboarding bio-attributes

Only the modalities configured will be collected during operator onboarding.

```
mosip.registration.operator.onboarding.bioattributes=leftLittle,leftRing,
```

Pre-Registration sync

On every Pre-Registration application fetch in the registration page, clear all the captured data before the Pre-Registration application fetch. Set the field IDs which should not be cleared after the Pre-Registration application fetch. It is a comma-separated list of field ids as per UI-SPEC.

```
mosip.registration.fields.to.retain.post.prid.fetch=consent,consentText,p
```

Storage Location to store the downloaded Pre-Registration Packets in the local system

```
mosip.registration.registration_pre_reg_packet_location=PreRegPacketStore
```

Pre-registration applications fetch period, No. of days before the appointment date.

```
mosip.registration.pre_reg_no_of_days_limit=7
```

Scheduled Jobs

Comma-separated list of offline job IDs. Offline jobs are jobs that are not part of manual sync.

```
mosip.registration.jobs.offline=DEL_J00013, RDJ_J00010, ADJ_J00012, PVS_J0001
```

Comma separated list of untagged job IDs. Untagged jobs, which will be not part of manual sync but only from the scheduler.

```
mosip.registration.jobs.unTagged=PDS_J00003
```

Comma separated list of job IDs that need Registration Client restart.

```
mosip.registration.jobs.restart=RCS_J00005
```

Registration batch jobs scheduler.

```
mosip.registration.jobs.scheduler.enable=Y
```

Default CRON expression for scheduling the Jobs.

```
mosip.registration.sync_jobs_restart_freq=0 0 */11 ? * *
```

Document scan

All the identified scanner implementations will be used to list the identified devices. For each device dpi, width and height can be configured. If it is not configured, it defaults to 0.

Values in this config `mosip.registration.docscanner.id` map support regex.

```
mosip.registration.docscanner.id={ "id1" : "STUB-SCANNER", "id2" : "S600"  
mosip.registration.docscanner.dpi={ "id1" : 200, "id2" : 300 }  
mosip.registration.docscanner.width={ "id1" : 200, "id2" : 350 }  
mosip.registration.docscanner.height={ "id1" : 200, "id2" : 400 }
```

GPS Device Connection

- Enable GPS device for capturing the geo-location. If y, the GPS device will be enabled. If n, the GPS device will be disabled.

```
mosip.registration.gps_device_enable_flag=N
```

- Model of the GPS Device

```
mosip.registration.gps_device_model=GPSBU343Connector
```

- Timeout for connecting to GPS device

```
mosip.registration.gps_port_timeout=1000
```

- GPS Serial Port in Linux machine

```
mosip.registration.gps_serial_port_linux=/dev/ttyusb0
```

- GPS Serial Port in Windows machine

```
mosip.registration.gps_serial_port_windows=
```

- The Distance Parameter for GPS Verification

```
mosip.registration.distance.from.machine.to.center=900000
```

Other configurations

Resetting Password in Registration Client

To reset a password in the Registration Client, click **Reset Password** from the **Actions** menu in the top-right corner of the **Home** page. This redirects the operator to a configurable URL:

```
mosip.registration.reset_password_url=${mosip.api.internal.url}/keycloak/
```

 **Note:** The placeholder `"mosip.api.internal.url"` should be defined in `application-default.properties`.

Supervisor Packet Approval Configuration

This configuration determines whether supervisor approval is required before the Sync and Upload of registration packets.

- If **enabled (Y)**, the system requires a supervisor to review and approve the registration packets before it is synched and uploaded.

```
mosip.registration.supervisor_approval_config_flag=Y
```

- If **disabled (N)**, the registration proceeds auto approving, and packets are automatically uploaded in the next scheduled job.

Additionally, the system will cross-check the resident's biometrics with locally stored operator biometric templates to verify the registration.

```
mosip.registration.mds.deduplication.enable.flag=N
```

Minimum disk space that should be available in the machine to proceed with registration - in MB

```
mosip.registration.disk_space_size=5
```

Location to store registration packets in the client machine:

```
object.store.base.location=packets
```

Number of days allowed to start Registration Client without upgrade when software upgrade is available.

```
mosip.registration.softwareUpdateCheck_configured_frequency=60
```

Time in Seconds for forced log-out of the operator, if the operator is idle for the specified duration

```
mosip.registration.idle_time=900
```

Time in Seconds to display the warning message pop-up to the operator, if the operator is idle for the specified duration

```
mosip.registration.refreshed_login_time=600
```

Maximum no. of days for approved packet pending to be synced to a server beyond which Registration Client is frozen for registration

```
mosip.registration.last_export_registration_config_time=100
```

Maximum no. of packets pending EOD approval beyond which Registration Client is frozen for registration

```
mosip.registration.reg_pak_max_cnt_apprv_limit=100
```

Enable supervisor authentication feature. If y, supervisor approval will be enabled, else, will be disabled

```
mosip.registration.supervisor_approval_config_flag=Y
```

No. of days beyond audit creation date to delete audits

```
mosip.registration.audit_log_deletion_configured_days=10
```

No. of days beyond the registration date to delete synced and uploaded registration packet:

```
mosip.registration.reg_deletion_configured_days=1
```

No. of days beyond the appointment date to delete unconsumed pre-registration application data

```
mosip.registration.pre_reg_deletion_configured_days=1
```

The maximum duration to which registration is permitted without sync of master data

```
mosip.registration.sync_transaction_no_of_days_limit=5
```

Allowed a number of invalid login attempts:

```
mosip.registration.invalid_login_count=50
```

Used to configure the time (in minutes) for locking the account after crossing the configured invalid login count

```
mosip.registration.invalid_login_time=2
```

Configuration is used to check if any sync job is missed/failed beyond expected days, this configuration is checked every time the operator clicks on any registration process. We follow the below convention to create this config key.

```
mosip.registration.job api name as in sync_job_def table.frequency=value  
in days
```

#Maximum no. of days without running the Master Sync Job beyond which Registration Client is frozen for registration

```
mosip.registration.masterSyncJob.frequency=190
```

#Maximum no. of days without running the Pre-Registration Sync Job beyond which Registration Client is frozen for registration

```
mosip.registration.preRegistrationDataSyncJob.frequency=190
```

#Maximum no. of days without running the Packet Sync Status Job beyond which Registration Client is frozen for registration

```
mosip.registration.packetSyncStatusJob.frequency=190
```

#Maximum no. of days without running the Key Policy Sync Job beyond which Registration Client is frozen for registration

```
mosip.registration.keyPolicySyncJob.frequency=190
```

#Maximum no. of days without running the Registration Deletion Job beyond which Registration Client is frozen for registration

```
mosip.registration.registrationDeletionJob.frequency=190
```

#Maximum no. of days without running the Configuration Sync Job beyond which Registration Client is frozen for registration

```
mosip.registration.synchConfigDataJob.frequency=190
```

#Maximum no. of days without running the Audit Logs Deletion Job beyond which Registration Client is frozen for registration

```
mosip.registration.deleteAuditLogsJob.frequency=190
```

Date formats

Date format to be displayed on acknowledgment slip, default value - dd/MM/yyyy hh:mm a

```
mosip.registration.application_date_format
```

Date format to be displayed on Registration Client dashboard, default format - dd MMM hh:mm a

```
mosip.registration.dashboard_date_format
```

Settings page

Settings Schema

Default settings schema is configured as below:

```
[  
 {  
   "name": "scheduledjobs",  
   "description": {  
     "ara": "إعدادات الوظائف المجدولة",  
     "fra": "Paramètres des travaux planifiés",  
     "eng": "Scheduled Jobs Settings"  
   },  
   "label": {  
     "ara": "إعدادات الوظائف المجدولة",  
     "fra": "Paramètres des travaux planifiés",  
     "eng": "Scheduled Jobs Settings"  
   },  
   "fxml": "ScheduledJobsSettings.fxml",  
   "icon": "scheduledjobs.png",  
   "order": "1",  
   "shortcut-icon": "scheduledjobs-shortcut.png",  
   "access-control": [  
     "REGISTRATION_SUPERVISOR"  
   ]  
 },  
 {  
   "name": "globalconfigs",  
   "description": {  
     "ara": "إعدادات التكوين العامة",  
     "fra": "Paramètres de configuration globale",  
     "eng": "Global Config Settings"  
   },  
   "label": {  
     "ara": "إعدادات التكوين العامة",  
     "fra": "Paramètres de configuration globale",  
     "eng": "Global Config Settings"  
   },  
   "fxml": "GlobalConfigSettings.fxml",  
   "icon": "globalconfigs.png",  
   "order": "2",  
   "shortcut-icon": "globalconfigs-shortcut.png",  
   "access-control": [  
     "REGISTRATION_SUPERVISOR"  
   ]  
 },  
 {  
   "name": "devices",  
   "description": {  
     "ara": "إعدادات الجهاز",  
     "fra": "RégLAGes de l'appareil",  
     "eng": "Device Settings"  
   },  
 }]
```

```
"label":{  
    "ara": "إعدادات الجهاز",  
    "fra": "Réglages de l'appareil",  
    "eng": "Device Settings"  
},  
"fxml": "DeviceSettings.fxml",  
"icon": "devices.png",  
"order": "3",  
"shortcut-icon": "devices-shortcut.png",  
"access-control": [  
    "REGISTRATION_SUPERVISOR",  
    "REGISTRATION_OFFICER"  
]  
}  
]
```

Clicking on  in the home page opens a pop-up displaying configured **Settings** page as per the above sample settings schema.

settings pop-up

Device settings

- All the connected devices are listed in this page.
- Option to scan for SBI for any specific port range is available in this page.

- If more than one device is identified, operator can choose amongst the listed devices to set the default device for the current login session.
- Access control on this page is controlled via the settings-schema.

Global configuration

- All the Registration Client related configuration key-value pairs are listed in this page.
- Operator can set the local preference on the server configraton value, applicable only for permitted configuration keys.
- Access control on this page is controlled via the settings-schema.

Scheduled Jobs

- All the available background jobs are listed here along with their cron expression.
- Every job's next and previous trigger time is listed along with the job name.
- Privileged operator can update the cron expression of any job.
- `Synchornize Data` in home page will trigger all of these listed jobs with just one click.
- If the operator needs to trigger specific job, the same can be handled in this page.
- Access control on this page is controlled via the settings-schema.

Telemetry from Registration Client

Following are the metrics that are collected in the client application using the micrometer library:

- JVM Memory Metrics
- JVM Thread Metrics
- JVM GC Metrics
- JVM Heap Pressure Metrics
- Processor Metrics
- Class Loader Metrics
- Disk Metrics
- Packet Metrics based on client and server status

All the metrics collected are appended to `metrics.log` file. Rolling policy of the `metrics.log` is defined in registration-services `logback.xml`.

Below are the challenges faced in exporting the collected metrics from client application to the server for further analysis:

1. Unreliable network conditions on field.
2. Metrics files are mostly large files, and cannot afford retries on failed attempts.
3. Required HTTP based metrics export.

To overcome the above challenges, Registration Client is built with `tus-java-client` (version: 0.4.3) . Tusd server URL and the upload chunk-size are made configurable in the client application.

`mosip.registration.tus.server.url`: This is the server URL config which specifies to which URL the metrics files are to be uploaded.

`mosip.registration.tus.server.upload.chunksize`: This config defines the chunk size, which means, how much size of the file is to be uploaded at once. By default, this is given as 1024, which means 1KB

Note:

- The `Tus protocol` is designed to enable resumable uploads of large files over HTTP, which can be useful for web applications that need to handle file uploads in unreliable network conditions or with large files that might take a long time to upload. For more information on TUS, refer [here](#).
 - `Tusd` is a popular implementation of the Tus protocol that can be used as a standalone server. It is a part of the MOSIP deployment.

Each metric json logged into `metrics.log` file is tagged with machine name. Refer the below log lines with the machine names.

- A job is scheduled to upload collected metrics to server from the client application.
 - Job runs with a fixed delay of 15 minutes.
 - Resumable file URLs are stored under `.metrics` folder of registration client. Once the complete file is uploaded to server, the metrics file is deleted locally.

Android Registration Client

Overview

Overview

The Android Registration Client is a tablet application that serves as a portable version of the existing desktop [Registration Client](#). It has been developed to support accessibility on all Android devices. The creation of the Android Registration Client was driven by the need to meet the mobility requirements of countries adopting MOSIP.

The primary objective of the tablet version is to facilitate the registration process for residents, especially those who are unable to physically visit registration centers. It also serves remote locations where setting up registration centers is not feasible. To address this challenge, the Android Registration Client was created, enabling operators and supervisors to easily reach remote areas and maximize resident registrations across the country.

Features

The first developer release of the Android Registration Client offers the following key features:

1. **Operator/ Supervisor Login (offline and online):** Operators can securely login using their credentials, whether in offline or online mode, to carry out various registration transactions. To enable offline login, the operator must have previously logged in and synchronized their data over a network.
 2. **Multi-language Support:** The Android Registration Client supports multiple languages for content display and data entry.
- **Display Language:** Display Language refers to the language used for rendering UI elements such as labels and headings. With the Android Registration Client, Operators have the option to choose their preferred language for UI display. This language selection can be made on the login screen. Currently, the supported display languages include Arabic, French, and English.

- New languages can be added by following the below steps:
- Additional languages can be configured by adding localization files in lib/l10n folder present in the root project directory("android_registration_client").
- The languages that are rendered on the UI will be based on the country configuration (after master data sync). The default display language is English. Other languages will be available in the UI after the master data sync.

To know more, refer to [Flutter doc-Internationalizing Flutter apps](#).

- **Data Entry Language:** The Data Entry Language refers to the specific language utilized by the Operator while gathering data, which is then stored on the server in that selected language. During the registration process, the Operator can choose the language preference for the data collected, allowing applicants to provide information in their desired language. This language selection option becomes available upon initiating a new registration. The responsibility for managing the data entry language lies within the UI Spec, and any modifications or changes can be made through that specification.
3. **Auto-Sync/ manual sync:** On launching the Android Registration Client and logging in for the first time, the system automatically syncs the following data:
- **Configuration sync:** Sync of properties which drives in deciding the ARC UI functionality. For example: Invalid login attempts, idle timeout, thresholds, etc.
 - **Masterdata sync:** As a part of this sync, supporting information like Dynamic field data, templates, locations, screen authorization, blocklisted words, etc. are pulled in.
 - **UserDetails sync:** userID, along with their status is synced. Only the user details belonging to machine mapped center will be synced.
 - **Certificate sync:** Certificates used to validate the server signatures, device CA certificates, and public key (specific to a center and machine, also called policy key) used to encrypt the registration packet will be synced.
4. **New Registrations:** Operators can register a resident using the [New Registration](#) feature. The registration process can be customized through the Android Registration Client [UI specification](#). The required data for registering an applicant are as follows:
- **Consent:** Before the registration process, applicants must provide consent to the terms and conditions presented on the consent screen. This explicitly

asks the applicant to grant permission for storing and using their Personally Identifiable Information (PII).

- **Demographic Details:** Once the consent is obtained, the Operator will enter the demographic data of the applicant in the language preferred by the applicant. This includes details such as their name, gender, date of birth, and residential address.
- **Documents Upload:** Following the completion of the demographic details, the Operator can select the document type, input the reference, and upload the supporting documents provided by the applicant. Supporting documents may include Proof of Address, Proof of Identity, and Proof of Birth, based on the country-specific requirements.
- **Biometrics:** After the documents have been uploaded, the Operator will proceed to capture the applicant's biometrics. The biometrics captured are as follows:

- Fingerprints
- Iris
- Photograph
- Exception photograph

The acquisition of biometric data is regulated by the country. The country has control over the capture of each type of biometric (fingerprint, iris, or face) through the global configuration. When the Operator selects the **Capture** button, the biometric SBI application is accessed to capture the biometrics.

Once the biometrics are obtained, the data and control are returned to the Android Registration Client. To obtain the resident's biometrics, the quality of the captured image must exceed the threshold specified by the country. The biometrics can be captured a set number of times if necessary to meet the quality threshold. In situations where none of the captured images meet the threshold, the image with the highest quality score will be saved.

If the resident has a biometric exception (such as a missing finger/eye or very poor finger/iris quality), the Operator can designate that particular biometric as an exception. However, the Operator must still capture the resident's exception photo.

- **Preview section:** The Operator can review the data entered by the applicant, including demographic information, uploaded documents, and captured

biometrics. This preview allows the Operator to ensure the accuracy of the entered data. If any mistakes are found, the Operator can easily go back to the corresponding section and make the necessary corrections. If the data is correct, the Operator can proceed to the next step, which is to authenticate themselves.

- **Operator authentication:** Once both the Operator and applicant have confirmed that the data is accurately filled, the Operator is required to authenticate themselves using their credentials. After a successful authentication, the data packets are created and only then the sync and upload operations can be performed.
- **Packet sync:** After the applicant's registration form has been completed and the Operator has authenticated themselves, a packet sync must be performed. This can be done either manually or as a background job(auto sync and upload of packets). Packet sync ensures that the packet is prepared for uploading and the status of the uploaded packet is synchronized with the server.
- **Packet Upload:** Once the packet sync is completed, the system will proceed to upload the packet to the server when an internet connection is available. If there is no network access, the system will attempt to upload the packet as soon as connectivity is established.
- **Acknowledgment section:** Following the completion of the new registration process, an acknowledgment receipt is generated. This receipt includes the AID(Application ID), captured demographic data in the selected language, a photograph of the resident, and a ranking of each finger from 1 to 10, with 1 representing the finger with the best quality. The receipt is designed to be easily printed.

exceptionalBiometrics exceptions: If the resident has a biometric exception (such as a missing finger/eye or very poor finger/iris quality), the Operator can designate that particular biometric as an exception. However, the Operator must still capture the resident's exception photo.

- **Preview section:** The Operator can review the data entered by the applicant, including demographic information, uploaded documents, and captured biometrics. This preview allows the Operator to ensure the accuracy of the entered data. If any mistakes are found, the Operator can easily go back to the corresponding section and make the necessary corrections. If the data is correct, the Operator can proceed to the next step, which is to authenticate themselves.

- **Operator authentication:** Once both the Operator and applicant have confirmed that the data is accurately filled, the Operator is required to authenticate themselves using their credentials. After a successful authentication, the data packets are created and only then the sync and upload operations can be performed.
 - **Packet sync:** After the applicant's registration form has been completed and the Operator has authenticated themselves, a packet sync must be performed. This can be done either manually or as a background job(auto sync and upload of packets). Packet sync ensures that the packet is prepared for uploading and the status of the uploaded packet is synchronized with the server.
 - **Packet Upload:** Once the packet sync is completed, the system will proceed to upload the packet to the server when an internet connection is available. If there is no network access, the system will attempt to upload the packet as soon as connectivity is established.
 - **Acknowledgment section:** Following the completion of the new registration process, an acknowledgment receipt is generated. This receipt includes the AID(Application ID), captured demographic data in the selected language, a photograph of the resident, and a ranking of each finger from 1 to 10, with 1 representing the finger with the best quality. The receipt is designed to be easily printed.
5. **Operator onboarding:** To log in to the Android Registration Client, the operator must complete the onboarding process. This functionality is available only during the first online login. The operator must onboard by capturing their fingerprints, thumbprints, iris, and face. Once these are captured, the operator can start registering residents and using other services.
6. **Dashboard:** The Operator can access the dashboard where he can view the following:
- a. **Packets created:** This will show the total number of packets created from the time the Android Registration Client was installed.
 - b. **Packets Synced:** This will show the total number of packets synced from the time the Android Registration Client was installed.
 - c. **Packets Uploaded:** This will show the total number of packets uploaded from the time the Android Registration Client was installed.
 - d. **User details:**
 - i. **User ID:** This will show the list of User IDs of the Users mapped to the device.

- ii. **Username:** This will show the list of usernames of the Users mapped to the device.
 - iii. **Status:** This will show the status of Users mapped to the device. This can take values such as onboarded, active, inactive, etc.
7. **Supervisor's Approval:** Once the packet is created by the Operator, as an additional check, the Supervisor will have to go through each application to make sure the details filled are coherent. At this stage, the Supervisor can either Approve the Application or he can Reject it. If the Supervisor decides to reject it, they also will have to mandatorily mention the reason for rejection. Once the Application has been Approved or Rejected, the Supervisor will have to authenticate himself by entering his Username and Password. Once they have successfully authenticated, the Application will be removed from the "Supervisor's Approval" section and will be moved to the "Manage Application" Section.

 This feature will only be available for users having "Supervisor" role.

- 5. **Manual Application upload/export:** Once the Application is either Approved or Rejected by the Supervisor, those packets can be uploaded to the server from the "Manage Application" section. If there is internet connectivity, the packet will be synced and uploaded to the server but in case of lack of internet connectivity, the User can also export the packet to their local device storage.
- 6. **Update UIN:** In a scenario where the Resident wants to update their data, they can do so by letting the Operator know their UIN and the data that needs to be updated. Residents can update their demographic details, documents, and biometrics using this feature.
- 7. **Logout:** Using this feature, once the user is done with their registration and other activities, they can logout. If no background tasks are running, the user will be immediately logged out. If there are tasks (like sync) running in the background, the user will be notified about the same. From here if the User wants to cancel the logout, the background activities will keep running whereas if the user chooses to logout, they will be logged out and the background activities will be terminated.
- 8. **Update operator's biometrics:** In a scenario where the operator wants to update his biometric section from operational tasks to update.

9. **Handles Feature:** The Handles Feature is designed to streamline citizen registration and authentication. During registration, specific attributes such as email, phone number, or national ID—can be designated as a handle. This handle serves as a unique identifier that can later be used for authentication for various services. Handles can also be used to update data in case of data discrepancies. By allowing flexible and secure identification, the feature enhances the accuracy and integrity of citizen records while simplifying user interactions with government systems.

Configuration Guide

To read through the comprehensive list of configurable properties for the Android Registration Client, refer [Android Registration Client Configuration Guide](#).

UI Specifications

For more details on UI specifications for the Android Registration Client, refer [here](#).

Compatibility

The Android Registration Client is compatible with the following MOSIP platform versions:

1. 1.1.5.x
2. LTS 1.2.0 and above

Develop

Developer Guide

The documentation here will guide you through the pre-requisites and the other necessary details required for Android Registration Client developer setup.

The android-registration-client repository contains the Android Registration Client software for MOSIP. The feature-flutter branch focuses on integrating Flutter into the client.

Setup

To set up the Android Registration Client with Flutter and Android Studio, follow the steps below:

Prerequisites

- Flutter SDK (3.10.4): Install Flutter by following the official [Flutter installation guide](#).
- Android Studio (or Any IDE of your choice): Download and install Android Studio from the official [Android Studio website](#).

Step 1: Clone the Repository

The `develop` branch of android-reg-client is currently being actively developed. If you wish to access this branch, you can clone the repository by executing the following command in your terminal. Alternatively, you can download one of the releases available in the repository's release section.

```
git clone -b feature-flutter https://github.com/mosip/android-registration-client
```

Active Branches:

- [release-0.11.x](#)(developer release branch)
- [develop](#)(active development branch)

Step 2: Set up Flutter in Android Studio

1. To begin, launch Android Studio.
2. Next, select **Open an existing Android Studio project** and navigate to the cloned repository.
3. Open the `android-registration-client` directory as a project in Android Studio.
4. In order to integrate Flutter with Android Studio, install the Flutter plugin by accessing `File > Settings > Plugins` and searching for **Flutter**. Proceed to click on **Install** to install the plugin.
5. To ensure proper functionality, configure the Flutter SDK path by navigating to `File > Settings > Languages & Frameworks > Flutter` and specifying the Flutter SDK path as the location where you have installed Flutter.
6. Finally, save the changes by clicking on the "Apply" button.

Customizing the Registration Client

- Styling of the application can be configured by modifying these files `lib/utils/app_style.dart, lib/utils/app_config.dart`
- Application language bundles can be added to this path `lib/l10n` After adding the bundle run the below command to generate Localization data (Required for the first time).

```
flutter gen-l10n
```

- The label and application logo can be changed here `android/app/src/main/AndroidManifest.xml`

Step 3: Build and Run the Application

- The `pigeon.sh` file consists of the necessary commands for downloading dependencies and generating Flutter - Android native communication code. Please execute the `pigeon.sh` file or execute the commands within the file separately.
- Ensure you have connected an Android device or initiated an Android emulator.

- Open the terminal within Android Studio or use an external terminal.
- Navigate to the `android-registration-client` directory.
- Run the following command to build and execute the application:

```
flutter run
```

Step 4: Build, debug, and release APK

Execute the commands below to debug and release the APK

```
// Debug APK  
flutter build apk --debug  
  
// Release APK  
flutter build apk --release
```

Set up Mock MDS for Biometric Scan

The Mock MDS tool can be utilized to simulate the functionalities of biometric devices. The Mock MDS application is compliant with CTK standards and can serve as a substitute for Android SBI modules during testing and validation.

1. Install the Mock MDS application.
2. Access the **Settings** menu.
3. Under Device Configuration, choose **Registration** from the dropdown menu.
4. In P12 Configuration:
 - Enter the necessary credentials for the Device Key and upload the Device P12 file.
 - Enter the required credentials for the FTM Key and upload the FTM P12 file.
 - Complete all fields in MOSIP IDA Configuration.
5. In Modality Configuration, specify the quality score for Face, Finger, and Iris scans(these values can also be adjusted during testing).
6. Click on the **Save** button.
7. Go back to the Home Page and select `LOAD AND VALIDATE CERTIFICATES`.

A toast message will be displayed indicating the success of the validation process.

Note: To view the released version of the Mock SBI APK, click [here](#).

To download the Mock SBI APK, click on [camera-mds.zip](#).

Contributions

If you would like to contribute to the Android Registration Client, please follow the guidelines outlined [here](#).

License

The Android Registration Client is licensed under the [MIT License](#).

Support

If you encounter any issues or have any questions, please open an issue on the [GitHub repository](#).

Sources

- [GitHub- mosip/android-registration-client](#): Reference Android Registration Client Software - WIP
- [Flutter- Get started: Install](#)
- [Android Studio- Download](#)

UI Specification

Registration Client UI Specifications

Overview

The registration UI forms are rendered using respective UI specification JSON. This is derived from the [ID Schema](#) defined by a country. Here, we will be discussing the properties used in the UI specification of the Registration Client.

In the Registration Client, currently, Registration Tasks(process) forms are configurable using the UI specifications.

Each process has multiple screens and each screen is rendered with one or more fields.

Process/ Task spec JSON template

```
{  
    "id": "<NEW/UPDATE/LOST/BIOMETRIC_CORRECTION process name as passed in  
    //order in which the process is displayed on the registration client |  
    "order": 1,  
    "flow": "<NEW/UPDATE/LOST/CORRECTION>,"  
    //Multi-lingual labels displayed based on the logged in language  
    "label": {  
        "eng": "Registration",  
        "ara": "تسجيل",  
        "fra": "Inscription"  
    },  
    //screen details - follows screen spec structure  
    "screens": [  
        {}  
    ],  
    //caption displays on-hover content  
    "caption": {  
        "eng": "Registration",  
        "ara": "تسجيل",  
        "fra": "Inscription"  
    },  
    //icon is the symbol that appears before the process label  
    "icon": "registration.png",  
    "isActive": true,  
    //group names that should be by default selected during UPDATE UIN pro  
    "autoSelectedGroups": [  
        ""  
    ]  
}
```

Screen spec JSON template

```
//Order of the screen on the registration page
"order": 1,
"name": "<unique identifier for the screen>",
"label": {
    "ara": "شاشة عينة",
    "fra": "Exemple d'écran",
    "eng": "Sample screen"
},
"caption": {
    "ara": "شاشة عينة",
    "fra": "Exemple d'écran",
    "eng": "Sample screen"
},
//field details - follows field spec structure
"fields": [
    {}
],
"layoutTemplate": null,
//displays field to provide pre-reg application ID, data fetched from
"preRegFetchRequired": true,
//enable below flag to capture additionalInfo request ID , applicable
"additionalInfoRequestIdRequired": false,
//show or hide screens
"active": true
}
```

Field spec JSON template

```
{  
    "id": "<Unique identifier for the field, must be same as that described in the UI>",  
    //inputRequired is used to identify if UI input is needed or not  
    "inputRequired": true,  
    //type defines the datatype of the field,which must be the same as the type in the UI  
    "type": "<string/simpleType/documentType/biometricsType>",  
    "controlType": "textbox/fileupload/dropdown/checkbox/button/date/ageDate/  
    //minimum- applicable only for date controlType(defined in days)  
    "minimum": 0,  
    //maximum- applicable only for date controlType(defined in days)  
    "maximum": 365,  
    "description": "<Field description>,"  
    "label": {  
        "ara": "حقل العينة",  
        "fra": "Exemple de champ",  
        "eng": "Sample Field"  
    },  
    //fieldType is used to identify if it is a dynamic field  
    "fieldType": "<default/dynamic>",  
    //to validate the format should be in upper or lower case  
    "format": "<lowercase/uppercase/none>",  
    //list of validators for the field  
    "validators": [  
        {  
            //type of validation engine (currently, only regex is supported)  
            "type": "regex",  
            //expression for the validation  
            "validator": "^(\\d{10,30})$"  
            //list of arguments needed for the validator  
            "arguments": [],  
            //if null, its applicable for all languages, else validator is language specific  
            "langCode": null,  
            /*error code to be used to display specific error message, if validation fails.  
             * There error codes must be configured in registration module  
             */  
            "errorCode": "UI_100001"  
        }  
    ],  
    //determines sharing and longevity policies applicable as defined in the UI  
    "fieldCategory": "<pvt/evidence/kyc>",  
    "alignmentGroup": "<fields belonging to same alignment group are placed together in the UI>,"  
    //determines when to display and hide the field(set null if the field is always visible)  
    "visible": {  
        "engine": "MVEL",  
        "expr": "identity.get('ageGroup') == 'INFANT' && (identity.get('isChild') == true || identity.get('isTeen') == true)"  
    },  
    "contactType": null,  
    //used to group together the list of fields(only applicable in the Update UIN process)  
    "group": "<grouping used in update UIN process>,"  
}
```

```
"groupLabel": {  
    "eng": "Sample group",  
    "ara": "مجموعة العينة",  
    "fra": "Groupe d'échantillons"  
},  
/*on change of the field value, configured Action will be triggered on  
     Change action handlers should be implemented in registration  
"changeAction": null,  
//enable or disable auto-transliteration  
"transliterate": false,  
/*provide the templateName(applicable only for html controlType field)  
     These templates should be configured in templates table*/  
"templateName": null,  
"fieldLayout": null,  
"locationHierarchy": null,  
//On any biometric exception, Need to capture exception photo as proof  
"exceptionPhotoRequired": true,  
/*applicable only for BiometricsType field, defines the list of attributes  
     All the supported biometric attributes are listed down for reference  
"bioAttributes": [  
    "leftEye",  
    "rightEye",  
    "rightIndex",  
    "rightLittle",  
    "rightRing",  
    "rightMiddle",  
    "leftIndex",  
    "leftLittle",  
    "leftRing",  
    "leftMiddle",  
    "leftThumb",  
    "rightThumb",  
    "face"  
],  
//capture of above mentioned bioAttributes can be conditionally mandatory  
"conditionalBioAttributes": [  
    {  
        "ageGroup": "INFANT",  

```

```
"required": true,  
//if requiredOn is defined, the evaluation result of requiredOn.expr  
"requiredOn": [  
    {  
        "engine": "MVEL",  
        "expr": "identity.get('ageGroup') == 'INFANT' && (identity.get('gender') == 'M' || identity.get('gender') == 'U')"  
    }  
,  
    //used to identify the type of field  
    "subType": "<document types / applicant / heirarchy level names>"  
}
```

Sample correction process SPEC: Biometric Correction

```
{  
    "id": "BIOMETRIC_CORRECTION",  
    "order": 4,  
    "flow": "CORRECTION"  
    "label": {  
        "eng": "Biometric correction",  
        "ara": "التصحيح البيومترى",  
        "fra": "Correction biométrique"  
    },  
    "screens": [  
        {  
            "order": 1,  
            "name": "consentdet",  
            "label": {  
                "ara": "موافقة",  
                "fra": "Consentement",  
                "eng": "Consent"  
            },  
            "caption": {  
                "ara": "موافقة",  
                "fra": "Consentement",  
                "eng": "Consent"  
            },  
            "fields": [  
                {  
                    "id": "consentText",  
                    "inputRequired": true,  
                    "type": "simpleType",  
                    "minimum": 0,  
                    "maximum": 0,  
                    "description": "Consent",  
                    "label": {},  
                    "controlType": "html",  
                    "fieldType": "default",  
                    "format": "none",  
                    "validators": [],  
                    "fieldCategory": "evidence",  
                    "alignmentGroup": null,  
                    "visible": null,  
                    "contactType": null,  
                    "group": "consentText",  
                    "groupLabel": null,  
                    "changeAction": null,  
                    "transliterate": false,  
                    "templateName": "Registration Consent",  
                    "fieldLayout": null,  
                    "locationHierarchy": null,  
                    "conditionalBioAttributes": null,  
                }  
            ]  
        }  
    ]  
}
```

```
        "required": true,
        "bioAttributes": null,
        "requiredOn": [],
        "subType": "consentText"
    },
{
    "id": "consent",
    "inputRequired": true,
    "type": "string",
    "minimum": 0,
    "maximum": 0,
    "description": "consent accepted",
    "label": {
        "ara": "الاسم الكامل الكامل الكامل",
        "fra": "J'ai lu et j'accepte les termes et conditions",
        "eng": "I have read and accept terms and conditions"
    },
    "controlType": "checkbox",
    "fieldType": "default",
    "format": "none",
    "validators": [],
    "fieldCategory": "evidence",
    "alignmentGroup": null,
    "visible": null,
    "contactType": null,
    "group": "consent",
    "groupLabel": null,
    "changeAction": null,
    "transliterate": false,
    "templateName": null,
    "fieldLayout": null,
    "locationHierarchy": null,
    "conditionalBioAttributes": null,
    "required": true,
    "bioAttributes": null,
    "requiredOn": [],
    "subType": "consent"
},
{
    "id": "preferredLang",
    "inputRequired": true,
    "type": "string",
    "minimum": 0,
    "maximum": 0,
    "description": "user preferred Language",
    "label": {
        "ara": "لغة الإخطار",
        "fra": "Langue de notification",
        "eng": "User Preferred Language"
    }
}
```

```
        "eng": "Notification Langauge"
    },
    "controlType": "button",
    "fieldType": "dynamic",
    "format": "none",
    "validators": [],
    "fieldCategory": "pvt",
    "alignmentGroup": "group1",
    "visible": null,
    "contactType": null,
    "group": "PreferredLanguage",
    "groupLabel": null,
    "changeAction": null,
    "transliterate": false,
    "templateName": null,
    "fieldLayout": null,
    "locationHierarchy": null,
    "conditionalBioAttributes": null,
    "required": true,
    "bioAttributes": null,
    "requiredOn": [],
    "subType": "preferredLang"
}
],
"layoutTemplate": null,
"preRegFetchRequired": false,
"additionalInfoRequestIdRequired": false,
"active": false
},
{
    "order": 2,
    "name": "BiometricDetails",
    "label": {
        "ara": "التفاصيل البيومترية",
        "fra": "Détails biométriques",
        "eng": "Biometric Details"
    },
    "caption": {
        "ara": "التفاصيل البيومترية",
        "fra": "Détails biométriques",
        "eng": "Biometric Details"
    },
    "fields": [
        {
            "id": "individualBiometrics",
            "inputRequired": true,
            "type": "biometricsType",
            "minimum": 0,
            "maximum": 1
        }
    ]
}
```

```
        "maximum": 0,
        "description": "",
        "label": {
            "ara": "القياسات الحيوية الفردية",
            "fra": "Applicant Biometrics",
            "eng": "Applicant Biometrics"
        },
        "controlType": "biometrics",
        "fieldType": "default",
        "format": "none",
        "validators": [],
        "fieldCategory": "pvt",
        "alignmentGroup": null,
        "visible": null,
        "contactType": null,
        "group": "Biometrics",
        "groupLabel": null,
        "changeAction": null,
        "transliterate": false,
        "templateName": null,
        "fieldLayout": null,
        "locationHierarchy": null,
        "conditionalBioAttributes": [
            {
                "ageGroup": "INFANT",
                "process": "ALL",
                "validationExpr": "face",
                "bioAttributes": [
                    "face"
                ]
            }
        ],
        "required": true,
        "bioAttributes": [
            "leftEye",
            "rightEye",
            "rightIndex",
            "rightLittle",
            "rightRing",
            "rightMiddle",
            "leftIndex",
            "leftLittle",
            "leftRing",
            "leftMiddle",
            "leftThumb",
            "rightThumb",
            "face"
        ],
        "requiredOn": []
    }
}
```

```
        "required": false,
        "subType": "applicant"
    },
{
    "id": "proofOfException",
    "inputRequired": false,
    "type": "documentType",
    "minimum": 0,
    "maximum": 0,
    "description": "proofOfException",
    "label": {
        "ara": "إثبات الاستثناء",
        "fra": "Exception Proof",
        "eng": "Exception Proof"
    },
    "controlType": "fileupload",
    "fieldType": "default",
    "format": "none",
    "validators": [],
    "fieldCategory": "evidence",
    "alignmentGroup": null,
    "visible": null,
    "contactType": null,
    "group": "Documents",
    "groupLabel": null,
    "changeAction": null,
    "transliterate": false,
    "templateName": null,
    "fieldLayout": null,
    "locationHierarchy": null,
    "conditionalBioAttributes": null,
    "required": false,
    "bioAttributes": null,
    "requiredOn": [],
    "subType": "POE"
}
],
"layoutTemplate": null,
"preRegFetchRequired": false,
"additionalInfoRequestIdRequired": true,
"active": false
}
],
"caption": {
    "eng": "Biometric correction",
    "ara": "التصحيح البيومترى",
    "fra": "Correction biométrique"
},
"icon": "UINUpdate.png",
```

```
        "isActive": true,  
        "autoSelectedGroups": null  
    }
```

Enabling Handles Feature

Add the following in the properties section of IDSchema:

```
"selectedHandles":  
{  
    "fieldCategory": "none",  
    "format": "none",  
    "type": "array",  
    "items":  
    {  
        "type": "string"  
    },  
    "fieldType": "default"  
},
```

- ⓘ Enable handles on a particular field by setting property as: "handle": true

Example:

```
"email":  
{  
    "bioAttributes": [],  
    "validators":  
    [  
        {  
            "langCode": null,  
            "validator": "^[A-Za-z0-9_\\-]+(\\. [A-Za-z0-9_]+)*@[A-Za-z0-9.  
            "arguments": [],  
            "type": "regex"  
        }  
    ],  
    "fieldCategory": "pvt",  
    "format": "none",  
    "type": "string",  
    "fieldType": "default",  
    "handle": true  
}
```

Technology Stack

The table below outlines the frameworks, tools, and technologies employed by the Android Registration Client.

Tool /Technology	Version	Description	License
Flutter	3.10.4	Flutter transforms the development process. Build, test, and deploy beautiful mobile, web, desktop, and embedded experiences from a single codebase.	BSD License
Dart	3.0.3		BSD License

Android SDK Versions :

Target SDK Version : 31

Compile SDK Version : 31

Test

End User Guide

Login using OTPLogin using OTPThis user guide is designed to provide assistance to Operators and Supervisors in successfully installing, running, and registering applicants to obtain their Unique Identification Numbers (UIN) on tablet devices.

Prerequisites

- Reliable and consistent Internet connectivity.
- Tablets running Android version 10 to 13.
- Tablets with a minimum of 4 GB RAM.
- The tablets need to be capable of capturing fingerprints, iris, and face (photo) biometrics. Additionally, they should also have the ability to scan documents. However, if the tablets do not support these capabilities, MOCK SBI can be used as an alternative.

How to install Android Registration Client (ARC)

1. Download and install the APK on Android tablet.
2. Once ARC is installed, long press on the logo to copy the machine details.
3. On the [Admin Portal](#), using admin credentials, login and perform the following to add the device:
 - Go to `Resources/Machine` and click on **Create machine**
 - Add a new machine and enter the machine details:
 - Add the specs as **Mobile**
 - Map it to a Zone and Center
 - Add the Machine spec ID as **Mobile**
 - Enter Device name
 - Enter Public Key
 - Enter Sign Public Key
 - Create the role `Default` in KeyCloak with all the other roles.

- Create the Operator's user account in Keycloak set the password and assign the role as `Default`, `REGISTRATION_OFFICER`, `Registration Operator`, `REGISTRATION_SUPERVISOR`
- Login into Admin Portal to perform the following and add the user:
 - After login into the Admin Portal, go to `User Zone Mapping` and add the zone for the user and activate it.
 - Go to `User Center Mapping` and add the center for the user and activate it.

(i) Note: The user should be assigned to the same Zone and Center as the device.

4. The user should relaunch the ARC and log in using their valid credentials. Additionally, the operator has the option to select their preferred display language.

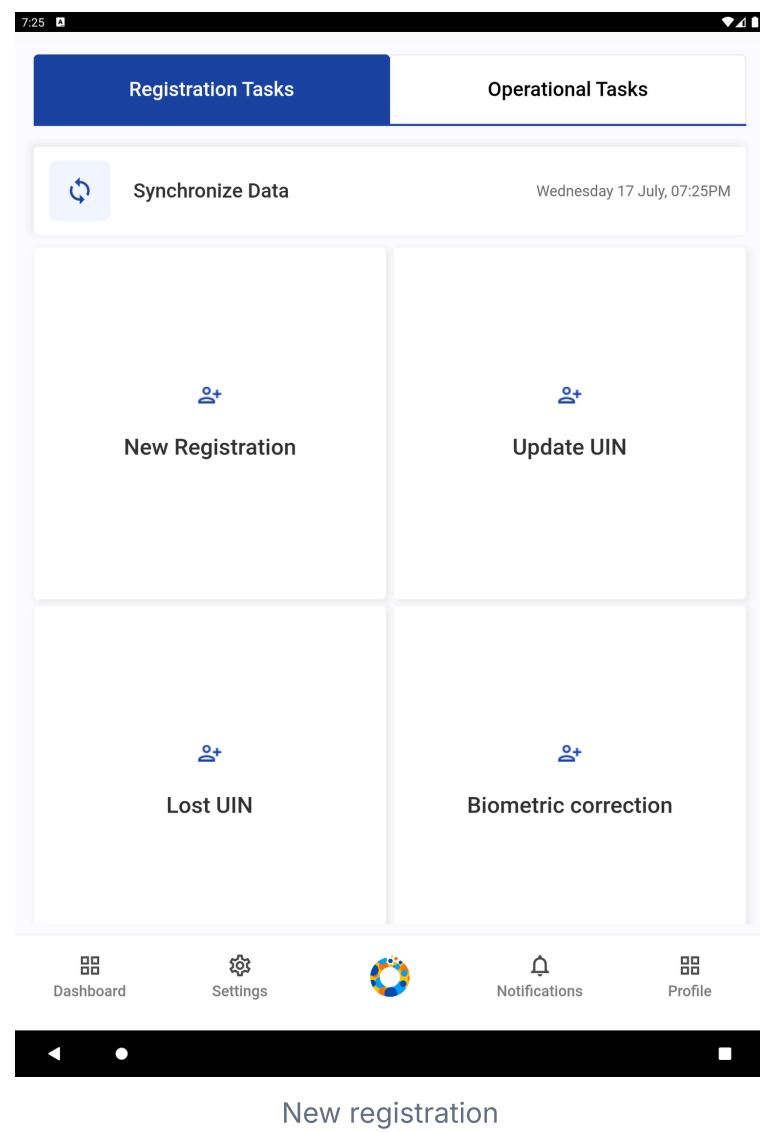
Upon successful login, the user will be directed to the Home page, which includes the following options:

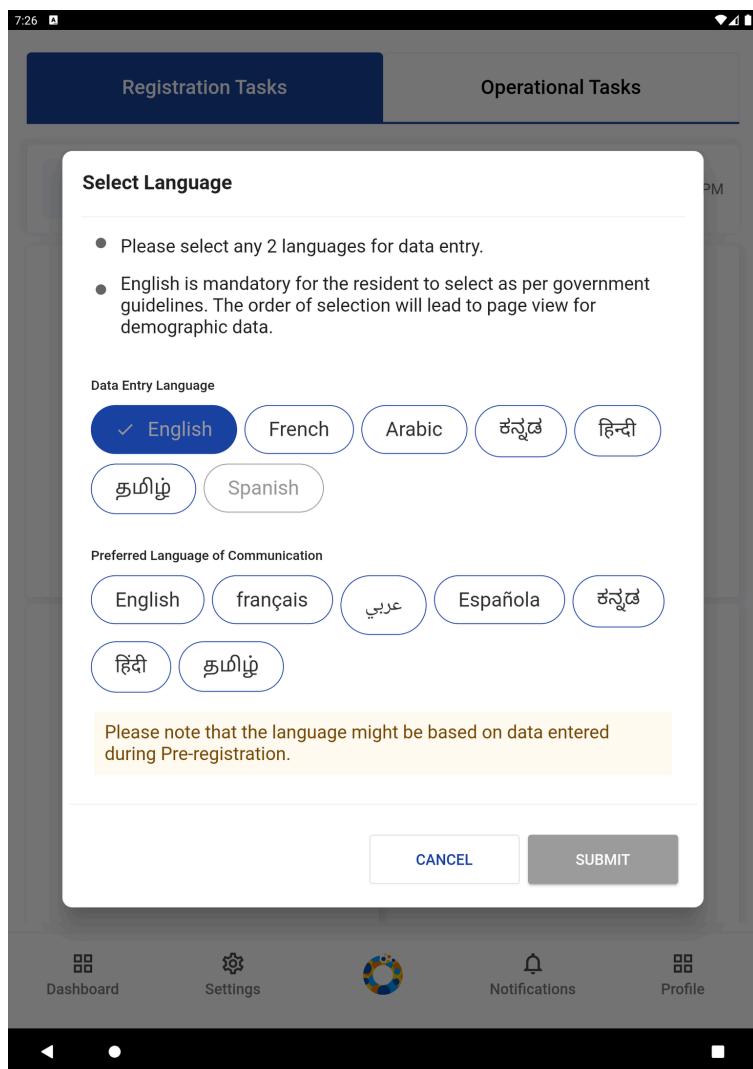
- New Registration
- Operational Tasks
- Dashboard
- Settings (Future scope)

New Registration

To begin the Registration process, the Operator is required to follow the steps outlined below.

1. Click on **New Registration card**.
2. Select the language to be used for data entry, which will be used to collect the resident's information. There will be a default language for data entry.
3. Choose the language in which the notification will be sent to the resident. Click **Submit** to proceed.





New registration



New registration

4. The operator will be redirected to the Consent page, where the resident must agree to the [terms and conditions](#) to proceed.
5. After accepting consent, the Operator will need to fill out the demographic data of the resident, including their name, age, date of birth, and address. Once all mandatory fields are completed, the **Continue** button will be enabled.

The screenshot shows a mobile application interface for 'New Registration'. At the top, there's a blue header bar with the title 'New Registration'. Below it is a navigation bar with tabs: 'Consent' (selected), 'Demographic Details', 'Document Upload', 'Biometric Details', 'Preview', and 'Authent >'. The main content area has three language tabs: 'English' (selected), 'French', and 'Arabic'.

The 'Consent' section contains two paragraphs of text:

- I understand that the data collected about me during registration by the said authority includes
 - Name
 - Date of birth
 - Gender
 - Address
 - Contact details
 - Documents
 - Biometrics
- I also understand that this information will be stored and processed for the purpose of verifying my identity in order to access various services, or to comply with a legal obligation. I give my consent for the collection of this data for this purpose

A callout box below the text states: 'I have read and accept terms and conditions to share my PII / J'ai lu et j'accepte les termes et conditions pour partager mes PII *' followed by the Arabic translation 'الاسم الكامل الكامل الكامل/ *'.

At the bottom, there are two buttons: 'GO BACK' and a large blue 'INFORMED' button. The status bar at the very bottom shows navigation icons.

New registration

The screenshot shows the 'New Registration' application interface on a mobile device. The top navigation bar includes tabs for 'Consent', 'Demographic Details' (which is selected), 'Document Upload', 'Biometric Details', 'Preview', and 'Authent >'. The main form area is titled 'Application ID' and contains a text input field labeled 'Enter Application ID' with a 'FETCH DATA' button and a QR code scanner icon.

Full Name/ Nom complet/ الاسم الكامل *

Full Name: _____ Nom complet: _____
الاسم الكامل: _____

DOB/ DOB/ DOB *

YYYY/MM/DD OR 0

Gender/ Le genre/ جنس *

English: Male, Female, Others
French: Mâle, Femelle, Autres
Arabic: ذكر, أنثى, آخرون

addressLine1/ addressLine1/ 1 عنوان دائم *

addressLine1: _____ addressLine1: _____
عنوان دائم: _____

CONTINUE

New registration

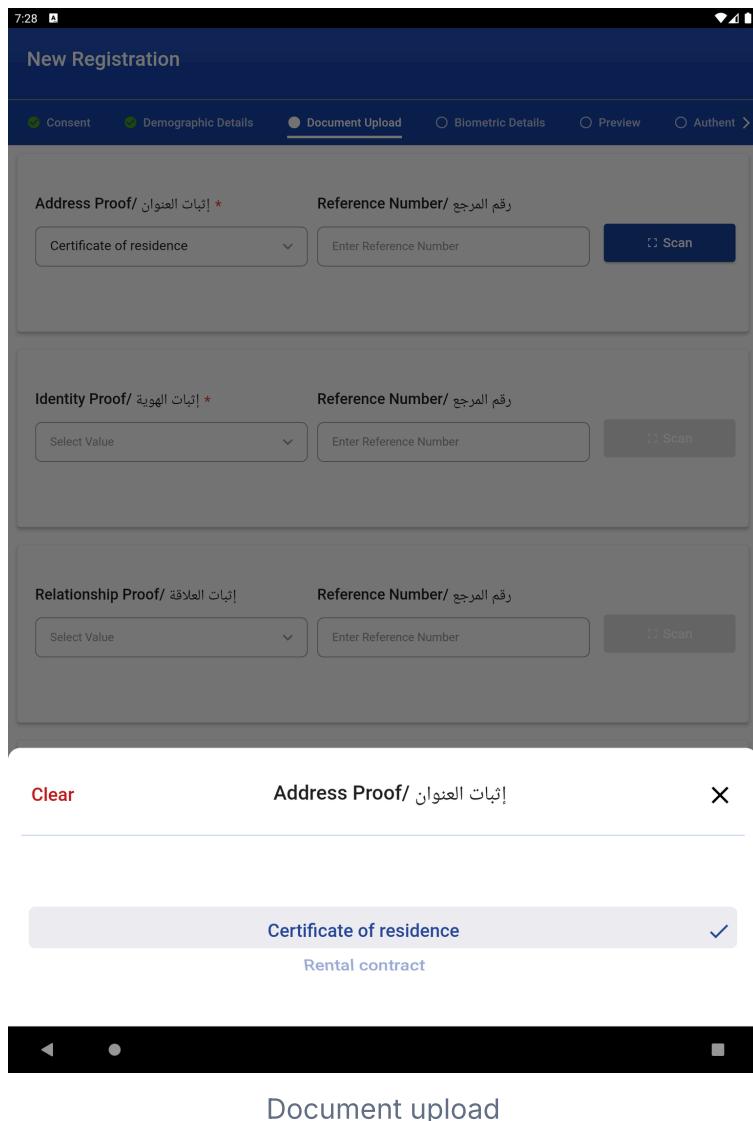
The screenshot shows the 'New Registration' screen of the MOSIP mobile application. At the top, there are tabs: 'Consent' (green), 'Demographic Details' (selected, blue), 'Document Upload', 'Biometric Details', 'Preview', and 'Authent >'. Below the tabs, there's a section for 'Application ID' with a text input field containing 'Enter Application ID' and a 'FETCH DATA' button. The 'Full Name' field contains 'Piyush' with a placeholder 'الاسم الكامل / First Name' and a 'Last Name' field. The 'DOB / DOB' field shows '2000/01/01' and an 'OR' option with '24'. The 'Gender' section has radio buttons for 'Male' (selected), 'Female', 'Others', and their Arabic counterparts 'ذكر', 'إناث', and 'آخرون'. Below these are three address lines: 'addressLine1 / 1' with 'EON', 'addressLine2 / 2' with 'Phase 3', and 'addressLine3 / 3' which is empty. At the bottom right is a large blue 'CONTINUE' button.

New registration

- Upon clicking the **Continue** button, the Operator will be navigated to the **Document upload** page where they will need to:

- Select the type of document (e.g. proof of identity, proof of address) from the drop-down menu.
- Enter the **Reference Number** of the document.

Document upload



4. Upload the document by clicking on the **Scan** button to open the camera. The Operator can take a picture of the document and then choose from the following actions:

- **Cancel:** Clicking on the "Cross" icon will remove the captured image and return the Operator to the previous screen.
- **Crop:** The Operator can drag from the four corners of the captured image to crop it as needed.
- **Save:** Clicking on the "Save" button will save the captured image and return the Operator to the previous Document Upload page.
- **Retake:** Clicking on the "Retake" button will remove the captured image, reopen the camera, and allow the Operator to take a new photo.

Document upload

The screenshot shows the 'New Registration' process in the MOSIP system. The current step is 'Document Upload'. There are four sections for different types of proofs:

- Address Proof / إثبات العنوان ***: Includes a dropdown for 'Certificate of residence', a text input for 'Enter Reference Number', and a 'Scan' button.
- Identity Proof / إثبات الهوية ***: Includes a dropdown for 'Reference Identity Card', a text input for 'Enter Reference Number', and a 'Scan' button.
- Relationship Proof / إثبات العلاقة ***: Includes a dropdown for 'Select Value', a text input for 'Enter Reference Number', and a 'Scan' button.
- DOB Proof / DOB**: Includes a dropdown for 'Certificate of Birth', a text input for 'Enter Reference Number', and a 'Scan' button.

A preview of a scanned document (a green and black checkered pattern) is visible in the first section. At the bottom right is a 'CONTINUE' button.

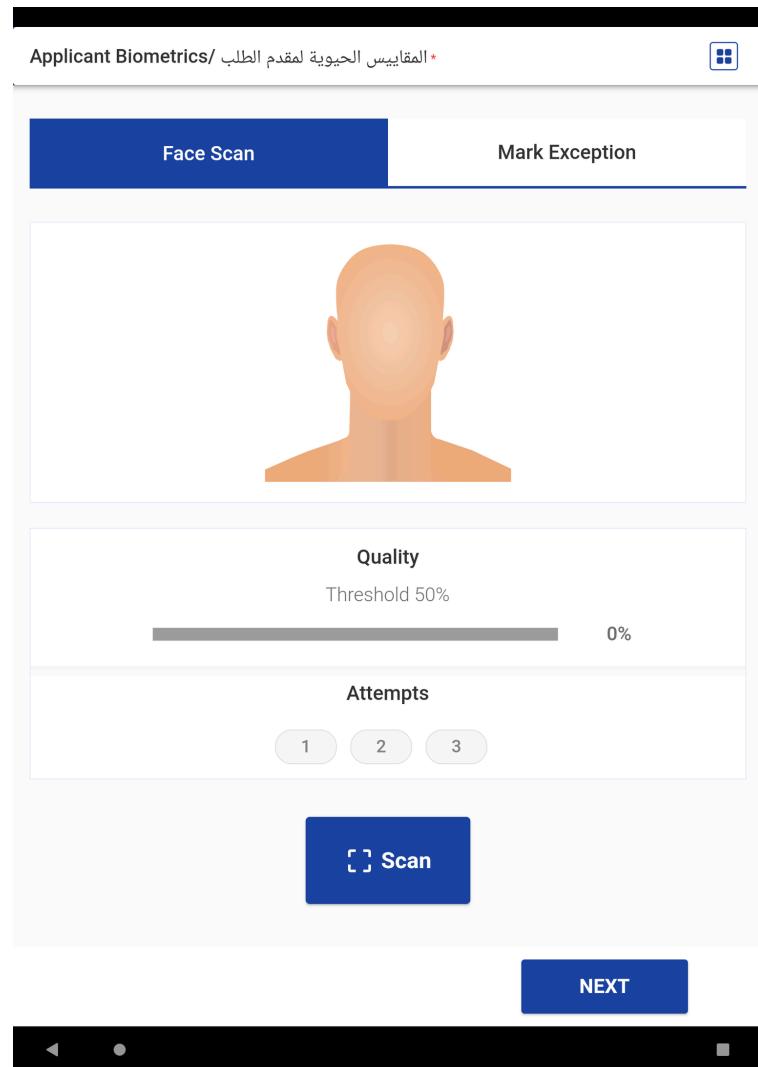
Document upload

- After ensuring all required information has been accurately entered into the **Document Upload** screen, the Operator can proceed by clicking on the **Continue** button to access the **Biometric Capture** page. Here, the Operator can capture the biometric data of the Resident, including a face photo, fingerprint, and iris scan.

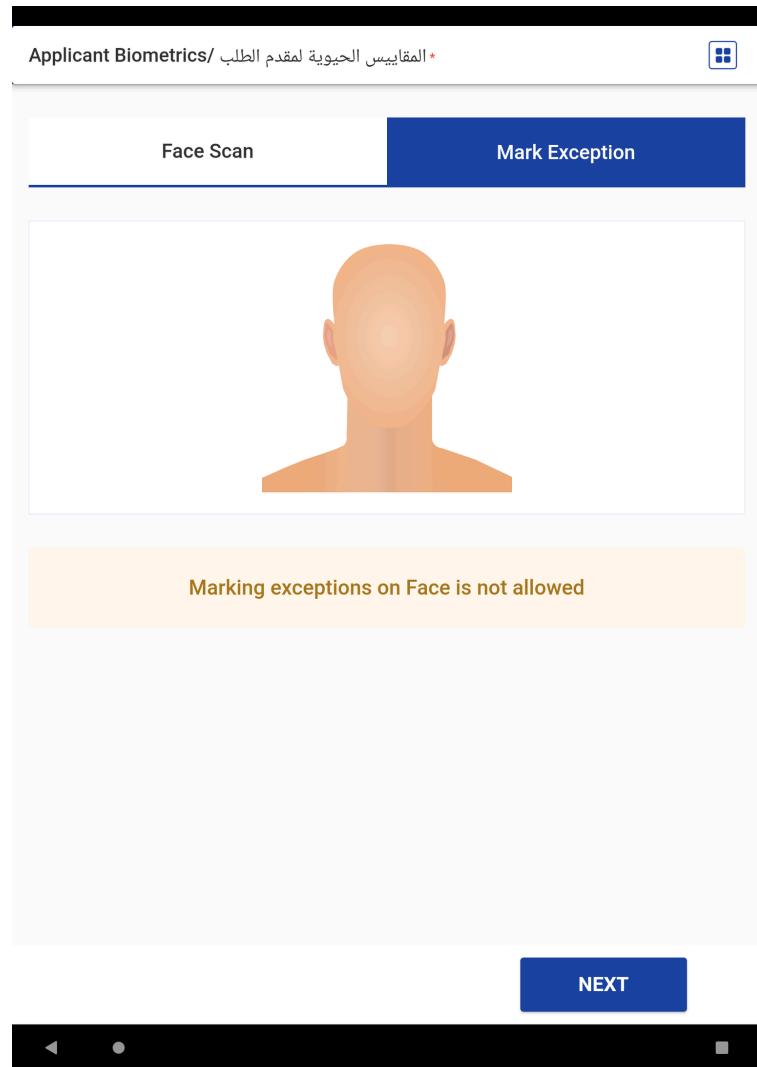
Face photo capture process

- To capture the face photo, the Operator should click on the **Scan** button to activate the camera and take a picture.
- The image quality will be displayed on the screen and must meet a certain threshold to be considered acceptable.
- The Operator has three attempts to capture the biometric image.

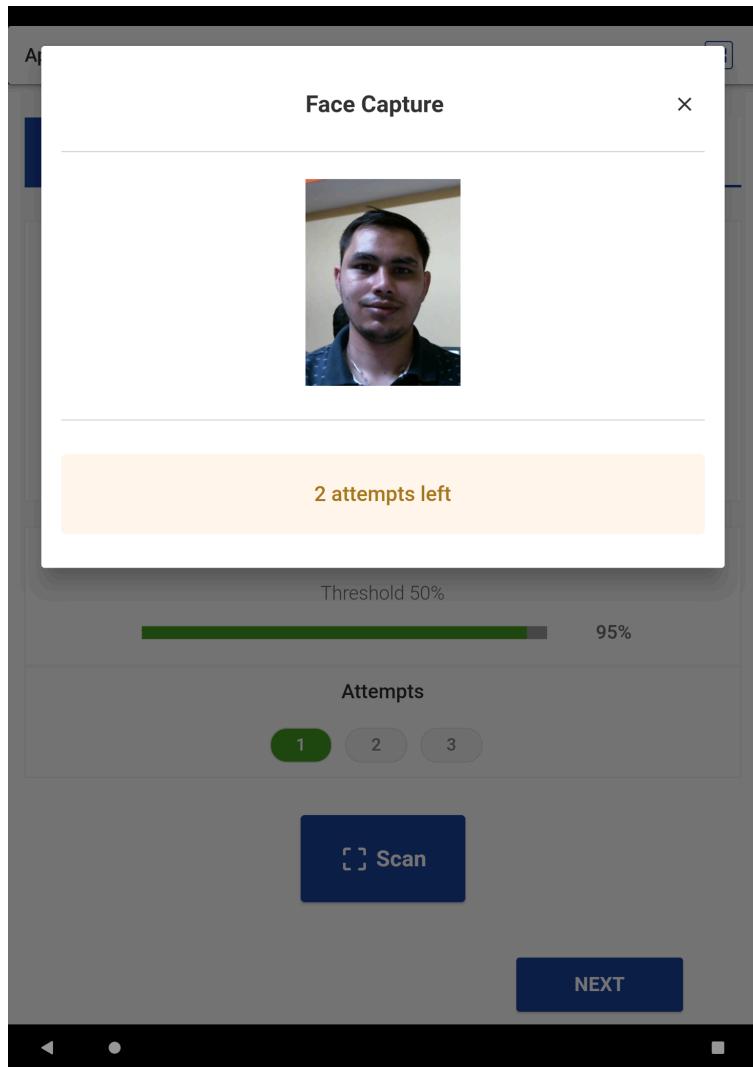
- It is important to note that no exceptions can be made for the face photo biometric capture process.



Face photo capture process



Face photo capture process

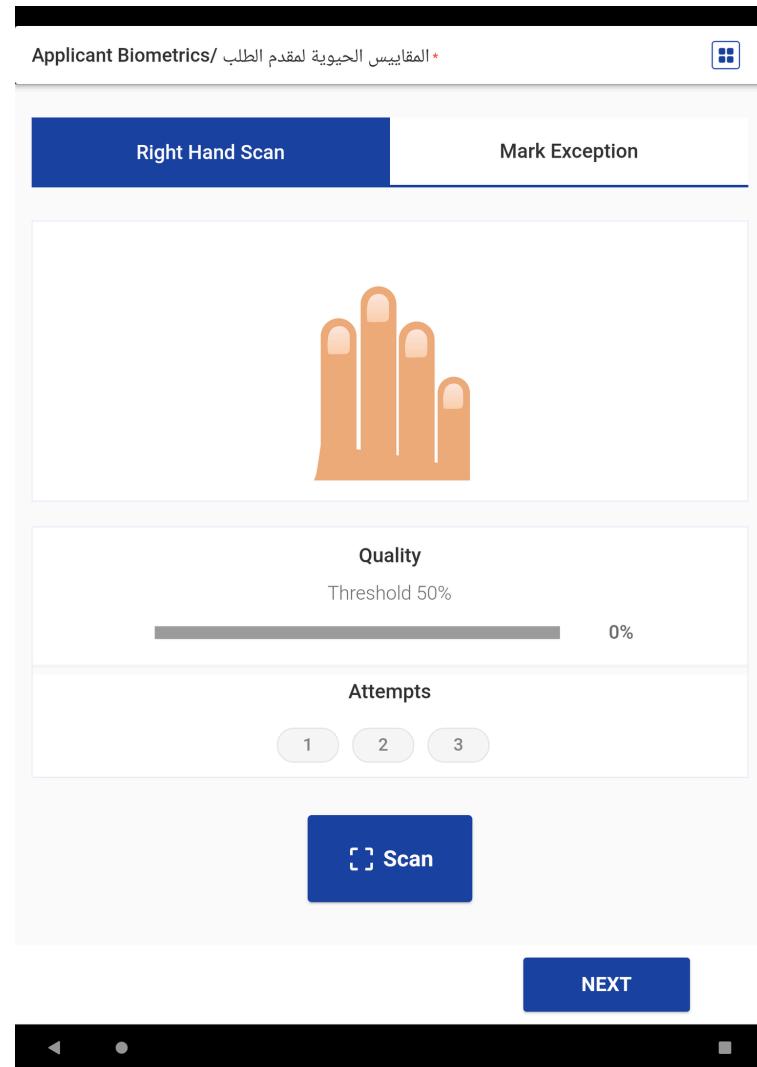


Face photo capture process

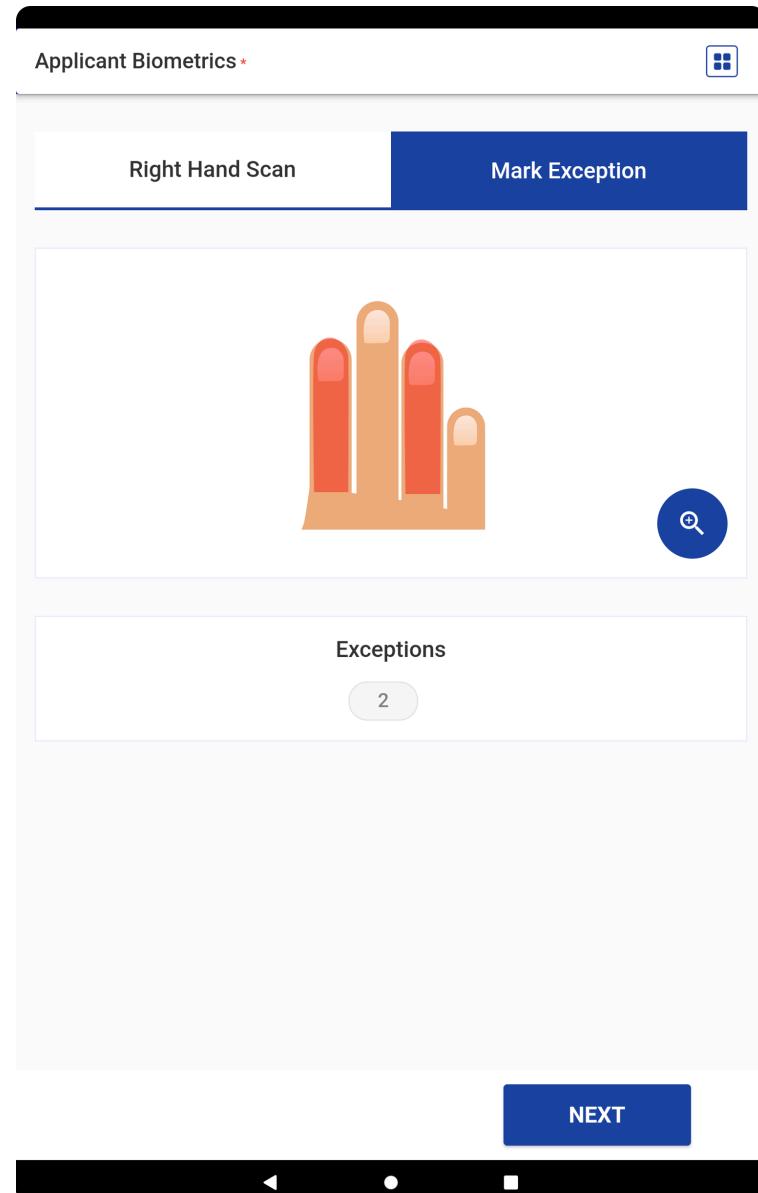
Biometric Data Capture Process:

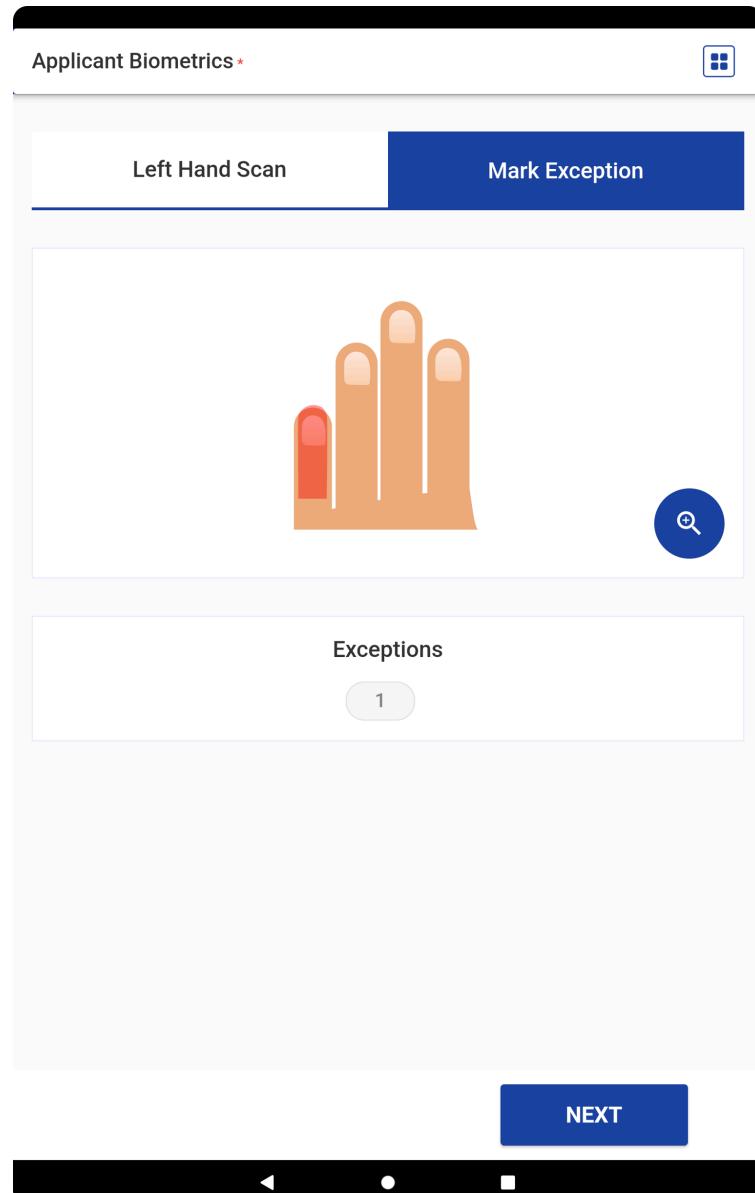
- To capture biometric data, the Operator should click on the **Scan** button.
- This will allow the Operator to capture the biometric information.
- Once the data is captured, the image quality will be displayed on the screen and must meet the acceptable threshold limit.

Note: Three attempts are provided to capture the biometric data.



Biometric data capture process

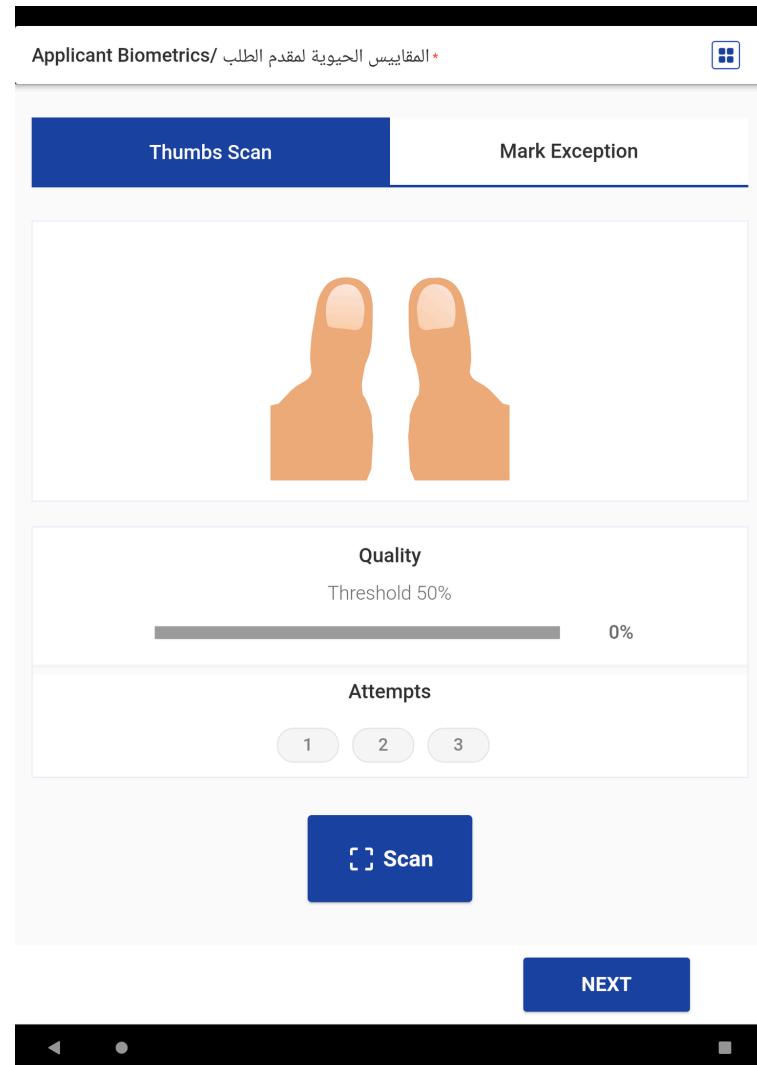




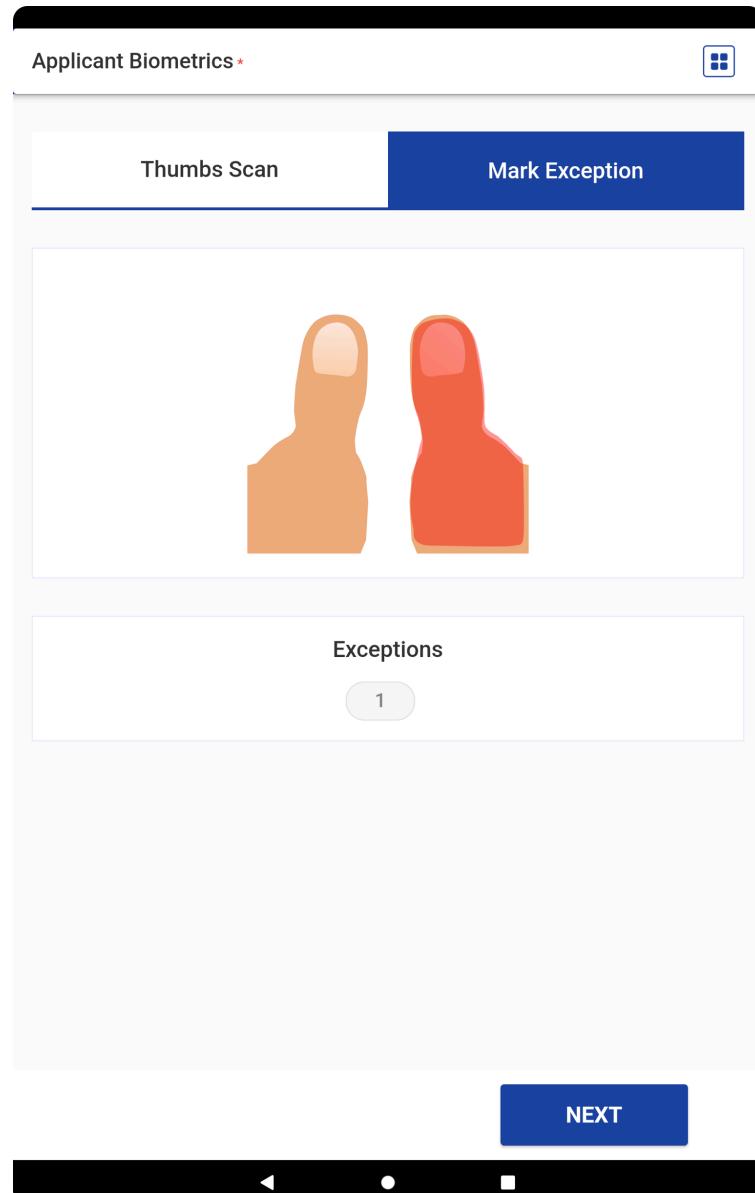
Biometric data capture process

Fingerprint Capture Process:

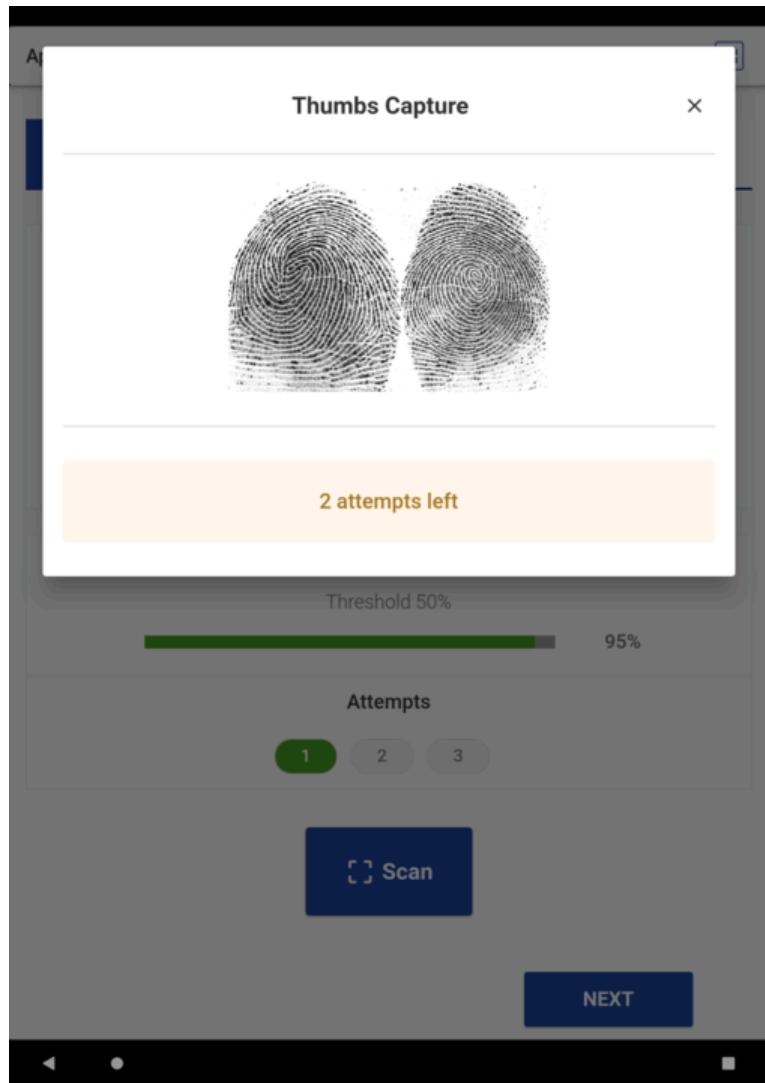
If a thumb is missing or experiencing difficulties that prevent its fingerprint from being captured, the Operator is authorized to indicate an **exception**. To mark an exception, the operator must select the affected thumb and specify the type of exception as either *Temporary* or *Permanent*. Additionally, the operator may include any relevant additional comments.



Fingerprint capture process



Fingerprint capture process



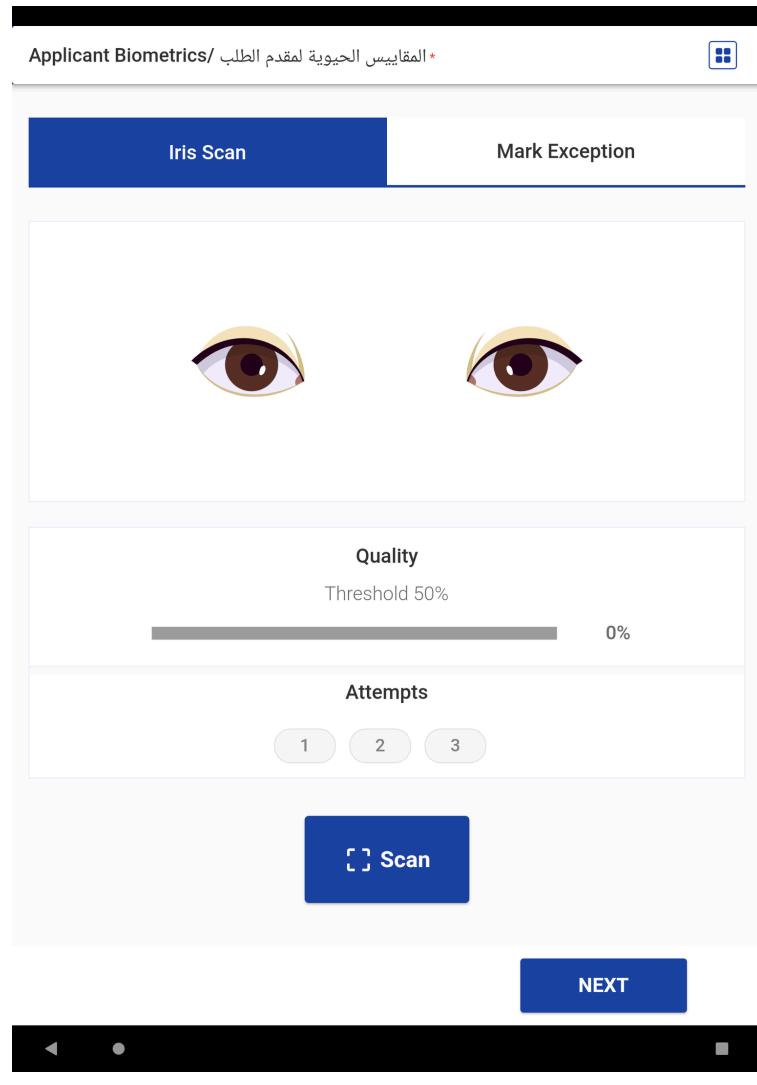
Fingerprint capture process

Iris Scanning Process:

- To initiate the Iris scan, the Operator is required to click on the **Scan** button.
- This action will allow the Operator to capture the Iris image.
- Once the Iris has been successfully captured, the quality of the image will be displayed on the screen.
- The quality score needs to meet the established threshold limit.
- The Operator has three opportunities to capture the biometric data.

If one or both of the Irises are not detected or encounter issues that prevent successful capture, the Operator has the option to mark an exception. To do so, the Operator must identify the specific Iris that is problematic and indicate the type of

exception- either *Temporary* or *Permanent*. Additionally, the Operator may provide any relevant comments.



Iris scanning process

Applicant Biometrics / المقاييس الحيوية لمقدم الطلب *

Iris Scan Mark Exception



Exceptions
1

Exception Type

Permanent Temporary

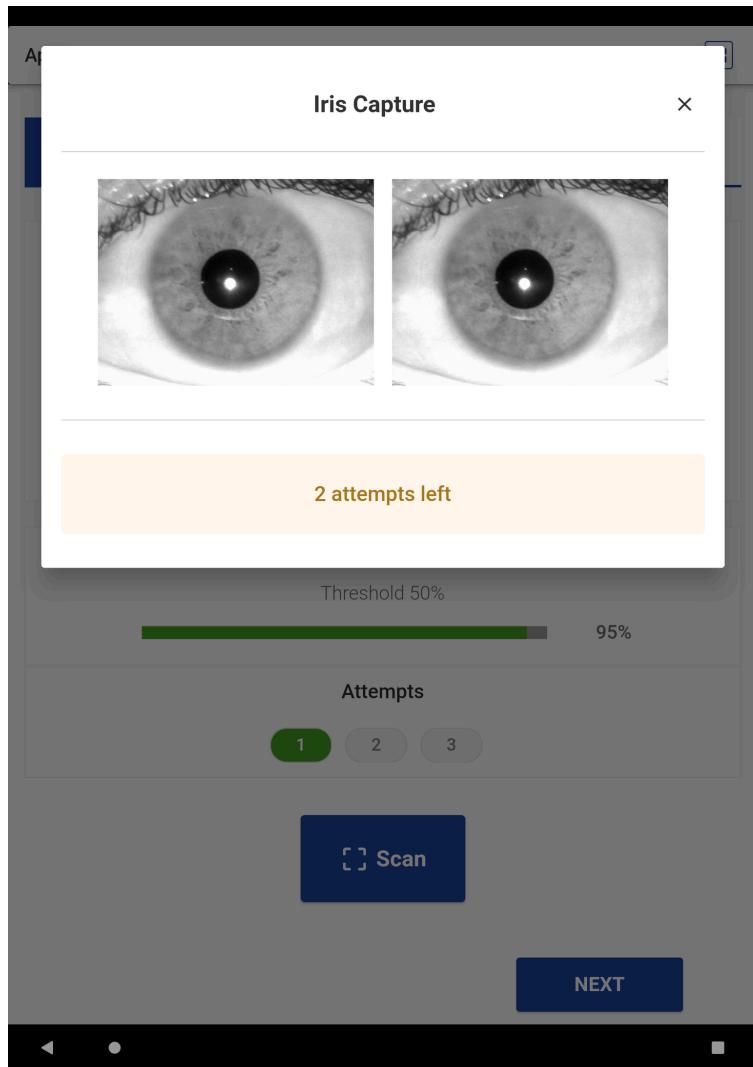
Comments

Add comments for marking the exception

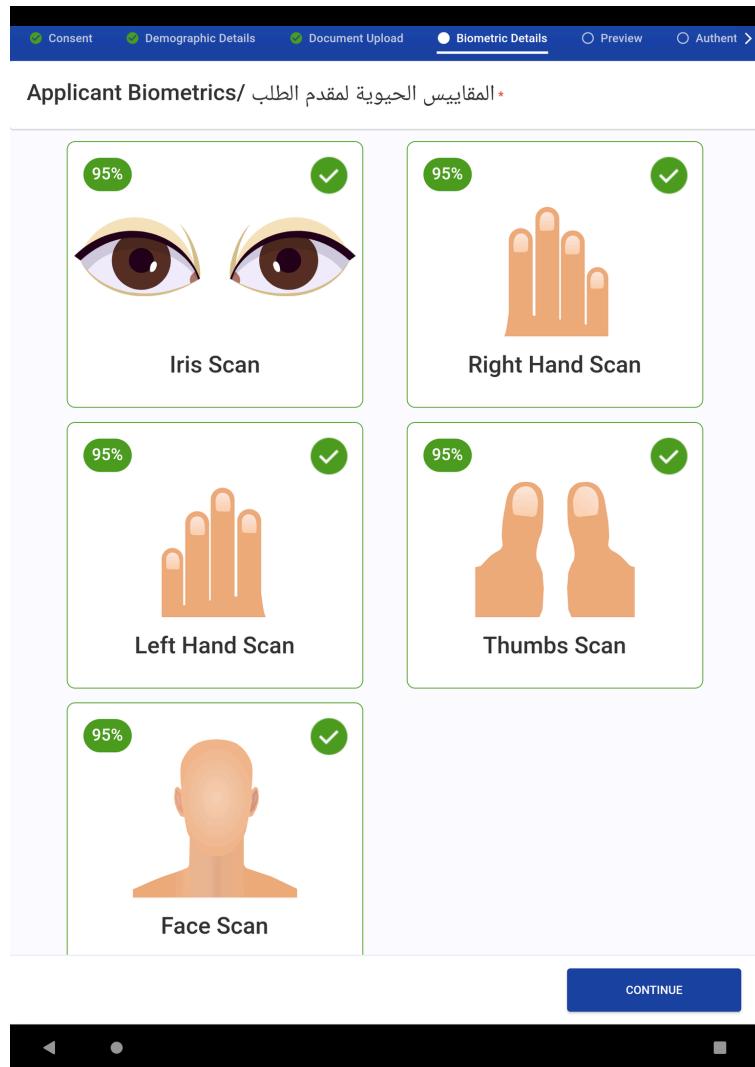
NEXT

◀ • ▶

Iris scanning process



Iris scanning process



Iris scanning process

New Registration

Consent Demographic Details Document Upload Biometric Details Preview Authent >

Application ID
10001126431003120240717135716

Date
17/07/2024 07:32 PM

Demographic Information

Full Name/**Piyush** Photo

DOB/DOB
2000/01/01

Gender/**Male**

Phone/**6575775564** Email/**piyush@text.com**

Address
addressLine1/**Rabat Sale Kenitra/Rabat Sale** addressLine2/**Kenitra** addressLine3/**Pune**

EON/**Phase 3/r** Province/**Pune** City/**Kenitra**

Region/**Rabat Sale Kenitra** Kenitra/Kenitra

Zone/**Kenitra** Postal/**14022**

Ben Mansour/Ben Mansour

Documents

Identity Proof/[**Reference Identity Card**] Address Proof/[**Certificate of residence**] DOB Proof/[**Certificate of Birth**]

Biometrics

Applicant Biometrics/**الملحقين الجودية لمقدم الطلب**
Fingers (10),Iris (2),Face (1)

Left Iris Right Iris

Left Slap Right Slap Thumbs

CONTINUE

Iris scanning process

Demographic Information

Full Name/الاسم باللغة العربية	Piyush/پیوش	Photo
DOB/DOB	2000/01/01	
Gender/جنس	Male/MLE	
Phone/رقم الهاتف	657577564	Email/البريد الإلكتروني
		piyush@text.com

Address

addressLine1/عنوان المكتب/المحل	addressLine2/عنوان المكتب/المحل	addressLine3/عنوان المكتب/المحل
EON/نمبر	Phase 3/رقم المرحلة	Pune/بنارس
Region/المنطقة	Province/الإقليم	City/المدينة
Rabat Sale Kenitra/Rabat Sale Kenitra	Kenitra/Kenitra	Kenitra/Kenitra
Zone/المنطقة	Postal/الرمز البريدي	
Ben Mansour/Ben Mansour	14022	

Documents

Identity Proof/بطاقات الهوية
Reference Identity Card
Address Proof/بيانات العنوان
Certificate of residence
DOB Proof/DOB
Certificate of Birth

Biometrics

Applicant Biometrics/المعلومات الحيوية المقترنة بالطلب
Fingers (10), Iris (2), Face (1)

Introducer Biometrics/بيانات المقترنة
Fingers (0), Iris (0), Face (0)

CONTINUE

Iris scanning process

- After all the biometric data has been properly captured or any exceptions have been noted, the **Continue** button will be activated. The Operator can then proceed by clicking on the **Continue** button, which will redirect them to the **Preview** page. The Preview page will display the following information:

- Application ID
- Timestamp of Registration
- Demographic data collected
- Documents submitted
- Biometric data recorded

From the Preview page, the Operator can navigate back to previous screens to make any necessary adjustments to the entered or captured data. Once the Operator has verified the accuracy of the entered data, they can proceed by

clicking on the **Continue** button, which will direct them to the **Operator Authentication** page.

10. On the **Operator Authentication** page, operators are required to input their credentials (username and password) that were used during the login process.

Upon successful verification of the credentials, the packet will be uploaded to the server and the operator will be redirected to the **Acknowledgment** screen. This screen includes the following information:

- Application ID
- Timestamp of Registration
- Demographic data captured
- Documents uploaded
- Biometric data captured
- Print option
- QR code for the Application ID
- Option to initiate a new registration process.

The screenshot shows a mobile application interface titled "New Registration". At the top, there is a navigation bar with several tabs: "Consent", "Demographic Details", "Document Upload", "Biometric Details", "Preview", and "Authent >". The "Authent >" tab is currently selected, indicated by a blue underline and a blue dot icon.

The main content area is titled "Authentication using Password". It contains two input fields: "Username *" and "Password *". Both fields have placeholder text "Enter Username" and "Enter Password" respectively. Below these fields is a blue button labeled "AUTHENTICATE".

At the bottom of the screen, there is a black navigation bar with three icons: a left arrow, a central dot, and a right square.

Authentication page

New Registration

Consent Demographic Details Document Upload Biometric Details Preview >

Registration Acknowledgement PRINT

Application ID
16075151221003120240905065733

Date
05/09/2024 12:39 PM

Demographic Information

Date of Birth:
1998/01/01
Gender:
MLE
Province:
Central Province

Photo

Documents

Identity Proof
Reference Identity Card

Biometrics

Applicant Biometrics
Fingers (10), Iris (0), Face (1)

Left Slap Right Slap Thumbs
Face

Introducer Biometrics
Fingers (0), Iris (0), Face (0)

Registration Officer
zambia Registration Center
16075

[Go To Home](#)

Acknowledgement page

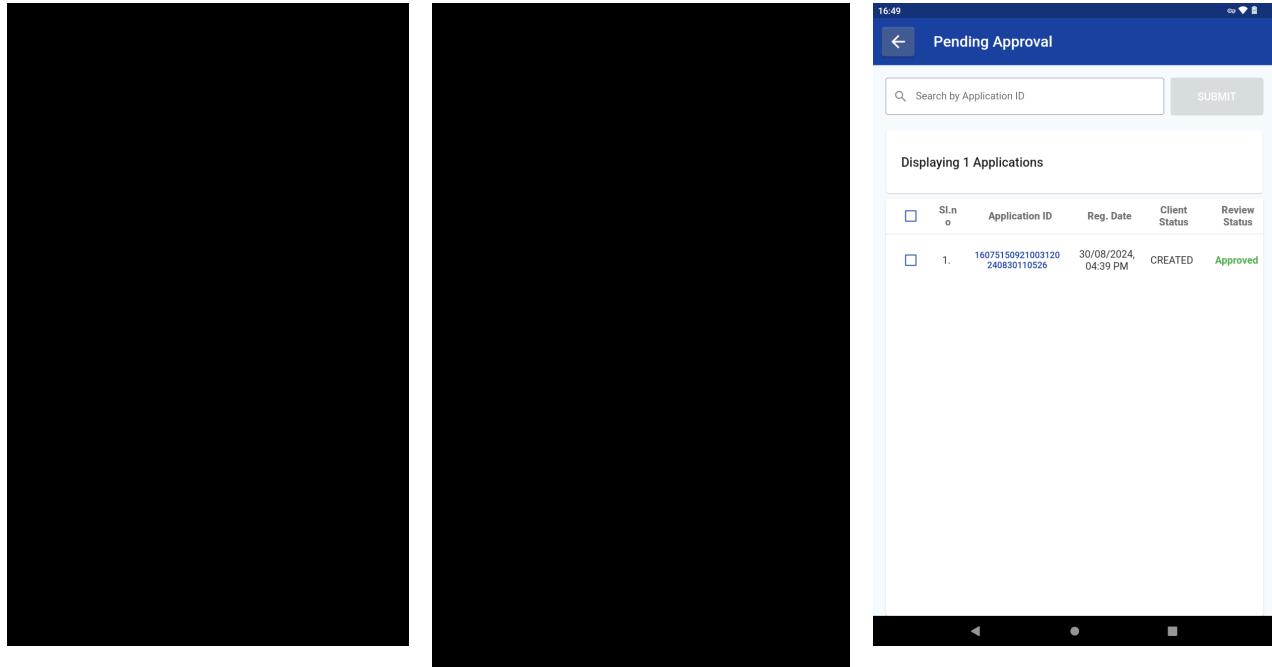
Pending Approval:

Upon successful verification of the credentials, the acknowledgment will be displayed, and the Application will be moved to the "Pending Approval" section. This feature will only be available for the User who has a Supervisor's role assigned to him.

Once the packet is created by the Operator, as an additional check, the Supervisor will have to go through each application to make sure the details filled are coherent.

Step 1: The user goes to the "Pending Approval" section from the Operational Tasks section. The user will be taken to the page where they can see the list of all the

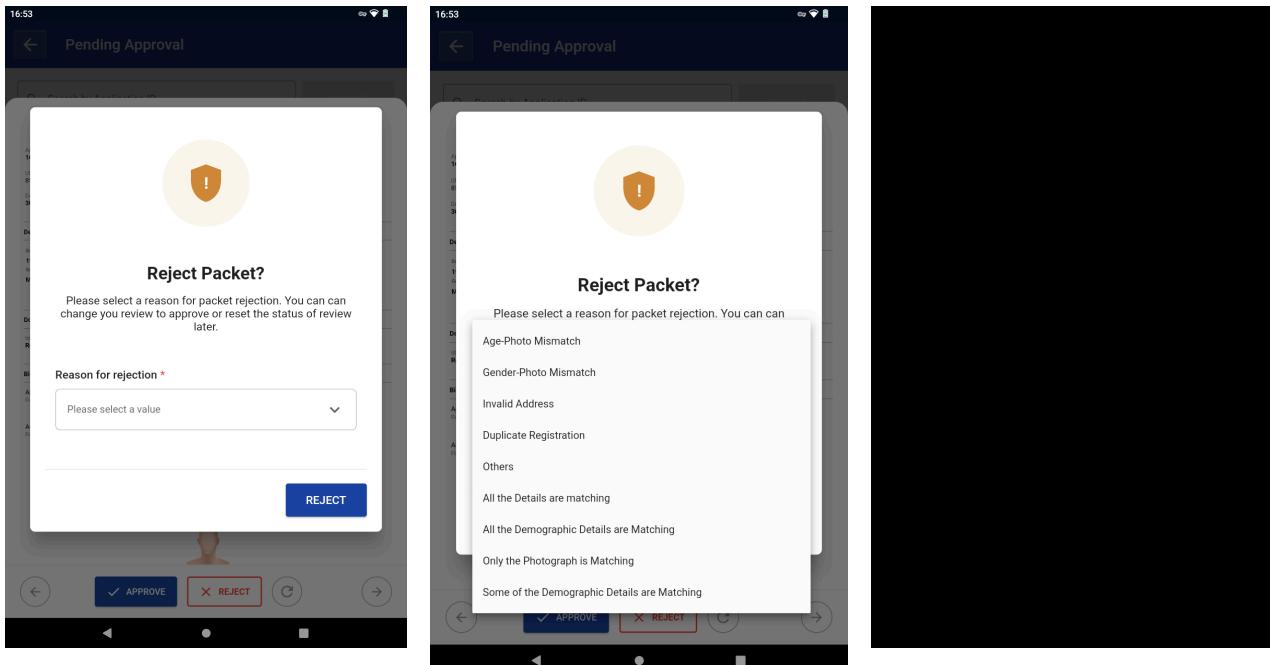
Applications created by the Operator. All of these Applications will be "Pending".



Supervisor's approval

Step 2: The Supervisor then clicks on the Application ID one by one. At this stage, the Supervisor can either Approve the Application or he can Reject it. If the Supervisor decides to reject it, they also will have to mandatorily mention the reason for rejection.

Step 3: Once the Application has been Approved or Rejected, the Supervisor will have to authenticate himself by clicking on the "Submit" button and thereby entering their Username and Password. The User can also bulk submit the Applications. The only pre-requisite is that the packet has to be in Approved or Rejected status (pending Applications cannot be submitted for uploading). Once they have successfully authenticated, the Application will be removed from the "Pending Approval" section and will be moved to the "Manage Application" Section.



Pending approval

Step 4: Once the Application is either Approved or Rejected by the Supervisor and is submitted, those packets can be uploaded to the server from the “Manage Application” section or can be exported to their local device storage.

Manual Application upload/export

Once the Application is either Approved or Rejected by the Supervisor, those packets can be uploaded to the server from the “Manage Application” section.



Manage application

Step 1: The user selects the packets they want to upload (bulk upload can also be done). Once selected, the user clicks on the “Upload” button, after which the packet syncs and gets uploaded if there is internet connectivity.

In case of a lack of internet connectivity, the User can also export the packet to their local device storage. They can also bulk export the packets by choosing the Applications and clicking on the Export button.

On choosing to upload, the packet is uploaded to the [Registration Processor](#). Once the packet has been successfully processed, a unique identification number (UIN) is generated.

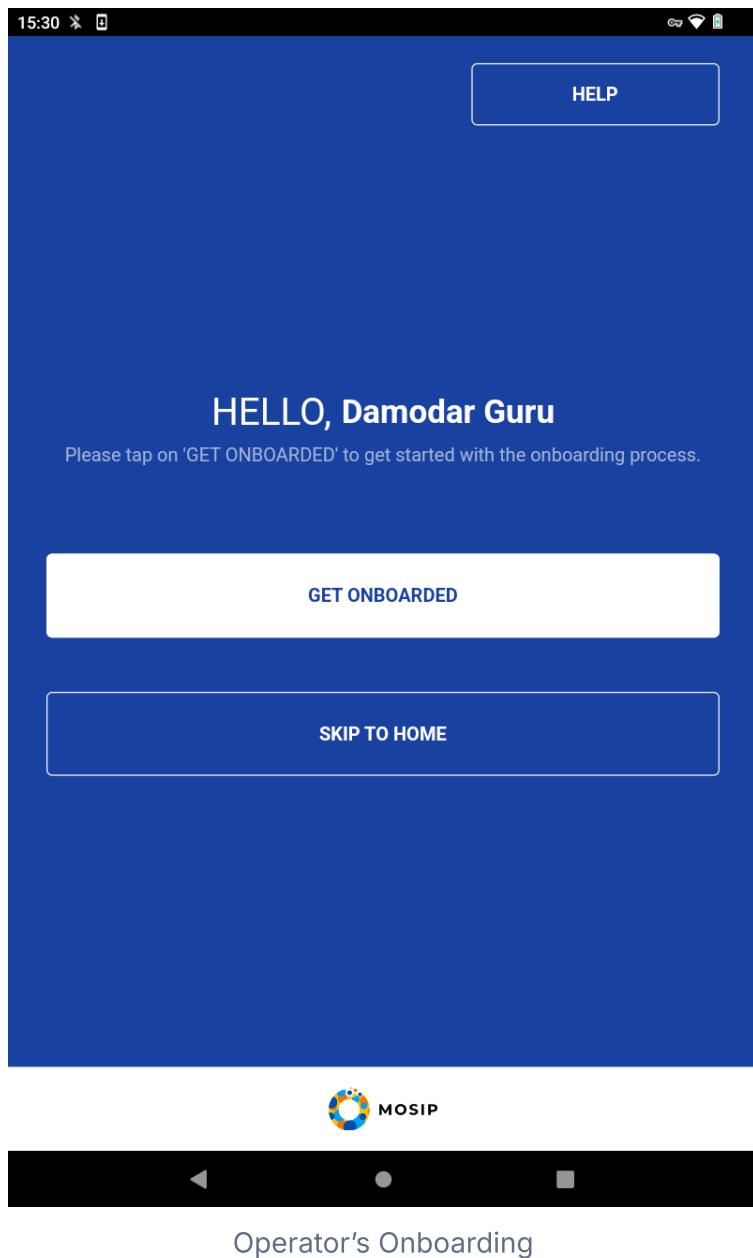
1. **Print-** The operator can click on this option to obtain a physical copy of the acknowledgment.
2. **New Registration-** The operator can initiate another registration by clicking on this option.

In summary, the user (Operator/ Supervisor) can follow the aforementioned steps to register an individual by capturing demographic data, documents, and biometric data to generate their UIN.

Operator Onboarding: To begin the Onboarding process, the Operator is required to follow the steps outlined below. The operator, to log in to the Android Registration Client, will have to onboard himself. This functionality will be available on first-time online login only.

a. On logging in for the first time, the Operator will be taken to the screen where they will have the following two options:

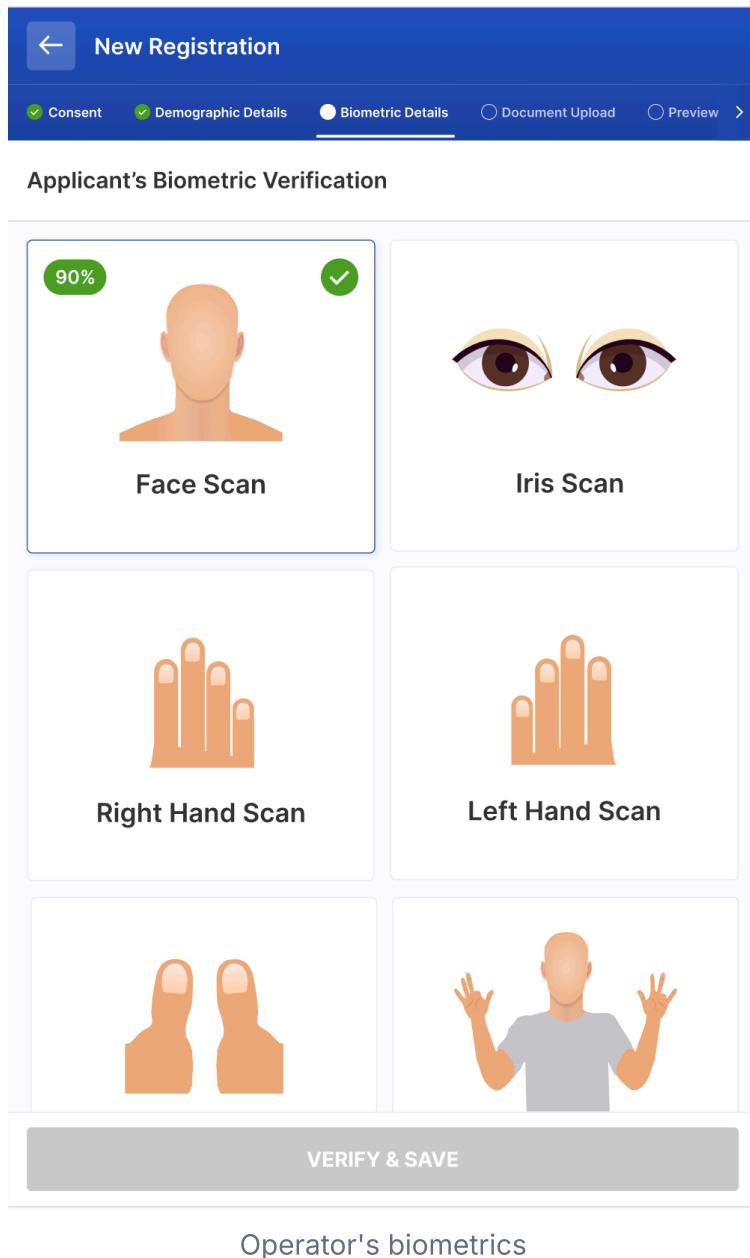
1. **Get onboard:** This flow is present for the system to verify the Operator's biometrics with their registered biometrics. This is to enable local deduplication. To get onboarded the operator must not be assigned "default" role.
2. **Skip to home:** This flow is to dodge "Operator's Onboarding". If the user selects this, they will be taken to the "Homepage" after which the user can get started with Resident registration. One of the prerequisites of this flow is to have the "Default" role mapped to the center.



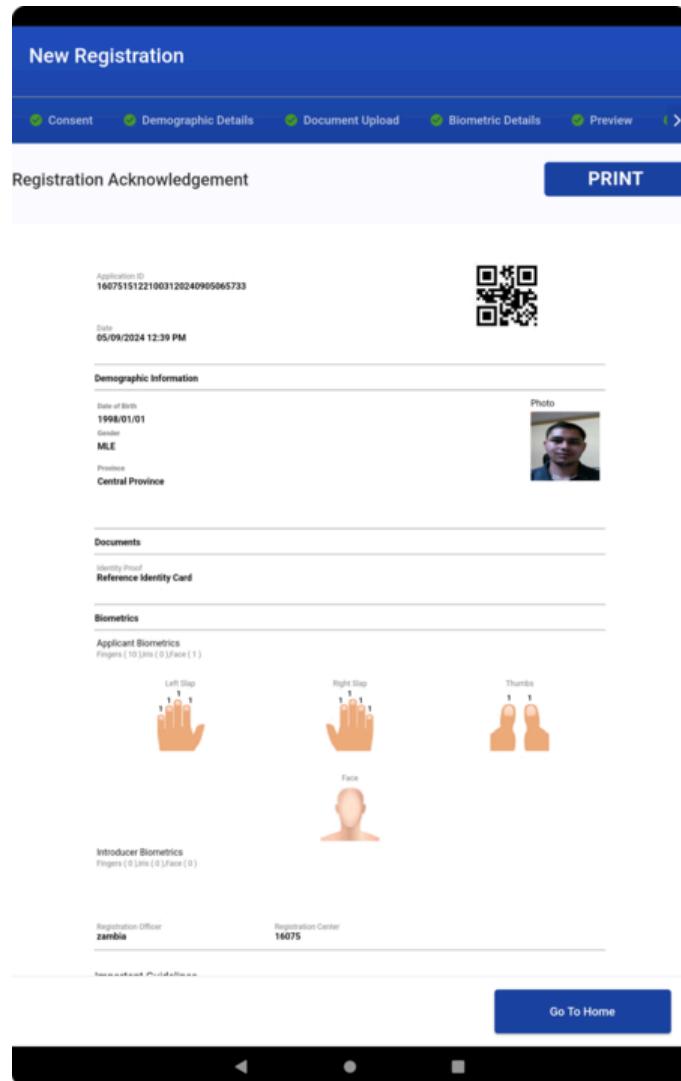
Operator's Onboarding

3. Steps to Onboard Operator's Biometrics:

- a. The user will be taken to the Biometrics Capture Homepage where he will be able to see all the below biometrics:
 - a. Face capture
 - b. Iris capture
 - c. Left-hand finger capture
 - d. Right-hand finger capture
 - e. Thumb capture



- b. The user will then have to capture all the above-listed biometrics one by one.
- c. Steps to capture the biometrics are given [here](#).
- d. Once all the biometrics are duly captured, the below acknowledgment message will be displayed on the screen.



Acknowledgement page

4. **Dashboard:** The Operator can access the dashboard where he can view the following:
 - a. **Packets created:** This will show the total number of packets created from the time the Android Registration Client was installed.
 - b. **Packets Synced:** This will show the total number of packets synced from the time the Android Registration Client was installed.
 - c. **Packets Uploaded:** This will show the total number of packets uploaded from the time the Android Registration Client was installed.
 - d. **User details:**
 - i. User ID: This will show the list of User IDs of the Users mapped to the device.
 - ii. Username: This will show the list of usernames of the Users mapped to the device.

- iii. Status: This will show the status of Users mapped to the device. This can take values such as onboarded, active, inactive, etc.

The screenshot shows the User's dashboard interface. At the top, there is a blue header bar with the word "Dashboard". Below it, there are three white cards with icons and counts: "Packets Created" (01), "Packets Synced" (00), and "Packets Uploaded" (00). The main content area is titled "Users" and contains a table with columns: "User ID", "User Name", and "Status". The table lists 15 user entries, all of whom are marked as "Active - Not Onboarded". At the bottom of the dashboard, there is a navigation bar with five items: "Dashboard" (selected), "Settings", "Profile", "Notifications", and a gear icon. The "Dashboard" item has a blue background and white text, while the others have white backgrounds and black text. Below the dashboard is a black footer bar with three small white dots.

User ID	User Name	Status
6262	6262	● Active - Not Onboarded
1616	1616	● Active - Not Onboarded
3341	3341	● Active - Not Onboarded
4646	4646	● Active - Not Onboarded
6543	6543	● Active - Not Onboarded
raja-test1	raja-test1	● Active - Not Onboarded
2222	2222	● Active - Not Onboarded
2409	2409	● Active - Not Onboarded
1919	1919	● Active - Not Onboarded
9343	9343	● Active - Not Onboarded
5151	5151	● Active - Not Onboarded
3636	3636	● Active - Not Onboarded
9657	9657	● Active - Not Onboarded
abcd	abcd	● Active - Not Onboarded
aviral	aviral	● Active - Not Onboarded

User's dashboard

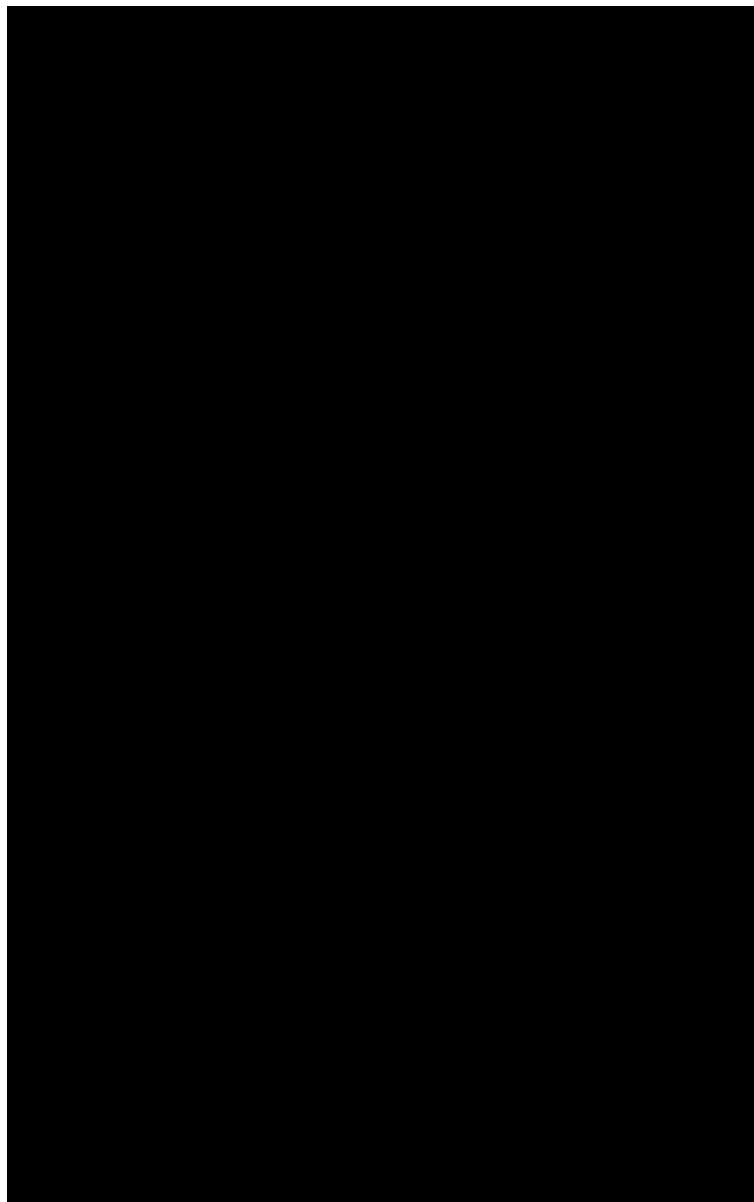
In summary, the aforementioned steps can be followed by the user (Operator/ Supervisor) to onboard themselves, update their biometrics, or view the Dashboard.

Update UIN

In a scenario where the Resident wants to update their data, they can do so by letting the Operator know their UIN along with the data that needs to be updated. Residents can update their demographic details, documents, and biometrics using this feature.

Step 1: Go to "Update UIN" from the Homepage

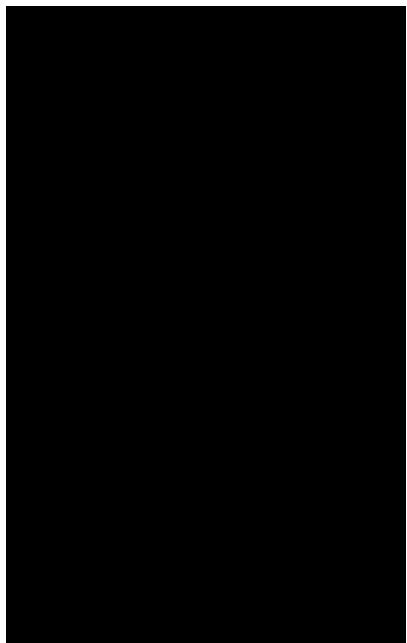
Step 2: Enter the UIN of the Resident and choose the data to be updated.



Update UIN

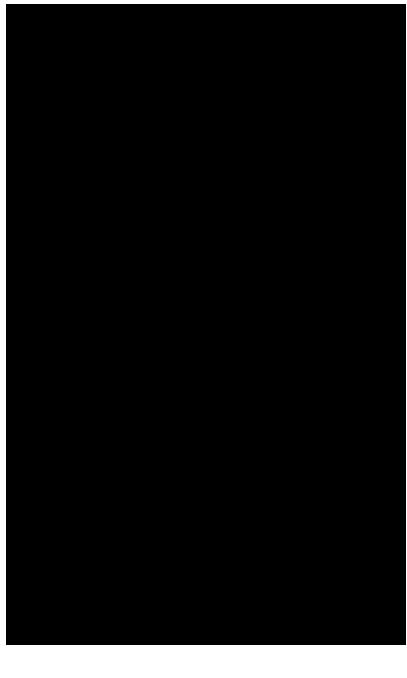
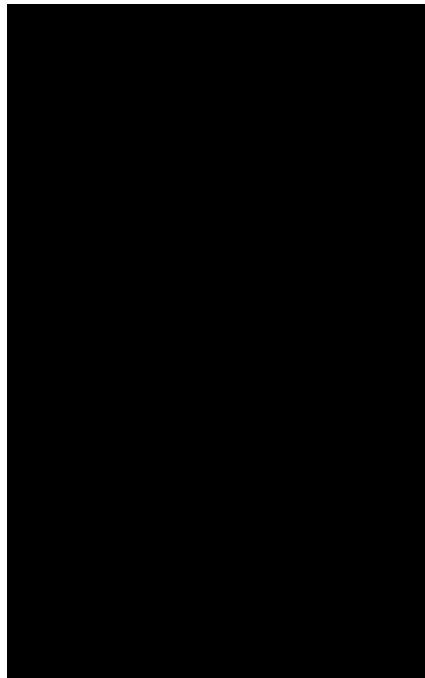
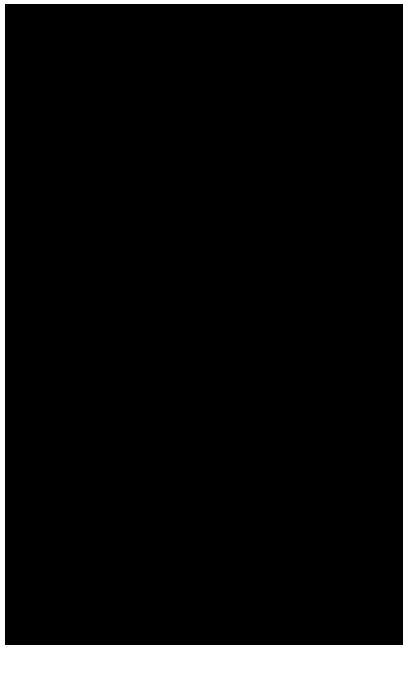
Step 3: Enter the data that the Resident wants to update. It could demographic data, documents, and biometrics.

The screenshot shows a mobile application interface titled "Update". At the top, there are three tabs: "Consent" (green dot), "Demographic Details" (blue dot, currently selected), and "Document Upload" (grey dot). Below the tabs, there are four form fields: "First Name *" with the value "sachin"; "Date of Birth *" with the value "1998/01/01" and an alternative input "26"; "Residence Status" with the value "Non-Foreigner"; and "Gender *" with the value "Male" selected from options "Male", "Female", and "Others". At the bottom right is a blue "CONTINUE" button.



Update data

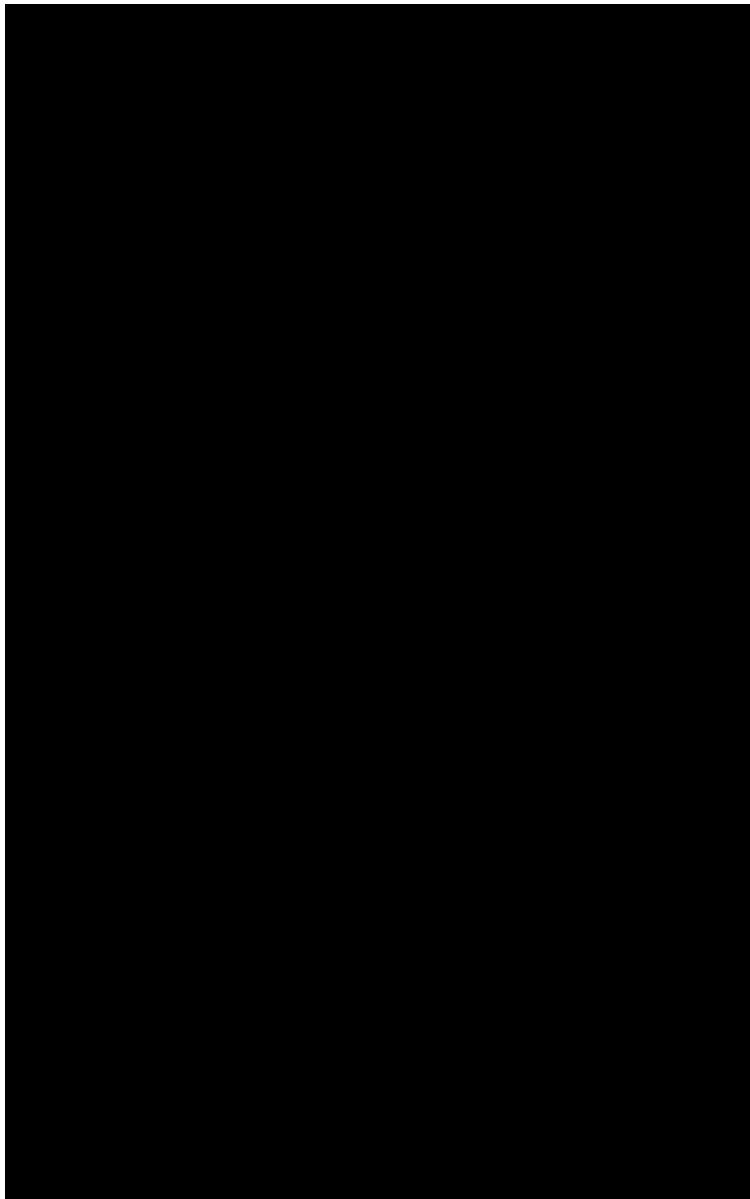
Step 4: Once all the required data is filled, the User will be taken to the Preview screen (data can still be modified) and then to the Acknowledgment screen (data cannot be updated hereafter).



Acknowledge data

Step 5: The user will then have to authenticate himself using his Username and Password. Once the authentication is successful, the packet will be uploaded to the server.

5. **Logout:** Using this feature, once the user is done with their registration and other activities, they can logout. If no background tasks are running, the user will be immediately logged out. If there are tasks (like sync) running in the background, the user will be notified about the same. From here if the User wants to cancel the logout, the background activities will keep running whereas if the user chooses to logout, they will be logged out and the background activities will be terminated.



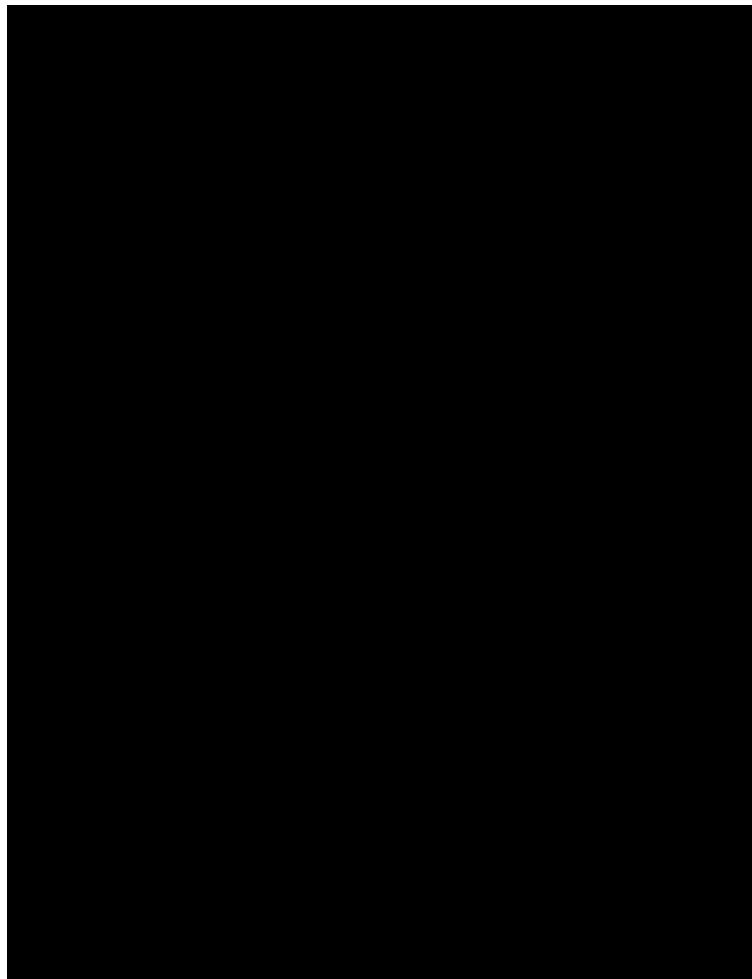
Logout screen

Update operators biometrics

This feature will be used by the operators to update their biometrics. They can follow the below steps for the same:

a. The user will be taken to the Biometrics Capture Homepage where he will be able to see all the below biometrics:

1. Face capture
2. Iris capture
3. Left-hand finger capture
4. Right-hand finger capture
5. Thumb capture



Update operator's biometrics

b. The user will then have to capture all the above-listed biometrics one by one.

c. Steps to capture the biometrics are given [here](#).

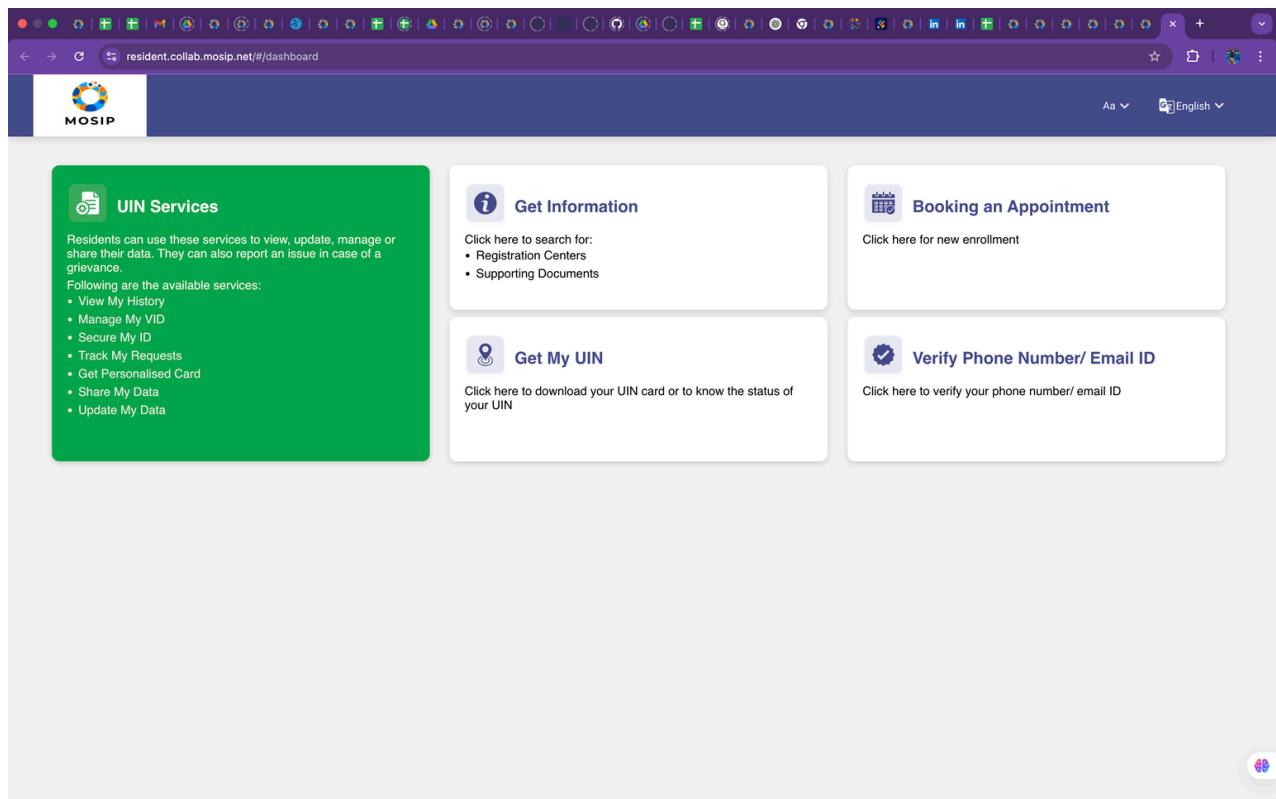
d. Once all the biometrics are duly captured, the below acknowledgment message will be displayed on the screen.

Handles Feature Authentication:

Assumption: The Handles feature is enabled, and the email ID is designated as a Handle during registration.

Scenarios: A resident attempts to log into the Resident Portal using their Handle (i.e., email ID).

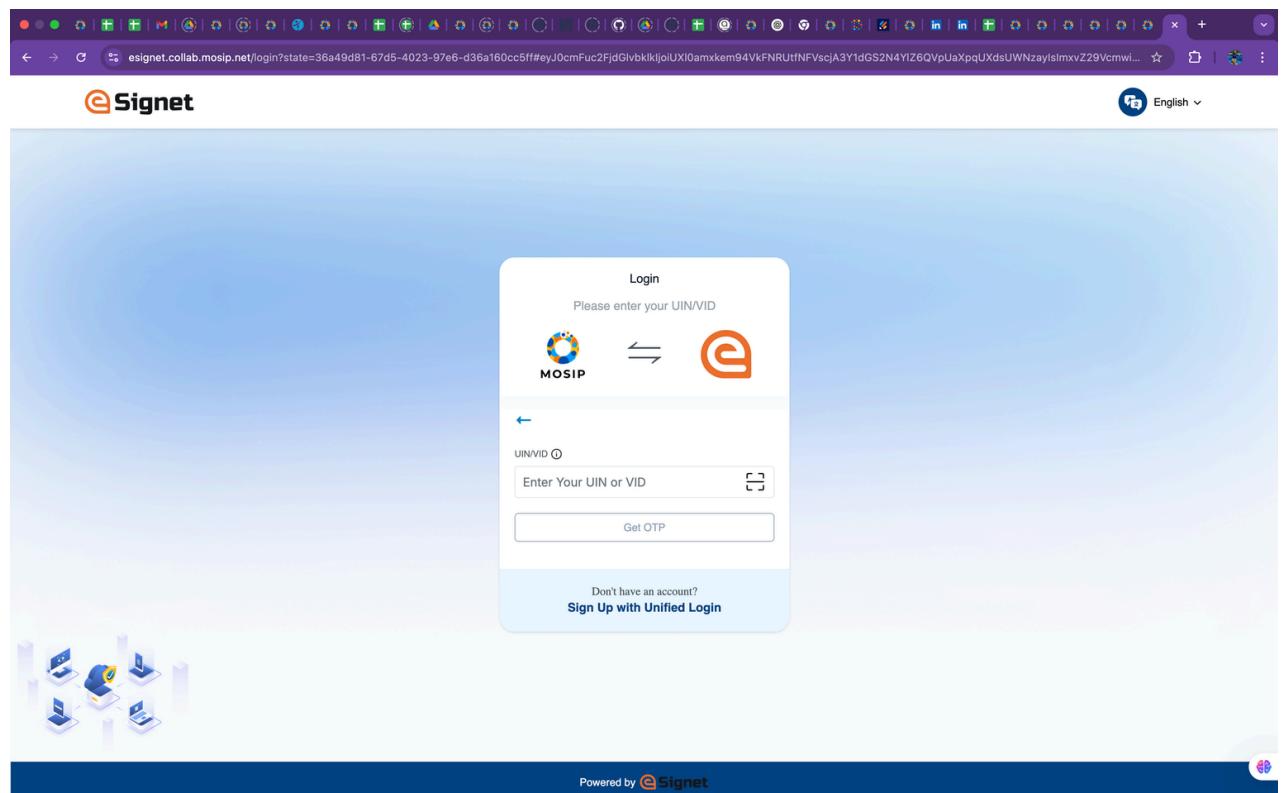
Step 1: Open the Resident Portal and navigate to "UIN Services."



Step 2: The resident will be taken to the eSignet login page.

eSignet Login Page

Step 3: Choose the option to "Login using OTP"

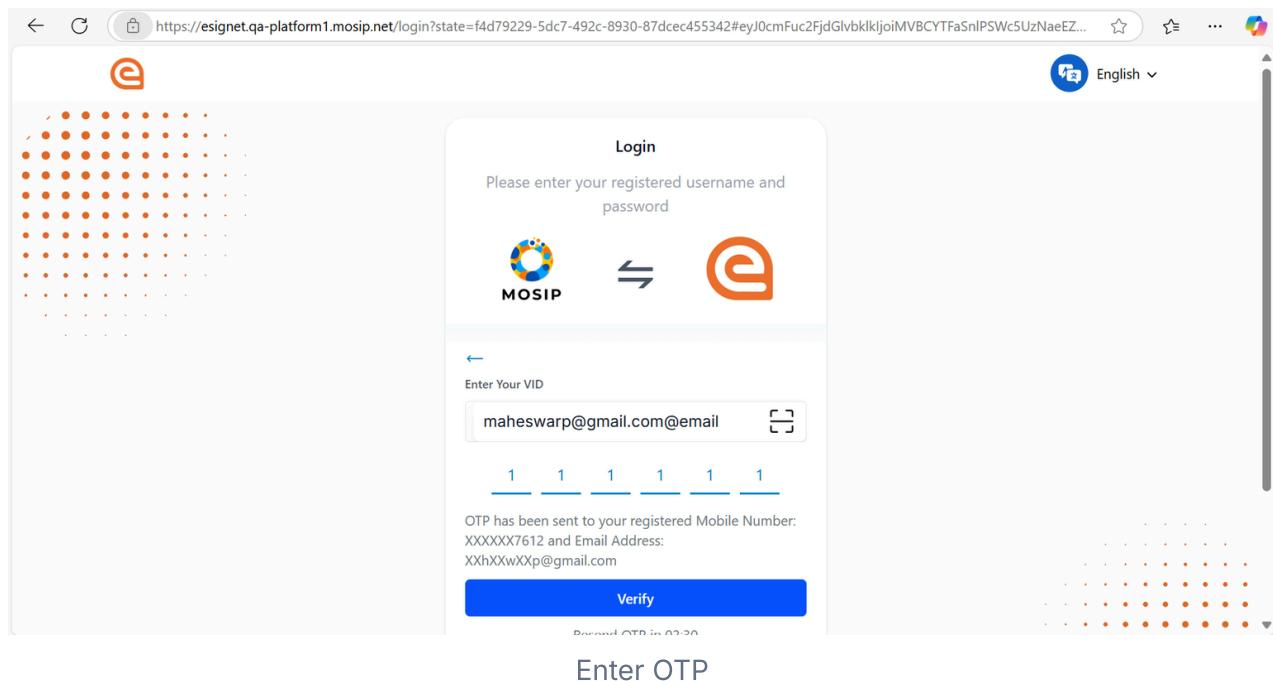


Login Using OTP

Step 4: Enter the attributes marked as Handle (email ID in this case) and click on "Get OTP". OTP will be sent to registered email ID and/or mobile number

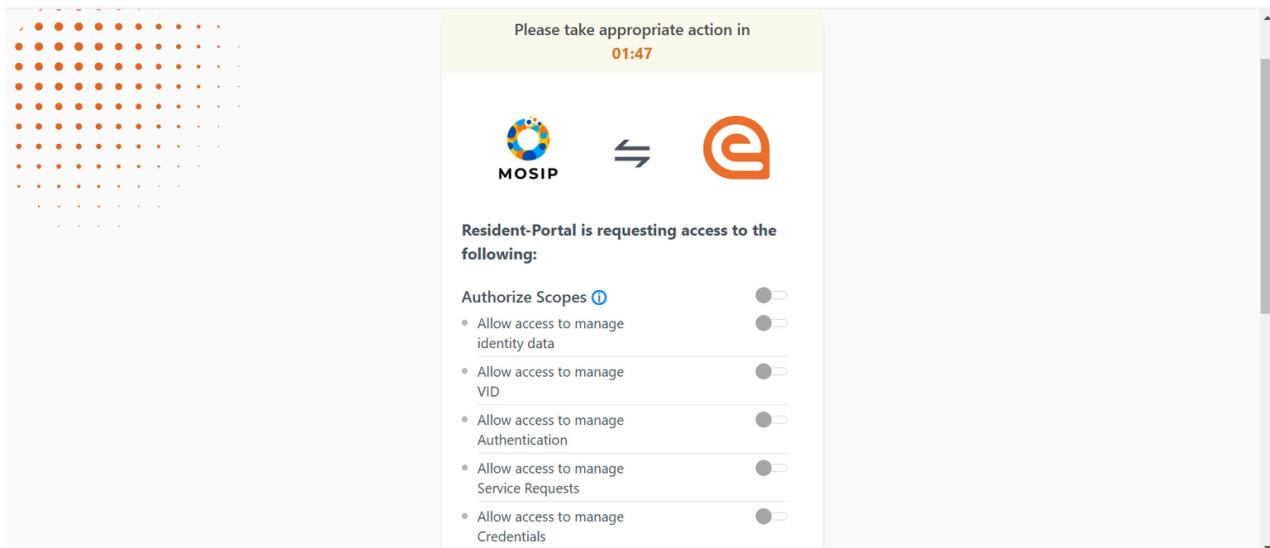
Get OTP

Step 5: Enter the OTP received over the registered email ID and/or mobile number



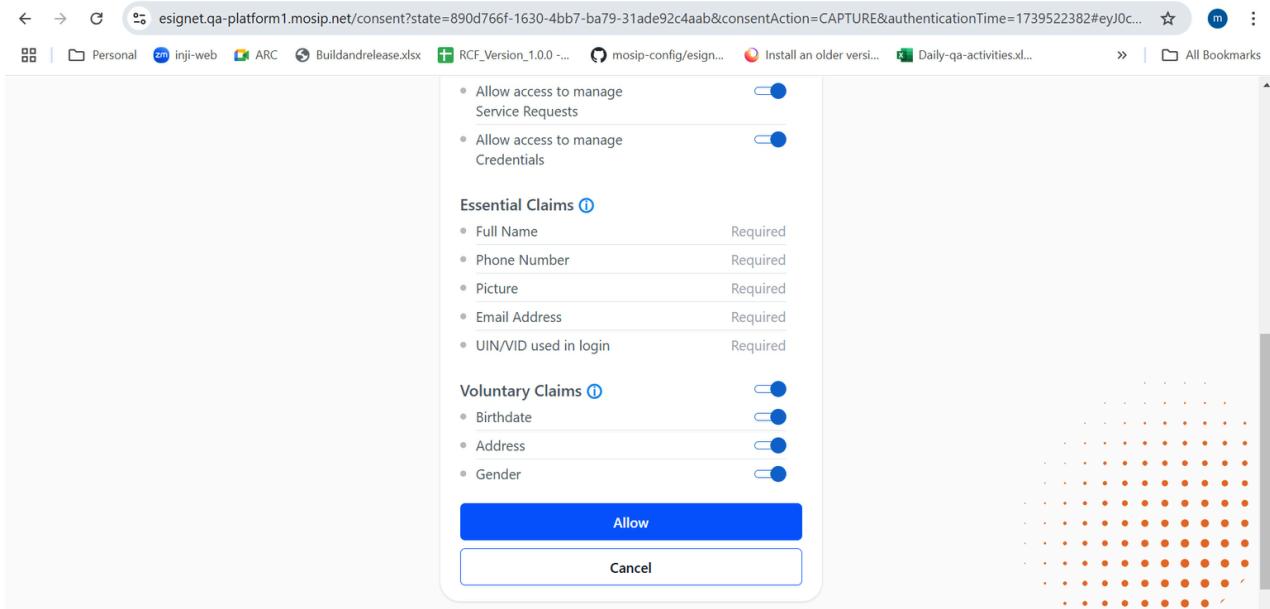
Enter OTP

Step 6: Select/De-select the Claims based on preference.



Select/Deselect Claims

And click on the "Allow" button.



Click Allow

Step 7: You have now successfully authenticated and logged into Resident Portal via eSignet using handles (email ID in this case).

Resident Services

Collab Guide

[Android Registration Client \(ARC\)](#) is a tablet application that serves as portable version of the existing desktop [Registration Client](#). It can be accessed through Android devices and also meets mobility requirements of countries adopting MOSIP Identity.

The primary objective of the tablet version is to facilitate the registration process for residents who are not able to physically visit Registration centers and also serve remote locations where setting up Registration center is not feasible. To address this challenge, the ARC was developed enabling Operators / Supervisors to easily access the remote areas and maximize resident registrations across the country.

This guide serves as a tool to demonstrate the impressive capabilities of MOSIP's system. Additionally, the primary user of this guide will be the Operator / Supervisor trying to register individuals for generating UIN.

Let's embark on this journey together to explore the potential of ARC.

 For developers setting up ARC locally, refer [Developers Guide](#).

Pre-requisites

- Reliable and consistent Internet connectivity
- Tablets running Android version 10 to 13
- Tablets with a minimum of 4 GB RAM
- The tablets should be capable of capturing fingerprints, IRIS, and face (photo) biometrics. Additionally, they should also have the ability to scan documents. However, if the tablets do not support these capabilities, MOCK SBI can be used as an alternative.

Note: For Mock MDS, click [here](#) to download the Mock MDS in your system folder, which will enable you to simulate biometric capture (without real biometric devices).

- The following details are required to access ARC in [Collab](#) environment:
 - UIN or a VID
 - Machine details
- To obtain your UIN credentials for Collab environment follow the below steps:
 - The provision of a Unique Identification Number (UIN) as a demonstration credential will enable you to have a firsthand experience of the ARC's capabilities and explore its various features.
 - Please fill the [form](#) with correct details and submit the form. Upon receiving the form, we will generate a demo credential for you. We will also register you as an Operator on Keycloak and map your device to the center to which your credential is required to be mapped.

Step-by-Step Process

Mentioned below are the steps to download, install, and use ARC.

Step 1: Download and install the APK on Android tablet. Visit [Android Registration Client](#) to access ARC in Collab environment.

Step 2: Install ARC and launch it.

Step 3: Login as an Operator with the credentials received and wait for synchronization to complete.

Step 4: Refer to our comprehensive [User Guide](#) document to learn how to register and use ARC.

Note: Please be advised that if the Android Registration Client is uninstalled and then re-installed, the aforementioned steps will need to be repeated from the start.

Additional Resources

- To know more about features of the Android Registration Client, click [here](#).

- To learn more about the ARC configurations, click [here](#).

Get in Touch

If you require any assistance or encounter any issues during the testing and integration process, kindly reach out to us through the support provided below.

- Navigate to [Community](#).
- Provide a detailed description about the support you require or provide complete information about the issue you have encountered, including steps to reproduce, error messages, logs and any other required details.

Thank you. Wishing you a pleasant experience!

Deploy

Configuration Guide

This guide provides a comprehensive list of configurable properties for the Android Registration Client. Please note that this list is not exhaustive but serves as a helpful checklist for reviewing commonly configured properties.

It is important to acknowledge that all properties listed in this guide are automatically synchronized with the Android Registration Client. These properties are sourced from the `registration-default.properties` file.

1. In the `registration-default.properties` file, update the following property to specify the field values on which you want to enable the handle. Ensure that these field values match the field values in the **IDSchema**.

```
mosip.registration.default-selected-handle-fields=email,phone
```

2. In the `id-authentication-default.properties` file, update the **Regex** to validate handles with the provided key as the postfix:

```
mosip.ida.handle-types.regex={ '@email' : '.*@email$', '@phonenumer' : '
```

3. In the `id-repository-default.properties` file, map the postfix values with the corresponding field values:

```
mosip.identity.fieldid.handle-postfix.mapping={'email':'@email', 'phone':
```

Configuration files

- `application-default.properties`
- `registration-default.properties`

SBI related configurations

Timeouts in milliseconds are set during any HTTP calls to SBI.

```
mosip.registration.mdm.connection.timeout=10000  
mosip.registration.mdm.RCAPTURE.connection.timeout=40000  
mosip.registration.mdm.MOSIPDINFO.connection.timeout=5000  
mosip.registration.mdm.MOSIPDISC.connection.timeout=5000
```

Quality score threshold based on modality, Possible values 1 to 100

```
mosip.registration.leftslap_fingerprint_threshold=40  
mosip.registration.rightslap_fingerprint_threshold=40  
mosip.registration.thumbs_fingerprint_threshold=40  
mosip.registration.iris_threshold=60  
mosip.registration.face_threshold=9
```

Retry attempts, Possible values 1 to 10

```
mosip.fingerprint_authentication.quality_score=30  
mosip.iris_authentication.quality_score=30  
mosip.face_authentication.quality_score=30
```

Quality score threshold based on the modality for operator authentication, Possible values 1 to 100`

```
mosip.fingerprint_authentication.quality_score=30  
mosip.iris_authentication.quality_score=30  
mosip.face_authentication.quality_score=30
```

Batch Size

Jobs like RID sync, packet upload, and status sync are carried out in batches, several registration records are to be executed in a batch on every trigger.

```
mosip.registration.rid_sync_batch_size=5  
mosip.registration.packet_upload_batch_size=5  
mosip.registration.status_sync_batch_size=5
```

Scheduled Jobs

Default CRON expression for scheduling the Jobs.

```
mosip.registration.sync_jobs_restart_freq=0 0 */11 ? * *
```

Other Configurations

- Enables/disables reviewer authentication on any biometric exception during registration

```
mosip.registration.review_authentication_configuration=Y
```

- If enabled, cross-check of resident's biometrics with locally stored operator biometric templates.

```
mosip.registration.mds.deduplication.enable.flag=N
```

- Minimum disk space required to create a packet - in MB

```
mosip.registration.disk_space_size=5
```

- Maximum no. of days for approved packet pending to be synced to a server beyond which Registration Client is frozen for registration

```
mosip.registration.last_export_registration_config_time=100
```

- No. of days beyond audit creation date to delete audits

```
mosip.registration.audit_log_deletion_configured_days=10
```

- The maximum duration to which registration is permitted without sync of master data

```
mosip.registration.sync_transaction_no_of_days_limit=5
```

- Allowed several invalid login attempts

```
mosip.registration.invalid_login_count=50
```

- Configuration is used to check if any sync job is missed/ failed beyond expected days, this configuration is checked every time the operator clicks on any registration process. We follow the below convention to create this config key.

```
mosip.registration.job api name as in sync_job_def
```

```
table.frequency=value in days
```

Date formats

- Date format to be displayed on acknowledgment slip, default value - dd/MM/yyyy hh:mm a

```
mosip.registration.application_date_format
```

- Date format to be displayed on Registration Client dashboard, default format - dd MMM hh:mm a

```
mosip.registration.dashboard_date_format
```

Supporting properties for 1.1.5.x server compatibility

Due to the absence of UI specifications in the 1.1.5.x versions, the android regclient addresses backward compatibility by migrating the schema of these versions to the LTS UI Spec structure.

To facilitate this migration, certain configurations and templates have been incorporated to ensure compatibility with the 1.1.5.x server. The list of these configurations is provided below.

- `mosip.registration.consent-screen-template-name=reg-consent-screen-content-template`

The consent screen is not a part of 1.1.5.x schema structure. So, we are completely fetching this consent screen content from `master.template` table, and the `templateTypeCode` for the consent screen content is mentioned in the above configuration.

- `mosip.registration.individual-biometrics-id=individualBiometrics`

The id of individual biometrics should be mentioned in the above property according to the configured 1.1.5.x schema.

- `mosip.registration.introducer-biometrics-id=guardianBiometrics`

The id of guardian/ introducer biometrics should be mentioned in the above property according to the configured 1.1.5.x schema.

- `mosip.registration.infant-agegroup-name=INFANT`

The age-group name for infants (aged below 5 years) which is configured in the configured server should be mentioned in the above property.

- `mosip.registration.agegroup-config={"INFANT":{"bioAttributes":["face"],"isGuardianAuthRequired":true}, "ADULT":{"bioAttributes":["leftEye","rightEye","rightIndex","rightLittle","rightRing","rightMiddle","leftIndex","leftLittle","leftRing","leftMiddle","leftThumb","rightThumb","face"],"isGuardianAuthRequired":false}, "SENIOR_CITIZEN":{"bioAttributes":["leftEye","rightEye","rightIndex","rightLittle","rightRing","rightMiddle","leftIndex","leftLittle","leftRing","leftMiddle","leftThumb","rightThumb","face"],"isGuardianAuthRequired":false}}`

The above property indicates a list of age groups, required bio-attributes, and a flag that indicates whether guardian authentication is required or not. This property should be changed according to the server configuration and requirements.

- `mosip.registration.allowed-bioattributes=leftEye,rightEye,rightIndex,rightLittle,rightRing,rightMiddle,leftIndex,leftLittle,leftRing,leftMiddle,leftThumb,rightThumb,face`

The above property defines the list of bio-attributes that are allowed for scanning during registration. If there are any changes in the server, it should be changed accordingly.

- `mosip.registration.default-app-type-code=000`

The above property defines the default applicantTypeCode. In LTS, we have applicanttype.mvel script to fetch the documents according to age, gender, and some other attributes. Based on the applicant details, the script returns an applicantTypeCode which can be any value from “000” to “014”, and respective documents will be fetched from `master.applicant_valid_document table`. Since we do not have this script defined in 1.1.5.x to handle this, we have added a default `applicantTypeCode`.

Templates

Ensure that the preview and acknowledge templates are present in the `template table` of `mosip_master` database with the following type code:

`reg-android-preview-template-part`

```
reg-android-ack-template-part
```

Logout

Logout from ARC will check for any running background tasks in the background. Ask the user if the user still wants to logout from the application.

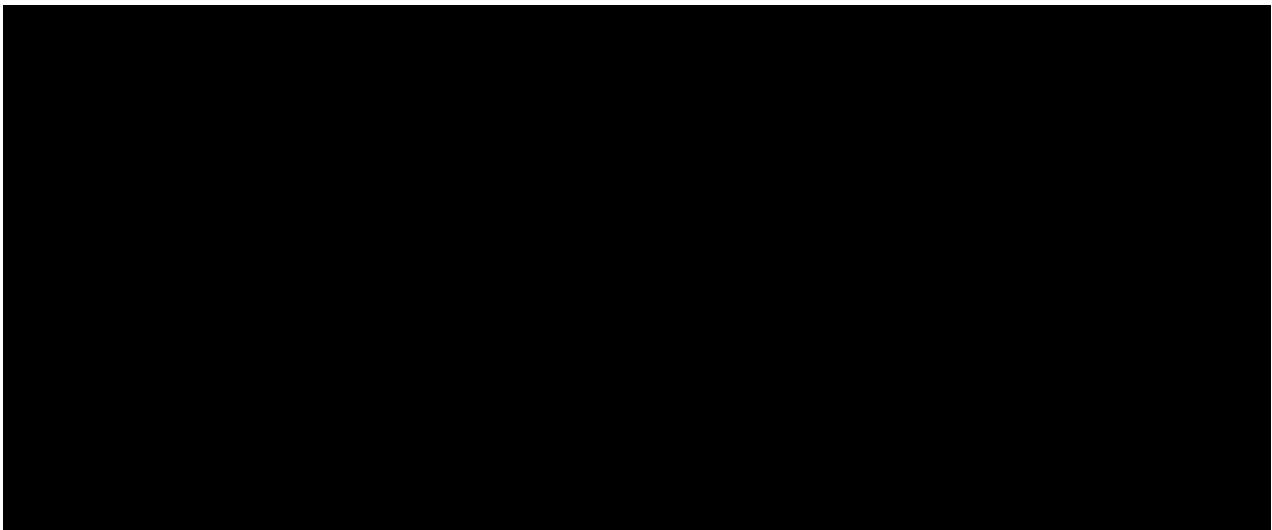
- If the user clicks on logout on the popup, all the jobs running and scheduled jobs will stop.
- If no jobs are running in the background, the user will simply log out and navigate to the login screen.
- No configuration changes are required to log out of ARC.

Registration Processor

Overview

Introduction

Registration Processor (Regproc) is a backend processing engine to enable ID Lifecycle management. The diagram below shows the Registration Processor along with the other modules that contribute to issuing a Unique Identification Number(UIN) for an individual. Internally, Regproc follows the [SEDA](#) architecture where data flows via multiple stages till the UIN is issued.



The relationship of Regproc with other services is explained here. *NOTE: The numbers do not signify a sequence of operations or control flow*

1. Registration packets are uploaded by the [Registration Client](#) to the [Packet Receiver](#).
2. After packet validation is done Regproc notifies the pre-registration application using the datasync service.
3. The quality of biometrics is checked using an external biometric SDK. This is done in Regproc's [Quality Classifier stage](#).
4. Regproc shares biometric data with [ABIS](#), Manual adjudication System, and Verification System. The policy for sharing this data is fetched from [PMS](#).
5. The above data is shared by providing a URL that partners use to fetch data. This URL is obtained from the [Datashare](#) service.

6. Regproc's [ABIS Middleware stage](#) communicates with [ABIS](#) via [Activemq](#). The ABIS performs deduplication and sends back the result to the Queue.
7. Regproc stores and updates applicant demographic and biometric information in the [ID Repository](#). Also, activate or deactivate the applicant's UIN.
8. Regproc calls IDA Internal Authentication Service to authenticate Applicant(for update flow), introducer, operator, and supervisor(when bio auth mode is used to create packet).
9. After the UIN is processed the [Printing Stage](#) calls [Credential Service](#) to create credentials for print. This credential will be pushed to websub and the Printing systems will consume the same.
10. The [Notification Service](#) is used to send email/sms notifications to the applicant after the request processing is completed on the server.
11. Regproc connects to the external "Manual Adjudication System" via a queue. Regproc sends applicant information required for adjudication in the queue and the external adjudication system consumes it. The data is shared from mosip to an external adjudication system based on policy.
12. Regproc calls [Key Manager](#) for decrypting packets and encrypting information.
13. Regproc uses [Masterdata Service](#) to validate the center, machine, user, etc.
14. Regproc connects to Virus Scanner for scanning packets in the [Packet Receiver Stage](#) and [Packet Uploader Stage](#)
15. Each Stage in regproc calls [Packet Manager](#) to read information from the packet.

1.2 LTS Registration Processor Demo + E-2-E Demo 09-02-2022



Stages and services

The Registration Processor contains several [stages and services](#).

The registration packet flows through the various stages depending on the type of flow. See [Registration Flows and Stage Sequence](#).

Note: The Print Stage has been renamed as the Credential Requestor Stage. For further information, please click [here](#).

Build and deploy

Refer to [repo](#).

Configurations

Refer to [Configuration Guide](#).

Developer Guide

To know more about the developer setup, read the [Registration Processor Developers Guide](#).

API

Refer to [API Documentation](#).

Source code

[Github repo.](#)

Develop

Build, integrate, and enhance solutions.

Registration Processor Developers Guide

Overview

[Registration Processor](#)(Regproc) is a backend processing engine to enable the ID Lifecycle management. It has several stages and services, registration packet flows through various stages depending on the type of flow.

The documentation here will guide you through the prerequisites required for the developer's setup.

Software setup

Below are a list of tools required in Registration Processor:

1. JDK 11
2. Any IDE (like Eclipse, IntelliJ IDEA)
3. Apache Maven (zip folder)
4. pgAdmin
5. Git
6. Notepad++ (optional)
7. lombok.jar (file)
8. settings.xml (document)

Follow the steps below to set up Registration Processor on your local system:

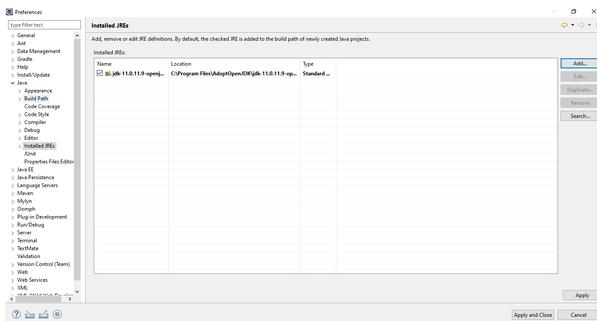
1. Download `lombok.jar` and `settings.xml` from [here](#).
2. Unzip Apache Maven and move the unzipped folder in `C:\Program Files` and `settings.xml` to `conf` folder `C:\Program Files\apache-maven-3.8.4\conf`.

3. Install Eclipse, open the `lombok.jar` file and wait for some time until it completes the scan for Eclipse IDE and then click `Install/Update`.



4. Check the Eclipse installation folder `C:\Users\userName\eclipse\jee-2021-12\eclipse` to see if the `lombok.jar` is added. By doing this, you don't have to add the dependency of `lombok` in your `pom.xml` file separately as it is auto-configured by Eclipse.

5. Configure the JDK (Standard VM) with your Eclipse by traversing through `Preferences → Java → Installed JREs`.



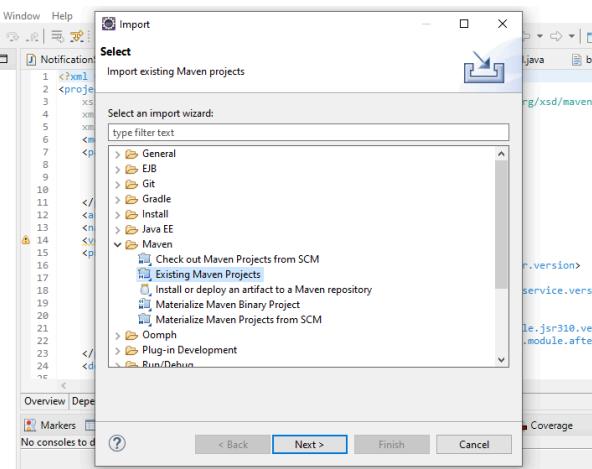
Code setup

For the code setup, clone the [registration](#) repository from and follow the guidelines mentioned in the [Code Contributions](#).

Importing and building of a project

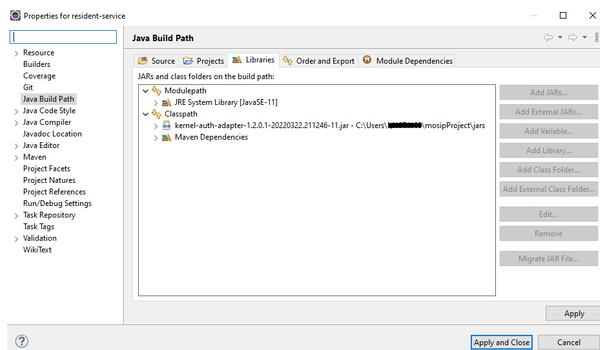
1. Open the project folder where `pom.xml` is present.
2. Open command prompt from the same folder.

3. Run the command `mvn clean install -Dgpg.skip=true -DskipTests=true` to build the project and wait for the build to complete successfully.
4. After building of a project, open Eclipse and select `Import Projects → Maven → Existing Maven Projects → Next → Browse to project directory → Finish`.
5. After successful importing of project, update the project by right-click on `Project → Maven → Update Project`.



Environment setup

1. For the environment setup, you need an external JAR that is available [here](#) with different versions. (E.g.: You can download `kernel-auth-adapter.jar` and add to project `Libraries → Classpath → Add External JARs → Select Downloaded JAR → Add → Apply and Close`).



2. Clone [mosip-config repository](#).
3. Create an empty folder inside the `mosip-config` with `sandbox-local` name and then copy and paste all the config files inside `sandbox-local` folder except

.gitignore, README and LICENSE .

4. As Registration Processor is using two properties files, `registration-processor-default` and `application-default`, you will have to configure them according to your environment. The same files are available [here](#) for reference.

5. To run the server, two files are required- [kernel-config-server.jar](#) and [config-server-start.bat](#).

6. Put both the files in the same folder and change the location attribute to `sandbox-local` folder in `config-server-start.bat` file and also check the version of `kernel-config-server.jar` towards the end of the command.

Example:

```
java -jar -Dspring.profiles.active=native -  
Dspring.cloud.config.server.native.search-  
locations=file:C:\Users\myDell\mosipProject\mosip-config\sandbox-local -  
Dspring.cloud.config.server.accept-empty=true -  
Dspring.cloud.config.server.git.force-pull=false -  
Dspring.cloud.config.server.git.cloneOnStart=false -  
Dspring.cloud.config.server.git.refreshRate=0 kernel-config-server-1.2.0-  
20201016.134941-57.jar .
```

7. Run the server by opening the `config-server-start.bat` file.

The server should now be up and running.

8. Before running any application of Registration Processor, replace the below properties in `bootstrap.properties`:

```
spring.cloud.config.uri=http://localhost:51000/config
```

```
spring.cloud.config.label=master
```

```
spring.profiles.active=default
```

```
spring.profiles.active can be dmz or mz(default)
```

9. An alternative approach to start the application is to place the dependency of application to be executed in the pom of `MOSIP stage executor` update maven and place above mentioned properties in the `bootstrap.properties` and then start `MOSIP stage executor`.

Test

Credential Requestor Stage

Rename - "Print Stage" to "Credential Requestor Stage"

Note: The change in nomenclature from 'Print Stage' to 'Credential Requestor Stage' was implemented to enhance clarity and better reflect its expanded functionality.

History of Print Stage in MOSIP

In MOSIP 0.9.0, the "print stage" was primarily designed to facilitate the submission of printing requests. The core functionality of print stage revolved around initiating a request for the physical printing of credentials. However, as the system evolved and incorporated additional features, the scope of the print stage expanded beyond its original purpose.

In the evolved approach, at the print stage, apart from processing a print request, the application enables partner-specific print capabilities. Contrary to its initial purpose, the print stage no longer serves the singular function of printing credentials. Instead, it has transformed into a multifaceted component with enhanced set of responsibilities.

In the current system, the print stage's role extends beyond traditional printing activities. Its primary function now revolves around initiating a request for credential generation once a Unique Identification Number (UIN) is generated. This request is not aimed at physical printing but serves as a mechanism to gather additional information for specific partners. These partners may require supplementary data beyond what is initially generated with the UIN.

Therefore, in the evolved approach, the print stage has transitioned from being a straightforward printing request to a more versatile component that manages the initiation of credential requests tailored to partner-specific information needs. This

adaptation reflects the system's responsiveness to changing requirements and the dynamic nature of credentialing processes. This reason led us to re-name the "Print Stage" to the "Credential Requestor Stage" as this name serves the purpose of the work executed by this stage.

Introduction

The Credential Requestor Stage in MOSIP, formerly known as the Print Stage, is a crucial component used to request credentials for configurable partners after the UIN is generated. This stage enables countries to share information with multiple partners, each with specific needs after UIN generation. Partners, such as Print Partners and Digital Card Partners, may require demographic or biometric information to perform operations.

What is the credential requestor stage?

The Credential Requestor Stage plays a crucial role in the MOSIP system, serving as a mechanism to solicit credentials from configurable partners post-UIN generation. In this context, partners, previously registered with MOSIP, require demographic and biometric data to execute their respective operations. For instance, a Print Partner necessitates specific demographic details for the purpose of printing cards. Similarly, digital card partners seek demographic information to generate digital cards. Additionally, DPGs might seek confirmation of successful UIN generation to integrate this information into their systems. The Credential Requestor Stage facilitates various use cases where the country aims to share pertinent information with multiple partners subsequent to UIN generation.

What are the latest changes done in the credential requestor stage?

Partner Profile Configuration

MOSIP has introduced a new partner profile for the Credential Requestor Stage. The partner profiles are maintained [here](#).

Sample Partner Profile:

```
jsonCopy code{
  "partners": [
    {
      "id": "digitalcardPartner",
      "partnerId": "mpartner-default-digitalcard",
      "credentialType": "PDFCard",
      "template": "RPR_UIN_CARD_TEMPLATE",
      "appIdBasedCredentialIdSuffix": ".pdf",
      "process": null,
      "metaInfoFields": null
    },
    {
      "id": "printPartner",
      "partnerId": "mpartner-default-print",
      "credentialType": "euin",
      "template": "RPR_UIN_CARD_TEMPLATE",
      "appIdBasedCredentialIdSuffix": null,
      "process": null,
      "metaInfoFields": null
    },
    {
      "id": "opencrvsPartner",
      "partnerId": "opencrvs-partner",
      "type": "opencrvs",
      "template": "RPR_UIN_CARD_TEMPLATE",
      "process": ["OPENCRVS_NEW"],
      "metaInfoFields": ["opencrvsBRN"]
    }
  ]
}
```

Field Description

- `id`: Logical unique identifier
- `partnerId`: Partner identifier configured in MOSIP
- `credentialType`: Type of credential configured in MOSIP
- `template`: Template used for generating the credential

- `appIdBasedCredentialIdSuffix`: Applicable for special conditions where the credential ID is the application ID itself, with an optional suffix (for example, `.pdf`). Currently, this is applicable for digital card credentials.
- `process`: If applicable for a particular process. If applicable for all processes, value is `null`
- `metaInfoFields`: Meta information fields to be sent as additional information while generating the credential

Configuration Changes

Once the partner profile is configured, the System Integrator (SI) should make changes to the following configurations:

- `mosip.registration.processor.credential.partner-profiles`: Specify the file name for the partner profiles. By default its → `registration-processor-credential-partners.json`. If a country intends to change the file name only then this configuration should be updated otherwise default configuration can be used
- `mosip.registration.processor.credential.default.partner-ids`: Specify default partner IDs for which credentials will be created automatically
- `mosip.registration.processor.credential.conditional.partner-id-map`: Define conditions for conditional partners. Credentials for these partners will be requested only if the conditions are met. Use `MVEL` expressions for conditions
- `mosip.registration.processor.credential.conditional.no-match-partner-ids`: Specify a partner ID to be used when no conditions are met for conditional partners

Conditional Partner Requests

- The stage will create credentials for default partner IDs by default
- For conditional partners, credentials will be requested only if they match a particular `MVEL` expression
- `MVEL` expressions can be written on any identity field as well as meta info field
- If there is no condition match for conditional partners, SI can configure a no-match partner, which will be used when no conditional partner match is found

Configuration File Locations

1. **Credential Requestor Stage Configuration:** Click [here](#) to view credential requestor stage configuration details.
2. **Partner Profile Configuration:** Click [here](#) to view partner profile configuration details.

Conclusion

The Credential Requestor Stage configuration is an essential part of MOSIP's functionality, enabling seamless communication with partners and ensuring the secure exchange of information post-UIN generation.

Note: Ensure that configured IDs are logically unique and consistent across future configurations.

Manual Adjudication and Verification

- When biometric duplicates are found in ABIS, the MOSIP system sends a request for manual adjudication to the Manual Adjudication System via a queue.
- The system integrator can build the Manual Adjudication System, which would be listening to the `MOSIP-to-ManualAdjudication` queue for any manual adjudication requests and sends a response back in the `ManualAdjudication-to-MOSIP` system after verifying the data.
- The data sent to the Manual Adjudication System is driven by a policy defined in MOSIP and the specification is similar to ABIS identify request.

The manual adjudication stage in [registration processor](#) performs the following functions:

- Sends the applicant's demographic and biometric information (via queue + Datashare) to Manual Adjudication System (MAS).
- Receives decision from MAS and accordingly forwards the packets.
- If rejected, notifies the applicant.

Manual Adjudication request to Queue is as follows:

```
{
  "id": "mosip.manual.adjudication.adjudicate",
  "version": "1.0",
  "requestId": "987654321-89AB-CDEF-0123-456789ABCDEF",
  "requesttime": "2019-02-14T12:40:59.768Z",
  "referenceId": "27847657360002520181208123456",
  "referenceURL": "<datashare url for regid>",
  "gallery": {
    "referenceIds": [
      {
        "referenceId": "27847657360002520181208123451",
        "referenceURL": "<data share for matchedRegId>"
      },
      {
        "referenceId": "27847657360002520181208123452",
        "referenceURL": "<data share for matchedRegId>"
      }
    ],
    "addtional": [
      {
        "abisId": "<abis app code>",
        "response": "<abis response text received>"
      }
    ]
  }
}
```

Request parameters

`requestId` : request_id that is associated with each request present in `reg_manual_verification` table.

`referenceId` : reg_id that is associated with each request present in `reg_manual_verification` table.

`referenceURL` : Datashare url of biometrics, demographics(identity), audits, metainfo, documents of reg_id

`gallery` : contains the matched ref_id and referenceURL of matched ref_id.

Sample request

```
{  
  "id": "mosip.manual.adjudication.adjudicate",  
  "version": "1.0",  
  "requestId": "4d4f27d3-ec73-41c4-a384-bf87fce4969e",  
  "referenceId": "10002100741000320210107125533",  
  "requesttime": "2021-01-19T07:16:22.930Z",  
  "referenceURL": "http://datashare-service/v1/datashare/get/mpolicy-default",  
  "addtional": null,  
  "gallery": {  
    "referenceIds": [  
      {  
        "referenceId": "10002100741000120210107111325",  
        "referenceURL": "http://datashare-service/v1/datashare/get/mpolicy-default"  
      }  
    ]  
  }  
}
```

Manual adjudication response structure from MAS

```
{
  "id": "mosip.manual.adjudication.adjudicate",
  "requestId": "987654321-89AB-CDEF-0123-456789ABCDEF",
  "responsetime": "2019-02-14T12:40:59.768Z",
  "returnValue": "1",
  "candidateList": [
    {
      "count": "1",
      "candidates": [
        {
          "referenceId": "27847657360002520181208123451",
          "analytics": {
            //This section is optional
            "primaryOperatorID": "110011",
            "primaryOperatorComments": "<comments provided by operator>",
            "secondaryOperatorID": "110012",
            "secondaryOperatorComments": "<comments provided by operator>",
            "key1": "value1",
            "key2": "value2"
          }
        }
      ],
      "analytics": {
        // This section is optional
        "key1": "value1",
        "key2": "value2"
      }
    }
  ]
}
```

Response parameters

`returnValue` : 1-Success, 2-Failure

`candidateList` : It contains matched candidate referencelds, count and analytics.

Scenario: No match

Response structure- Not Matched (No Duplicate Profile found)

```
{  
  "id": "mosip.manual.adjudication.adjudicate",  
  "requestId": "c278b2f1-29f7-4d1a-9fa7-e93e3f932816",  
  "responsetime": "2022-09-23T06:31:31.7456782+00:00",  
  "returnValue": "1",  
  "candidateList": {  
    "count": "0",  
    "candidates": [],  
    "analytics": null  
  }  
}
```

Scenario: There are matches

Response structure

```
{  
  "id": "mosip.manual.adjudication.adjudicate",  
  "requestId": "58d5bb0e-e65e-4907-b452-81edbf3ae46",  
  "responsetime": "2022-09-23T06:33:11.9869624+00:00",  
  "returnValue": "1",  
  "candidateList": {  
    "count": "1",  
    "candidates": [  
      {  
        "referenceId": "10001100010000620220923053704",  
        "analytics": {  
          "primaryOperatorID": "admin",  
          "primaryOperatorComments": "MATCHED",  
          "Face": "F",  
          "Finger": "Right_Thumb,Right_Index,Right_Middle,Right_Ring,Right_Pinky",  
          "Iris": "Right_Iris,Left_Iris"  
        }  
      }  
    ],  
    "analytics": null  
  }  
}
```

Datashare structure

Datashare contains biometrics, identity documents, metainfo, audits related to particular rid and the datashare URL contains encrypted form of this data.

Note: Datashare encryption using partner key and decryption in MAS is using private key of that partner.

Sample Request URL

GET https://datashare-service/v1/datashare/get/mpolicy-default-adjudication/mpartner-default-adjudication/mpartner-default-adjudicationmpolicy-default-adjudication202011110619201EpLEjvD

Sample Encrypted Response

- ⓘ **The structure of the encrypted data downloaded from referenceURL in MOSIP 1.2.0 or later versions.** The data downloaded would be URL-safe base64 encoded. Hence, after decoding the data will be in the below format. It will be divided into two parts after splitting using #KEY_SPLITTER#.

Encrypted Key Data	KEY_SPLITTER	Encrypted Actual Data
Block 1	#KEY_SPLITTER#	Block 2

ⓘ **Block 1:**

Block 1, i.e. the encrypted key data is again split into three parts, • The 1st part is **VER_BYTES** (version bytes). The Current version constant is set as VER_R2 and this is present in the first 6 bytes of Block 1.

- The 2nd part is the **Certificate Thumbprint** i.e. the key identifier which is present in the next 32 bytes after VER_BYTES.
- The 3rd part is the **Encrypted Random AES Key**, encrypted with the RSA OAEP - SHA256-MFG1. This constitutes the remaining 256 bytes of Block 1.

Block 2:

Block 2, i.e. the encrypted actual data is again split into two parts,

- The 1st part is the random 32 bytes which will be used as **AAD** in AES encryption (first 32 bytes). From this 32 bytes AAD data, the first 12 bytes is **IV/Nonce**.
- The 2nd part is the encrypted data which is encrypted using AES GCM PKCS5Padding.

The structure of the encrypted data downloaded from referenceURL in MOSIP 1.1.5.5 or prior versions.

The data downloaded would be base64 encoded. Hence, after decoding the data will be in the below format. It will be divided into two parts after splitting using #KEY_SPLITTER#.

Encrypted Key Data	KEY_SPLITTER	Encrypted Actual Data
--------------------	--------------	-----------------------

Block 1	#KEY_SPLITTER#	Block 2
---------	----------------	---------

(i) Block 1:

Block 1, i.e. the encrypted key data is again split into two parts,

- The first part is the **Certificate Thumbprint** i.e. the key identifier which is the first 32 bytes in Block 1.
- The second part is the **Encrypted Random AES Key** which is encrypted with RSA OAEP - SHA256-MFG1. This constitutes the remaining 256 bytes of Block 1.

Block 2:

Block 2, i.e. the encrypted actual data is again split into two parts,

- The 1st part is the **Encrypted data**, encrypted using AES GCM PKCS5Padding.
- The 2nd part is **IV/Nonce** i.e. the last 32 bytes appended after encrypted data.

Sample response in case of Authentication Failure

```
{
  "id": null,
  "version": null,
  "responsetime": "2021-02-05T06:29:48.257Z",
  "metadata": null,
  "response": null,
  "errors": [
    {
      "errorCode": "KER-ATH-401",
      "message": "Authentication Failed"
    }
  ]
}
```

Possible Error Codes and messages from Datashare URL

Error Code	Error Message
DAT-SER-003	File does not exist or File is empty
DAT-SER-006	Data share not found
DAT-SER-006	Data share usage expired
KER-ATH-401	Authentication failed
KER-ATH-403	Forbidden

Policy structure

```
partner Id: mpartner-default-adjudication policy Id: mpolicy-default-adjudication
```

```
{  
  "shareableAttributes": [  
    {  
      "attributeName": "biometrics",  
      "group": "CBEFF",  
      "source": [  
        {  
          "attribute": "registration-client/NEW/individualBiometrics",  
          "filter": [  
            {  
              "type": "Iris"  
            }  
          ]  
        },  
        {  
          "attribute": "CNIE/verification/biometrics",  
          "filter": [  
            {  
              "type": "Finger"  
            }  
          ]  
        },  
        {  
          "attribute": "CNIE/verification/biometrics",  
          "filter": [  
            {  
              "type": "Face"  
            }  
          ]  
        }  
      ],  
      "encrypted": true,  
      "format": "extraction"  
    },  
    {  
      "attributeName": "fullName",  
      "source": [  
        {  
          "attribute": "fullName"  
        }  
      ],  
      "encrypted": true  
    },  
    {  
      "attributeName": "dateOfBirth",  
      "source": [  
        {  
          "attribute": "dateOfBirth"  
        }  
      ]  
    }  
  ]  
}
```

```
        }
    ],
    "encrypted": true
},
{
    "attributeName": "gender",
    "source": [
        {
            "attribute": "gender"
        }
    ],
    "encrypted": true
},
{
    "attributeName": "phone",
    "source": [
        {
            "attribute": "phone"
        }
    ],
    "encrypted": true
},
{
    "attributeName": "email",
    "source": [
        {
            "attribute": "email"
        }
    ],
    "encrypted": true
},
{
    "attributeName": "addressLine1",
    "source": [
        {
            "attribute": "addressLine1"
        }
    ],
    "encrypted": true
},
{
    "attributeName": "addressLine2",
    "source": [
        {
            "attribute": "addressLine2"
        }
    ],
    "encrypted": true
}
```

```
        },
        {
            "attributeName": "addressLine3",
            "source": [
                {
                    "attribute": "addressLine3"
                }
            ],
            "encrypted": true
        },
        {
            "attributeName": "region",
            "source": [
                {
                    "attribute": "region"
                }
            ],
            "encrypted": true
        },
        {
            "attributeName": "province",
            "source": [
                {
                    "attribute": "province"
                }
            ],
            "encrypted": true
        },
        {
            "attributeName": "city",
            "source": [
                {
                    "attribute": "city"
                }
            ],
            "encrypted": true
        },
        {
            "attributeName": "postalCode",
            "source": [
                {
                    "attribute": "postalCode"
                }
            ],
            "encrypted": true
        }
    ],
    "dataSharePolicies": {
```

```

    "typeUIShare": "Data Share",
    "validForInMinutes": "30",
    "transactionsAllowed": "100000",
    "encryptionType": "none",
    "shareDomain": "datashare.datashare",
    "source": "Packet Manager"
}
}

```

Configuration used in Manual adjudication

```

registration.processor.queue.manual.adjudication.request
registration.processor.queue.manual.adjudication.request.messageTTL
registration.processor.manual.adjudication.policy.id
registration.processor.manual.adjudication.subscriber.id
registration.processor.manual.adjudication.subscriber.id
mosip.regproc.data.share.protocol
mosip.regproc.data.share.internal.domain.name
registration.processor.queue.manual.adjudication.request
registration.processor.queue.manual.adjudication.request.messageTTL
registration.processor.manual.adjudication.policy.id
registration.processor.manual.adjudication.subscriber.id
registration.processor.manual.adjudication.subscriber.id
mosip.regproc.data.share.protocol
mosip.regproc.data.share.internal.domain.name

```

In `registration-processor-default.properties`, the possible Error codes are as follows:

Error Code	Error Message
RPR-MVS-000	manual verification failed
RPR-MVS-001	Registration Id should not empty or null
RPR-MVS-002	No matched reference id found for given RID
RPR-MVS-025	Manual adjudication failed
RPR-MVS-022	Registration Id should not empty or null
RPR-MVS-022	<code>TableNotFoundException</code> in Manual verification

RPR-MVS-021	Manual verification rejected
RPR-MVS-025	Manual verification resend to queue
---	---

Verification

This stage is applicable only if required biometrics are not present in the packet as per country configuration.

- Sends applicant's demographic documents (via Queue + Datashare) to external Verification System (VS).
- Receives decision from VS and accordingly forwards the packets.
- If rejected, notifies the applicant.

Verification request to Queue is as follows:

```
{
  "id": "mosip.verification.adjudicate",
  "version": "1.0",
  "requestId": "987654321-89AB-CDEF-0123-456789ABCDEF",
  "requesttime": "2019-02-14T12:40:59.768Z",
  "referenceId": "27847657360002520181208123456",
  "referenceURL": "<datashare url for regid>",
  "additional": [
    {
      "abisId": "<abis app code>",
      "response": "<abis response text received>"
    }
  ]
}
```

`requestId` : verification_request_id that is associated with each request present in `reg_verification` table.

`referenceId` : reg_id that is associated with each request present in `reg_verification` table.

`referenceURL` : Datashare URL of biometrics, demographics(identity) ,audits, metainfo, documents of reg_id .

Sample request

```
{
  "id": "mosip.verification.adjudicate",
  "version": "1.0",
  "requestId": "4d4f27d3-ec73-41c4-a384-bf87fce4969e",
  "referenceId": "10002100741000320210107125533",
  "requesttime": "2021-01-19T07:16:22.930Z",
  "referenceURL": "http://datashare-service/v1/datashare/get/mpolicy-default",
  "addtional": null,
}
```

Sample Response

```
{
  "id": "mosip.verification.adjudicate",
  "requestId": "4d4f27d3-ec73-41c4-a384-bf87fce4969e",
  "responsetime": "2021-01-19T13:16:22.930Z",
  "returnValue": "1"
}
```

Response parameters

`returnValue` : 1-success, 2-failure

Datashare structure

Datashare contains biometrics, identity, documents, metainfo, audits related to particular `rid` and datashare URL contains encrypted form of this data.

Note: Datashare encryption using partner key and decryption in MAS is using private key of that partner.

Sample request URL

GET https://datashare-service/v1/datashare/get/mpolicy-default-adjudication/mpartner-default-adjudication/mpartner-default-adjudicationmpolicy-default-adjudication20201110619201EpLEjvD

Sample Encrypted Response

- ① **The structure of the encrypted data downloaded from referenceURL in MOSIP 1.2.0 or later versions.** The data downloaded would be URL-safe base64 encoded. Hence, after decoding the data will be in the below format. It will be divided into two parts after splitting using #KEY_SPLITTER#.

Encrypted Key Data	KEY_SPLITTER	Encrypted Actual Data
Block 1	#KEY_SPLITTER#	Block 2

① Block 1:

Block 1, i.e. the encrypted key data is again split into three parts,

- The 1st part is **VER_BYT**ES**** (version bytes). The Current version constant is set as VER_R2 and this is present in the first 6 bytes of Block 1.

- The 2nd part is the **Certificate Thumbprint** i.e. the key identifier which is present in the next 32 bytes after VER_BYT**E**S.
- The 3rd part is the **Encrypted Random AES Key**, encrypted with the RSA OAEP - SHA256-MFG1. This constitutes the remaining 256 bytes of Block 1.

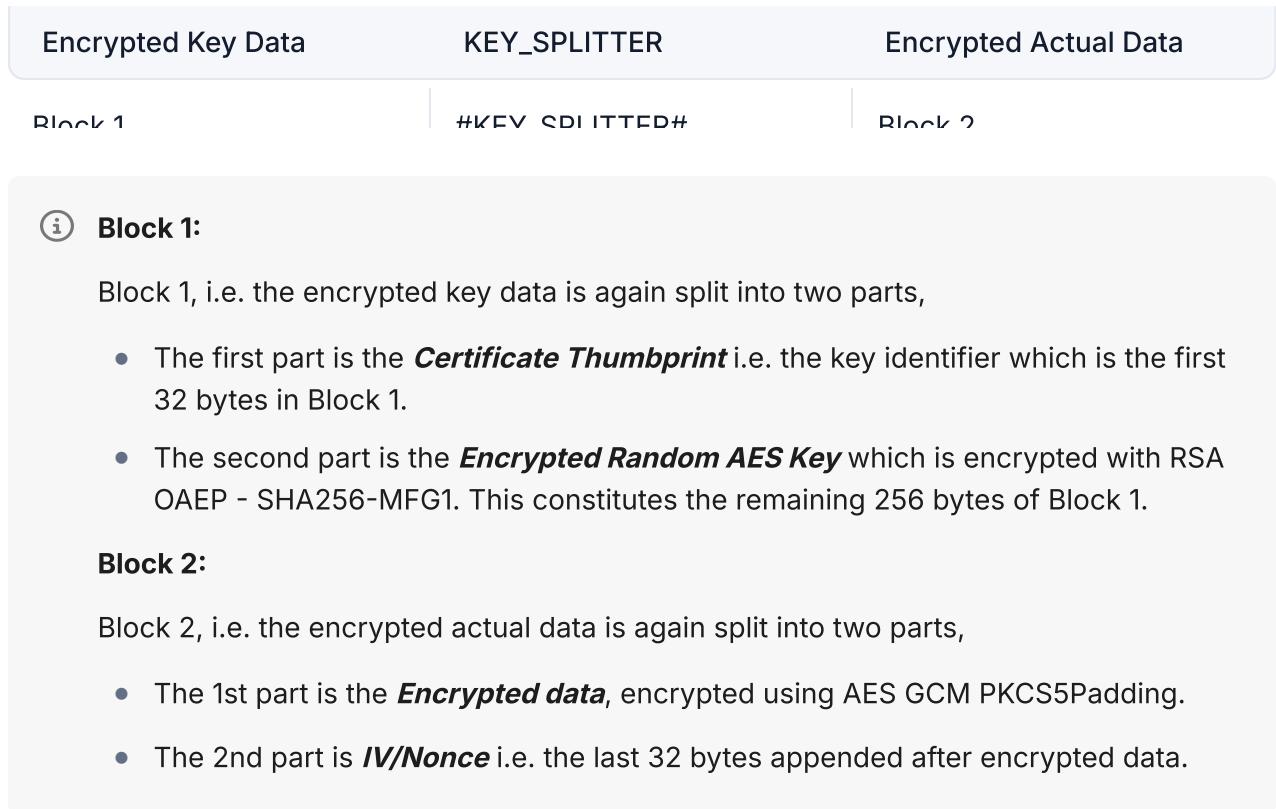
Block 2:

Block 2, i.e. the encrypted actual data is again split into two parts,

- The 1st part is the random 32 bytes which will be used as **AAD** in AES encryption (first 32 bytes). From this 32 bytes AAD data, the first 12 bytes is **IV/Nonce**.
- The 2nd part is the encrypted data which is encrypted using AES GCM PKCS5Padding.

The structure of the encrypted data downloaded from referenceURL in MOSIP 1.1.5.5 or prior versions.

The data downloaded would be base64 encoded. Hence, after decoding the data will be in the below format. It will be divided into two parts after splitting using #KEY_SPLITTER#.



Sample Response in case of Authentication Failure

```
{
  "id": null,
  "version": null,
  "responsetime": "2021-02-05T06:29:48.257Z",
  "metadata": null,
  "response": null,
  "errors": [
    {
      "errorCode": "KER-ATH-401",
      "message": "Authentication Failed"
    }
  ]
}
```

Possible Error codes and Messages from Datashare URL

Error Code	Error Message
DAT-SER-003	File does not exists or File is empty

DAT-SER-006	Data share not found
DAT-SER-006	Data share usage expired
KER-ATH-401	Authentication Failed

Policy structure

`partner Id`: mpartner-default-adjudication `policy Id`: mpolicy-default-adjudication

```
{  
  "shareableAttributes": [  
    {  
      "attributeName": "biometrics",  
      "group": "CBEFF",  
      "source": [  
        {  
          "attribute": "registration-client/NEW/individualBiometrics",  
          "filter": [  
            {  
              "type": "Iris"  
            }  
          ]  
        },  
        {  
          "attribute": "CNIE/verification/biometrics",  
          "filter": [  
            {  
              "type": "Finger"  
            }  
          ]  
        },  
        {  
          "attribute": "CNIE/verification/biometrics",  
          "filter": [  
            {  
              "type": "Face"  
            }  
          ]  
        }  
      ],  
      "encrypted": true,  
      "format": "extraction"  
    },  
    {  
      "attributeName": "fullName",  
      "source": [  
        {  
          "attribute": "fullName"  
        }  
      ],  
      "encrypted": true  
    },  
    {  
      "attributeName": "dateOfBirth",  
      "source": [  
        {  
          "attribute": "dateOfBirth"  
        }  
      ]  
    }  
  ]  
}
```

```
        }
    ],
    "encrypted": true
},
{
    "attributeName": "gender",
    "source": [
        {
            "attribute": "gender"
        }
    ],
    "encrypted": true
},
{
    "attributeName": "phone",
    "source": [
        {
            "attribute": "phone"
        }
    ],
    "encrypted": true
},
{
    "attributeName": "email",
    "source": [
        {
            "attribute": "email"
        }
    ],
    "encrypted": true
},
{
    "attributeName": "addressLine1",
    "source": [
        {
            "attribute": "addressLine1"
        }
    ],
    "encrypted": true
},
{
    "attributeName": "addressLine2",
    "source": [
        {
            "attribute": "addressLine2"
        }
    ],
    "encrypted": true
}
```

```
        },
        {
            "attributeName": "addressLine3",
            "source": [
                {
                    "attribute": "addressLine3"
                }
            ],
            "encrypted": true
        },
        {
            "attributeName": "region",
            "source": [
                {
                    "attribute": "region"
                }
            ],
            "encrypted": true
        },
        {
            "attributeName": "province",
            "source": [
                {
                    "attribute": "province"
                }
            ],
            "encrypted": true
        },
        {
            "attributeName": "city",
            "source": [
                {
                    "attribute": "city"
                }
            ],
            "encrypted": true
        },
        {
            "attributeName": "postalCode",
            "source": [
                {
                    "attribute": "postalCode"
                }
            ],
            "encrypted": true
        }
    ],
    "dataSharePolicies": {
```

```

    "typeUIShare": "Data Share",
    "validForInMinutes": "30",
    "transactionsAllowed": "100000",
    "encryptionType": "none",
    "shareDomain": "datashare.datashare",
    "source": "Packet Manager"
}
}

```

Configuration used in Verification

```

registration.processor.queue.verification.request
registration.processor.queue.verification.request.messageTTL
registration.processor.verification.policy.id
registration.processor.verification.subscriber.id
activemq.message.format
mosip.regproc.data.share.protocol
mosip.regproc.data.share.internal.domain.name

```

Error Codes

Error Code	Error Message
RPR-MVS-004	No Assigned Record Found
RPR-MVS-025	Multiple rids found for a reference id
RPR-MVS-022	TablenotAccessibleException in Manual verification
RPR-VER-002	Verification failed
RPR-VER-004	Resend for verification
RPR-MVS-016	Reg Id should not be null or empty
RPR-MVS-021	Manual verification rejected

Deploy

Effortlessly deploy and configure with comprehensive guides , repositories and more.

Deploy

Effortlessly deploy and configure with comprehensive guides , repositories and more.

Build and deploy

Refer to [repo](#).

Configurations

Refer to [Configuration Guide](#).

ID Repository

Overview

ID Repository contains the records of the identity of an individual and provides API based mechanism to store, retrieve, and update identity details by other MOSIP modules. ID Repository is used by [Registration Processor](#), [ID Authentication](#), and [Resident Services](#).

1.2 LTS- ID Repo - Demo-15-02-2022

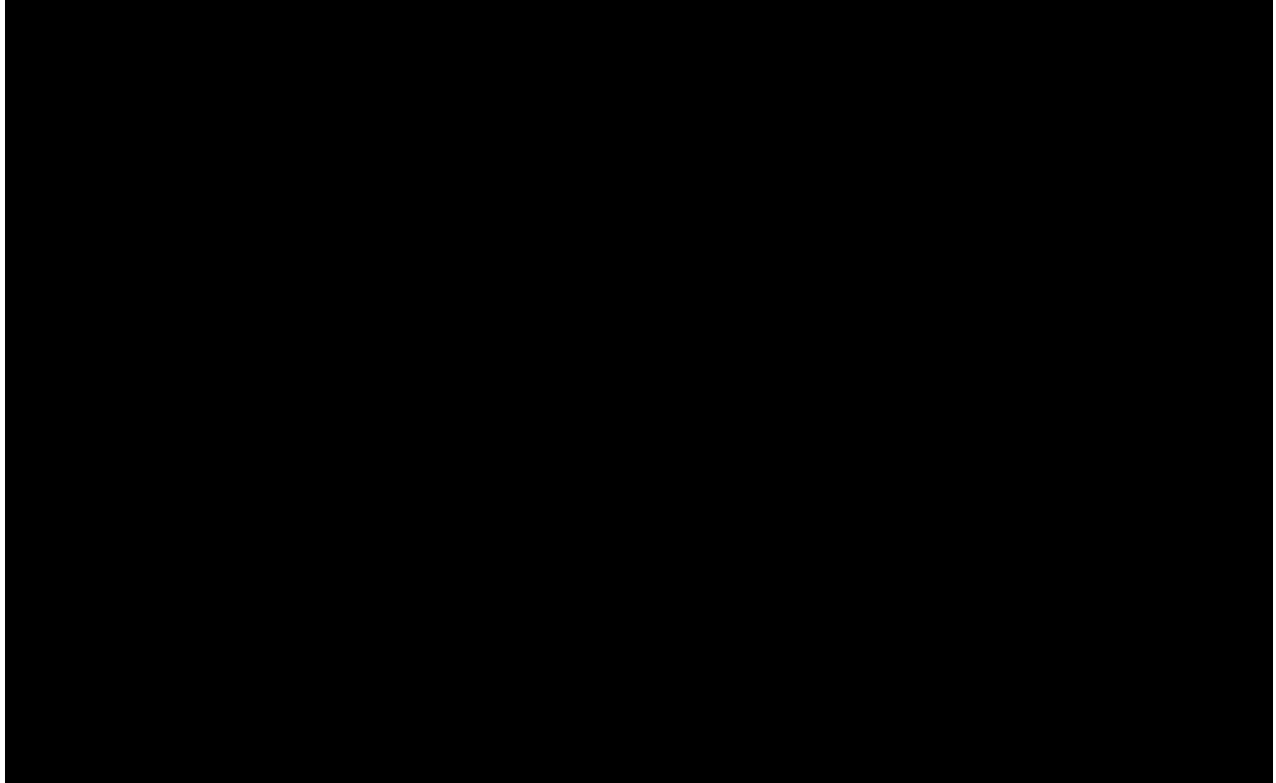


Services

ID Repository module consists of the following components:

1. Identity service
2. VID service
3. Credential service
4. Credential Request Generator service
5. Credential Feeder

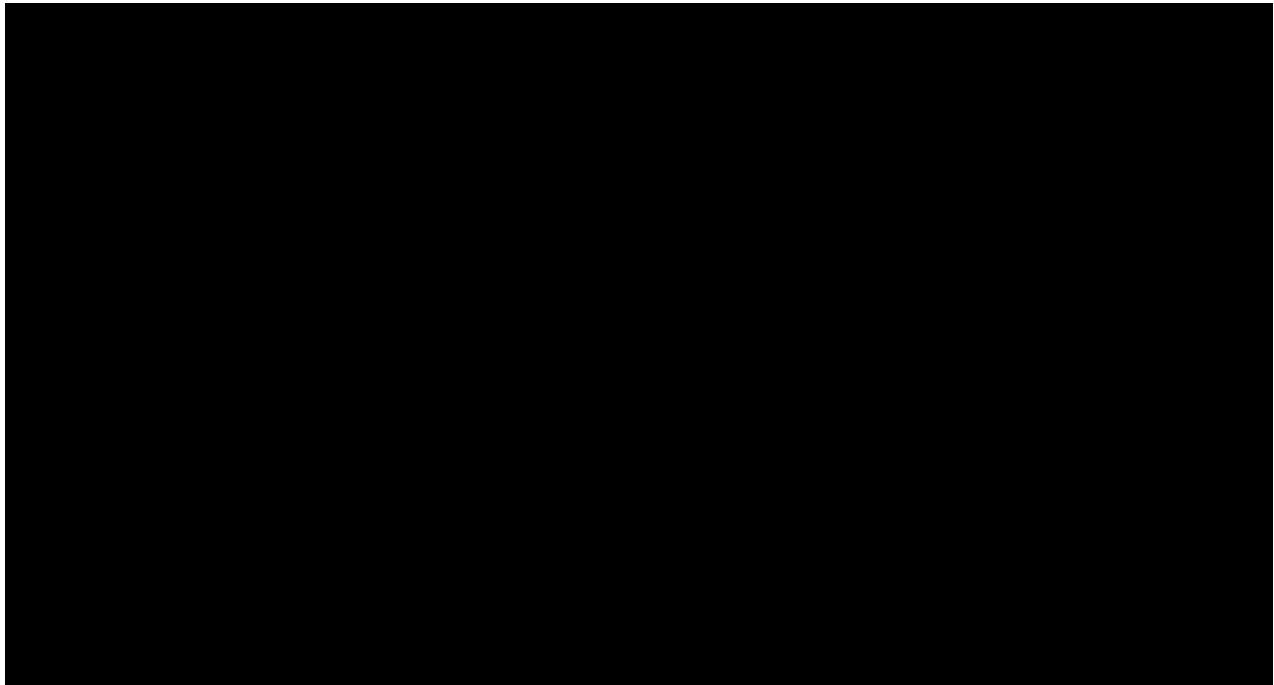
6. Salt generator



Identity service

- Stores, updates, and retrieves identity information.
- Also, retrieves and updates [UIN](#) status.

Identity service uses Biometric SDK (server) to extract templates from provided biometric data.

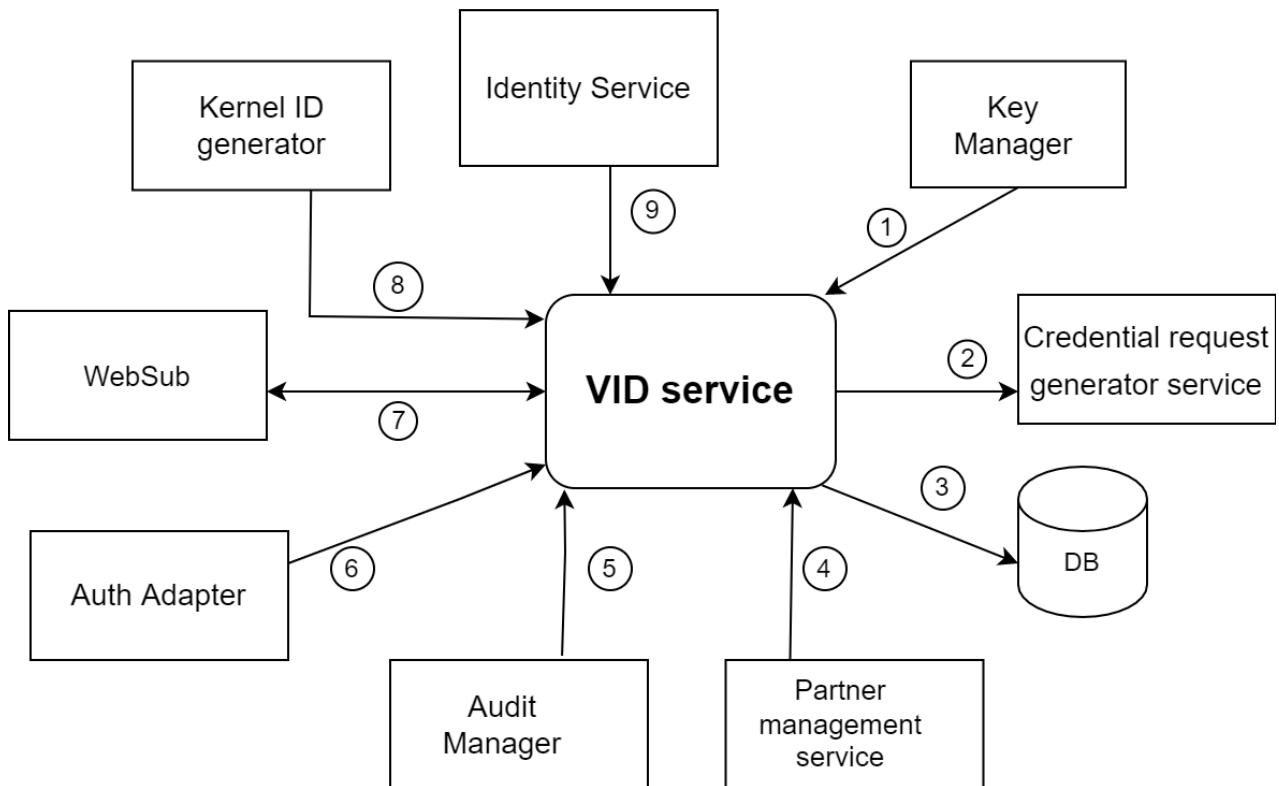


Above is the entity relationship diagram illustrated for the Identity service. *NOTE:* The numbers do not signify a sequence of operations or control flow. Arrows indicate the data flow.

1. [Key Manager](#) encrypts/decrypts data.
2. The credential request generator issues credentials for new/updated UIN data.
3. [Object Store](#) stores/retrieves biometrics and demographic documents.
4. All demographic data of UIN and references to biometric and demographic files stored in the object store are stored in `mosip_idrepo` DB.
5. [Partner management service](#) retrieves online verification partners to issue credentials.
6. Audit logs are logged into Audit Manager.
7. Biometric SDK extracts the templates for input biometric data.
8. Auth Adapter integrates with KeyCloak for authentication.
9. Masterdata service retrieves Identity schema based on input schema version.
10. [WebSub](#) publishes events related to UIN updation and auth type status updates.
11. Kernel ID generator generates UIN.
12. VID service fetches the list of VIDs associated with UIN to issue credential of update UIN and to create and activate draft VID.

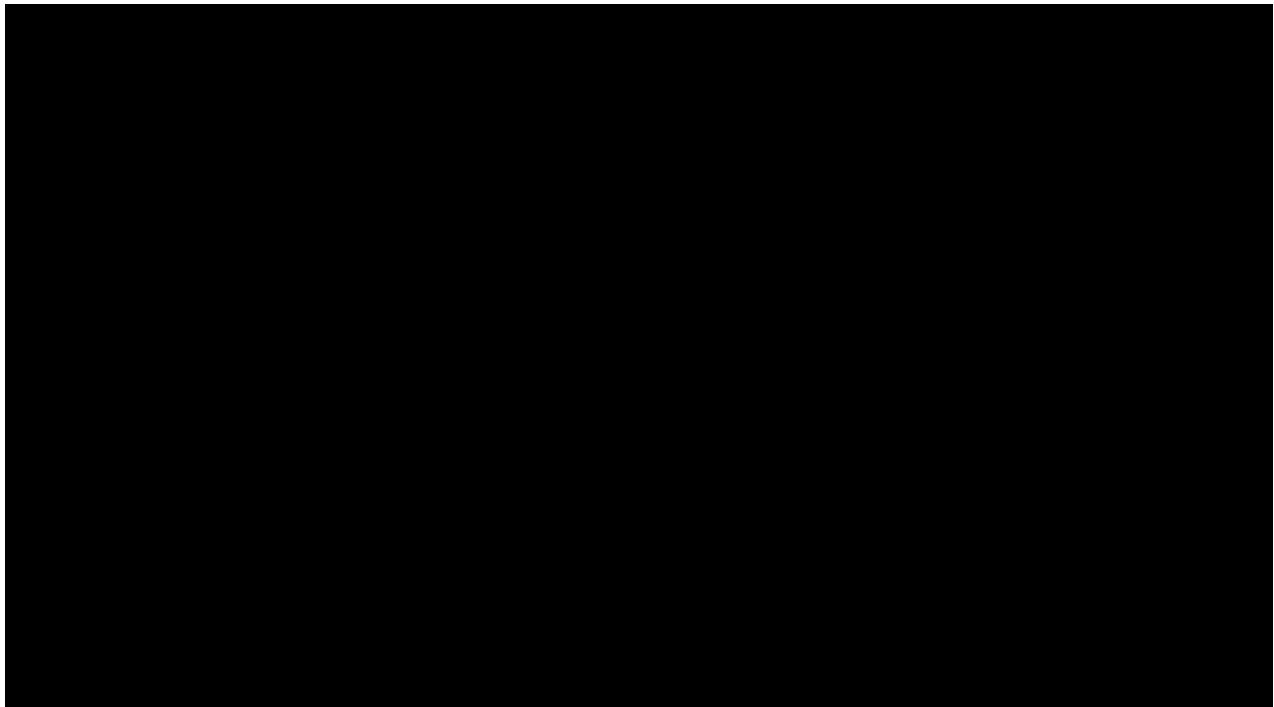
VID service

VID Service provides functionality to create/update Virtual IDs mapped against a UIN. It also provides the facility to update the status of VID. VIDs are created based on the VID policy defined in the configuration.



1. Key Manager encrypts/decrypts data.
2. The credential request generator issues credentials for new/updated UIN data.
3. All VID related data is stored in `mosip_idmap` DB.
4. Partner management service retrieves online verification partners to issue credentials.
5. Audit logs are logged into Audit Manager.
6. The Auth Adapter integrates with KeyCloak for authentication.
7. WebSub publishes events related to VID updation.
8. The kernel ID generator generates VID.
9. The identity service checks the status of UIN to create a VID.

Credential service



1. Key Manager encrypts/decrypts data and also used to sign data.
2. WebSub subscribes to get notifications related to credential status from IDA.
3. [DataShare](#) creates datashare url for sharable attributes.
4. Identity service retrieves identity data for UIN/VID.
5. Partner management service retrieves policies related to credential type and also retrieves policy for bio-extraction.
6. Auth Adapter integrates with KeyCloak for authentication.

Credential types

A credential can be defined as any document, object, or data structure that vouches for the identity of a person through some method of trust and authentication. Simply put, a credential is the thing that a person presents—in person or remotely—to say "this is who I am." The types of credentials issued in an ID system vary along multiple dimensions, depending on whether they are physical (i.e., they must be physically carried by a person in order to use them), or digital (i.e., they are machine readable and therefore can be used in a digital environment).

A credential type essentially maps to partner and data share policy.

Default credential types provided as part of [sandbox deployment](#) are given below:

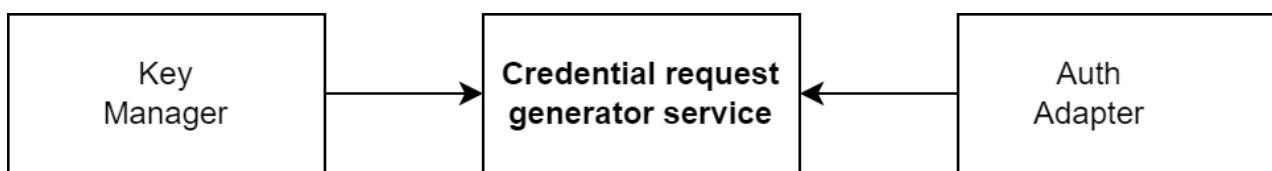
1. `auth` : Represents individual's data shared with Online Verification Partners (further used for Authentication and eKYC).
2. `qrcode` : qrcode type is used for qrcode partners to issue qrcode related credential data.
3. `euin` : It is used to issue credential data to partners who wish to download euin card using euin policy.
4. `reprint` : Reprint auth type is used for issuing credential information to reprint partners.
5. `vercred` : To issue verifiable credentials to partners, vercred credential type is used.

These types are defined in [partner_policy_credential_type table](#) of [mosip_pms database](#).

New credential types may be defined as per needs of a country.

Credential request generator service

This service creates request for credential issuance.



1. Key Manager encrypts/decrypts data.
2. The Auth Adapter integrates with KeyCloak for authentication.

Credential feeder

This job will feed the existing UIN/ VID identity information to the newly deployed IDA instance.

Salt generator

This is a one-time job that populates salts that are used to hash and encrypt data for Identity and VID services. This job must be executed before deploying these services. The following tables are populated:

- `uin_hash_salt` in `mosip_idrepo` DB.
- `uin_encrypt_salt` in `mosip_idmap` DB.

In the MOSIP sandbox, the job is run [here](#).

Developer Guide

To know more about the developer setups, read:

1. [Credential Request Generator Service Developers Guide](#)
2. [Identity Service Developers Guide](#)
3. [VID Service Developers Guide](#)
4. [Custom Handle Implementation Guide](#)

API

Refer to [API Documentation](#).

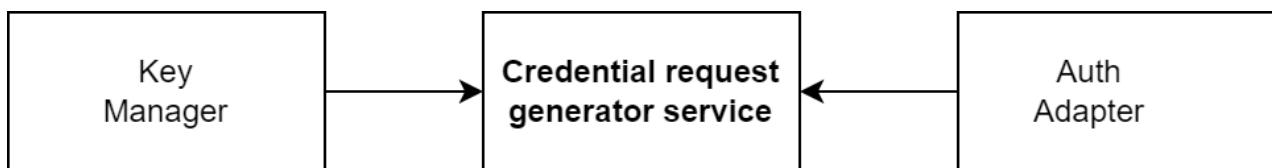
Source code

[Github repo](#)

Credential Request Generator Service Developers Guide

Overview

This service creates request for credential issuance.



1. Keymanager encrypts/decrypts data.
2. Auth Adapter integrates with Keycloak for authentication.

The documentation here will guide you through the prerequisites required for the developer' setup.

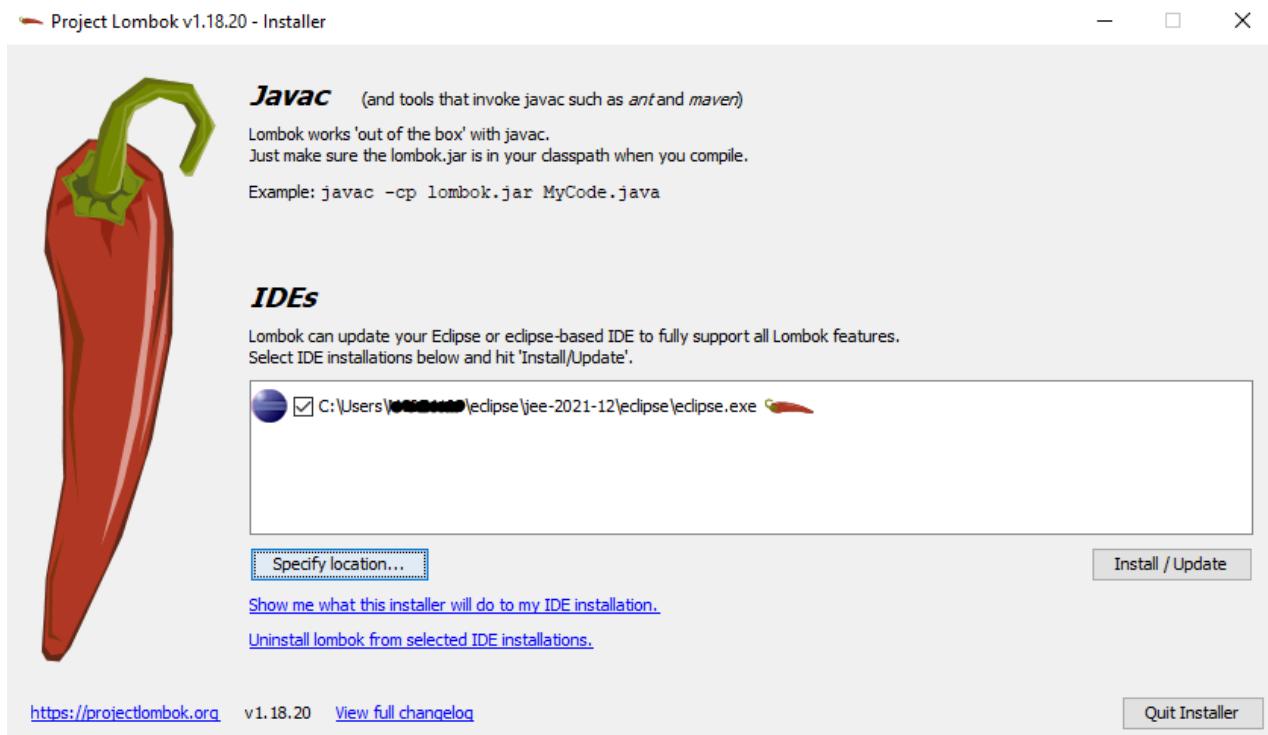
Software setup

Below are a list of tools required in ID Repository (Credential Request Generator Service) setup:

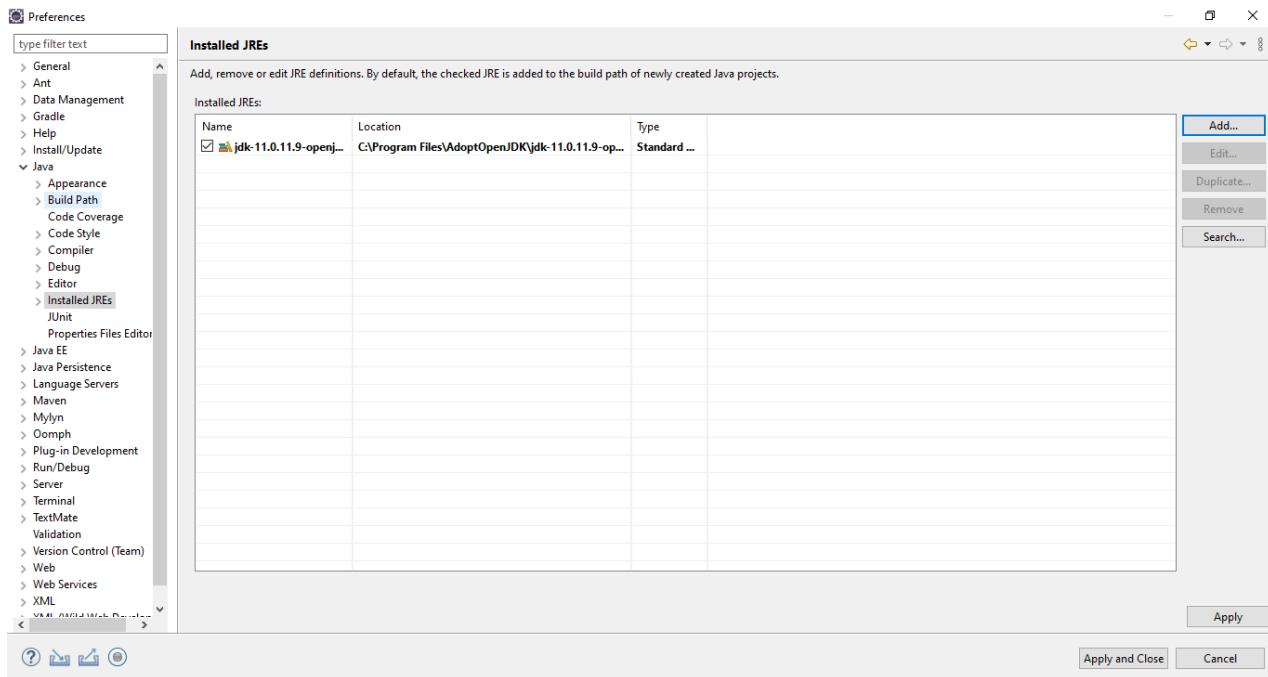
1. JDK 11
2. Any IDE (like Eclipse, IntelliJ IDEA)
3. Apache Maven (zip folder)
4. pgAdmin
5. Postman
6. Git
7. Notepad++ (optional)
8. lombok.jar (file)
9. settings.xml (document)

Follow the steps below to set up ID Repository Services on your local system:

1. Download `lombok.jar` and `settings.xml` from [here](#).
2. Unzip Apache Maven and move the unzipped folder in `C:\Program Files` and `settings.xml` to "conf" folder `C:\Program Files\apache-maven-3.8.4\conf`.
3. Install Eclipse, open the `lombok.jar` file and wait for some time until it completes the scan for Eclipse IDE and then click `Install/Update`.



4. Check the Eclipse installation folder `C:\Users\userName\eclipse\jee-2021-12\eclipse` to see if the `lombok.jar` is added. By doing this, you don't have to add the dependency of `lombok` in your `pom.xml` file separately as it is auto-configured by Eclipse.
5. Configure the JDK (Standard VM) with your Eclipse by traversing through `Preferences → Java → Installed JREs`.

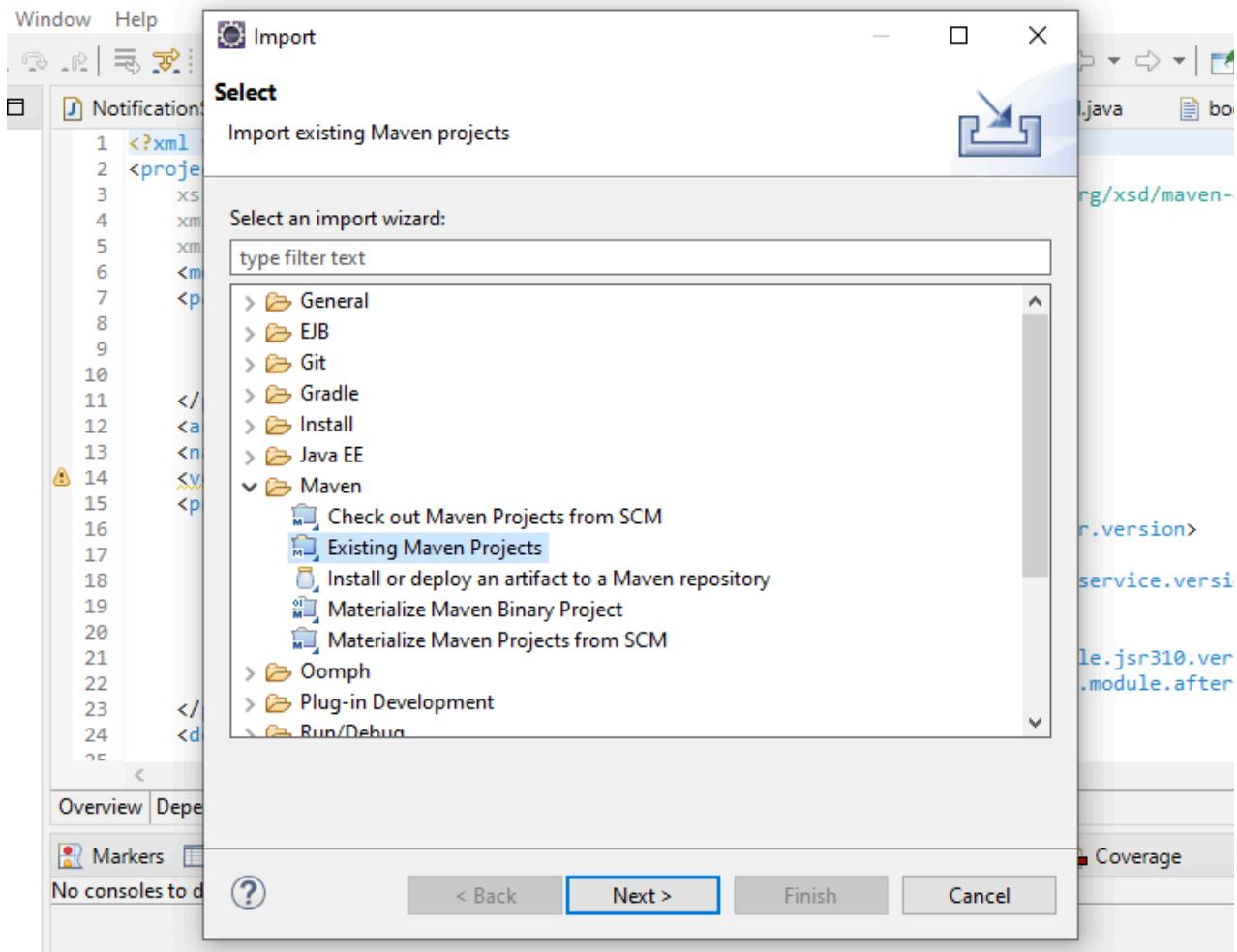


Code setup

For the code setup, clone the repository and follow the guidelines mentioned in the [Code Contributions](#).

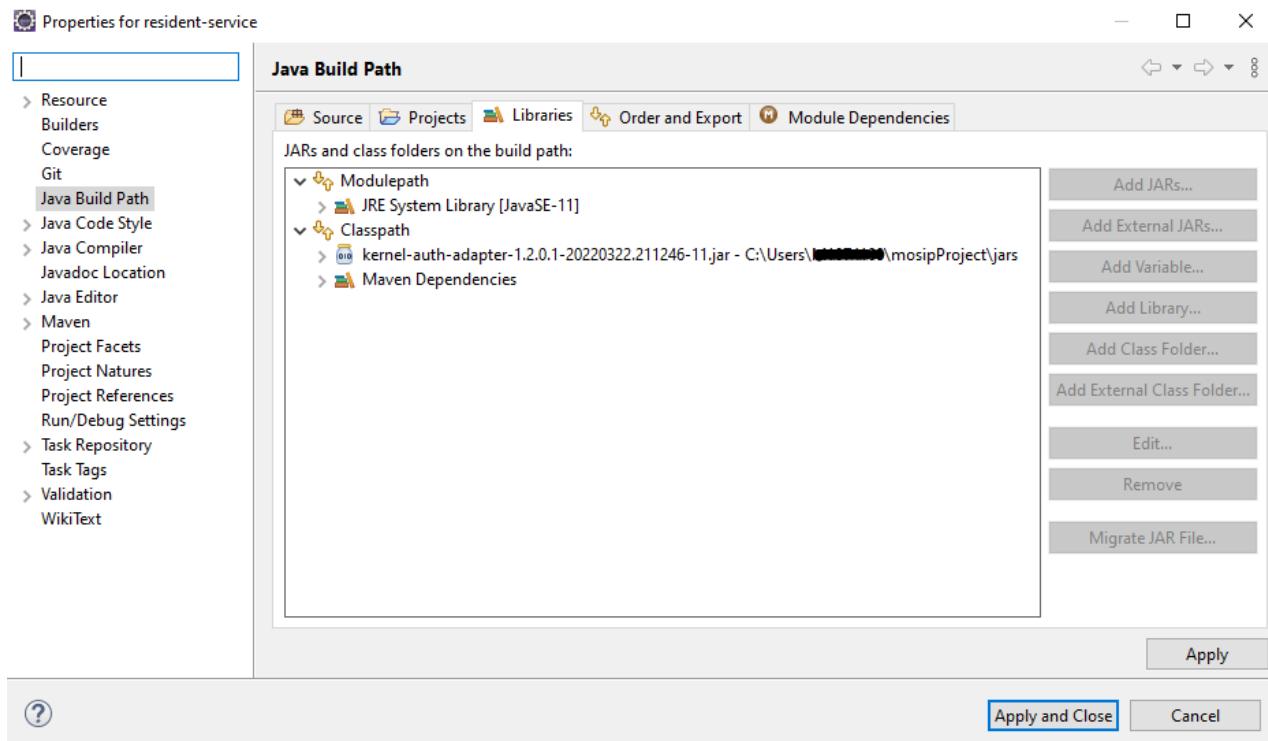
Importing and building of a project

1. Open the project folder where `pom.xml` is present.
2. Open command prompt from the same folder.
3. Run the command `mvn clean install -Dpgp.skip=true` to build the project and wait for the build to complete successfully.
4. After building of a project, open Eclipse and select `Import Projects → Maven → Existing Maven Projects → Next → Browse to project directory → Finish`.
5. After successful importing of project, update the project by right-click on `Project → Maven → Update Project`.



Environment setup

1. For the environment setup, you need an external JAR that is available [here](#) with different versions. (E.g.: You can download `kernel-auth-adapter.jar` and add to project `Libraries → Classpath → Add External JARs → Select Downloaded JAR → Add → Apply and Close`).



2. Clone [mosip-config repository](#).

3. Create an empty folder inside the `mosip-config` with `sandbox-local` name and then copy and paste all config files inside `sandbox-local` folder except `.gitignore`, `README` and `LICENSE`.

4. As ID Repository is using two properties files, `id-repository-default` and `application-default`, you will have to configure them according to your environment. The same files are available [here](#) for reference.

Properties to be updated:

`application-default.properties`

- `mosip.mosip.resident.client.secret = <current_password>`.
- `db.dbuser.password=<password>`.
- `mosip.kernel.xsdstorage-uri=file:///home/user/Desktop/tspl/mosip-config/sandbox-local/` (i.e. `sandbox-local` folder location).
- Comment this out `auth.server.admin.issuer.internal.uri` in `application-default.properties` file because you already have this

`auth.server.admin.issuer.uri`, and hence there is no need of `auth.server.admin.issuer.internal.uri`.

- `mosip.identity.mapping-file=<Path_to_identity_mapping_json_file>`. (For Example: `file:///home/user/Desktop/tspl/mosip-config/sandbox-local/identity-mapping.json`)

`id-repository-default.properties`

- `mosip.credential.service.database.hostname`
- `mosip.credential.service.database.port`

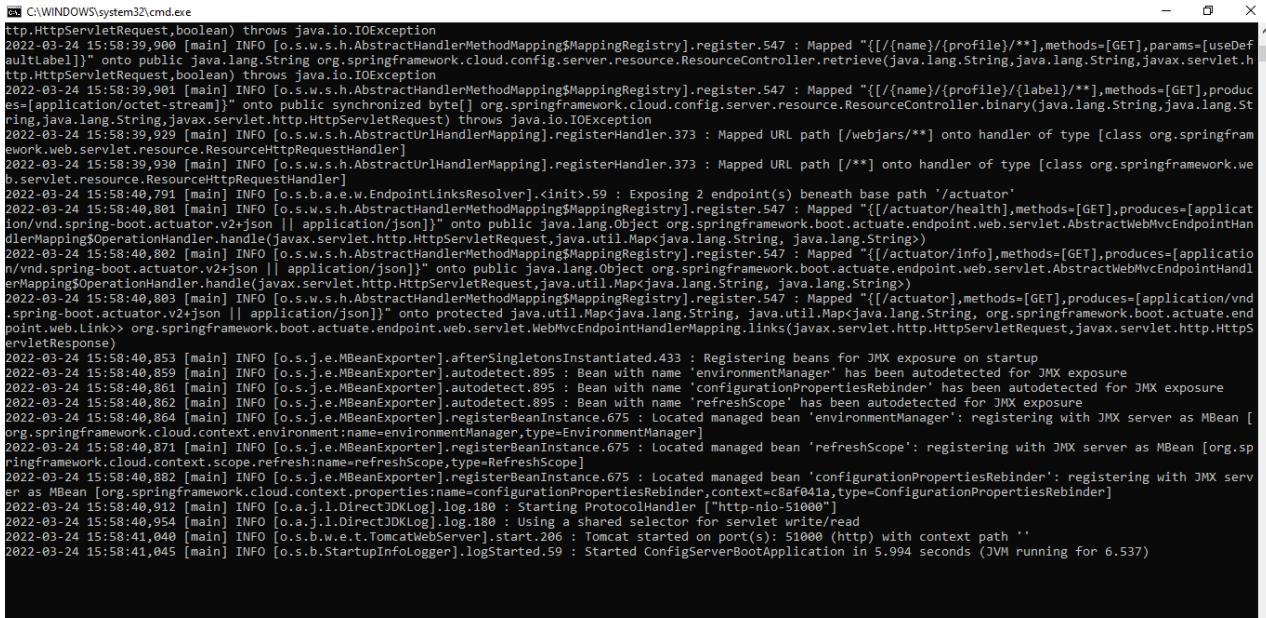
5. To run the server, two files are required- [kernel-config-server.jar](#) and [config-server-start.bat](#).

6. Put both the files in the same folder and change the location attribute to `sandbox-local` folder in `config-server-start.bat` file and also check the version of `kernel-config-server.jar` towards the end of the command.

Example:

```
java -jar -Dspring.profiles.active=native -
Dspring.cloud.config.server.native.search-
locations=/home/vipul/Desktop/tspl/mosip-config/sandbox-local -
Dspring.cloud.config.server.accept-empty=true -
Dspring.cloud.config.server.git.force-pull=false -
Dspring.cloud.config.server.git.cloneOnStart=false -
Dspring.cloud.config.server.git.refreshRate=0 kernel-config-server-1.2.0-
20201016.134941-57.jar.
```

7. Run the server by opening the `config-server-start.bat` file.



```

C:\WINDOWS\system32\cmd.exe
http.HttpServletRequest,boolean) throws java.io.IOException
2022-03-24 15:58:39.900 [main] INFO [o.s.w.s.h.AbstractHandlerMethodMapping$MappingRegistry].register.547 : Mapped "[{[/name}/{profile}**},methods=[GET],params=[useDefaultLabel]]" onto public java.lang.String org.springframework.cloud.config.server.resource.ResourceController.retrieve(java.lang.String,java.lang.String,javax.servlet.http.HttpServletResponse,java.io.IOException)
2022-03-24 15:58:39.901 [main] INFO [o.s.w.s.h.AbstractHandlerMethodMapping$MappingRegistry].register.547 : Mapped "[{[/name}/{profile}/{label}**},methods=[GET],produces=[application/octet-stream]]" onto public synchronized byte[] org.springframework.cloud.config.server.resource.ResourceController.binary(java.lang.String,java.lang.String,java.lang.String,java.lang.String,java.servlet.http.HttpServletResponse) throws java.io.IOException
2022-03-24 15:58:39.929 [main] INFO [o.s.w.s.h.AbstractUrlHandlerMapping].registerHandler.373 : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2022-03-24 15:58:39.930 [main] INFO [o.s.w.s.h.AbstractUrlHandlerMapping].registerHandler.373 : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2022-03-24 15:58:40.791 [main] INFO [o.s.w.a.e.w.EndpointLinksResolver].<init>.59 : Exposing 2 endpoint(s) beneath base path '/actuator'
2022-03-24 15:58:40.801 [main] INFO [o.s.w.s.h.AbstractHandlerMethodMapping$MappingRegistry].register.547 : Mapped "[{[actuator/health],methods=[GET],produces=[application/vnd.spring-boot.actuator.v2+json || application/json]}]" onto public java.lang.Object org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMvcEndpointHandlerMappings$OperationHandler.handle(javax.servlet.http.HttpServletRequest,java.util.Map<java.lang.String, java.lang.String>)
2022-03-24 15:58:40.802 [main] INFO [o.s.w.s.h.AbstractHandlerMethodMapping$MappingRegistry].register.547 : Mapped "[{[actuator/info],methods=[GET],produces=[application/vnd.spring-boot.actuator.v2+json || application/json]}]" onto public java.lang.Object org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMvcEndpointHandlerMappings$OperationHandler.handle(javax.servlet.http.HttpServletRequest,java.util.Map<java.lang.String, java.lang.String>)
2022-03-24 15:58:40.803 [main] INFO [o.s.w.s.h.AbstractHandlerMethodMapping$MappingRegistry].register.547 : Mapped "[{[actuator],methods=[GET],produces=[application/vnd.spring-boot.actuator.v2+json || application/json]}]" onto protected java.util.Map<java.lang.String, java.util.Map<java.lang.String, org.springframework.boot.actuate.endpoint.web.Link>> org.springframework.boot.actuate.endpoint.web.servlet.WebMvcEndpointHandlerMapping.links(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
2022-03-24 15:58:40.853 [main] INFO [o.s.j.e.MBeanExporter].afterSingletonsInstantiated.433 : Registering beans for JMX exposure on startup
2022-03-24 15:58:40.859 [main] INFO [o.s.j.e.MBeanExporter].autodetect.895 : Bean with name 'environmentManager' has been autodetected for JMX exposure
2022-03-24 15:58:40.861 [main] INFO [o.s.j.e.MBeanExporter].autodetect.895 : Bean with name 'configurationPropertiesRebinder' has been autodetected for JMX exposure
2022-03-24 15:58:40.862 [main] INFO [o.s.j.e.MBeanExporter].autodetect.895 : Bean with name 'refreshScope' has been autodetected for JMX exposure
2022-03-24 15:58:40.864 [main] INFO [o.s.j.e.MBeanExporter].registerBeanInstance.675 : Located managed bean 'environmentManager': registering with JMX server as MBean [org.springframework.cloud.context.environment:name=environmentManager,type=EnvironmentManager]
2022-03-24 15:58:40.871 [main] INFO [o.s.j.e.MBeanExporter].registerBeanInstance.675 : Located managed bean 'refreshScope': registering with JMX server as MBean [org.springframework.cloud.context.scope.refresh:name=refreshScope,type=RefreshScope]
2022-03-24 15:58:40.882 [main] INFO [o.s.j.e.MBeanExporter].registerBeanInstance.675 : Located managed bean 'configurationPropertiesRebinder': registering with JMX server as MBean [org.springframework.cloud.context.properties:name=configurationPropertiesRebinder,context=8af0d41a,type=ConfigurationPropertiesRebinder]
2022-03-24 15:58:40.912 [main] INFO [o.a.j.l.DirectJDKLog].log.180 : Starting ProtocolHandler ["http-nio-51000"]
2022-03-24 15:58:40.954 [main] INFO [o.a.j.l.DirectJDKLog].log.180 : Using a shared selector for servlet write/read
2022-03-24 15:58:41.040 [main] INFO [o.s.b.w.e.TomcatWebServer].start.206 : Tomcat started on port(s): 51000 (http) with context path ''
2022-03-24 15:58:41.045 [main] INFO [o.s.b.StartupInfoLogger].logStarted.59 : Started ConfigServerBootApplication in 5.994 seconds (JVM running for 6.537)

```

The server should now be up and running.

Below are the configurations to be done in Eclipse:

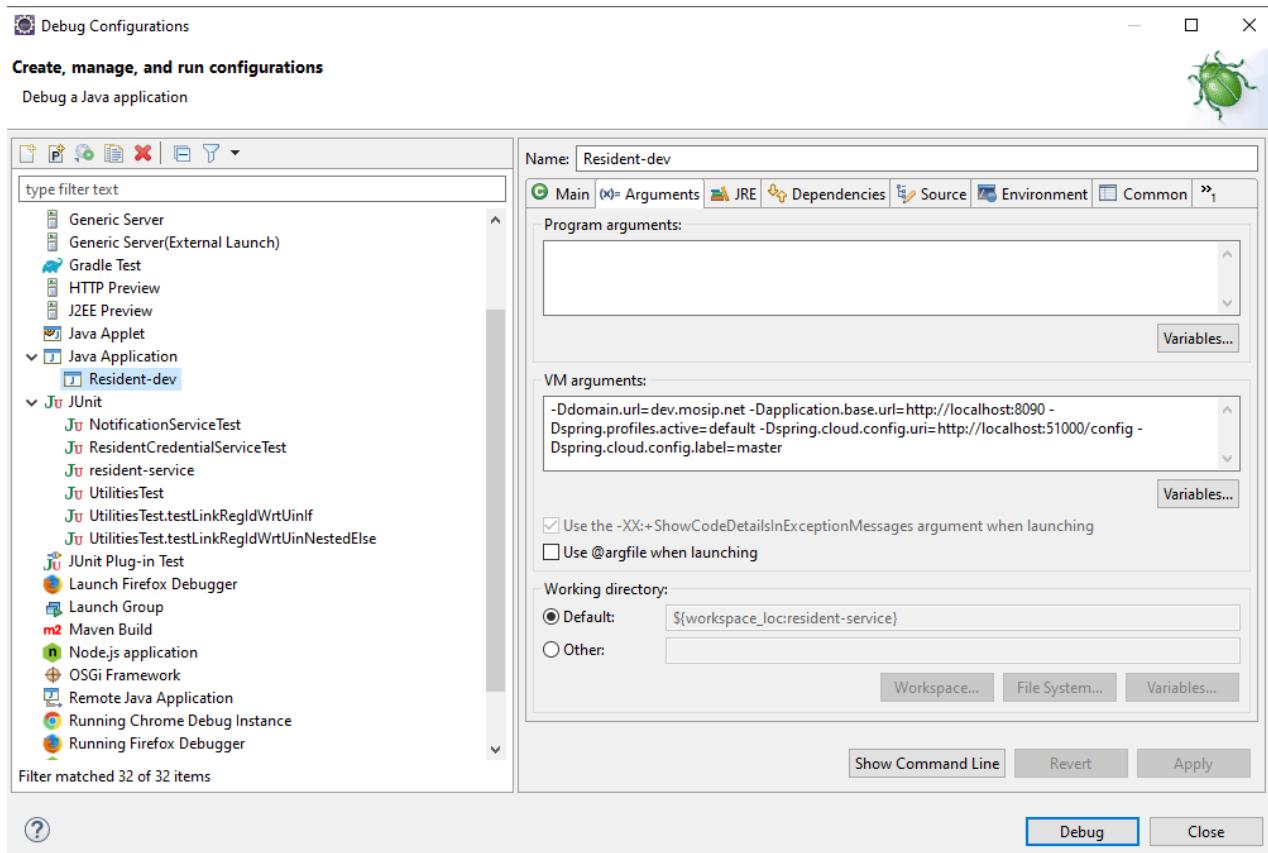
1. Open Eclipse and run the project for one time as `Java application`, so that it will create a Java application which you can see in debug configurations and then change its name. (e.g.: project name with environment - "Credential-request-generator-dev").

2. Open the arguments and pass this -Ddomain.url=dev.mosip.net -

```
Dapplication.base.url=http://localhost:8090 -
Dspring.profiles.active=default -
Dspring.cloud.config.uri=http://localhost:51000/config -
Dspring.cloud.config.label=master
```

in VM arguments.

3. Here, the domain URL represents the environment on which you are working (eg., it can be dev2.mosip.net or qa3.mosip.net).



4. Click Apply and then debug it (starts running).

Credential Request Generator Service API

- For API documentation, refer [here](#).
- The APIs can be tested with the help of **Swagger-UI**.
- Swagger is an interface description language for describing restful APIs expressed using JSON. You can access Swagger-UI of credential-request-generator-services for localhost from

```
https://localhost:8094/v1/credentialrequest/swagger-ui/index.html?  
configUrl=/v1/credentialrequest/v3/api-docs/swagger-config#/.
```

Identity Service Developers Guide

Overview

- Identity Service stores, updates, retrieves identity information.
- Also, retrieves and updates UIN status.

The documentation here will guide you through the prerequisites required for the developer' setup.

Software setup

Below are a list of tools required in ID Repository Services (Identity Service) setup:

1. JDK 11
2. Any IDE (like Eclipse, IntelliJ IDEA)
3. Apache Maven (zip folder)
4. pgAdmin
5. Postman
6. Git
7. Notepad++ (optional)
8. lombok.jar (file)
9. settings.xml (document)

Follow the steps below to set up ID Repository- Identity Services on your local system:

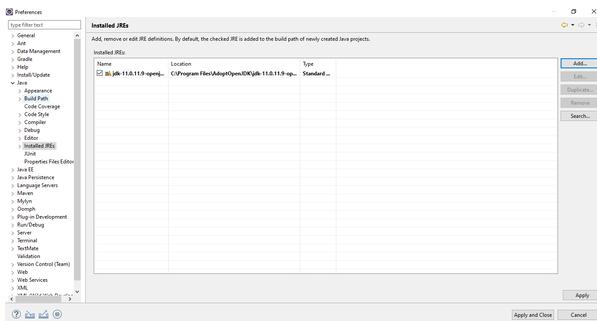
1. Download `lombok.jar` and `settings.xml` from [here](#).
2. Unzip Apache Maven and move the unzipped folder in `C:\Program Files` and `settings.xml` to "conf" folder `C:\Program Files\apache-maven-3.8.4\conf`.

3. Install Eclipse, open the `lombok.jar` file and wait for some time until it completes the scan for Eclipse IDE and then click `Install/Update`.



4. Check the Eclipse installation folder `C:\Users\userName\eclipse\jee-2021-12\eclipse` to see if the `lombok.jar` is added. By doing this, you don't have to add the dependency of `lombok` in your `pom.xml` file separately as it is auto-configured by Eclipse.

5. Configure the JDK (Standard VM) with your Eclipse by traversing through `Preferences → Java → Installed JREs`.



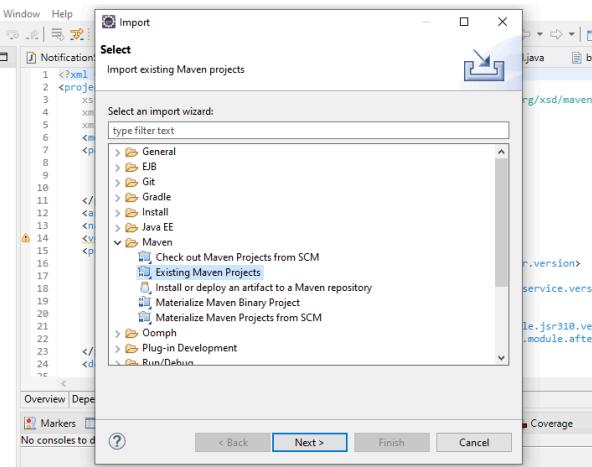
Code setup

For the code setup, clone the repository and follow the guidelines mentioned in the [Code Contributions](#).

Importing and building of a project

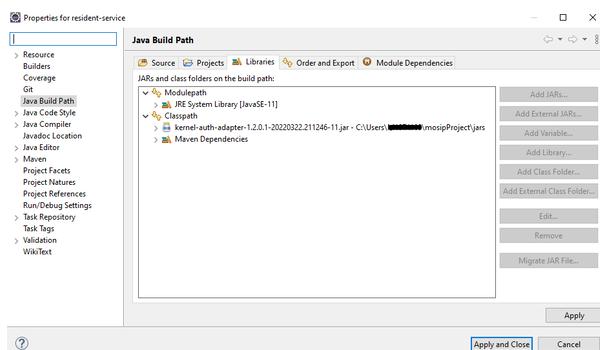
1. Open the project folder where `pom.xml` is present.
2. Open command prompt from the same folder.

3. Run the command `mvn clean install -Dgpg.skip=true` to build the project and wait for the build to complete successfully.
4. After building of a project, open Eclipse and select `Import Projects → Maven → Existing Maven Projects → Next → Browse to project directory → Finish`.
5. After successful importing of project, update the project by right-click on `Project → Maven → Update Project`.



Environment setup

1. For the environment setup, you need an external JAR that is available [here](#) with different versions. (E.g.: You can download `kernel-auth-adapter.jar` and add to project `Libraries → Classpath → Add External JARs → Select Downloaded JAR → Add → Apply and Close`).



2. Clone [mosip-config repository](#).
3. Create an empty folder inside the `mosip-config` with `sandbox-local` name and then copy and paste all config files inside `sandbox-local` folder except

.gitignore, README and LICENSE .

4. As Id Repository is using two properties files, `id-repository-default` and `application-default`, you will have to configure them according to your environment. The same files are available [here](#) for reference.

Properties to be updated:

`application-default.properties`

- `mosip.mosip.resident.client.secret = <current_password>` .
- `db.dbuser.password=<password>` .
- `mosip.kernel.xsdstorage-uri=file:///home/user/Desktop/tspl/mosip-config/sandbox-local/` (i.e. `sandbox-local` folder location).
- Comment this out `auth.server.admin.issuer.internal.uri` in `application-default.properties` file because you already have this `auth.server.admin.issuer.uri` , and hence there is no need of `auth.server.admin.issuer.internal.uri` .
- `mosip.identity.mapping-file=<Path_to_identity_mapping_json_file>` . (For Example: `file:///home/user/Desktop/tspl/mosip-config/sandbox-local/identity-mapping.json`)

`id-repository-default.properties`

- `mosip.idrepo.db.url`
- `mosip.idrepo.db.port`
- Comment out all the lines containing `mosip.biometric.sdk.providers.finger` , `mosip.biometric.sdk.providers.face` and `mosip.biometric.sdk.providers.iris` .
- Comment out this property `mosip.kernel.idobjectvalidator.referenceValidator` .
- `mosip.idrepo.mosip-config-url=file:///home/user/Desktop/tspl/mosip-config/sandbox-local/` (i.e. `sandbox-local` folder location).

5. To run the server, two files are required- [kernel-config-server.jar](#) and [config-server-start.bat](#).

6. Put both the files in the same folder and change the location attribute to `sandbox-local` folder in `config-server-start.bat` file and also check the version of `kernel-config-server.jar` towards the end of the command.

Example:

```
java -jar -Dspring.profiles.active=native -  
Dspring.cloud.config.server.native.search-  
locations=file:C:\Users\myDell\mosipProject\mosip-config\sandbox-local -  
Dspring.cloud.config.server.accept-empty=true -  
Dspring.cloud.config.server.git.force-pull=false -  
Dspring.cloud.config.server.git.cloneOnStart=false -  
Dspring.cloud.config.server.git.refreshRate=0 kernel-config-server-1.2.0-  
20201016.134941-57.jar .
```

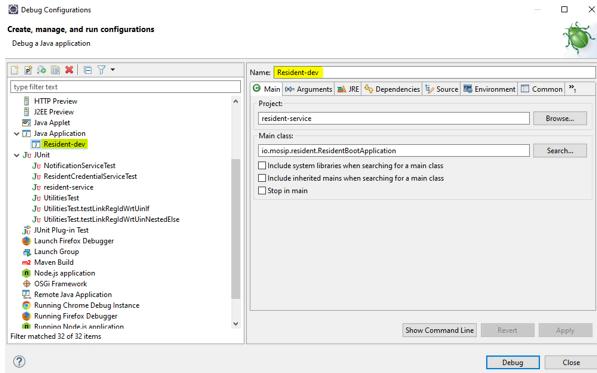
7. Run the server by opening the `config-server-start.bat` file.

```
[2022-04-24 15:58:49,450] [main] INFO [o.s.b.SpringApplication:run] - Starting ApplicationContext with context path '' using shared selector for servlet write/read
[2022-04-24 15:58:49,450] [main] INFO [o.s.b.SpringApplication:run] - Started ConfigServerBootApplication in 5.994 seconds (20M running for 6.537)
[2022-04-24 15:58:49,450] [main] INFO [o.s.b.StartupInfoLogger:logStarted] - Started ConfigServerBootApplication with context path ''
```

The server should now be up and running.

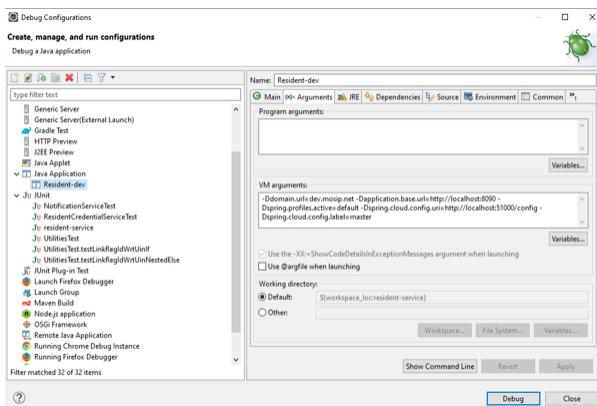
Below are the configurations to be done in Eclipse:

1. Open Eclipse and run the project for one time as `Java application`, so that it will create a Java application which you can see in debug configurations and then change its name. (e.g.: project name with environment - "Identity-Service-dev").



2. Open the arguments and pass this `-Ddomain.url=dev.mosip.net -Dapplication.base.url=http://localhost:8090 -Dspring.profiles.active=default -Dspring.cloud.config.uri=http://localhost:51000/config -Dspring.cloud.config.label=master` in VM arguments.

3. Here, the domain URL represents the environment on which you are working (eg., it can be `dev2.mosip.net` or `qa3.mosip.net`).



4. Click Apply and then debug it (starts running).

Identity service API

- For API documentation, refer [here](#).
- The APIs can be tested with the help of **Swagger-UI**.
- Swagger is an interface description language for describing restful APIs expressed using JSON. You can access Swagger-UI of identity-services for localhost from `http://localhost:8090/idrepository/v1/identity/swagger-`

```
ui/index.html?configUrl=/idrepository/v1/identity/v3/api-docs/swagger-
config#/ .
```

VID Service Developers Guide

Overview

VID Service provides functionality to create/update Virtual IDs mapped against an UIN. It also provides the facility to update the status of VID. VIDs are created based on the VID policy defined in the configuration.

The documentation here will guide you through the prerequisites required for the developer' setup.

Software setup

Below are a list of tools required in ID Repository VID Services setup:

1. JDK 11
2. Any IDE (like Eclipse, IntelliJ IDEA)
3. Apache Maven (zip folder)
4. pgAdmin
5. Postman
6. Git
7. Notepad++ (optional)
8. lombok.jar (file)
9. settings.xml (document)

Follow the steps below to set up ID Repository (VID Services) on your local system:

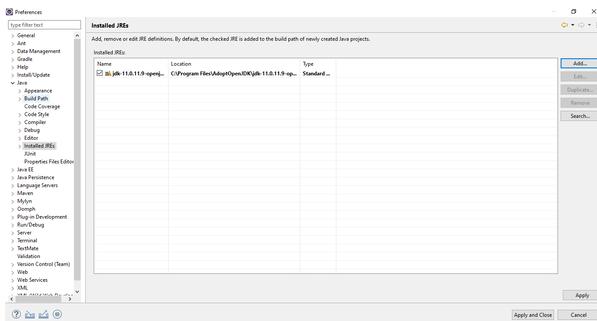
1. Download `lombok.jar` and `settings.xml` from [here](#).
2. Unzip Apache Maven and move the unzipped folder in `C:\Program Files` and `settings.xml` to "conf" folder `C:\Program Files\apache-maven-3.8.4\conf`.

3. Install Eclipse, open the `lombok.jar` file and wait for some time until it completes the scan for Eclipse IDE and then click `Install/Update`.



4. Check the Eclipse installation folder `C:\Users\userName\eclipse\jee-2021-12\eclipse` to see if the `lombok.jar` is added. By doing this, you don't have to add the dependency of `lombok` in your `pom.xml` file separately as it is auto-configured by Eclipse.

5. Configure the JDK (Standard VM) with your Eclipse by traversing through `Preferences → Java → Installed JREs`.



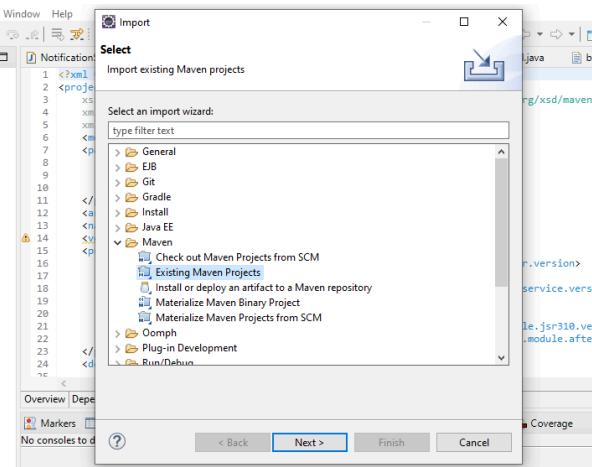
Code setup

For the code setup, clone the repository and follow the guidelines mentioned in the [Code Contributions](#).

Importing and building of a project

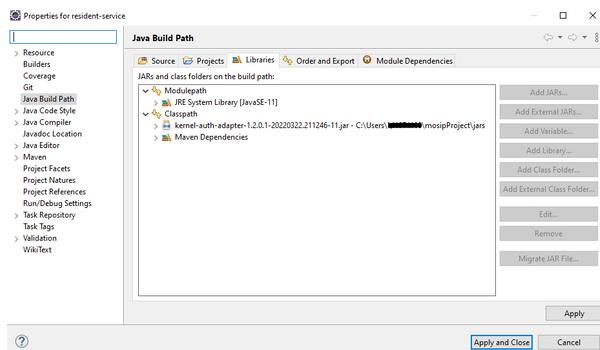
1. Open the project folder where `pom.xml` is present.
2. Open command prompt from the same folder.

3. Run the command `mvn clean install -Dgpg.skip=true` to build the project and wait for the build to complete successfully.
4. After building of a project, open Eclipse and select `Import Projects → Maven → Existing Maven Projects → Next → Browse to project directory → Finish`.
5. After successful importing of project, update the project by right-click on `Project → Maven → Update Project`.



Environment setup

1. For the environment setup, you need an external JAR that is available [here](#) with different versions. (E.g.: You can download `kernel-auth-adapter.jar` and add to project `Libraries → Classpath → Add External JARs → Select Downloaded JAR → Add → Apply and Close`).



2. Clone [mosip-config repository](#).

3. Create an empty folder inside the `mosip-config` with `sandbox-local` name and then copy and paste all config files inside `sandbox-local` folder except

.gitignore, README and LICENSE .

4. As Id Repository is using two properties files, `id-repository-default` and `application-default`, you will have to configure them according to your environment. The same files are available [here](#) for reference.

Properties to be updated:

`application-default.properties`

- `mosip.mosip.resident.client.secret = <current_password>` .
- `db.dbuser.password=<password>` .
- `mosip.kernel.xsdstorage-uri=file:///home/user/Desktop/tspl/mosip-config/sandbox-local/` (i.e. `sandbox-local` folder location).
- Comment this out `auth.server.admin.issuer.internal.uri` in `application-default.properties` file because you already have this `auth.server.admin.issuer.uri` , and hence there is no need of `auth.server.admin.issuer.internal.uri` .
- `mosip.identity.mapping-file=<Path_to_identity_mapping_json_file>` . (For Example: `file:///home/user/Desktop/tspl/mosip-config/sandbox-local/identity-mapping.json`)

`id-repository-default.properties`

- `mosip.idrepo.db.url`
- `mosip.idrepo.db.port`
- Comment out all the lines containing `mosip.biometric.sdk.providers.finger` , `mosip.biometric.sdk.providers.face` and `mosip.biometric.sdk.providers.iris` .
- `mosip.idrepo.mosip-config-url=file:///home/user/Desktop/tspl/mosip-config/sandbox-local/` (i.e. `sandbox-local` folder location).

5. To run the server, two files are required- [kernel-config-server.jar](#) and [config-server-start.bat](#).

6. Put both the files in the same folder and change the location attribute to `sandbox-local` folder in `config-server-start.bat` file and also check the version of `kernel-config-server.jar` towards the end of the command.

Example:

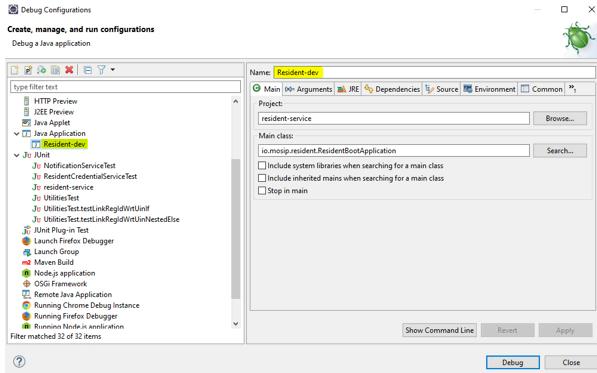
```
java -jar -Dspring.profiles.active=native -  
Dspring.cloud.config.server.native.search-  
locations=file:C:\Users\myDell\mosipProject\mosip-config\sandbox-local -  
Dspring.cloud.config.server.accept-empty=true -  
Dspring.cloud.config.server.git.force-pull=false -  
Dspring.cloud.config.server.git.cloneOnStart=false -  
Dspring.cloud.config.server.git.refreshRate=0 kernel-config-server-1.2.0-  
20201016.134941-57.jar .
```

7. Run the server by opening the `config-server-start.bat` file.

The server should now be up and running.

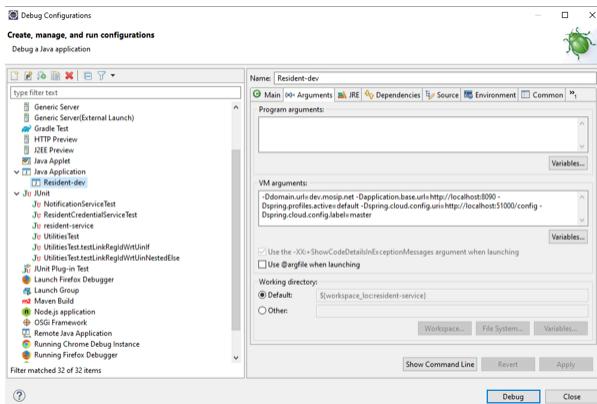
Below are the configurations to be done in Eclipse:

1. Open Eclipse and run the project for one time as `Java application`, so that it will create a Java application which you can see in debug configurations and then change its name. (e.g.: project name with environment - "vid-service-dev").



2. Open the arguments and pass this `-Ddomain.url=dev.mosip.net -Dapplication.base.url=http://localhost:8090 -Dspring.profiles.active=default -Dspring.cloud.config.uri=http://localhost:51000/config -Dspring.cloud.config.label=master` in VM arguments.

3. Here, the domain URL represents the environment on which you are working (eg., it can be `dev2.mosip.net` or `qa3.mosip.net`).



4. Click Apply and then debug it (starts running).

VID Service API

- For API documentation, refer [here](#).
- The APIs can be tested with the help of **Swagger-UI**.
- Swagger is an interface description language for describing restful APIs expressed using JSON. You can access Swagger-UI of resident-services for

localhost from `http://localhost:8091/idrepository/v1/swagger-ui/index.html?configUrl=/idrepository/v1/v3/api-docs/swagger-config#/`.

.well-known

What is .well-known folder?

The ".well-known" folder is a convention used in web development to provide a standardized location for certain files or resources that need to be publicly accessible and discoverable. It typically resides at the root level of a website or web server. The purpose of this folder is to make it easy for web clients (browsers, applications, or services) to find important files or resources related to web services and security.

Why should we use ".well-known" directory in MOSIP?

MOSIP can use the ".well-known" directory to serve the following purposes:

- **Standardization:** To provide a standardized location for specific public files and resources related to web services and security. It makes it easier for developers and web clients using MOSIP to know where to look for important information.
- **Security:** Security-related files and resources can be placed in the ".well-known" directory, such as the public certificate for encryption and signature verification can be placed here.
- **Interoperability:** By following the ".well-known" convention, web developers using MOSIP can ensure interoperability with various web standards and protocols. For example, MOISP shares the context file which contains the structure of its Verifiable Credentials.
- **Ease of Configuration:** Web servers can be configured to serve files from the ".well-known" directory without needing custom configurations for each specific resource. This simplifies the server setup and maintenance process.
- **Transparency:** For matters related to security policies and contact information, such as in the "security.txt" file, placing them in a well-known location makes it transparent and easily accessible to anyone interested in the website's security practices.

How is ".well-known" directory used in MOISP?

MOSIP's ".well-known" directory contains three files,

- controller.json
- mosip-context.json
- public-key.json

controller.json

The "controller.json" file is used in MOSIP to share the details of the public key using which a MOSIP-issued verifiable credential can be asserted.

```
{  
  "@context": "https://w3id.org/security/v2",  
  "id": "https://api.collab.mosip.net/.well-known/controller.json",  
  "assertionMethod": [  
    "https://api.collab.mosip.net/.well-known/public-key.json"  
  ]  
}
```

mosip-context.json

The "mosip-context.json" file contains the schema of the subject in the MOSIP-issued verifiable credential.

```
{  
  "@context": [{"  
    "@version": 1.1  
  }, "https://www.w3.org/ns/odrl.jsonld", {"  
    "mosip": "https://api.collab.mosip.net/mosip#",  
    "schema": "http://schema.org/",  
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",  
    "vcVer": "mosip:vcVer",  
    "UIN": "mosip:UIN",  
    "addressLine1": {"  
      "@id": "https://api.collab.mosip.net/mosip#addressLine1",  
      "@context": {"value": "rdf:value", "lang": "@language"}  
    },  
    "addressLine2": {"  
      "@id": "https://api.collab.mosip.net/mosip#addressLine2",  
      "@context": {"value": "rdf:value", "lang": "@language"}  
    },  
    "addressLine3": {"  
      "@id": "https://api.collab.mosip.net/mosip#addressLine3",  
      "@context": {"value": "rdf:value", "lang": "@language"}  
    },  
    "city": {"  
      "@id": "https://api.collab.mosip.net/mosip#city",  
      "@context": {"value": "rdf:value", "lang": "@language"}  
    },  
    "gender": {"  
      "@id": "https://api.collab.mosip.net/mosip#gender",  
      "@context": {"value": "rdf:value", "lang": "@language"}  
    },  
    "residenceStatus": {"  
      "@id": "https://api.collab.mosip.net/mosip#residenceStatus",  
      "@context": {"value": "rdf:value", "lang": "@language"}  
    },  
  
    "dateOfBirth": "mosip:dateOfBirth",  
    "email": "mosip:email",  
    "fullName": {"  
  
      "@id": "https://api.collab.mosip.net/mosip#fullName",  
  
      "@context": {"value": "rdf:value", "lang": "@language"}  
    },  
    "phone": "mosip:phone",  
    "postalCode": "mosip:postalCode",  
    "province": {"  
  
      "@id": "https://api.collab.mosip.net/mosip#province",  
      "@context": {"value": "rdf:value", "lang": "@language"}  
    }  
}
```

```
},
"region": {
    "@id": "https://api.collab.mosip.net/mosip#region",
    "@context": {"value": "rdf:value", "lang": "@language"}
},
"biometrics": "mosip:biometrics"
}]
}
```

public-key.json

The "public-key.json" file contains the public key using which the signature of the MOSIP-issued verifiable credential can be asserted.

```
{
    "@context": "https://w3id.org/security/v2",
    "type": "RsaVerificationKey2018",
    "id": "https://api.collab.mosip.net/.well-known/public-key.json",
    "controller": "https://api.collab.mosip.net/.well-known/controller.json",
    "publicKeyPem": "-----BEGIN PUBLIC KEY-----\r\nMIIBIjANBgkqhkiG9w0BAQEFA",
}
```

Custom Handle Implementation Guide

What is a handle?

UIN / VID are system-generated unique identifiers provided to Residents. Residents are allowed to authenticate themselves using either UIN / VID.

What if residents are given the flexibility to create their handle (username) and use their unique handle to authenticate?

- Handles can include resident's phone number, e-mail ID, or any linked functional ID / sectoral ID.
- The handle can also be a custom username created through the resident portal.

Overview

Countries that have an established user base can now register users onto a relying portal using their distinctive identifiers referred to as handles. These handles are tailored to meet the specific requirements of each country, enabling users to easily access digital services and receive prompt benefits from both the government and private sector. This approach eliminates the need for users to remember a new or system generated IDs.

How does it work?

The implementation of custom handles involves below steps:

1. Mark the fields that can be used as user handles. A new attribute is introduced in identity schema, **handle** which accepts boolean value. More than one field in the identity schema can be marked as handle.

With phone as an example:

```
{"fieldCategory": "phone number", "format": "none", "type": "string",
"fieldType": "default", "requiredOn" : "", **"handle" : true**},
```

2. When the user registers, collected user data should contain **selectedHandles**, as more than one field can be marked as handle, user can choose amongst the handle fields to use. User can also choose all of them. Client UI's collecting user data during registration can decide to provide this option to the user or it can also set selected handles to default values as decided by the country.

`selectedHandles` is also a field in schema, `identity`.

```
"selectedHandles" : {"fieldCategory": "none", "format": "none", "type": "array",
"items" : { "type" : "string" }, "fieldType": "default" }
```

3. When the collected identity object is sent to the ID repository, it validates the data and accepts the handle provided it is unique amongst the registered handles.

Note: If duplicated, a request to register the user is rejected.

4. Once identity is successfully processed and stored in an ID-repository, identity credentials are issued to IDA to store user credentials for each ID (UIN & VID) as well for each selected handle.
5. ID-repository can be configured to disable issuance of user credential to IDA for both UIN or VID using below properties.

```
mosip.idrepo.identity.disable-uin-based-credential-request=true
```

What is handles collision? How handles collision can be avoided?

If the system is configured to use more than one functional ID as a handle and if two different functional ID systems followed the same format /pattern to generate an ID, handles collision may occur.

Collision between two different functional IDs will result in denying the creation / updating of a handle for a resident.

Solution: Every handle stored is postfixed with handle type and the handle type is chosen based on the handle field ID in the identity schema. On every authenticate request, IDA will expect handle postfixed with handle_type as input.

Property mentioned below is introduced in ID repository to postfix handle type on every creation of identity.

```
mosip.identity.fieldid.handle-postfix.mapping={'phone': '@phone'}
```

Property mentioned below is introduced in `Id-authentication-default.properties` file to validate the handle value based on the postfix provided in the `individualId` input.

```
mosip.ida.handle-types.regex={'@phone': '^\\+91[1-9][0-9]{7,9}@phone$'}
```

Conclusion

Implementing custom handles provides a user-friendly approach to user authentication without burdening end users with the need to remember additional or system generated complex IDs.

Identity Verification

ID Authentication Services

Overview

ID Authentication is built as an independent service that can be seeded with data for authentication by any system, including MOSIP. In the current design, we can have multiple IDA modules running from a single issuer.

The ID Authentication (IDA) module of MOSIP consists of the following services:

1. Authentication Services
2. OTP Service
3. Internal Services

To learn more about it, refer to the below video:

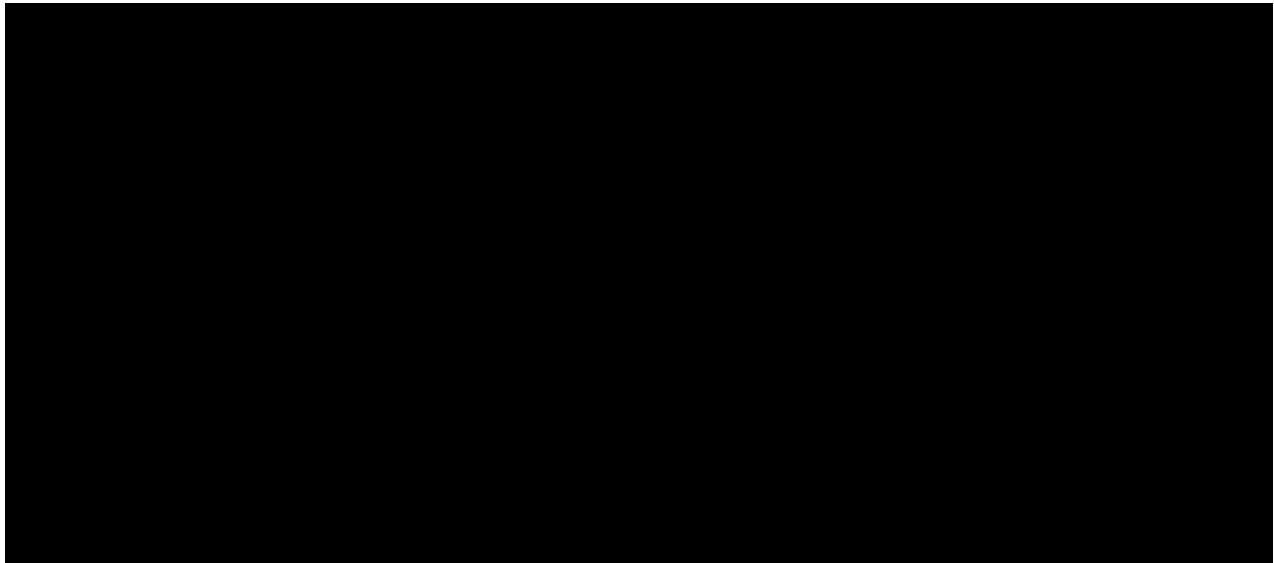
1.2 LTS ID-Auth Demo 10-02-2022



Authentication Services

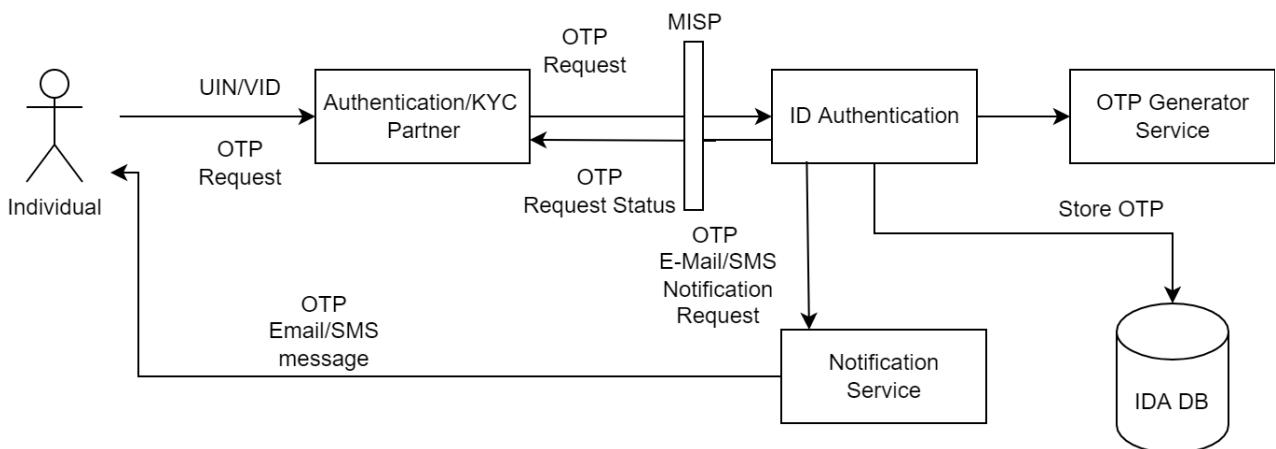
The services mentioned below are used by Authentication or e-KYC Partners.

- Authentication Service: used to authenticate an individual's UIN/VID using one or more authentication types.
- KYC Authentication Service: used to request e-KYC for an individual's UIN/VID using one or more authentication types.



OTP Request Service

OTP Request Service is used by Authentication/e-KYC Partners to generate OTP for an individual's UIN/VID. The generated OTP is stored in IDA DB for validation during OTP Authentication.



Internal Services

1. Internal Authentication Service - The authentication service used by internal MOSIP modules such as Resident Service, Registration Processor and Registration Client to authenticate individuals.
2. Internal OTP Service - used by Resident Service to generate OTP for an Individual for performing OTP Authentication.
3. Authentication Transaction History Service - used by Resident Service to retrieve a paginated list of authentication and OTP Request transactions for an individual.

Credential issuance callback

- [ID Authentication](#) uses the credential data of the individuals for performing authentication.
- This credential is requested by [ID Repository](#) upon any UIN insertion/update or VID creation.
- The credential is created by Credential Service uploaded to [Datashare](#) service and the Datashare URL is sent to ID-Authentication using [WebSub](#) message.
- WebSub invokes the credential-issuance callback in [ID Authentication](#) where the credential data is downloaded from Datashare and then stored in IDA DB.

Key generation

ID Authentication needs the below [keys](#) to be generated during the deployment for usage in Authentication Service.

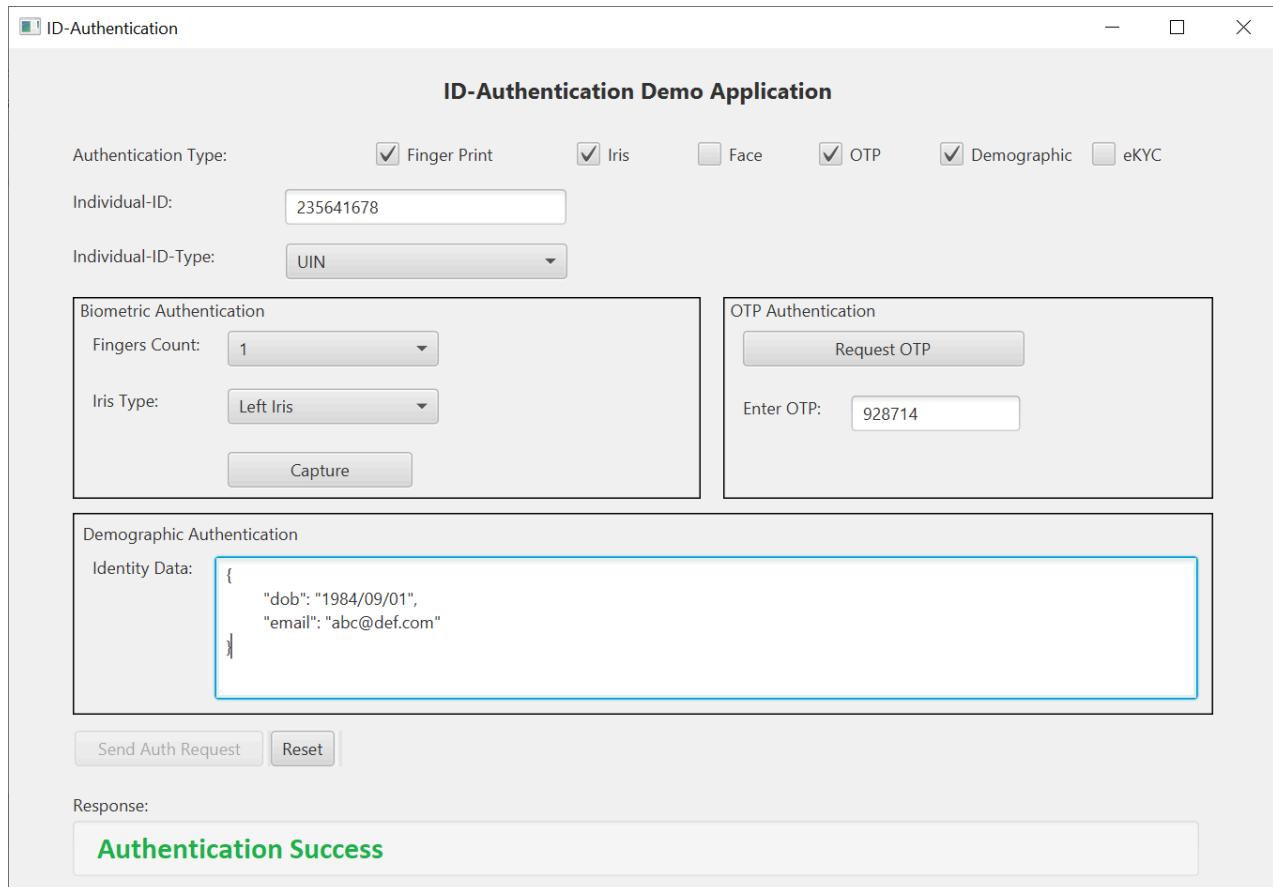
1. `IDA_IDENTITY_CACHE` (K18) symmetric key to encrypt and decrypt the Zero-knowledge 10K random keys
2. `IDA_ROOT` master key(K15)), `IDA_module` master key(K16), `IDA-SIGN` master key
3. Base keys `CRED_SERVICE` (K22), `IDA-FIR` (K21), `INTERNAL` (K19), `PARTNER` (K20)

Authentication client demo app

This is a reference application to demonstrate how authentication and KYC can be performed by [Authentication Partners](#).

Refer to the [repository](#) for more details.

Below is the sample authentication demo UI image.



Authentication Error Eventing

The ID Authentication service now offers an **Authentication Error Eventing** feature. When an authentication related error occurs, a message will prompt to the user to retry after a few minutes. In the meantime, Kafka event will be triggered to publish the data to the designated topic, allowing subscribers to receive a message for further processing.

This feature can be utilized for different use cases such as on demand template extraction, report generations, to identify any fraudulent occurrence etc.

One such use case is on demand template extraction. In an instance where a user has successfully registered and obtained a valid UIN/VID but encounters an error during authentication due to unavailability of the entered UIN/VID in the IDA DB, this feature comes into play. This issue tends to occur particularly during periods of high registration and UIN generation volumes, where additional time is needed for data transmission from the ID Repo to the IDA DB. This authentication error eventing feature will help in capturing the errors related to this issue and event will be created. subscribers can capture this event and process them accordingly to enable the template extraction to proceed with the authentication/verification process.

This feature is designed to be a plugin feature in IDA, which can be configured based on the requirement. To enable the feature below property should be marked as `True` :

```
mosip.ida.authentication.error.eventing.enabled=true
```

Once this property is enabled, related kafka property setup should be installed to utilize the feature.

For further guidance on this feature, you can refer [here](#)

Subscribers who will be subscribing to the event should be onboarded as authentication partners. To on board subscribers below steps needed to be followed:

Steps to onboard the subscribers:

1. Create a policygroup by the name `mpolicygroup-default-tempextraction`
2. The policy should be configured to not allow any authentication to be carryout but the partner except reading the kafka event. To attain this, `allowedAuthTypes` should be marked as `null`

For example:

```
{"authTokenType": "partner", "allowedKycAttributes": [{"attributeName": "fullName"}, {"attributeName": "gender"}],
```

```
{"attributeName": "residenceStatus"}, {"attributeName": "dateOfBirth"},  
{"attributeName": "photo"}], "kycLanguages": ["ara", "eng"], "allowedAuthTypes":  
[]}
```

3. Publish the policygroup and policy
4. Refer this [link](#) to onboard the subscribers as authentication partners. The name of the partner should be `mpartner-default-tempextraction`

Note: This feature is exclusively available in ID Authentication version 1.2.1.0 only. To configure the latest version of IDA and access this new feature, please refer to this link [here](#)

Configuration

Refer to [ID Authentication Configuration Guide](#).

Developer Guide

To know more about the developer setups, read:

1. [ID Authentication Service Developers Guide](#)
2. [ID Authentication OTP Service Developers Guide](#)
3. [ID Authentication Internal Service Developers Guide](#)

API

Refer [API Documentation](#).

Source code

[Github repo.](#)

ID Authentication Demographic Data Normalization

Demographic data normalization is the process of applying rules for formatting of the demographic data (such as the address) into a common format before demographic data matching is verified during the demographic authentication in IDA. For example, for address lines, the '1st Street' can be replaced with '1 st' and 'C/o' can be removed from both the input and database data before the match is verified. These rules will be different for different languages, and may be configured/implemented differently.

The ID-Authentication Demographic data normalization mentioned here is specific to the [Demo-SDK reference implementation](#) of the [Kernel Demographic API](#). It takes the below configuration to apply the name and address normalization rules.

For any other custom implementation of the normalization, the Demo-SDK needs to be implemented accordingly.

Demographic name/address normalization using regular expressions and their replacement configurations

The below configuration is used to define the separator for normalizing regex (pattern) and the replacement word. The default is set to '='.

```
ida.norm.sep==
```

The format for configuring the name/address normalization rules for any language is given below:

```
ida.demo.<name/address/common>.normalization.regex.<languageCode/any>
[<sequential index starting from 0>]=<regular expression>${ida.norm.sep}
<replacement string>
```

- * name/address/common - type of normalization, common applies to both name and address
- * languageCode - this is the code for languages like hin, eng, any('any')

If replacement string is not specified, the regular expression will be replaced with empty string.

Note: It is recommended that the sequence is not broken in the middle otherwise all normalization properties will not be read for the particular type.

Normalization rules for English language

```
ida.demo.address.normalization.regex.eng[0]=[CcSsDdWwHh]/[Oo]
ida.demo.address.normalization.regex.eng[1]=(M|m|D|d)(rs?)(.)
ida.demo.address.normalization.regex.eng[2]=(N|n)(O|o)(\\.)?
ida.demo.address.normalization.regex.eng[3]=[aA][pP][aA][rR][tT][mM][eE][iI][nN]
ida.demo.address.normalization.regex.eng[4]=[sS][tT][rR][eE][eE][tT]${ida.norm.sep}1
ida.demo.address.normalization.regex.eng[5]=[rR][oO][aA][dD]${ida.norm.sep}2
ida.demo.address.normalization.regex.eng[6]=[mM][aA][iI][nN]${ida.norm.sep}3
ida.demo.address.normalization.regex.eng[7]=[cC][rR][oO][sS][sS]${ida.norm.sep}4
ida.demo.address.normalization.regex.eng[8]=[oO][pP][pP][oO][sS][iI][tT]${ida.norm.sep}5
ida.demo.address.normalization.regex.eng[9]=[mM][aA][rR][kK][eE][tT]${ida.norm.sep}6
ida.demo.address.normalization.regex.eng[10]=1[sS][tT]${ida.norm.sep}1
ida.demo.address.normalization.regex.eng[11]=1[tT][hH]${ida.norm.sep}1
ida.demo.address.normalization.regex.eng[12]=2[nN][dD]${ida.norm.sep}2
ida.demo.address.normalization.regex.eng[13]=2[tT][hH]${ida.norm.sep}2
ida.demo.address.normalization.regex.eng[14]=3[rR][dD]${ida.norm.sep}3
ida.demo.address.normalization.regex.eng[15]=3[tT][hH]${ida.norm.sep}3
ida.demo.address.normalization.regex.eng[16]=4[tT][hH]${ida.norm.sep}4
ida.demo.address.normalization.regex.eng[17]=5[tT][hH]${ida.norm.sep}5
ida.demo.address.normalization.regex.eng[18]=6[tT][hH]${ida.norm.sep}6
ida.demo.address.normalization.regex.eng[19]=7[tT][hH]${ida.norm.sep}7
ida.demo.address.normalization.regex.eng[20]=8[tT][hH]${ida.norm.sep}8
ida.demo.address.normalization.regex.eng[21]=9[tT][hH]${ida.norm.sep}9
ida.demo.address.normalization.regex.eng[22]=0[tT][hH]${ida.norm.sep}0
# Note: the common normalization attributes will be replaced at the end.
# Special characters are removed : . , - * ( ) [ ] ` ' / \ # "
# Replace special char with space. Trailing space is removed from property
ida.demo.common.normalization.regex.any[0]=[\\".\\,|\\\"-|\\\"*|\\\"(|\\\")|\\\"[\\\"]]
# Trailing space is removed from property. As a workaround first replacing
ida.demo.common.normalization.regex.any[1]=\\s+${ida.norm.sep} .
ida.demo.common.normalization.regex.any[2]=\\. ${ida.norm.sep}
```

Normalization rules for Non-English language

For non-english languages, the non-english words needs to be converted into UTF-16 and then copied to the configuration. For example, convert the Unicode characters to UTF-16.

Before conversion:

```
ida.demo.address.normalization.regex.hin[0]=پہلی${ida.norm.sep}پہلا
```

After conversion:

```
ida.demo.address.normalization.regex.hin[0]=\u092a\u0939\u0932\u0940${ida.norm.sep}\u092a\u0939\u0932\u093e
```

ID Authentication Service Developers Guide

Overview

The services mentioned below are used by Authentication/e-KYC Partners.

- Authentication service- used to authenticate an individual's UIN/VID using one or more authentication types.
- KYC Authentication service- used to request e-KYC for an individual's UIN/VID using one or more authentication types.

The documentation here will guide you through the prerequisites required for the developer' setup.

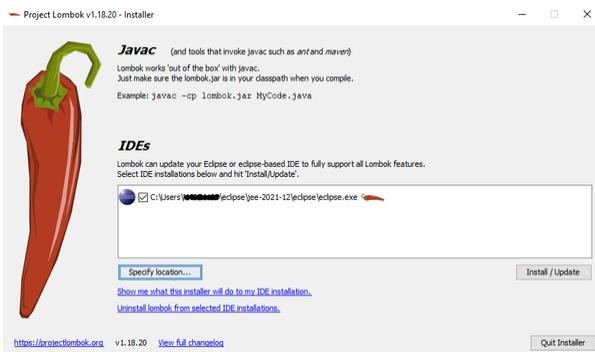
Software setup

Below are a list of tools required in ID Repository Services:

1. JDK 11
2. Any IDE (like Eclipse, IntelliJ IDEA)
3. Apache Maven (zip folder)
4. pgAdmin
5. Postman
6. Git
7. Notepad++ (optional)
8. lombok.jar (file)
9. settings.xml (document)

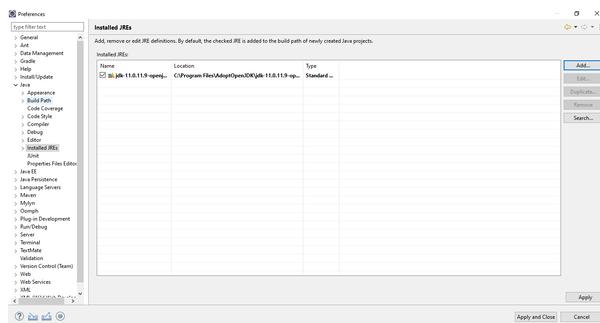
Follow the steps below to set up ID Repository Services on your local system:

1. Download `lombok.jar` and `settings.xml` from [here](#).
2. Unzip Apache Maven and move the unzipped folder in `C:\Program Files` and `settings.xml` to "conf" folder `C:\Program Files\apache-maven-3.8.4\conf`.
3. Install Eclipse, open the `lombok.jar` file and wait for some time until it completes the scan for Eclipse IDE and then click `Install/Update`.



4. Check the Eclipse installation folder `C:\Users\userName\eclipse\jee-2021-12\eclipse` to see if the `lombok.jar` is added. By doing this, you don't have to add the dependency of `lombok` in your `pom.xml` file separately as it is auto-configured by Eclipse.

5. Configure the JDK (Standard VM) with your Eclipse by traversing through `Preferences → Java → Installed JREs`.

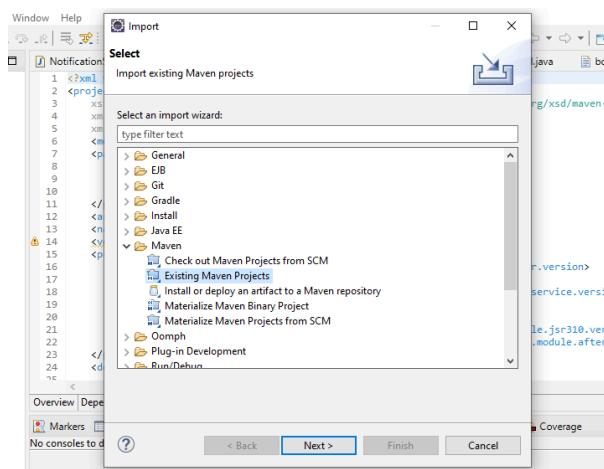


Code setup

For the code setup, clone the repository and follow the guidelines mentioned in the [Code Contributions](#).

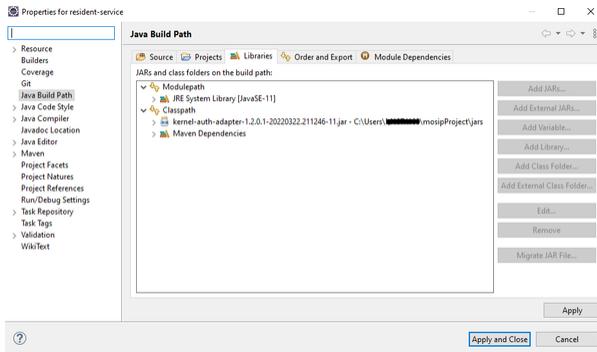
Importing and building of a project

1. Open the project folder where `pom.xml` is present.
2. Open command prompt from the same folder.
3. Run the command `mvn clean install -Dgpg.skip=true` to build the project and wait for the build to complete successfully.
4. After building of a project, open Eclipse and select `Import Projects → Maven → Existing Maven Projects → Next → Browse to project directory → Finish`.
5. After successful importing of project, update the project by right-click on `Project → Maven → Update Project`.



Environment setup

1. For the environment setup, you need an external JAR that is available [here](#) with different versions. (E.g.: You can download `kernel-auth-adapter.jar` and add to project `Libraries → Classpath → Add External JARs → Select Downloaded JAR → Add → Apply and Close`).



2. Clone [mosip-config repository](#).

3. Create an empty folder inside the `mosip-config` with `sandbox-local` name and then copy and paste all config files inside `sandbox-local` folder except `.gitignore`, `README` and `LICENSE`.

4. As ID Authentication is using two properties files, `id-authentication-default` and `application-default`, you will have to configure them according to your environment. The same files are available [here](#) for reference.

Properties to be updated:

`application-default.properties`

- `mosip.mosip.resident.client.secret = <current_password>`.
- `db.dbuser.password=<password>`.
- `mosip.kernel.xsdstorage-uri=file:///home/user/Desktop/tspl/mosip-config/sandbox-local/` (i.e. `sandbox-local` folder location).
- Comment this out `auth.server.admin.issuer.internal.uri` in `application-default.properties` file because you already have this `auth.server.admin.issuer.uri`, and hence there is no need of `auth.server.admin.issuer.internal.uri`.
- `mosip.identity.mapping-file=<Path_to_identity_mapping_json_file>`. (For Example: `file:///home/user/Desktop/tspl/mosip-config/sandbox-local/identity-mapping.json`)

`id-authentication-default.properties`

-

-

5. To run the server, two files are required- [kernel-config-server.jar](#) and [config-server-start.bat](#).

6. Put both the files in the same folder and change the location attribute to `sandbox-local` folder in `config-server-start.bat` file and also check the version of `kernel-config-server.jar` towards the end of the command.

Example:

```
java -jar -Dspring.profiles.active=native -
Dspring.cloud.config.server.native.search-
locations=file:C:\Users\myDell\mosipProject\mosip-config\sandbox-local -
Dspring.cloud.config.server.accept-empty=true -
Dspring.cloud.config.server.git.force-pull=false -
Dspring.cloud.config.server.git.cloneOnStart=false -
Dspring.cloud.config.server.git.refreshRate=0 kernel-config-server-1.2.0-
20201016.134941-57.jar .
```

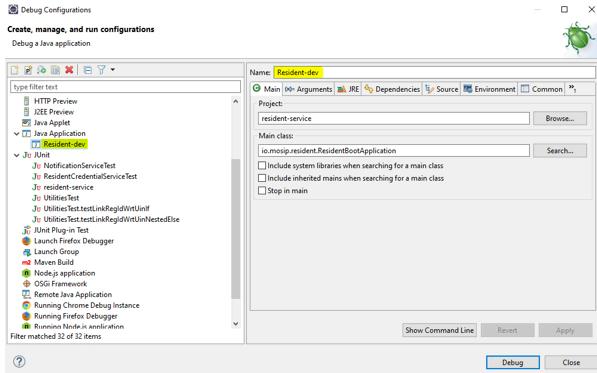
7. Run the server by opening the `config-server-start.bat` file.

```
C:\Windows\system32\cmd.exe
java -jar -Dspring.profiles.active=native -
Dspring.cloud.config.server.native.search-
locations=file:C:\Users\myDell\mosipProject\mosip-config\sandbox-local -
Dspring.cloud.config.server.accept-empty=true -
Dspring.cloud.config.server.git.force-pull=false -
Dspring.cloud.config.server.git.cloneOnStart=false -
Dspring.cloud.config.server.git.refreshRate=0 kernel-config-server-1.2.0-
20201016.134941-57.jar .
```

The server should now be up and running.

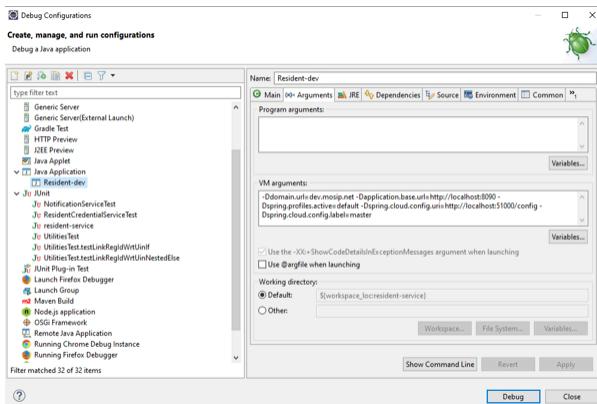
Below are the configurations to be done in Eclipse:

1. Open Eclipse and run the project for one time as `Java application`, so that it will create a Java application which you can see in debug configurations and then change its name. (e.g.: project name with environment - "Auth-Service-Dev").



2. Open the arguments and pass this `-Ddomain.url=dev.mosip.net -Dapplication.base.url=http://localhost:8090 -Dspring.profiles.active=default -Dspring.cloud.config.uri=http://localhost:51000/config -Dspring.cloud.config.label=master` in VM arguments.

3. Here, the domain URL represents the environment on which you are working (eg., it can be `dev2.mosip.net` or `qa3.mosip.net`).



4. Click Apply and then debug it (starts running).

Authentication Service API

- For API documentation, refer [here](#).

ID Authentication OTP Service Developer Guide

Overview

OTP Request Service is used by Authentication/e-KYC Partners to generate OTP for an individual's UIN/VID. The generated OTP is stored in IDA DB for validation during OTP Authentication.

The documentation here will guide you through the prerequisites required for the developer' setup.

Software setup

Below are a list of tools required in ID Repository Services:

1. JDK 11
2. Any IDE (like Eclipse, IntelliJ IDEA)
3. Apache Maven (zip folder)
4. pgAdmin
5. Postman
6. Git
7. Notepad++ (optional)
8. lombok.jar (file)
9. settings.xml (document)

Follow the steps below to set up ID Repository Services on your local system:

1. Download `lombok.jar` and `settings.xml` from [here](#).

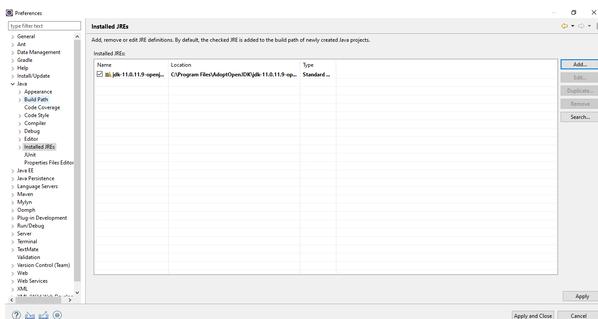
2. Unzip Apache Maven and move the unzipped folder in `C:\Program Files` and `settings.xml` to "conf" folder `C:\Program Files\apache-maven-3.8.4\conf`.

3. Install Eclipse, open the `lombok.jar` file and wait for some time until it completes the scan for Eclipse IDE and then click `Install/Update`.



4. Check the Eclipse installation folder `C:\Users\userName\eclipse\jee-2021-12\eclipse` to see if the `lombok.jar` is added. By doing this, you don't have to add the dependency of `lombok` in your `pom.xml` file separately as it is auto-configured by Eclipse.

5. Configure the JDK (Standard VM) with your Eclipse by traversing through `Preferences → Java → Installed JREs`.

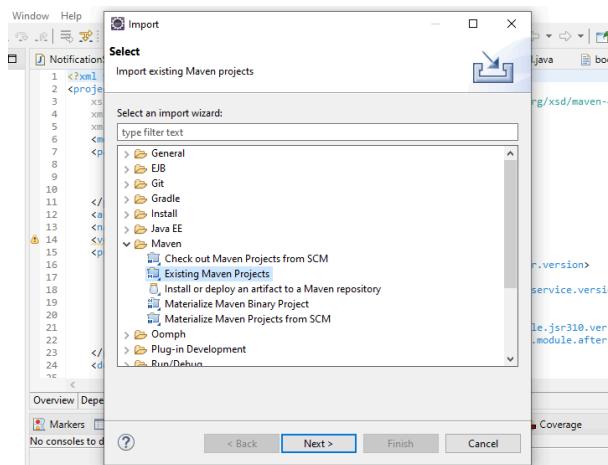


Code setup

For the code setup, clone the repository and follow the guidelines mentioned in the [Code Contributions](#).

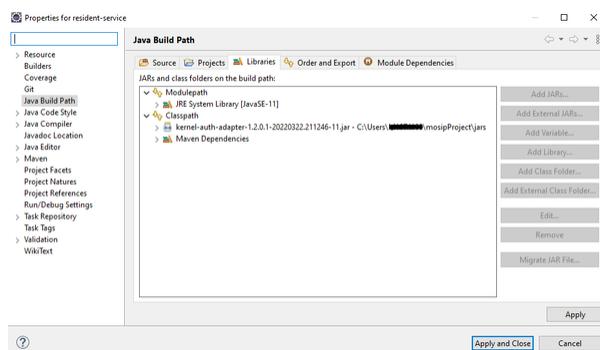
Importing and building of a project

1. Open the project folder where `pom.xml` is present.
2. Open command prompt from the same folder.
3. Run the command `mvn clean install -Dgpg.skip=true` to build the project and wait for the build to complete successfully.
4. After building of a project, open Eclipse and select `Import Projects → Maven → Existing Maven Projects → Next → Browse to project directory → Finish`.
5. After successful importing of project, update the project by right-click on `Project → Maven → Update Project`.



Environment setup

1. For the environment setup, you need an external JAR that is available [here](#) with different versions. (E.g.: You can download `kernel-auth-adapter.jar` and add to project `Libraries → Classpath → Add External JARs → Select Downloaded JAR → Add → Apply and Close`).



2. Clone [mosip-config repository](#).

3. Create an empty folder inside the `mosip-config` with `sandbox-local` name and then copy and paste all config files inside `sandbox-local` folder except `.gitignore`, `README` and `LICENSE`.

4. As ID Authentication is using two properties files, `id-authentication-default` and `application-default`, you will have to configure them according to your environment. The same files are available [here](#) for reference.

Properties to be updated:

`application-default.properties`

- `mosip.mosip.resident.client.secret = <current_password>`.
- `db.dbuser.password=<password>`.
- `mosip.kernel.xsdstorage-uri=file:///home/user/Desktop/tspl/mosip-config/sandbox-local/` (i.e. `sandbox-local` folder location).
- Comment this out `auth.server.admin.issuer.internal.uri` in `application-default.properties` file because you already have this `auth.server.admin.issuer.uri`, and hence there is no need of `auth.server.admin.issuer.internal.uri`.
- `mosip.identity.mapping-file=<Path_to_identity_mapping_json_file>`. (For Example: `file:///home/user/Desktop/tspl/mosip-config/sandbox-local/identity-mapping.json`)

`id-authentication-default.properties`

-
-

5. To run the server, two files are required- [kernel-config-server.jar](#) and [config-server-start.bat](#).

6. Put both the files in the same folder and change the location attribute to `sandbox-local` folder in `config-server-start.bat` file and also check the version of `kernel-config-server.jar` towards the end of the command.

Example:

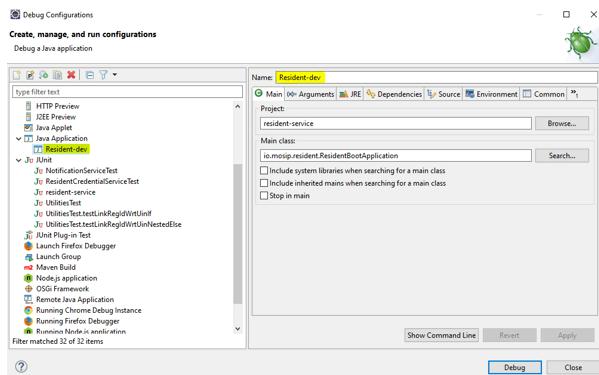
```
java -jar -Dspring.profiles.active=native -  
Dspring.cloud.config.server.native.search-  
locations=file:C:\Users\myDell\mosipProject\mosip-config\sandbox-local -  
Dspring.cloud.config.server.accept-empty=true -  
Dspring.cloud.config.server.git.force-pull=false -  
Dspring.cloud.config.server.git.cloneOnStart=false -  
Dspring.cloud.config.server.git.refreshRate=0 kernel-config-server-1.2.0-  
20201016.134941-57.jar .
```

7. Run the server by opening the `config-server-start.bat` file.

The server should now be up and running.

Below are the configurations to be done in Eclipse:

1. Open Eclipse and run the project for one time as `Java application`, so that it will create a Java application which you can see in debug configurations and then change its name. (e.g.: project name with environment - "Auth-Otp-Service-Dev").



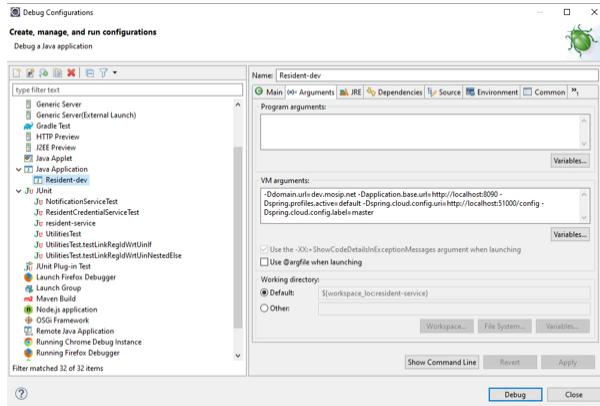
2. Open the arguments and pass this -Ddomain.url=dev.mosip.net -

Dapplication.base.url=http://localhost:8090 -

Dspring.profiles.active=default -

`Dspring.cloud.config.uri=http://localhost:51000/config -`
`Dspring.cloud.config.label=master` in VM arguments.

3. Here, the domain URL represents the environment on which you are working (eg., it can be `dev2.mosip.net` or `qa3.mosip.net`).



4. Click Apply and then debug it (starts running).

Authentication OTP Service API

- For API documentation, refer [here](#).

ID Authentication Internal Service Developers Guide

Overview

- Internal Authentication Service: The authentication service used by internal MOSIP modules such as Resident Service, Registration Processor, and Registration Client to authenticate individuals.
- Internal OTP Service: used by Resident Service to generate an OTP for an Individual for performing OTP Authentication.
- Authentication Transaction History Service: used by Resident Service to retrieve a paginated list of authentication and OTP Request transactions for an individual.

The documentation here will guide you through the prerequisites required for the developer's setup.

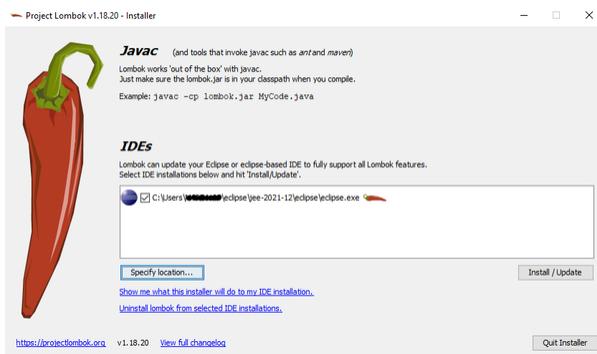
Software setup

Below is a list of tools required in ID Repository Services:

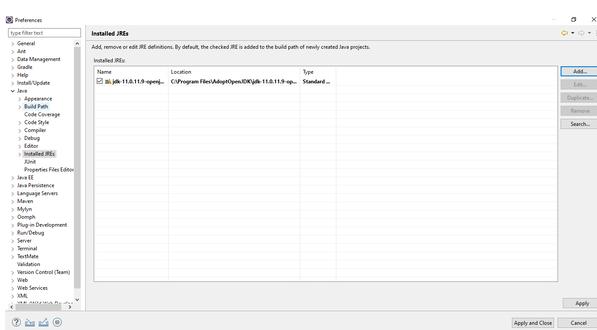
1. JDK 11
2. Any IDE (like Eclipse or IntelliJ IDEA)
3. Apache Maven (zip folder)
4. pgAdmin
5. Postman
6. Git
7. Notepad++ (optional)
8. lombok.jar (file)
9. settings.xml (document)

Follow the steps below to set up ID Repository Services on your local system:

1. Download `lombok.jar` and `settings.xml` from [here](#).
2. Unzip Apache Maven and move the unzipped folder in `C:\Program Files` and `settings.xml` to "conf" folder `C:\Program Files\apache-maven-3.8.4\conf`.
3. Install Eclipse, open the `lombok.jar` file, wait for some time until it completes the scan for the Eclipse IDE, and then click `Install/Update`.



4. Check the Eclipse installation folder `C:\Users\userName\eclipse\jee-2021-12\eclipse` to see if the `lombok.jar` is added. By doing this, you don't have to add the dependency of `lombok` in your `pom.xml` file separately, as it is auto-configured by Eclipse.
5. Configure the JDK (Standard VM) with your Eclipse by traversing through `Preferences → Java → Installed JREs`.

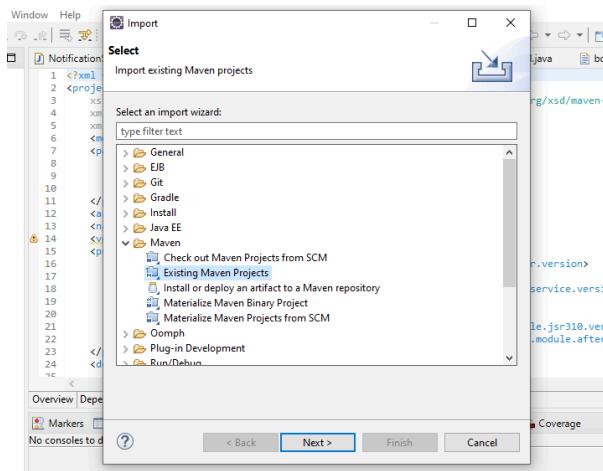


Code setup

For the code setup, clone the repository and follow the guidelines mentioned in the [Code Contributions](#).

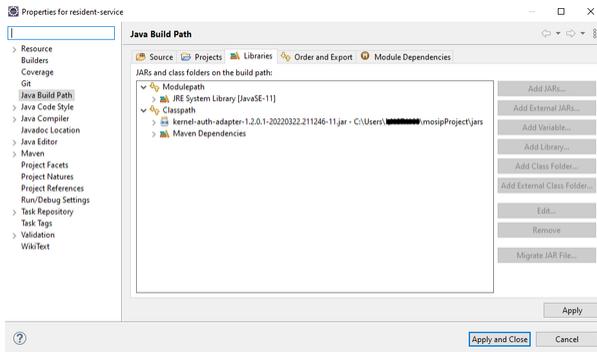
Importing and building a project

1. Open the project folder where `pom.xml` is present.
2. Open the command prompt from the same folder.
3. Run the command `mvn clean install -Dgpg.skip=true` to build the project and wait for the build to complete successfully.
4. After building a project, open Eclipse and select `Import Projects → Maven → Existing Maven Projects → Next → Browse to project directory → Finish`.
5. After successfully importing of project, update the project by right-clicking on `Project → Maven → Update Project`.



Environment setup

1. For the environment setup, you need an external JAR that is available [here](#) with different versions. (E.g.: You can download `kernel-auth-adapter.jar` and add to the project `Libraries → Classpath → Add External JARs → Select Downloaded JAR → Add → Apply and Close`).



2. Clone [mosip-config repository](#).

3. Create an empty folder inside the `mosip-config` with `sandbox-local` name and then copy and paste all config files inside `sandbox-local` folder except `.gitignore`, `README` and `LICENSE`.

4. As ID Authentication is using two property files, `id-authentication-default` and `application-default`, you will have to configure them according to your environment. The same files are available [here](#) for reference.

Properties to be updated:

`application-default.properties`

- `mosip.mosip.resident.client.secret = <current_password>`.
- `db.dbuser.password=<password>`.
- `mosip.kernel.xsdstorage-uri=file:///home/user/Desktop/tspl/mosip-config/sandbox-local/` (i.e. `sandbox-local` folder location).
- Comment this out `auth.server.admin.issuer.internal.uri` in `application-default.properties` file because you already have this `auth.server.admin.issuer.uri`, and hence there is no need for `auth.server.admin.issuer.internal.uri`.
- `mosip.identity.mapping-file=<Path_to_identity_mapping_json_file>`. (For Example: `file:///home/user/Desktop/tspl/mosip-config/sandbox-local/identity-mapping.json`)

`id-authentication-default.properties`

-

-

5. To run the server, two files are required- [kernel-config-server.jar](#) and [config-server-start.bat](#).

6. Put both the files in the same folder and change the location attribute to `sandbox-local` folder in `config-server-start.bat` file and also check the version of `kernel-config-server.jar` towards the end of the command.

Example:

```
java -jar -Dspring.profiles.active=native -
Dspring.cloud.config.server.native.search-
locations=file:C:\Users\myDell\mosipProject\mosip-config\sandbox-local -
Dspring.cloud.config.server.accept-empty=true -
Dspring.cloud.config.server.git.force-pull=false -
Dspring.cloud.config.server.git.cloneOnStart=false -
Dspring.cloud.config.server.git.refreshRate=0 kernel-config-server-1.2.0-
20201016.134941-57.jar .
```

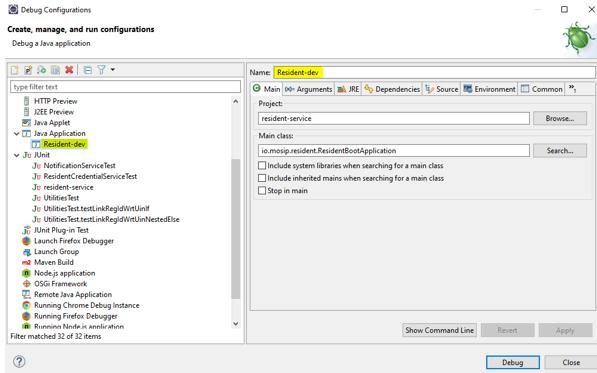
7. Run the server by opening the `config-server-start.bat` file.

```
C:\Users\myDell\mosipProject\mosip-config\sandbox-local>java -jar -Dspring.profiles.active=native -
-Dspring.cloud.config.server.native.search-
locations=file:C:\Users\myDell\mosipProject\mosip-config\sandbox-local -
Dspring.cloud.config.server.accept-empty=true -
Dspring.cloud.config.server.git.force-pull=false -
Dspring.cloud.config.server.git.cloneOnStart=false -
Dspring.cloud.config.server.git.refreshRate=0 kernel-config-server-1.2.0-
20201016.134941-57.jar .
```

The server should now be up and running.

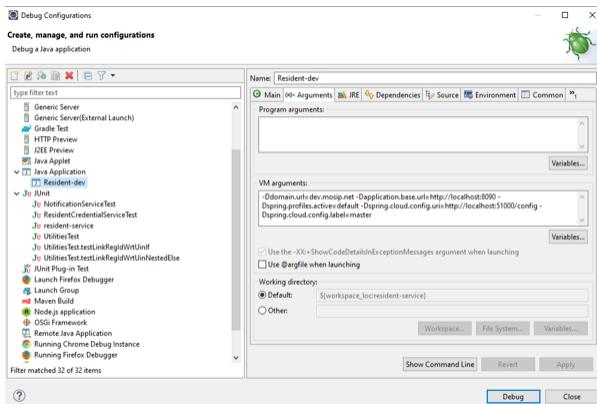
Below are the configurations to be done in Eclipse:

1. Open Eclipse and run the project for one time as `Java application`, so that it will create a Java application which you can see in debug configurations and then change its name. (e.g.: project name with the environment - "Auth-Internal-Service-Dev").



2. Open the arguments and pass this `-Ddomain.url=dev.mosip.net -Dapplication.base.url=http://localhost:8090 -Dspring.profiles.active=default -Dspring.cloud.config.uri=http://localhost:51000/config -Dspring.cloud.config.label=master` in VM arguments.

3. Here, the domain URL represents the environment on which you are working (eg., it can be `dev2.mosip.net` or `qa3.mosip.net`).



4. Click Apply and then debug it (starts running).

Authentication Internal Service API

- For API documentation, refer [here](#).

MOSIP Authentication SDK

Overview

The **MOSIP Authentication SDK** is a (Python-based) wrapper designed to simplify interaction with the **MOSIP Authentication Service**, enabling seamless integration of robust identity verification workflows into Python applications. This SDK abstracts complex details such as request/response structures, encryption/decryption mechanisms, and error handling, allowing developers to implement authentication workflows quickly and efficiently.

Currently, the SDK supports OTP authentication and demographic authentication. Future updates will expand its functionality to include biometric authentication.

Additionally, although the SDK is currently Python-based, we will soon be expanding support to other languages to offer broader compatibility.

Purpose

This page provides an overview of the Authentication SDK, outlining its functionality and providing a [detailed process for installing](#) and testing the IDA API using the SDK.

Why Use This SDK?

While building your solution around MOSIP, it is recommended to use eSignet, MOSIP's OAuth- and OIDC-based solution, for most online and scalable authentication needs due to its modern, standards-compliant design. However, the MOSIP Authentication SDK offers its own advantages, particularly in the flexibility it provides, making it an invaluable tool for addressing a wide range of identity verification requirements.

1. **Ease of Integration:** Simplifies the process of working with MOSIP's APIs, reducing the learning curve for developers

2. **Consistency:** Provides a uniform interface for different authentication operations, ensuring a consistent experience
3. **Security:** Manages encryption and decryption of requests and responses, adhering to MOSIP's security standards
4. **Flexibility:** Supports multiple authentication methods, including demographic authentication, offering versatility in identity verification workflows

Key Features

1. **Simplified API Interaction:** Abstracts the complexity of direct API calls to MOSIP services
2. **Support for Multiple Authentication Workflows:** Includes controllers for both KYC-based and general authentication
3. **Comprehensive Configuration:** Allows customization via a configuration file (authenticator-config.toml)
4. **Secure Handling:** Automatically encrypts requests and decrypts responses to ensure secure communication
5. **Error Management:** Provides clear error messages and handling mechanisms

Controllers

The SDK provides two primary controllers, each designed for a specific authentication workflow:

1. kyc-auth-controller
Used for **Know Your Customer (KYC)** authentication. This controller facilitates verification using demographic data or OTP verification.
Reference: [KYC Auth Controller API Documentation](#)
2. auth-controller
Used for general authentication of individuals, allowing verification based on a wide range of identifiers such as demographic authentication and OTP authentication.
Reference: [Auth Controller API Documentation](#)

Method Reference

The SDK provides two key methods for authentication:

1. **kyc Method:** Used for KYC-based authentication by verifying an individual's demographic data and OTP.
2. **auth Method:** Handles general authentication requests with similar parameters as kyc.

Both methods require the individual's ID (individual_id), ID type (individual_id_type), demographic data (DemographicsModel), optionally an OTP, biometric data, and consent confirmation. These methods streamline identity verification processes for diverse use cases.

Please refer below to know more about the methods.

kyc Method

Authenticates an individual using KYC-based workflow.

```
kyc(  
    individual_id: str,  
    individual_id_type: str,  
    demographic_data: DemographicsModel,  
    otp_value: Optional[str] = None,  
    biometrics: Optional[List[BiometricModel]] = None,  
    consent: bool = False  
) -> Response
```

auth Method

Performs a general authentication.

```
auth(  
    individual_id: str,  
    individual_id_type: str,  
    demographic_data: DemographicsModel,  
    otp_value: Optional[str] = None,  
    biometrics: Optional[List[BiometricModel]] = None,  
    consent: bool = False  
) -> Response
```

Common Parameters

- **individual_id (str)**: The unique ID of the individual (e.g., VID, UIN)
- **individual_id_type (str)**: Specifies the type of ID used (e.g., VID, UIN)
- **demographic_data (DemographicsModel)**: A model containing demographic details such as name and address
- **otp_value (Optional[str])**: The One-Time Password (OTP) for authentication, if applicable
- **consent (bool)**: Indicates if the individual has given consent for authentication

Installation Process

Pre-requisites:

Before beginning the installation and configuration of this SDK, the user must complete the following steps:

1. **Register as an Authentication Partner (AP)**: Register their organization as an Authentication Partner. Please refer to this link [here](#) and follow the steps for registration.
2. **Obtain the IDA-FIR(K21) Certificate**: The user must possess the IDA-FIR(K21) certificate. The certificate can be obtained [here](#).
3. **Provide required details in the request:**
 - a. app id: IDA
 - b. ref : IDA-FIR
4. **Install pip on the machine**: The user should install pip to manage Python packages. Installation instructions can be found [here](#).

Configuration

During installation, the SDK must be configured by updating the authenticator-config.toml file. Please refer to this link [here](#) for the configuration file, This file contains essential details, such as:

- **Service Endpoints**
- **Encryption Keys**
- **Timeout Settings**
- **Logging Settings**

Refer to this link [here](#) for a sample configuration file to guide you in the setup process.

Installation

Install the SDK using pip:

```
pip install git+https://github.com/mosip/ida-auth-sdk.git@v0.9.0
```

Usage

Users who wish to try out this SDK should follow these steps:

1. **Initialize the Authenticator:** Set up the authentication instance to begin interacting with the SDK
2. **Create Demographic Data:** Prepare the necessary demographic information required for authentication
3. **Perform Authentication:** Execute the authentication request using the SDK
4. **Handle the Response:** Process and utilize the response received from the authentication service

For detailed guidance on performing these steps during the installation process, please refer to the model implementation below.

Basic Example:

```
from mosip_auth_sdk import MOSIPAuthenticator
from mosip_auth_sdk.models import DemographicsModel, BiometricModel

# Initialize the authenticator
authenticator = MOSIPAuthenticator(config={
    # Configuration settings (refer to authenticator-config.toml for details)
})

# Create demographic data
demographics_data = DemographicsModel(
    name="John Doe",
    address="123 Main Street"
    # Add other fields as required
)

# Perform KYC authentication
response = authenticator.kyc(
    individual_id="123456789",
    individual_id_type="VID",
    demographic_data=demographics_data,
    otp_value="1234",
    consent=True
)

# Handle the response
if response.status_code == 200:
    decrypted_data = authenticator.decrypt_response(response.json())
    print("Authentication successful:", decrypted_data)
else:
    print("Authentication failed:", response.json())
```

Error Handling

The SDK provides clear error messages and codes to help diagnose issues effectively. Review the errors field in the response for details.

Encryption and Decryption

All communication with the MOSIP service is securely encrypted. Use the decrypt_response method to handle encrypted responses appropriately.

Conclusion

The **MOSIP Authentication SDK** simplifies the integration of robust authentication workflows into Python applications, ensuring secure, efficient, and compliant identity verification. By abstracting the complexities of direct API interaction, the SDK enables developers to focus on building impactful solutions without having to manage intricate implementation details.

Get in Touch

If you require any assistance or encounter any issues during the testing and integration process, kindly reach out to us through the support provided below.

- Navigate to [Community](#).
- Provide a detailed description about the support you require or provide complete information about the issue you have encountered, including steps to reproduce, error messages, logs and any other required details.

Thank you. Wishing you a pleasant experience!

ID Authentication

Overview

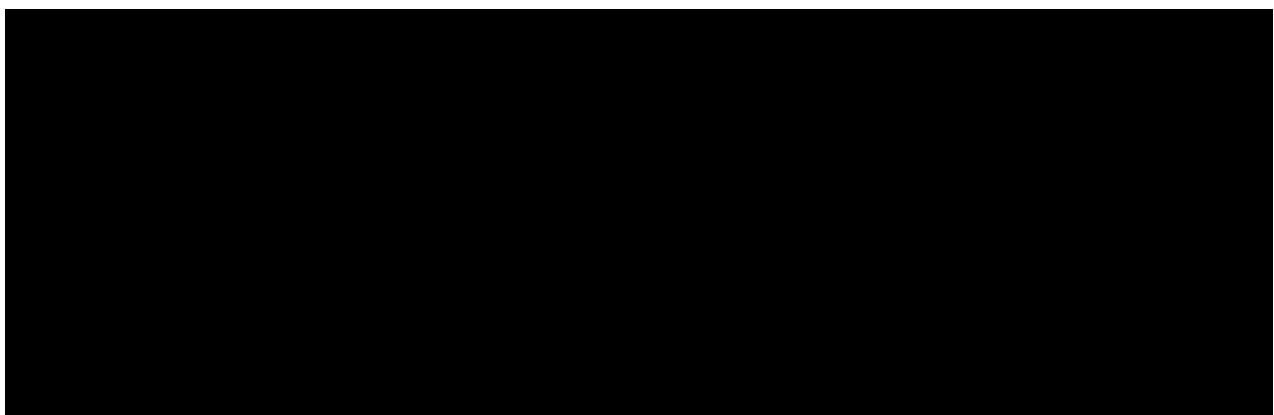
What is ID Authentication?

[IDA \(ID Authentication\)](#) module in MOSIP is an independent service that enables seamless identity verification using data from any system. Multiple IDA modules can run from a single issuer, providing services like authentication, OTP generation, and internal processes.

MOSIP recommends using eSignet for authentication, which leverages the IDA module to ensure secure and unified identity verification. eSignet simplifies the process by streamlining authentication across services, ensuring compliance with MOSIP's security protocols. This enables both governmental and private entities to securely authenticate residents without needing multiple solutions.

Read more about eSignet and its capabilities [here](#).

The typical authentication flow is illustrated below:



Authentication Process

Authentication types

The following types of authentication are offered by MOSIP's ID Authentication (IDA) module and are utilized through **eSignet** for ID verification by external parties:

1. Yes/No Authentication

MOSIP offers a *yes/no* API that can be used for the verification of attributes supplied along with authentication factors. The API verifies the identifier and the provided demographic attributes and also validates other authentication factors such as the OTP or biometrics and responds with a *yes* or a *no*. Successful verification of the data results in a *yes*. This kind of API can be typically used to support the verification of a limited set of demographic data about the person or for simple presence verification when biometrics are used.

2. KYC Authentication

MOSIP additionally offers a KYC API, which can be used to get an authorized set of attributes for the resident in the response of the API. This API is intended for use by authorized relying parties to perform KYC requests. The authentication includes an identifier along with authentication factors such as OTP and biometrics. The information returned is governed by a policy. Different relying parties can be provided with different KYC data based on their needs. The policy helps implement selective disclosure as part of the KYC data. The data thus returned is digitally signed by the server and can be used by the relying party with confidence.

3. Multifactor authentication

The authentication APIs support multiple factors. These can be:

- Biometric: Finger, face, iris
- Demographic: Name, date of birth, age, gender, etc.
- OTP: One-Time Password Based on the level of assurance needed for the transaction, the relying party can decide which factors are sufficient for identity verification.
- Password-based authentication.

Biometric authentication is performed using third-party matcher SDK that performs 1:1 matches on a given modality. Each biometric modality is treated as an independent factor in authentication.

VID

All authentications in MOSIP operate on a 1:1 matching principle. This requires the authentication request to include an identifier for the individual, along with authentication factors to verify the identity. MOSIP supports multiple identifiers for each person, which enhances privacy and prevents profiling. The individual can be authenticated using their UIN or alternate identifiers like VIDs. When a VID is used, additional checks ensure it hasn't expired or been revoked. The expiration of a VID is governed by the policy set during its creation.

To understand VIDs and their characteristics, read more about [VID](#).

Tokenization

In certain contexts, identity verification can be performed anonymously for one-time use. However, when identity verification is tied to transactions requiring identity assurance, it becomes necessary to link the user's identity to the transaction. This is done by providing relying parties with a "sticky" token identifier, which can serve as a reference ID for the individual in their system. When authentication is successful, the APIs return a token. Depending on the relying party's policies, the token may be random or "sticky." The relying party is expected to store this token for future reference, customer identification, and audit or redressal purposes.

Learn more about the [Token ID](#).

Relying parties and policies

Read more about [parties and policies](#).

Consent

MOSIP has a provision for specifying the user consent associated with an authentication transaction. This can be stored for audit purposes and the authentication flow can be extended to verify the consent if needed.

Hotlisting

MOSIP has a provision to help protect against the misuse of identity. For any report of lost identity or detection of fraudulent activity by fraud, the module can require the temporary suspension of authentication activities on a user. This is enabled by the hotlisting feature. The authentication service checks if the identifier used is hotlisted and if so, the authentication process is aborted and fails. The hotlisting service can be used by helpdesk and fraud solutions to list and delist the identifiers that need to be blocked temporarily.

Identity Management

ID Schema

Overview

ID Schema is a standard [JSON schema](#) that defines dataset that can be stored in the Identity repository for a resident. The schema allows the adopters to customize the fields that's needed for running the identity program.

Defining the ID Schema is the first step towards creating a foundational ID system. Once defined, all applications built on top of the MOSIP platform must conform to the same.

One should not confuse ID Schema with what is seen on the screen of the Registration client/ Pre-registration. ID Schema is the final data that you would like to store against each user in the final ID Repository. Quite often we collect more data than listed in ID Schema. This is essential to validate the user's claim. We should consider these data as transactional and it will never reach the final ID Repository.

Understanding ID Schema

This guide is intended for adopters who would customize the default ID Schema to suit the needs of a specific deployment.

Terminology

- **Field:** Unit of data collected from residents (eg. `fullName`, `dateOfBirth`, `proofOfIdentity` etc).
- **Field attribute:** Qualification of Field (e.g.: `fieldCategory`, `fieldType`, etc).
- **Definition:** Custom data types are defined for collecting different types of data:
 - `simpleType`: Multiple languages.
 - `documentType`: Document metadata.

- `biometricType`: Biometric file [CBEFF XML](#) metadata

JSON keys

- `bioAttributes`:
 - `leftEye`, `rightEye`, `leftIndex`, `leftRing`, `leftLittle`, `leftMiddle`,
`rightIndex`, `rightRing`, `rightMiddle`, `rightLittle`, `rightThumb`,
`leftThumb`, `face`,
- `fieldCategory`
 - `none`: Cannot be used for any purpose. But will be stored in id.json (default subpacket). These are used in exceptional cases when we need to store some data for future reference in case of investigating an ID fraud or if law mandates the storage of such data.
 - `pvt`: Private information, can be used in authentication. A limited data that can be used for authentication & kyc.
 - `kyc`: Information that can be disclosed to partners either as a credential or through the ekyc API's.
 - `evidence`: Field is treated as proof and may be subjected to removal. In certain countries law mandates deletion of such data once the Identity of the user is verified. Marking some of the fields as `evidence` helps in deleting them without violating the source of truth signatures.
 - `optional`: Field is treated as proof and will be removed after a predefined interval (defined as policy).
- `format`
 - `lowercased`: Value stored in lower case format
 - `uppercased`: Value stored in upper case format
 - `none`: No format applied to the data
- `validators`
 - `type`: Validation engine type. Supported type as of now is `regex`.
 - `validator`: Based on the type the actual script is placed here. In case the `type` is `regex` then the actual regex pattern is used here.
 - `arguments`: Array to hold parameter or dependent field IDs required for validation.

- `subType`
 - For every `documentType` field, `document category code` must be the value of this key. This document category code is used to validate the provided document types in the ID object.

Refer to the [sample ID Schema](#). A guide to customising the same is given below.

ID schema is loaded as a part of master data to `identity_schema` table in `mosip_masterdata` DB.

Dependencies

If any changes are made to the default ID Schema, make sure the following dependencies are updated as well:

1. UI Specifications
 - [Pre-registration](#)
 - [Registration Client](#)
2. [Identity Mapping JSON](#)
3. [ID Authentication Mapping JSON](#)
4. [Applicanttype MVEL](#)

Versions of ID Schema

ID Schema is identified based on its version in the MOSIP system. On publishing of an ID Schema, the schema is versioned. Every ID Object stores the ID Schema version which is validated during ID Object validation.

Good Practice

The following is the list of good practices that MOSIP recommends for creating your ID Schema.

- As a privacy by design practice, it is recommended that the number of fields is kept to a usable minimum in order to avoid profiling.
- Larger number of data results in serious data quality issues.
- Keeping the fields minimum ensures everyone is inclusively added to the foundational identity.
- As a best guide, limit the total number of fields to be less than 10.
- Stick to one version of ID Schema for better compatibility.

Identifiers

Overview

In the context of MOSIP, identifiers are alphanumeric digital handles for identities in the system. While a person's identity is represented as a collection of biographic and biometric attributes that can uniquely identify the person, the identity is referred to using identifiers.

UIN

Unique Identification Number (UIN), as the name suggests, is a unique number assigned to a resident. UIN never changes and is non-revocable. UIN is randomized such that one should not be able to derive any Personal Identifiable Information (PII) from the number itself.

The rules that govern generation of a UIN are listed [here](#).

VID

The VID / Virtual ID is an alias identifier that can be used for authentication transactions. The key characteristic of such an identifier is that it expires. This allows for free disclosure and usage of this identifier in various contexts. It is privacy friendly in a way such that it can be revoked, configured for one time usage and is not linkable. Since these are used for authentication transactions, such identifiers are to be known to the user only or generated with their participation.

RID / AID

The Application ID (AID) refers to the unique identifier given to a resident during any ID lifecycle event, such as ID Issuance, ID Update, or Lost ID retrieval, at the registration center. It serves as a distinguishing factor for each specific event and

can later be utilized by the resident to check the progress or status of the event. Previously, in MOSIP, the Application ID was referred to as the RID (Registration ID) and/or PRID (Pre-registration ID).

PRID

The PRID is a specialized RID used in the pre-registration system.

Token ID (PSUT - Partner Specific User Token)

The Token identifier/PSUT is a system provided customer reference number for relying parties to uniquely identify the users in their system. The token identifier is an alias meant for the partner/relying party typically unique (Configured through PMS policy, in case uniqueness is not the need then partner policy can be set to provide random number) to them. This identifier is included in the response of the authentication transactions. One key differentiator is that the PSUT is not accepted as an identifier for authentication transactions. This ensures that the partner who has the PSUT can not reauthenticate.

Resident Portal

Overview

Resident services are self-service tools utilized by residents through an online portal. The [Resident Portal](#) is a web-based user interface application that offers services related to the residents' Unique Identification Number (UIN). Through this portal, residents can perform various operations related to their UIN or VID and can also raise any concerns they may have.

The key features provided on the Resident portal are:

1. Avail **UIN services** using UIN/ VID (through [e-Signet](#)):

- **View My History:** This feature enables the Resident to view the history of transactions associated with their UIN.
- **Manage My VID:** Residents can create, delete, and download VID cards based on requirements.
- **Secure My ID:** Residents can lock or unlock their authentication modalities such as fingerprint authentication, iris authentication, email OTP authentication, SMS OTP authentication, thumbprint authentication, and face authentication.
- **Track My Requests:** This feature enables the Residents to enter an Event ID associated with the logged-in user's UIN to track the status of the event.
- **Get Personalized Card:** The residents can download a personalized card which essentially means that they can choose the attributes that they would want to be added to their cards.
- **Share My Data:** This feature enables Residents to choose the data that they want to share with a MOSIP-registered partner.
- **Update My Data:** This feature enables the Resident to update their identity data, address, email ID, phone number, and notification language preference.
- **Logout:** Once the Resident is done with the activities that he wanted to perform, he can end the active session by logging out from the portal.

2. Get Information

- About Registration Centers: Residents can get a list of Registration Centers near them or Registration Centers based on the location hierarchy.
 - List of supporting documents: Residents can get the list of all the supporting documents as Proof of Identity, Proof of Address, Proof of Relationship, etc.
3. **Get My UIN (using UIN/ VID/ AID):** Using this feature, the Resident can download their password-protected UIN card if the UIN card is ready or they can view the status of their Application ID (AID) if the UIN card is still under progress.
 4. **Verify email ID and/ or phone number:** Using this feature, the Resident can verify if the email ID/ Phone number given during registration is correct or not. This will be done by verifying the OTP sent over the registered email ID/ Phone number.
 5. **Book an appointment for new enrollment (via the pre-registration portal):** Using this feature, the Resident can book an appointment to visit the Registration center.

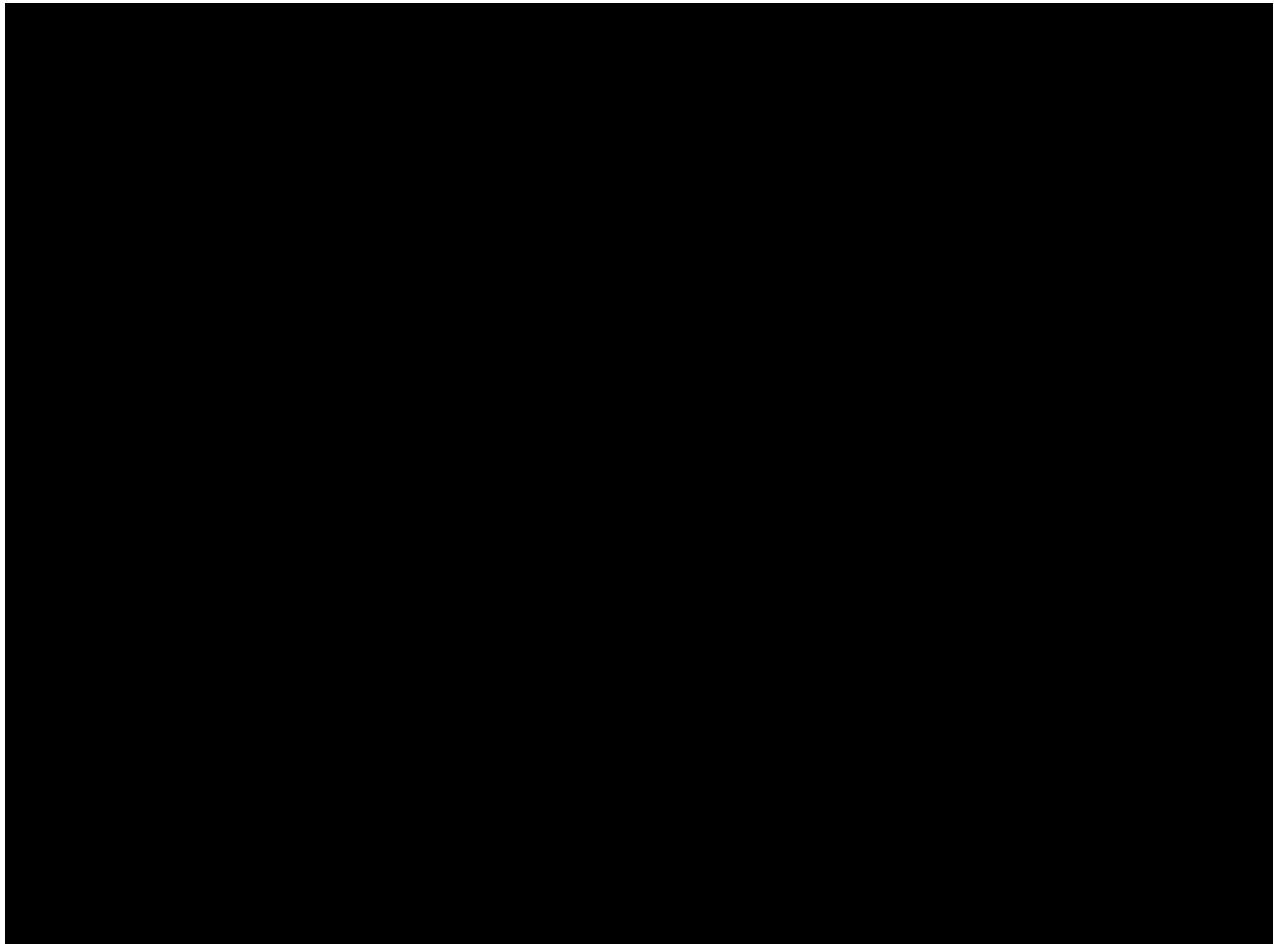
6. Ancillary features

- **Multi-lingual support:** Residents can view and use the Resident Portal in multiple languages including RTL languages.
- **Get Notifications (email and bell notifications):** Residents will be getting bell-icon notifications for the asynchronous events if they have an active session i.e. they have logged into the Resident Portal.
- **View profile details of the logged-in user (name, photo, and last login details):** The Resident will be able to view the name, and photo of the logged-in user. They will also be able to see the last login details of the Resident.
- **Responsive UI support:** Support for the application to work seamlessly on various resolutions.

Below is an image summarizing the features provided in the Resident portal.

The relationship of Resident services with other services is listed below.

- ⓘ **Note:** The numbers do not signify the sequence of operations or the control flow.



1. **Audit Manager:** Resident services sends all the audit logs to the Audit Manager.
2. **Digital card service:** Resident services use this service to download the PDF of the UIN card or VID card.
3. **Credential Request Generator Service:** This service is used to share the credentials with various partners like print partners, authentication partners, and digital card partners.
4. **ID Repository Identity Service:** Resident services use this service to retrieve the identity information of a credential and to lock/unlock authentication types.
5. **ID Repository VID service:** This service is used to generate/revoke various types of VIDs.
6. **ID Authentication:** This service is used by Resident services to authenticate users.
7. **MOSIP e-Signet:** This is used to authenticate and authorize the users in the event of login using UIN/ VID.
8. **Resident UI:** This is the interface through which users can interact with the Resident Services.

9. **WebSub:** This is used to get asynchronous notifications from IDA for acknowledgment purposes.
10. **Registration Processor:** This is used to sync and upload packets for features about changes in identity data.
11. **Packet Manager:** Resident services use this service to create packets.
12. **Partner Management Service:** Resident services use this service to get information about various partners and policies.
13. **Keycloak:** Resident services use this to authenticate to access the MOSIP internal APIs. The Resident Services communicates with endpoints of other MOSIP modules via a token obtained from Keycloak.
14. **Master data service:** Resident services invoke the Master Data services to get various templates and machine details.
15. **Notification service:** Resident services use this service to send various notifications through email or SMS.
16. **Key Manager:** Resident services use Key Manager to encrypt or decrypt the data used across features.

Design Principles

The design of the Resident portal embodies the following principles:

- **One-stop solution:** The Resident portal is designed to have components that aim to solve all the queries, issues, or discrepancies of the residents and act as a one-stop solution for all the requirements.
- **Self-Sovereign:** Once the ID is issued by an authority, the user/resident/citizen chooses to control and manage their data in their choice of devices.
- **Inclusive:** The Resident portal aims to be available in all browsers while also catering to the needs of visually impaired, dyslexic, and color-blind folks.
- **Presence assurance:** This web-based UI application would put in all its efforts to ensure easy access to all the residents with high availability.
- **Works Remote:** The Resident portal should be able to share credentials when data needs to be shared remotely without physical presence.
- **Trusted:** The identity verification process on the device should be trusted so that it can be used in service delivery without any concerns.

- **Grievance redressal:** The Resident portal ensures that in case of any concerns or grievances, the issue is raised and resolved through the portal itself.

Documentation

- [Release Notes](#)
- [Resident Services Developers Guide](#)
- [Resident Services UI Developers Guide](#)
- [Resident Portal Configuration Guide](#)
- [Resident Services Deployment Guide](#)
- [Configuring Resident OIDC Client](#)
- [Resident Portal User Guide](#)
- [Functional Overview](#)

Develop

Developers Guide

Overview

[Resident Services](#) are the self-services used by residents themselves via a portal.

The Resident Portal is a web-based UI application that provides residents of a country with services related to their Unique Identification Number (UIN).

The documentation here will guide you through the prerequisites required for the developer's setup.

Software setup

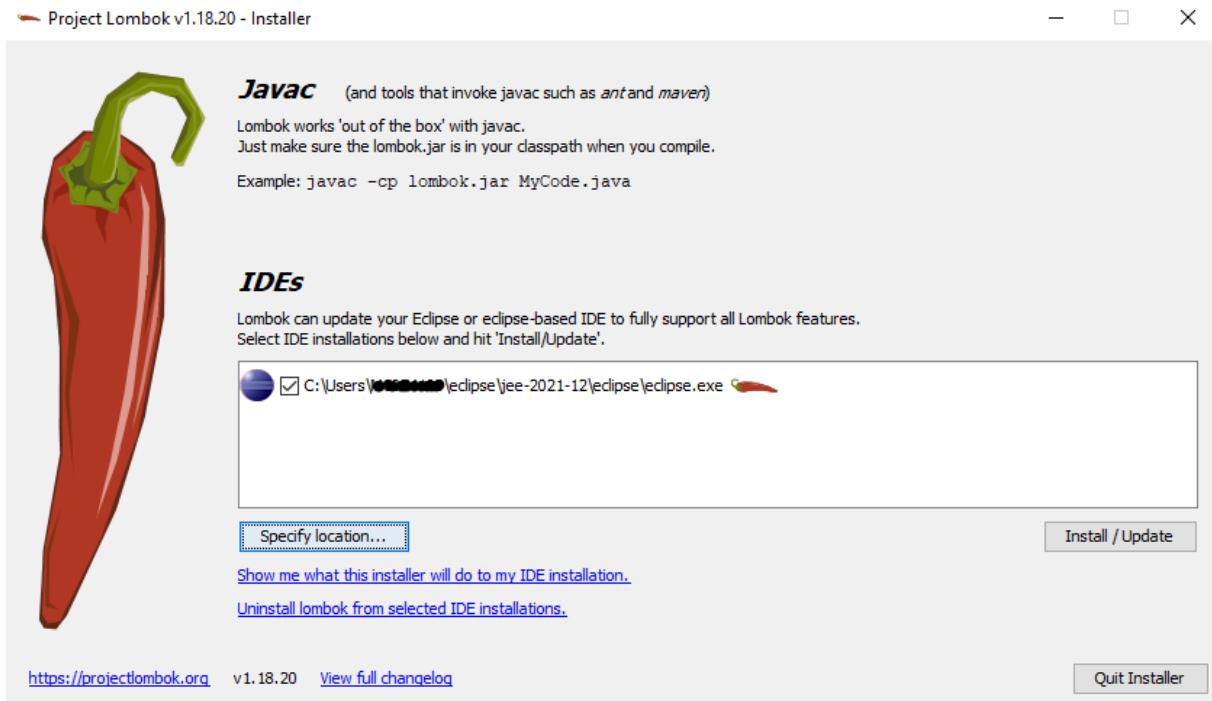
Below is a list of tools required in Resident Services:

1. JDK 11
2. Any IDE (like Eclipse, IntelliJ IDEA)
3. Apache Maven (zip folder)
4. pgAdmin
5. Postman
6. Git
7. Notepad++ (optional)
8. lombok.jar (file)
9. settings.xml (document)

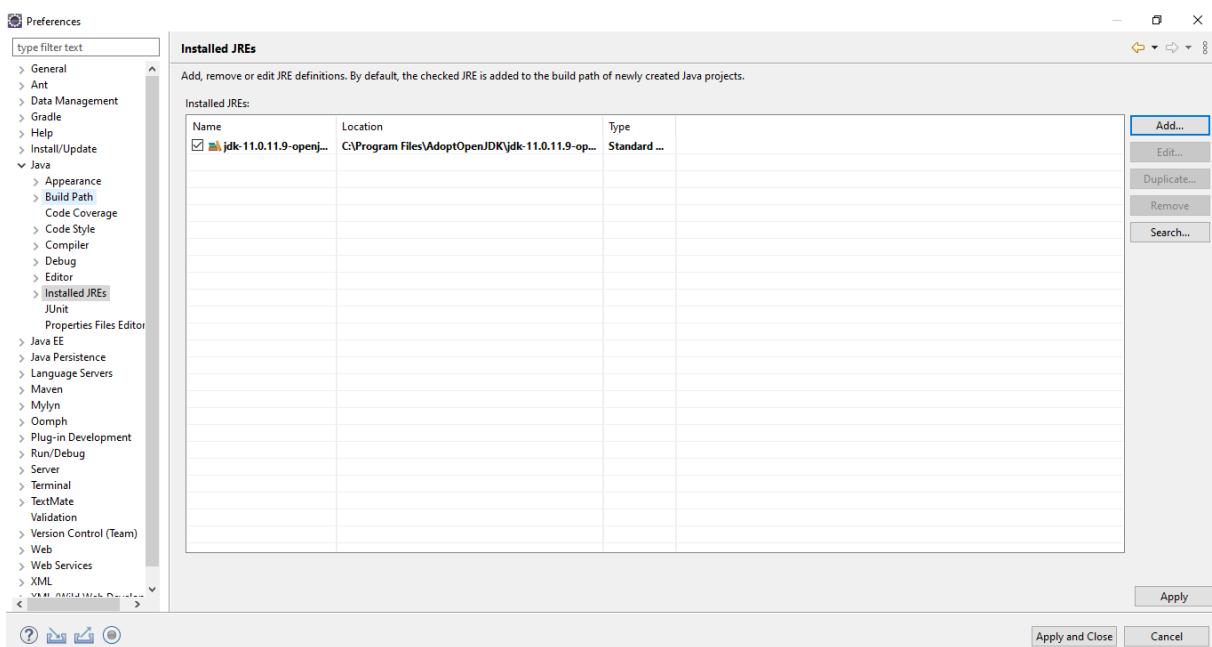
Follow the steps below to set up Resident Services on your local system:

1. Download `lombok.jar` and `settings.xml` from [here](#).
2. Install Apache Maven.
3. Copy the `settings.xml` to ".m2" folder `C:\Users\<username>\.m2`.
4. Install Eclipse.

5. Open the `lombok.jar` file and wait for some time until it completes the scan for Eclipse IDE and then click `Install/Update`. Specify the eclipse installation location if required by clicking the 'Specify location...' button. Then, click `Install/Update` the button to proceed.



6. Check the Eclipse installation folder `C:\Users\userName\eclipse\jee-2021-12\eclipse` to see if `lombok.jar` is added. By doing this, you will not have to add the dependency of `lombok` in your `pom.xml` file separately as it is auto-configured by Eclipse.
7. Configure the JDK (Standard VM) with your Eclipse by traversing through `Preferences → Java → Installed JREs`.

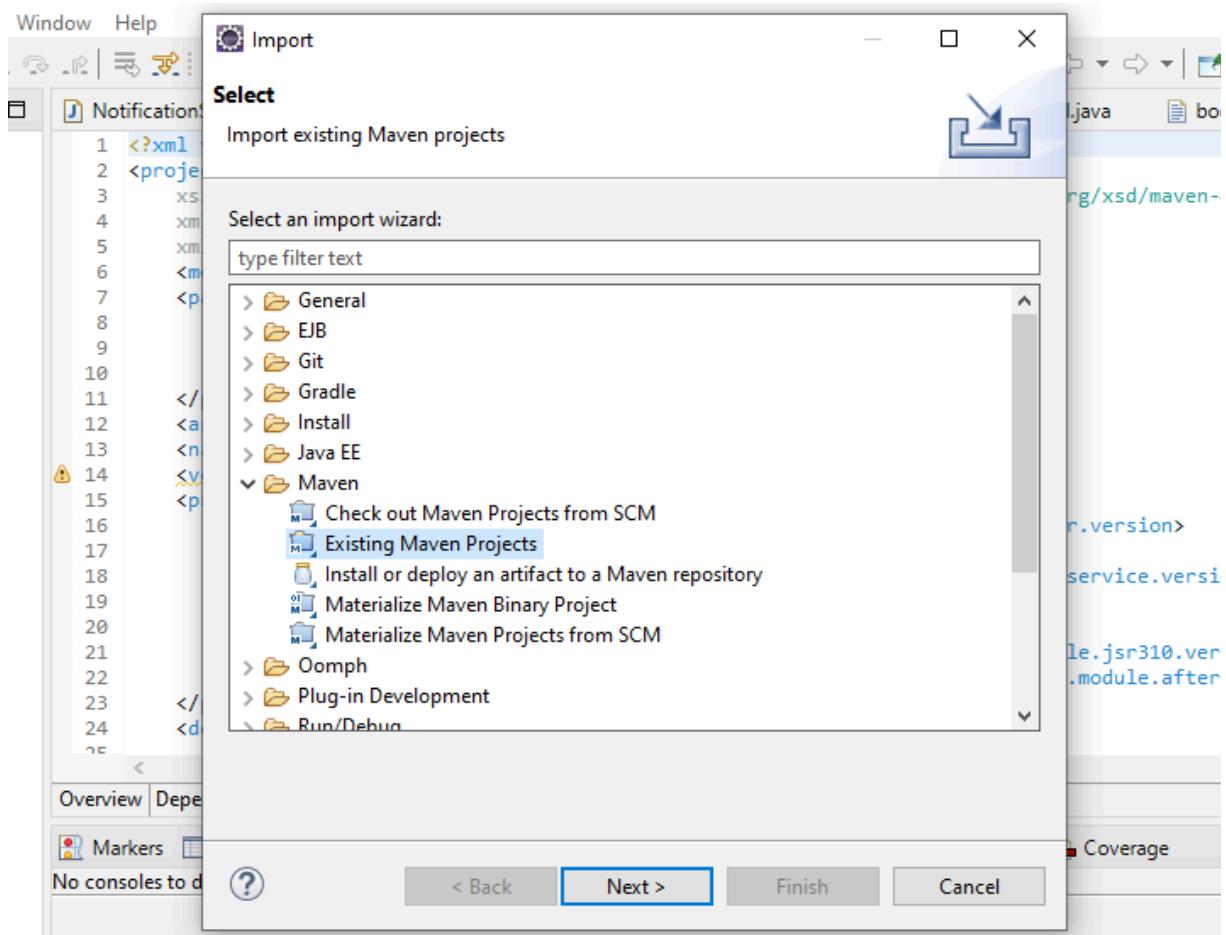


Code setup

For the code setup, clone the repository and follow the guidelines mentioned in the [Code Contributions](#).

Importing and building a project

1. Open the project folder where `pom.xml` is present.
2. Open the command prompt from the same folder.
3. Run the command `mvn clean install -Dgpg.skip=true -DskipTests=true` to build the project and wait for the build to complete successfully.
4. After building a project, open Eclipse and select `Import Projects → Maven → Existing Maven Projects → Next → Browse to project directory → Finish`.



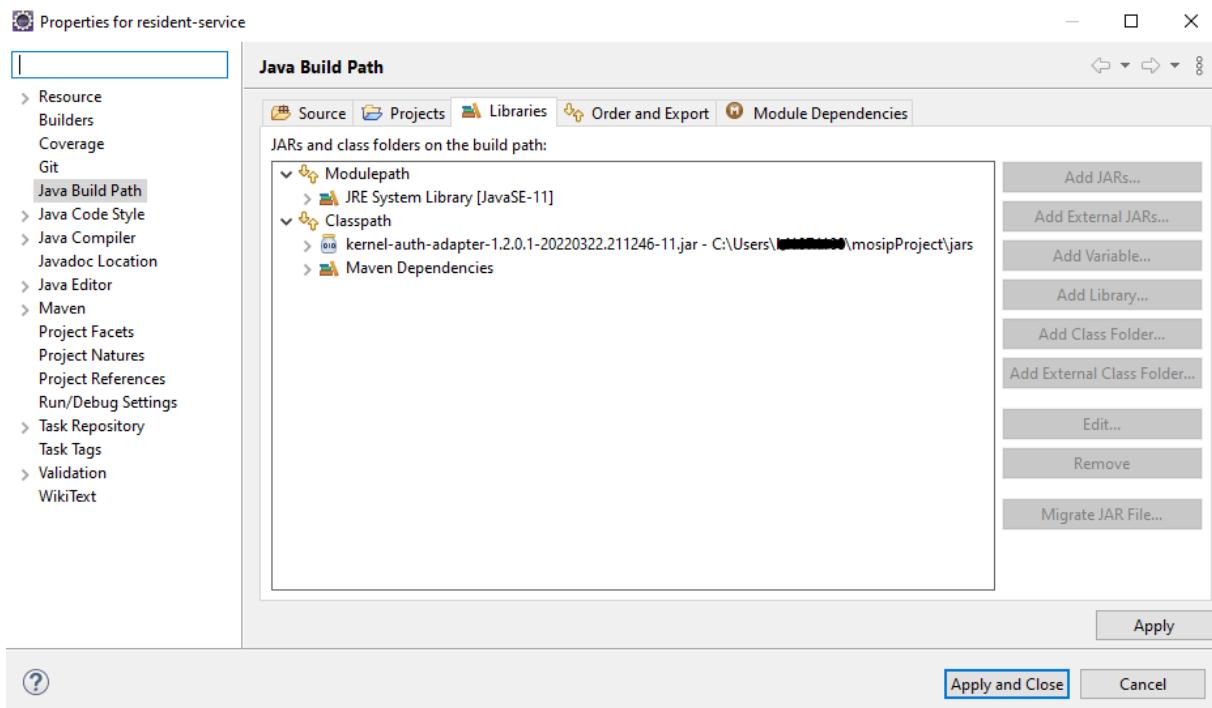
5. After successful importing of the project, update the project by right-clicking on `Project → Maven → Update Project`.

Environment setup

1. For the environment setup, you need an external JAR that is available [here](#) with different versions. Download the below-mentioned JARs with appropriate latest/appropriate versions. You will need to input the appropriate artifact ID and version and other inputs.
 - a. `icu4j.jar`
 - b. `kernel-auth-adapter.jar`
 - c. `kernel-ref-idobjectvalidator.jar`
 - d. `kernel-transliteration-icu4j.jar`

E.g.: You can download `kernel-auth-adapter.jar` and add to the project

`Libraries → Classpath → Add External JARs → Select Downloaded JAR → Add → Apply and Close).`



2. Clone [mosip-config repository](#).

- a. As Resident Services is using two properties files- `resident-default.properties` and `application-default.properties`. But for the local running of the application, you need to provide additional/overriding properties such as secrets, passwords, and properties passed by the environment which can be added to new files `application-dev-default.properties` (common

properties for all modules) and `resident-dev-default.properties` (Resident service-specific properties).

b. You will have to create both the property files according to your environment and put them in `mosip-config folder` (cloned). The same files are available below for reference.

These two files are loaded by the application by specifying the application names in the Application VM arguments like-

```
Dspring.cloud.config.name=application,resident,application-dev,
resident-dev
```

(also detailed in a later section).

3. To run the server, two files are required- `kernel-config-server.jar` and `config-server-start.bat`.

4. Put both files in the same folder and point to the property-

```
Dspring.cloud.config.server.native.search-locations to mosip-config
folder in config-server-start.bat file and also check the version of kernel-
config-server.jar towards the end of the command.
```

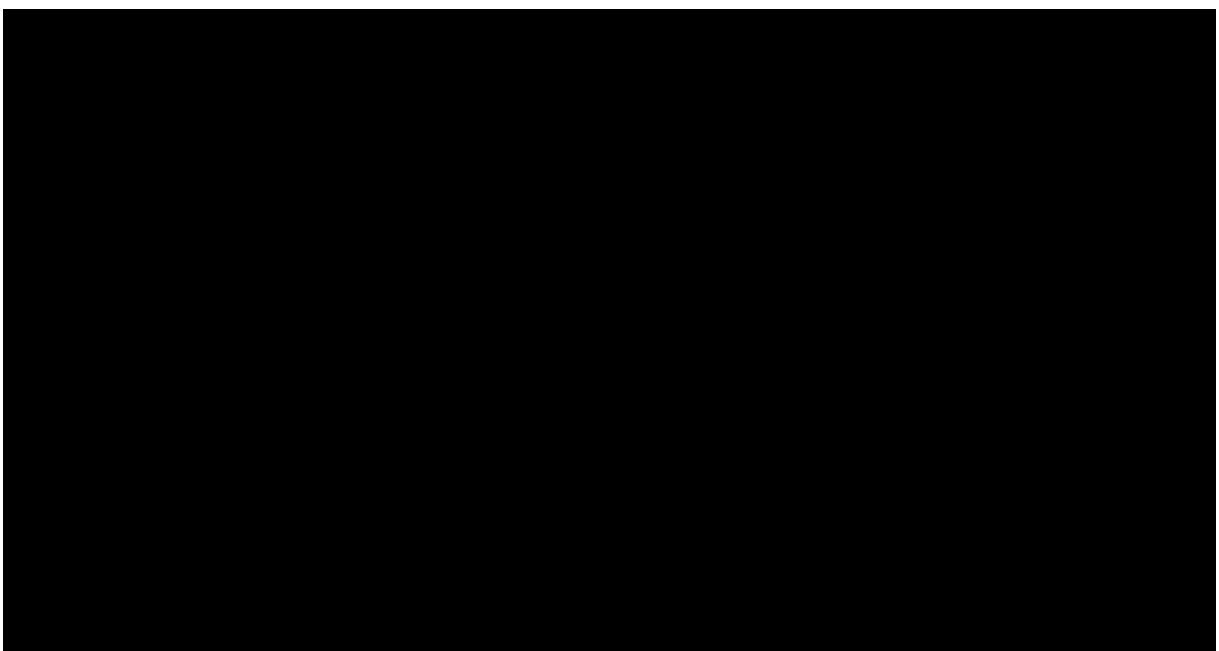
Example:

```
java -jar -Dspring.profiles.active=native -
Dspring.cloud.config.server.native.search-
locations=file:C:\Users\myDell\mosipProject\mosip-config -
Dspring.cloud.config.server.accept-empty=true -
Dspring.cloud.config.server.git.force-pull=false -
Dspring.cloud.config.server.git.cloneOnStart=false -
Dspring.cloud.config.server.git.refreshRate=0 kernel-config-server-1.2
```

- As mentioned earlier, you will have to create property files according to your environment like `resident-env-default` and `application-env-default` (here env represents environment name). Both files will contain different configurations such as `resident-env-default` will have config properties (e.g., secrets, passcodes, etc) used for the resident-services module only and `application-env-default` is used for environment-specific changes and can be used for other modules as well.
- In this example, currently, these two files are created for the dev environment and hence the files have suffixes of `-dev`. If you want to run it for a different environment such as qa, create these two files with `-qa` suffixes, and then you will also need to provide the appropriate VM argument for that referring to qa environment.

For instance,

- Add `mosip.resident.client.secret=*****` property to be able to use a decrypted passcode and run it on your local machine.
 - If you check the URLs present in `application-default` file, they are set to module-specific URLs, but you need to use internal/external environment URLs to access the APIs by using an `application-dev-default` file.
 - In `application-dev-default` file, assign environment domain URL to `mosipbox.public.url`, and change all other URLs with `${mosipbox.public.url}`.
 - It results in `mosipbox.public.url=internal/externalAPI` (e.g., `mosipbox.public.url=https://api-internal.dev.mosip.net`) and it will connect with the Development environment.
5. Run the server by opening the `config-server-start.bat` file.



Configurations to be done in Eclipse

1. Open Eclipse and run the project for one time as a Java application, so that it will create a Java application which you can see in debug configurations, and then change its name. (e.g.: project name with the environment - "Resident-dev").

2. Open the Arguments tab and specify Application VM arguments: For example, for a development environment:

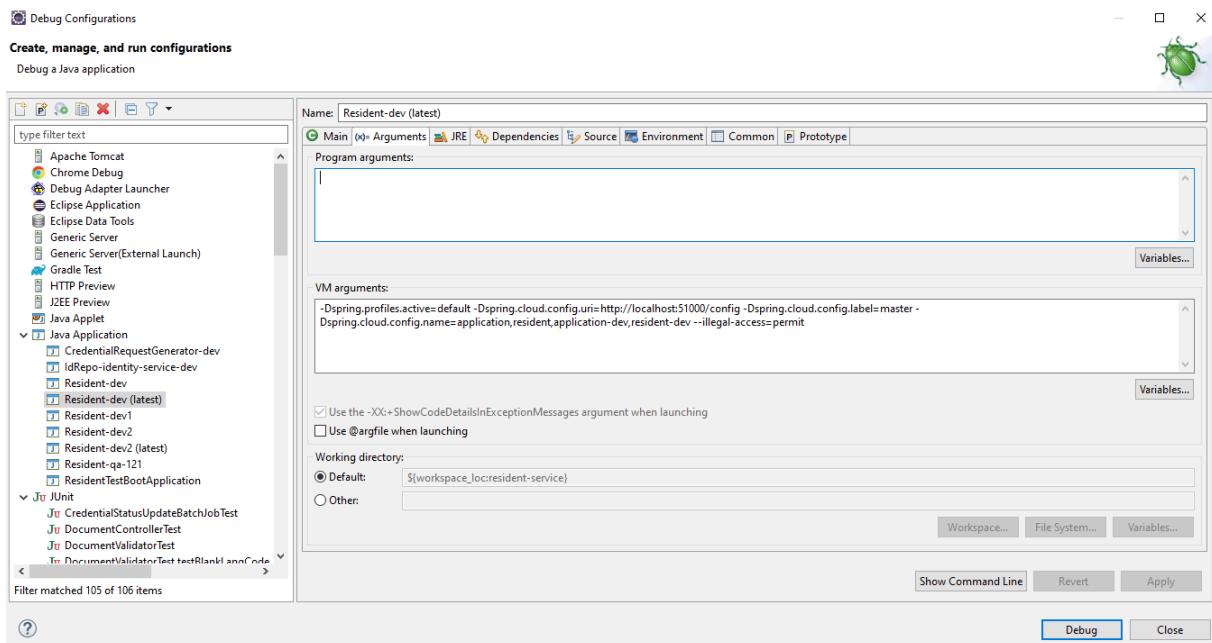
```
-Dspring.profiles.active=default -  
Dspring.cloud.config.uri=http://localhost:51000/config -  
Dspring.cloud.config.label=master -Dsubdomain=dev -  
Dspring.cloud.config.name=application,resident,application-dev,reside
```

Save this run configuration as 'Resident-dev'.

For `qa` environment, you can create `Resident-qa` run configuration with VM argument as below.

Example:

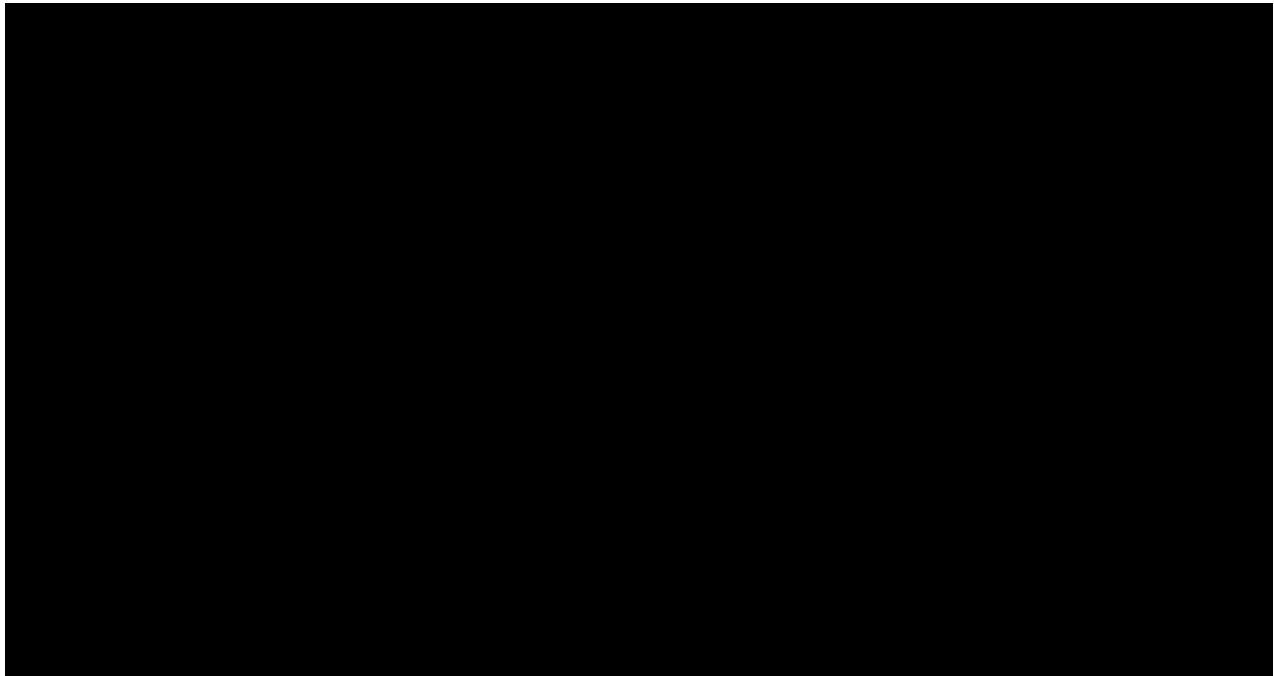
```
-Dspring.profiles.active=default -  
Dspring.cloud.config.uri=http://localhost:51000/config -  
Dspring.cloud.config.label=master -Dsubdomain=qa -  
Dspring.cloud.config.name=application,resident,application-qa,resident
```



- Click **Apply** and then debug it (starts running). In the console, you can see a message like `Started ResidentBootApplication in 34.078 seconds (JVM running for 38.361)`.

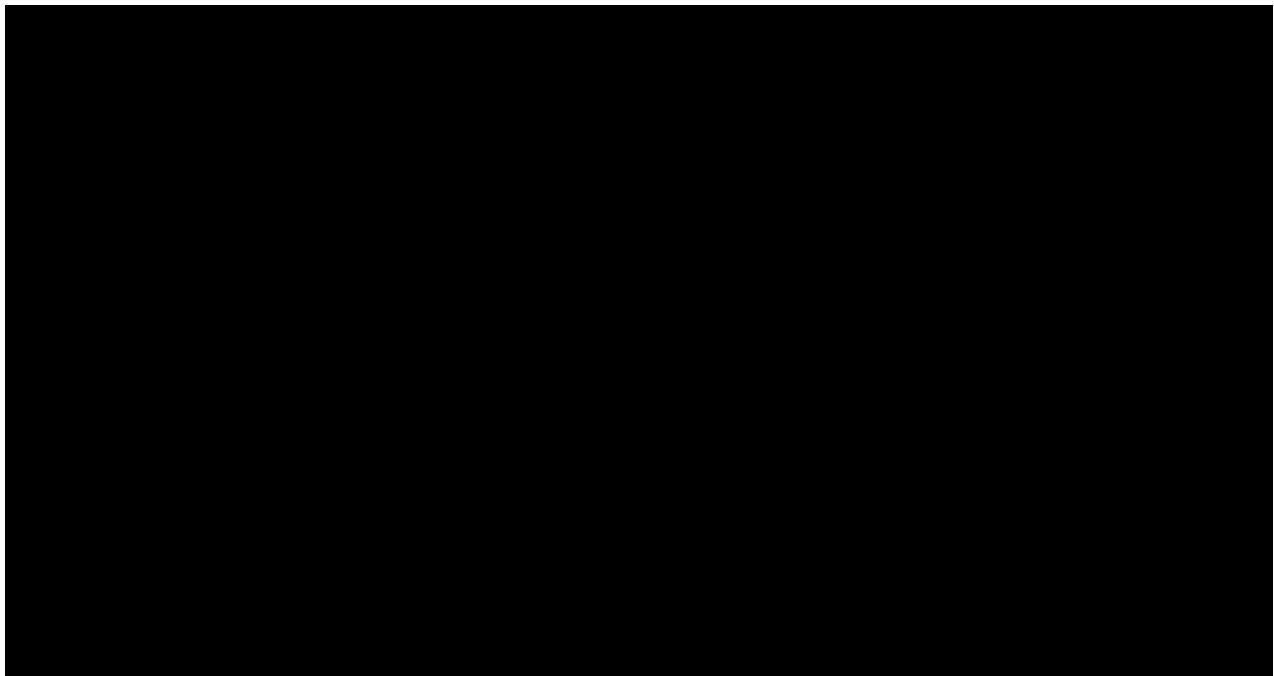
Resident services API

- For API documentation, refer [here](#).
- The APIs can be tested with the help of Postman or Swagger-UI.
- Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster. It is a widely used tool for API testing. Below you will find the APIs postman collection of resident-services.
- Swagger is an interface description language for describing restful APIs expressed using JSON. You can access Swagger-UI of resident-services for the dev-environment from <https://api-internal.dev.mosip.net/resident/v1/swagger-ui.html> and localhost from <http://localhost:8099/resident/v1/swagger-ui.html>.
- Download the JSON collection available below and import it to your postman. [Resident-Service-APIs.postman_collection-latest.json](#).



- Create an environment as shown in the image below.

This environment is created for dev. Give the variable name as `url` and set both values as `https://api-internal.dev.mosip.net`.



- Similarly, create another environment as shown below.

This environment is created for localhost. Give the variable name as `url` and set both values as `http://localhost:8099`.



UI Developers Guide

This [repository](#) contains the UI code for the Resident portal. To know more about the features and functions present on the portal, refer [here](#).

Build and Deployment

 The code is written in Angular JS.

- Install `node.js` - To build the angular code using angular cli that runs on node, recommended Node: 14.17.3, Package Manager: npm 6.14.13
- Install `angular cli` – To install angular cli for building the code into deployable artifacts. Follow the following steps to install angular cli on your system.
 - `npm install -g @angular/cli` (to install angular cli)
 - `ng --version` (to verify angular is installed in the system) We recommend Angular CLI: 7.2.1
- Check out the source code from GIT – To download the source code from git, follow the steps below to download the source code on your local system.
 - `git clone https://github.com/mosip/resident-ui.git` (to clone the source code repository from git)

For Production build

- **Build the code**

Follow the steps below to build the source code on your system.

- Navigate to the resident-ui directory inside the cloned repository.
- Run the command `ng build "--prod" "--base-href" "." "--output-path=dist"` in that directory to build the code.

- **Build Docker image**

Follow the steps below to build the docker image on your system.

- `docker build -t name .` (replace `name` with the name of the image you want, `"."` signifies the current directory from where the docker file has to be read.)
 - Example: `docker build -t residentui .`
- **Run the Docker image**

Follow the steps to build a docker image on your system.

 - `docker run -d -p 80:80 --name container-name image-name` (to run the docker image created with the previous step, `-d` signifies to run the container in detached mode, `-p` signifies the port mapping left side of the `:` is the external port that will be exposed to the outside world and the right side is the internal port of the container that is mapped with the external port. Replace `container-name` with the name of your choice for the container, replace `image-name` with the name of the image specified in the previous step)
 - Example: `docker run -d -p 8080:8080 --name nginx residentui`
- Now you can access the user interface over the internet via a browser.
 - Example: `http://localhost:8080/#/dashboard`

For Local build

- Build & deploy the code locally

Follow the steps below to build the source code on your system.

- Navigate to the `resident-ui` directory inside the cloned repository. Then, run the following command in that directory:
 - `npm install`
 - `ng serve`
- Now, you can access the user interface via the browser.
 - Example: `http://localhost:4200`

UI Specifications

Overview

UI specs of resident module are used to configure the form fields across Resident Portal. UI specs are saved as a JSON file with a list of fields. Each field has a set of attributes/ properties that can be configured which affects the look and feel along with the functionality of the field.

Below is the list of all the properties available for each field in the Resident Portal UI specs:

Property Name	Details	Sample Value
<code>id</code>	The id property is the unique id provided to a field to uniquely identify it. The id can be alpha-numeric without any spaces between them.	"id": "zone"
<code>description</code>	This is a non-mandatory property used to describe the field.	"description": "zone"
<code>labelName</code>	This property defines label name for the field. This property has sub-attributes as the language code (eng, fra, ara) to store data in different languages based on the country's configuration.	"labelName": { "eng": "Zone", "ara": "منطقة", "fra": "Zone"}
	This property defines the kind of UI component to be used to capture data in UI. Currently the values that can be used are: • textbox (creates multiple textboxes for each field to capture input in all the	

<code>controlType</code>	languages configured for the setup) • dropdown • fileupload • date (creates a date picker) • ageDate (creates a date picker along with number toggle to provide age directly) • checkbox (creates a toggle checkbox for the field which can be checked or unchecked) • button (creates dropdown options as buttons, which user can select easily)
<code>inputRequired</code>	This property decides if the field is to be displayed in the UI form or not. It is useful for some internal fields which do not need any input from the user.
<code>required</code>	This is a mandatory property which decides if the field is a required form field or not. If true, user must provide some value for the field.
<code>type</code>	This property defines the data type of the value corresponding to this field. The data types supported are "number", "string" and "simpleType". The type "simpleType" means that language specific value will be saved along with the language code.
	This property is relevant when control type is "dropdown" or "button". It defines if the field is of type "default" or "dynamic".

<code>fieldType</code>	<p>
If it is "dynamic" then all the options for the dropdown are populated from the "master.dynamic_field" table otherwise they are populated from corresponding table example "master.gender"</p>
<code>subType</code>	<p>This is relevant for 2 cases:</p> <ul style="list-style-type: none">
1. When control type is "dropdown"/ "button" and the type is "dynamic" then "subtype" can be used to populate the options for the field with the data available in "master.dynamic_field" table.
2. When the control type is "fileupload", then the property "subtype" is used to map the field to a "code" in the "master.doc_category" table.
	<ul style="list-style-type: none"> • This property enables us to add the list of language specific validators for the field. <p>
* Each validator can have the below fields,
"langCode",
"type",
"validator",
"arguments",
"errorMessageCode"

* The "type" defines the validation engine type.</p> <p>
* The "validator" gives the pattern/methodName/scriptName/expression
* The "arguments" array to is used to hold parameter</p> <pre data-bbox="1023 1897 1341 2055">
"validators": [{
"langCode": "eng",
"type": "regex",
"validator": "^(?=.</pre>

validators

or dependent field ids required for validation
* The "errorMessageCode" can be given to add custom error message which will be shown to the user when the validation fails. The error message corresponding to this code will be picked from language specific i18n translation files. In case "errorMessageCode" is not given then generic error message will be displayed to the user when validation fails.

Currently, regex is supported by MOSIP.
If "langCode" is not added then same "validator" is used for all languages of the field.

```
{0,50}".$",
<br>"arguments": [],
<br>"errorMessageCode":
"UI_1000"<br>},{

<br>"langCode": "ara",
<br>"type": "regex",
<br>"validator": "^[A-Z]+$",
<br>"arguments": []<br>},{

<br>"langCode": "fra",
<br>"type": "regex",
<br>"validator": "^[A-Z]+$",
<br>"arguments": []<br>}]
```

locationHierarchyLevel

This attribute is mandatory for the location dropdown fields.
The value will be as per corresponding location hierarchy level from the master.loc_hierarchy_list table.

```
{<br>"id":"region",
<br>"controlType":"dropdown",
<br>"fieldType":"default",
<br>"type":"simpleType",
<br>"parentLocCode":"MOR",
<br>"locationHierarchyLevel":1<br>..}
```

This attribute is to be used only for location dropdown fields and it is optional.
The corresponding location dropdown will be pre populated in UI based on the value in

`parentLocCode`

"parentLocCode".
If this attribute is NOT mentioned in UI specs, then the dropdown will be populated based on selection in its parent dropdown, as before.
For the first dropdown, in case this attribute is not mentioned in UI specs then the value from "mosip.country.code" configuration will be used for backward compatibility.

```
{<br>"id":"region",
<br>"controlType":"dropdown",
<br>"fieldType":"default",
<br>"type":"simpleType",
<br>"parentLocCode":"MO
R",
<br>"locationHierarchyLeve
l":1<br>..}
```

`alignmentGroup`

- * This property is used to group the fields on the screen.
* If it is skipped, then all the fields will appear in same sequence (horizontally layout) as they appear in UI specs.
* If you want the first and fifth field to be in same row in the screen, you can add this attribute with same group name.
* The UI is responsive so it will accommodate as many fields in one row as they will fit comfortably.

`containerStyle`

This is used to optionally apply some CSS styles to the UI field container.

```
"containerStyle":
{<br>"width": "600px",
<br>"margin-right": "10px"
<br>}
```

Technology Stack

UI & Rest end points:

The table below outlines the frameworks, tools, and technologies employed by Resident Portal.

Tool / Technology	Version	Description	License
Angular JS	7.2.1	AngularJS is a toolset for building the framework most suited to your application development. It is fully extensible and works well with other libraries.	MIT License
Node JS	16.2.0	<i>Node.js</i> is a JavaScript runtime built on Chrome's V8 JavaScript engine.	MIT License

Resident Services:

The table below outlines the frameworks, tools, and technologies employed by Resident Services.

Tool / Technology	Version	Description	License
Java SE 11	OpenJDK 11	Language Runtime in Docker Image	GNU General Public License, version 2, with the Classpath Exception
Ubuntu Server	20.04	Docker base image Operating System	Free
Spring	5	Application Framework	Apache License 2.0

Apache commons	Version compatible with Spring 5	Utilities	Apache License 2.0
Hibernate	5.2.17.Final	ORM	Apache Software License 2.0
Hibernate validator	6.0.12.Final	validator	Apache Software License 2.0
mvel2	2.4.7.Final	Expression language	Apache License 2.0
Jackson	2.9.x	JSON marshal/unmarshal	Apache Software License 2.0
Junit	4.x and above	Unit Testing	Common Public License - v 1.0
mockito	2.22.0	Junit - Mock Objects	MIT
power-mock	2.0.7	Junit - Mock Static Classes	Apache Software License 2.0
logback	1.2.3	Log	GNU Lesser GPL Version 2.1
velocity	1.7	Templating	Apache Software License 2.0
Swagger	Open API - 3	API Documentation	Apache Software License 2.0
Tomcat server	8	Application Server	Apache Software License 2.0
PostgreSQL	Server: 10	Database	Postgres License BSD 2-clause "Simplified License"
Sonar	7.2	Code quality Checking	Open Source License
Micrometer Prometheus	1.4.2	Metrics	Apache Software License 2.0

gson	2.8.4	JSON parser	Apache Software License 2.0
h2 database	1.4.197	JUnit Test DB	EPL 1.0, MPL 2.0
lombok	1.18.8	Development - reduce the boilerplate code	MIT
IText PDF	5.5.13.3	PDF Generation	AGPL 3.0

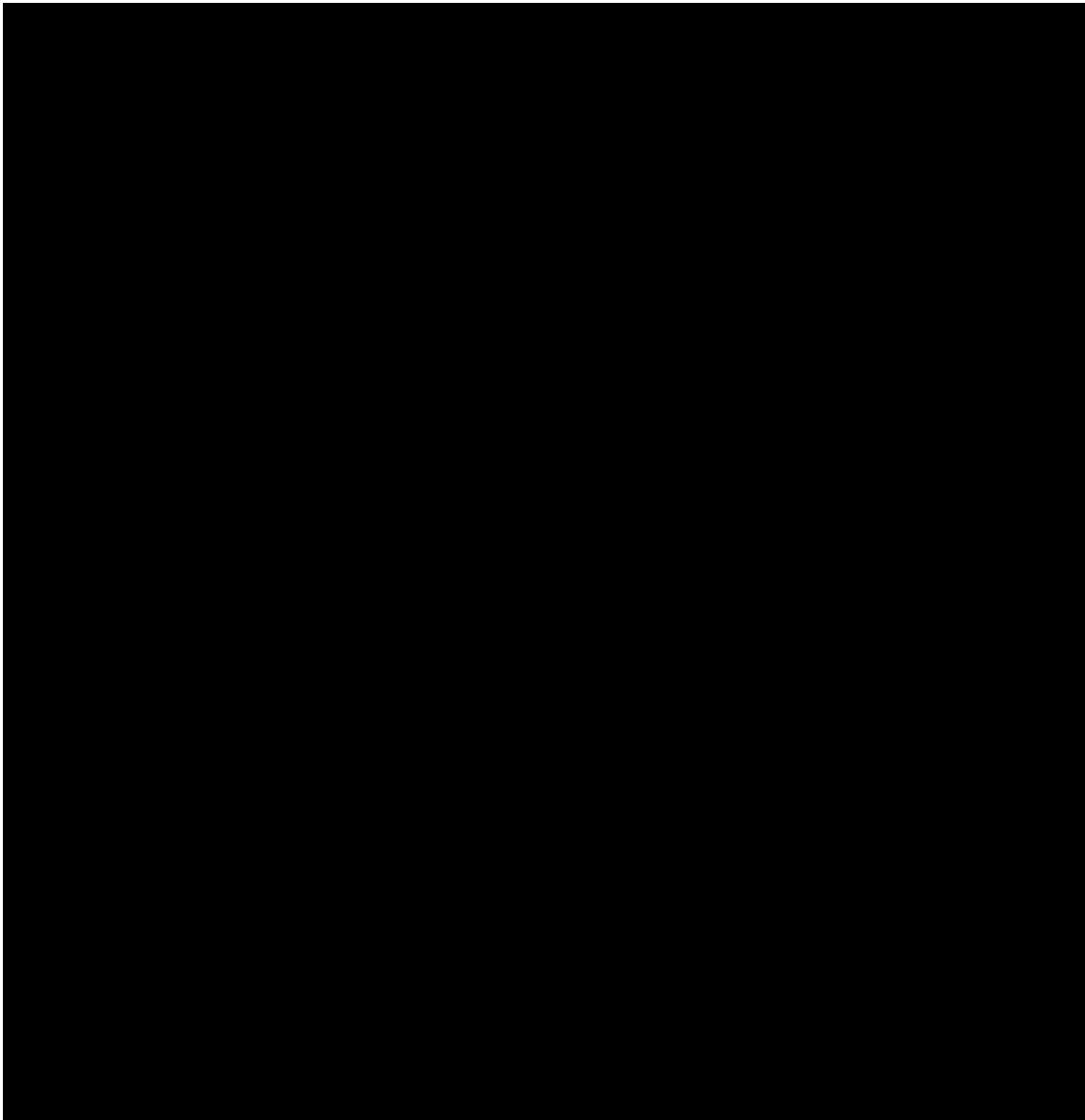
Test

Functional Overview

Resident Portal is a self-help web-based portal that can be used by the residents of a country to avail of the services related to their Unique Identification Number (UIN). The architecture, interface overview, and key services provided are discussed below:

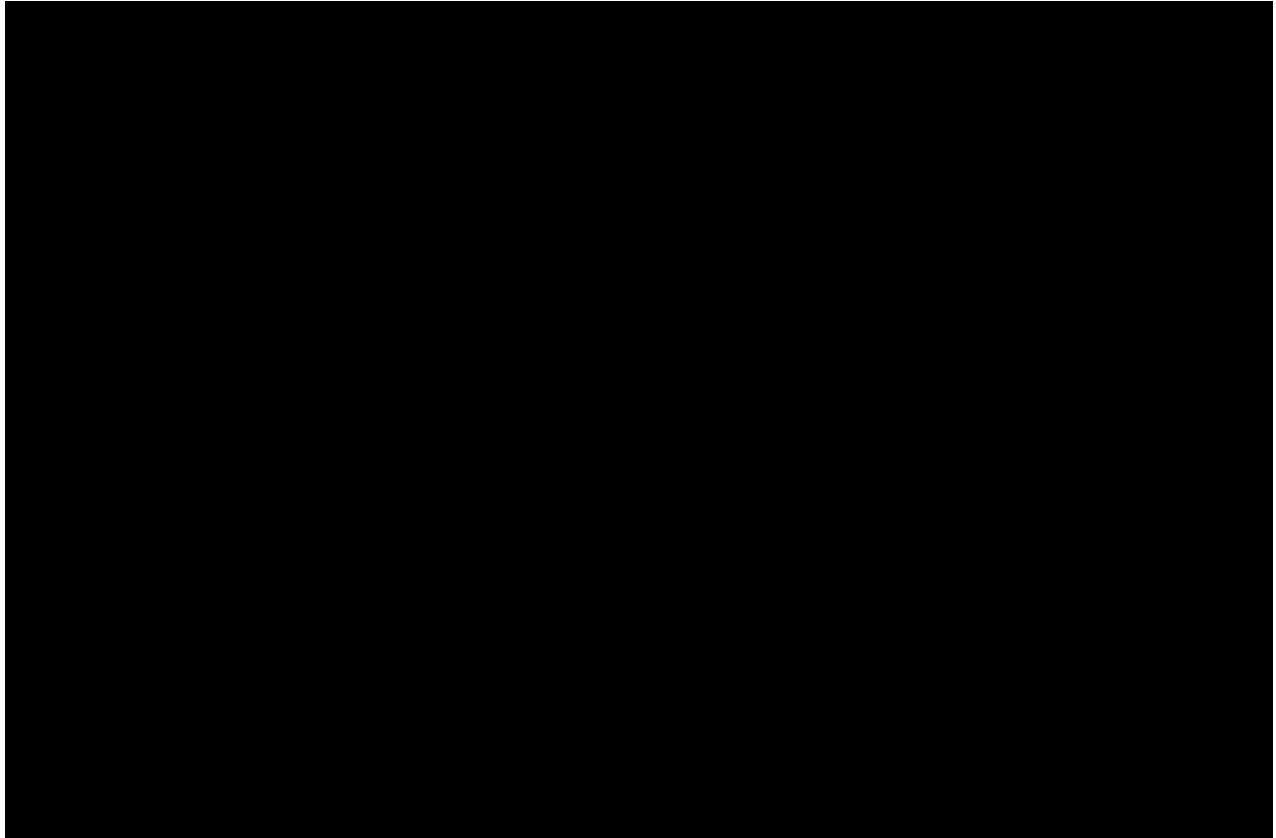
1. Architecture
2. Interface overview
3. Key Services
 - a. UIN services
 - i. View My History
 - ii. Secure My ID
 - iii. Manage My VID
 - iv. Track My Request
 - v. Download My Personalized Card
 - vi. Update My Data
 - vii. Share My Credential
 - b. Get Information
 - i. Supporting Document
 - ii. Registration Center
 - c. Verify Email ID/Phone number
 - d. Get My UIN
 - e. Booking an Appointment

Below is a detailed explanation of each of the features along with the list of relevant APIs.



Modules/services that Resident services depends on

Architecture



Interface Overview

Menu Bar

The Resident Portal menu bar contains the following:

1. **Font Size**- Residents can alter the size of the font based on their preferences.
2. **Language**- Residents can select the language of preference.
3. **Bell icon Notification**- Residents can view the notifications of all the asynchronous events in chronological order.

Profile Icon:

1. **Profile Icon**- Residents can view the following:
 - a. Name of the logged in user
 - b. Photo of the logged in user

- c. Last login details
- d. Logout option

Workspace

A dashboard view to quickly locate the 'Key Services'

Key Services

UIN Services

View My History

The residents can view the history of all the transactions associated with their logged-in UIN/ AID/ VID. They can also view their details and if any unaccounted entry is found, a report can be raised against the same.



Secure My ID

On clicking "Secure My ID", the residents can view the status of all the authentication types. They can choose to lock or unlock authentication types like the following:

1. Email OTP authentication
 2. Phone OTP authentication
 3. Demographic authentication
 4. Fingerprint authentication
 5. Iris authentication
 6. Face authentication
- **Fetch Authentication Types - Lock status of the individual**



- **Applies the Authentication Types Lock/Unlock request in IDA**

Manage My VID

On clicking "Manage My VID", the resident will be taken to a page where they can view details of the existing VIDs, generate new VID, revoke existing VIDs, or download a VID card.

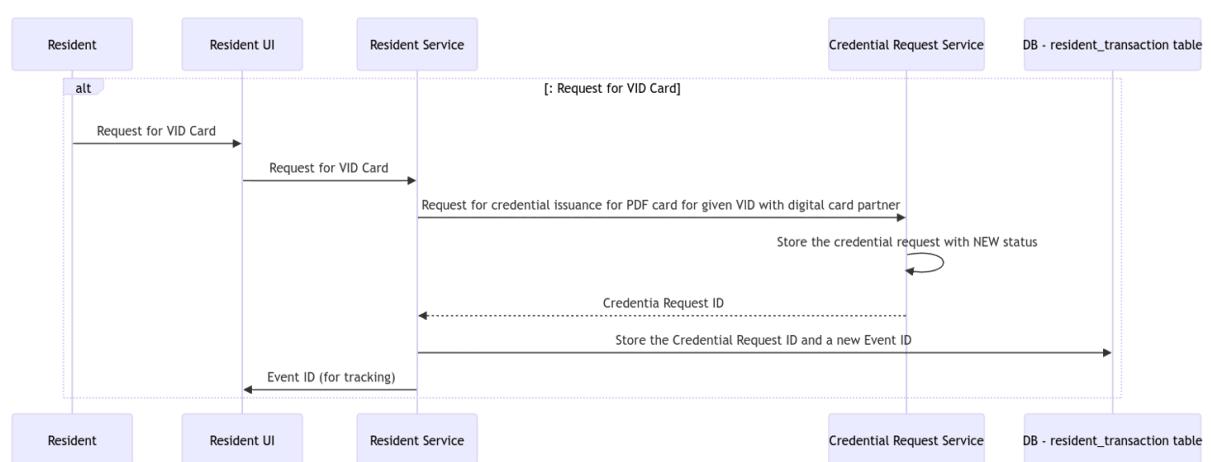
The following types of VIDs can be seen based on the VID policy:

1. Perpetual VID
 2. Temporary VID
 3. One-time VID
- **Fetch Active VIDs of the Individual**

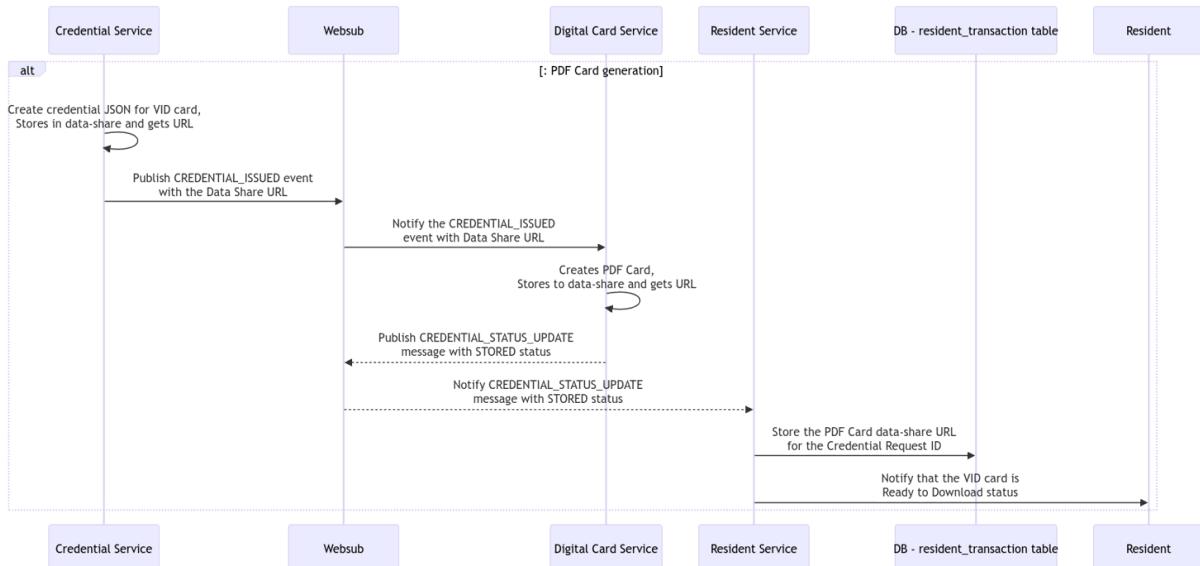


- **Revoke the VID of the Individual**

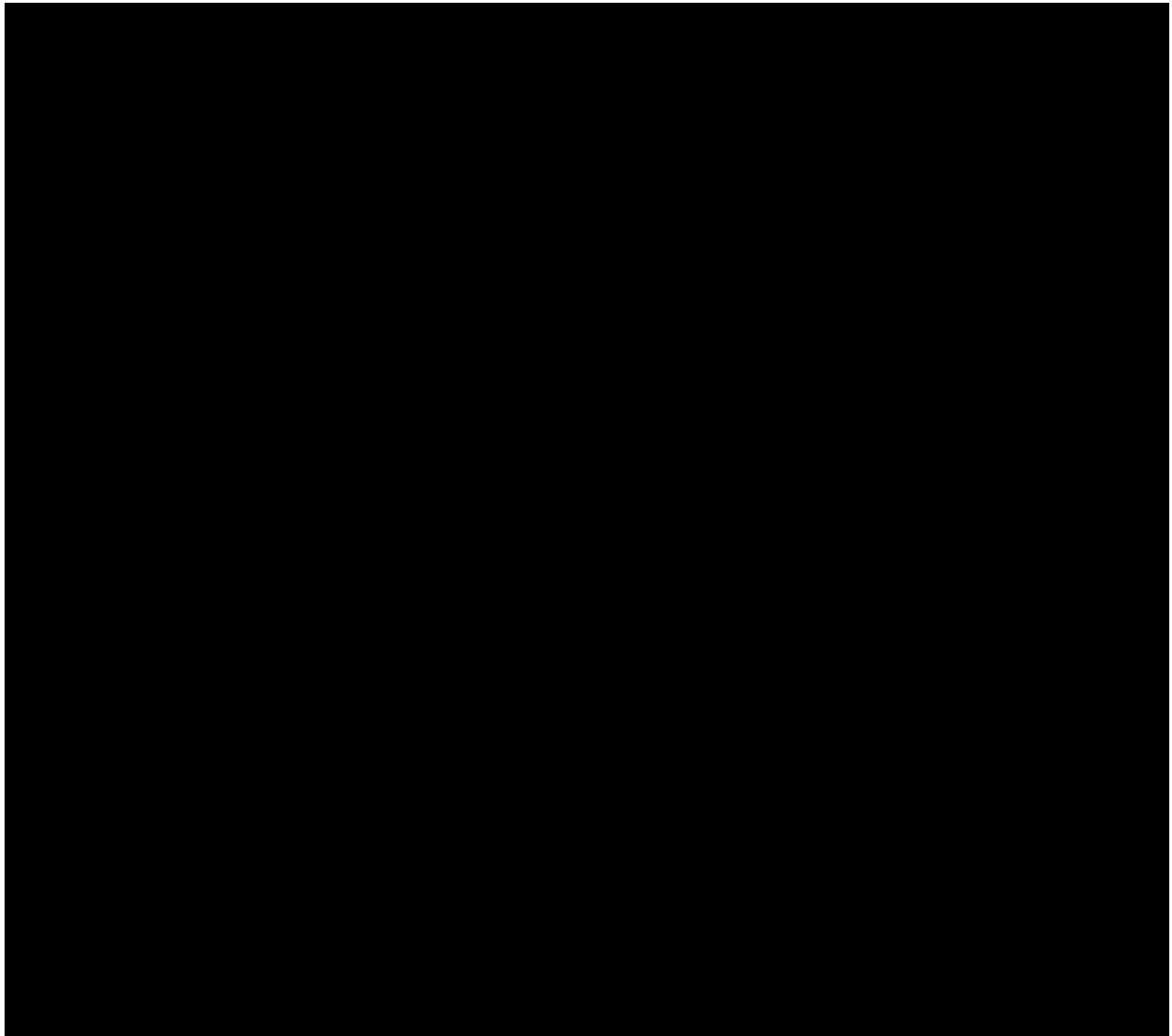
- **Store the Credential Request ID and a new Event ID**



- **Notify that the VID card is Ready to Download status.**



- **Download PDF Card**



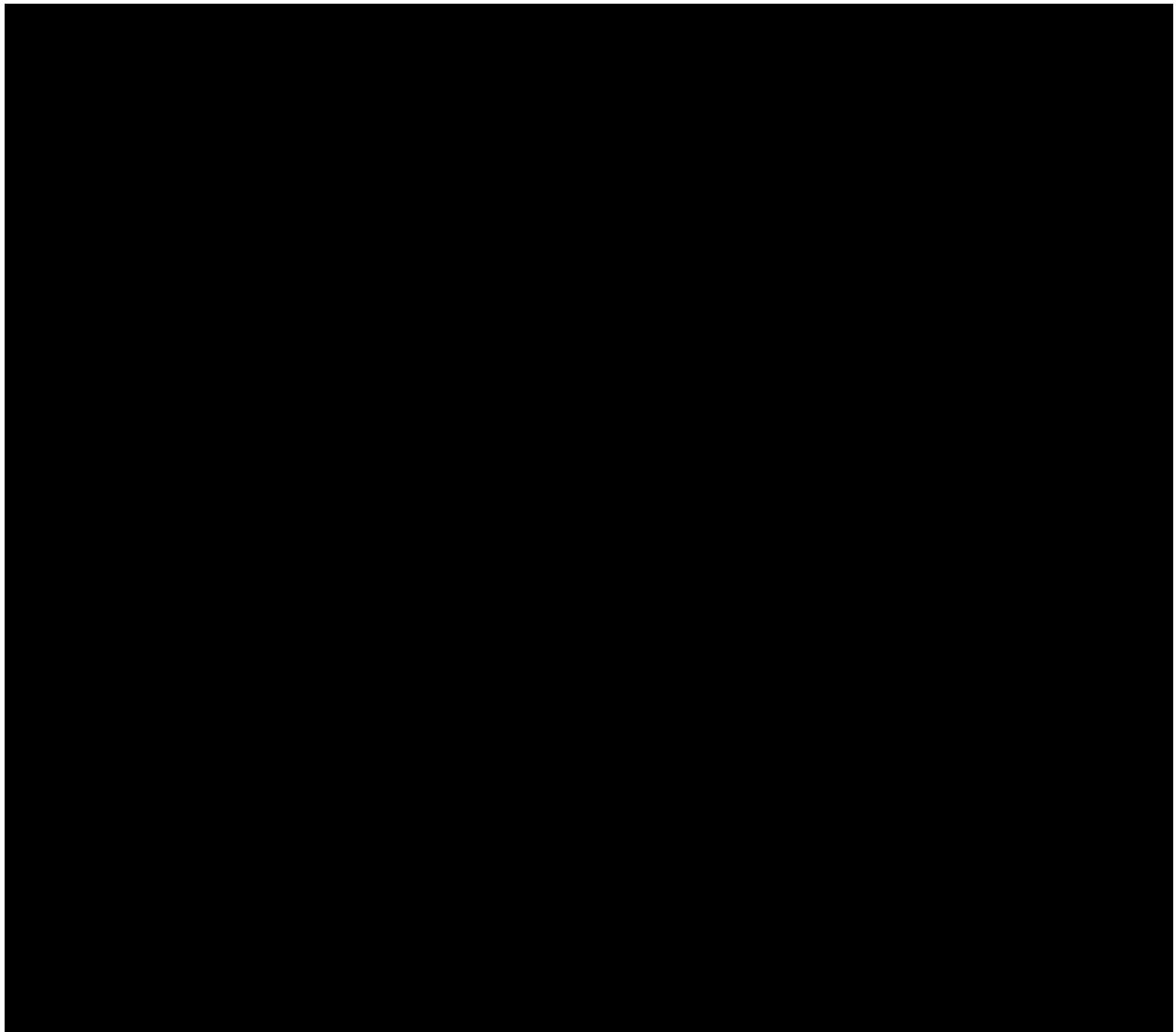
Track My Requests

On clicking “Track My Requests”, the residents can track the status of an Event ID (EID) associated with the logged-in UIN/ VID. They can also **view and download** the detailed information about the entered EID.

Download My Personalized Card

On clicking “Get Personalized Card”, the residents can select the data to be added to their credential. They can preview the chosen data and download it. Residents should select at least 3 attributes.

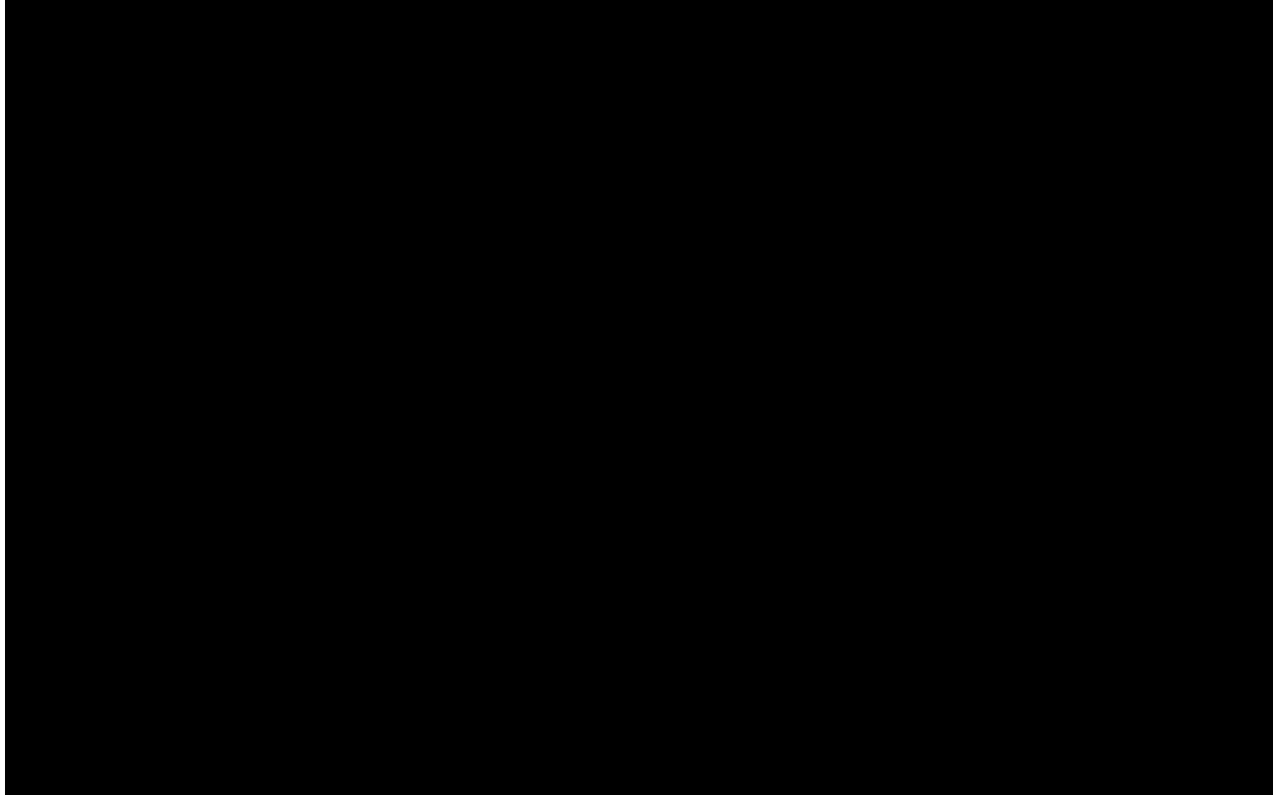
- **Creates the personalized card and signs it**



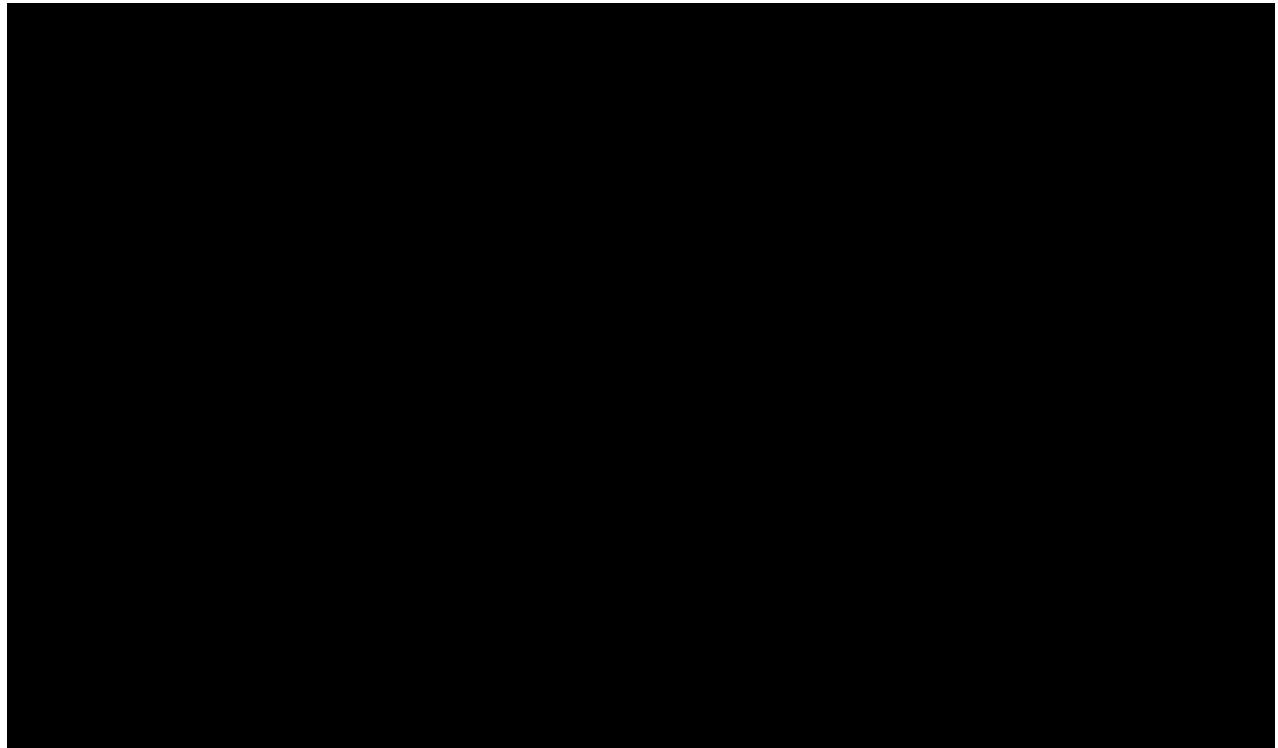
Share My Data

On clicking “Share My Data”, the residents can choose the data to be shared with any of the registered partners to avail various third party services.

- **Submits the share credential request**

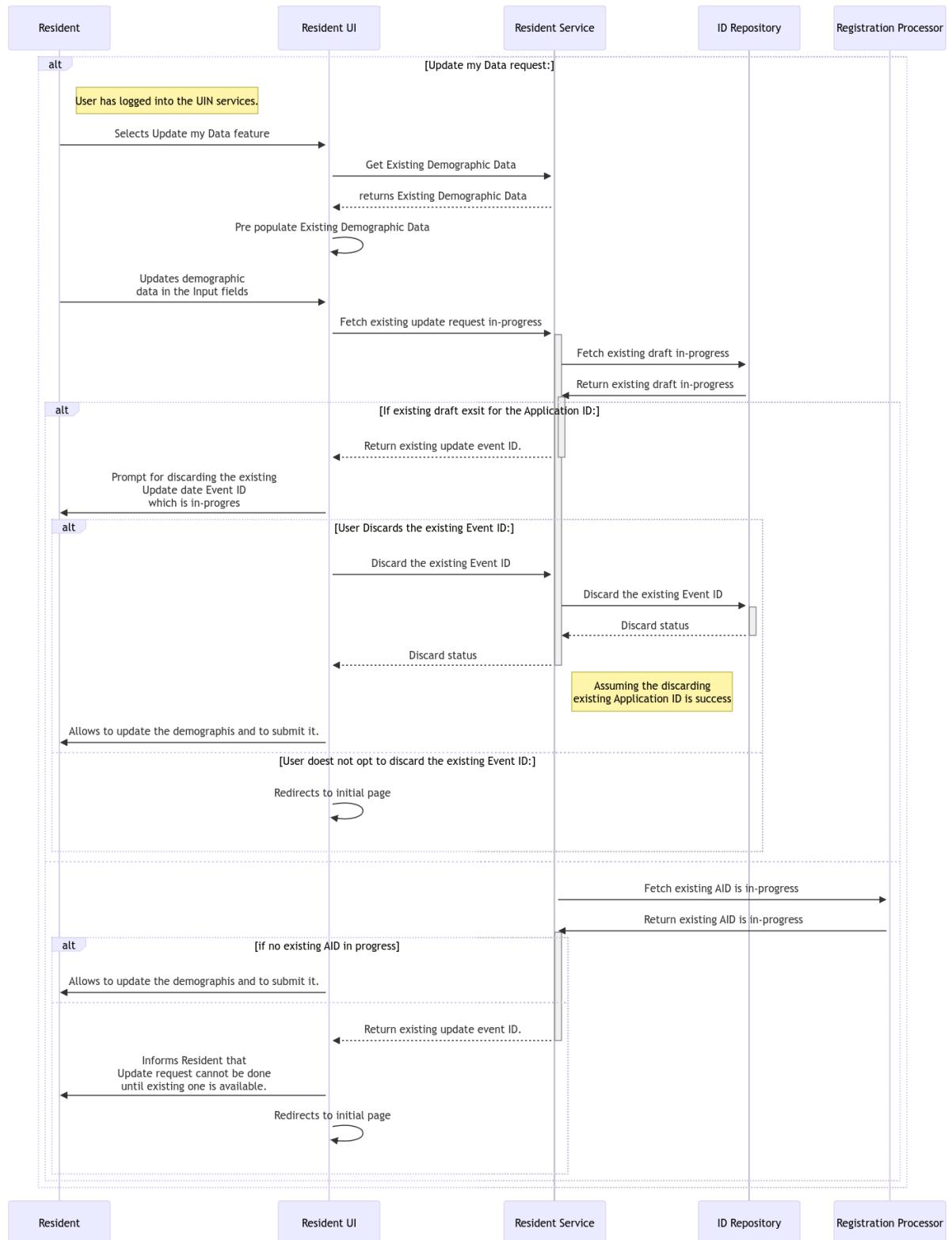


- **Notifies resident about the credential shared status for the event ID**

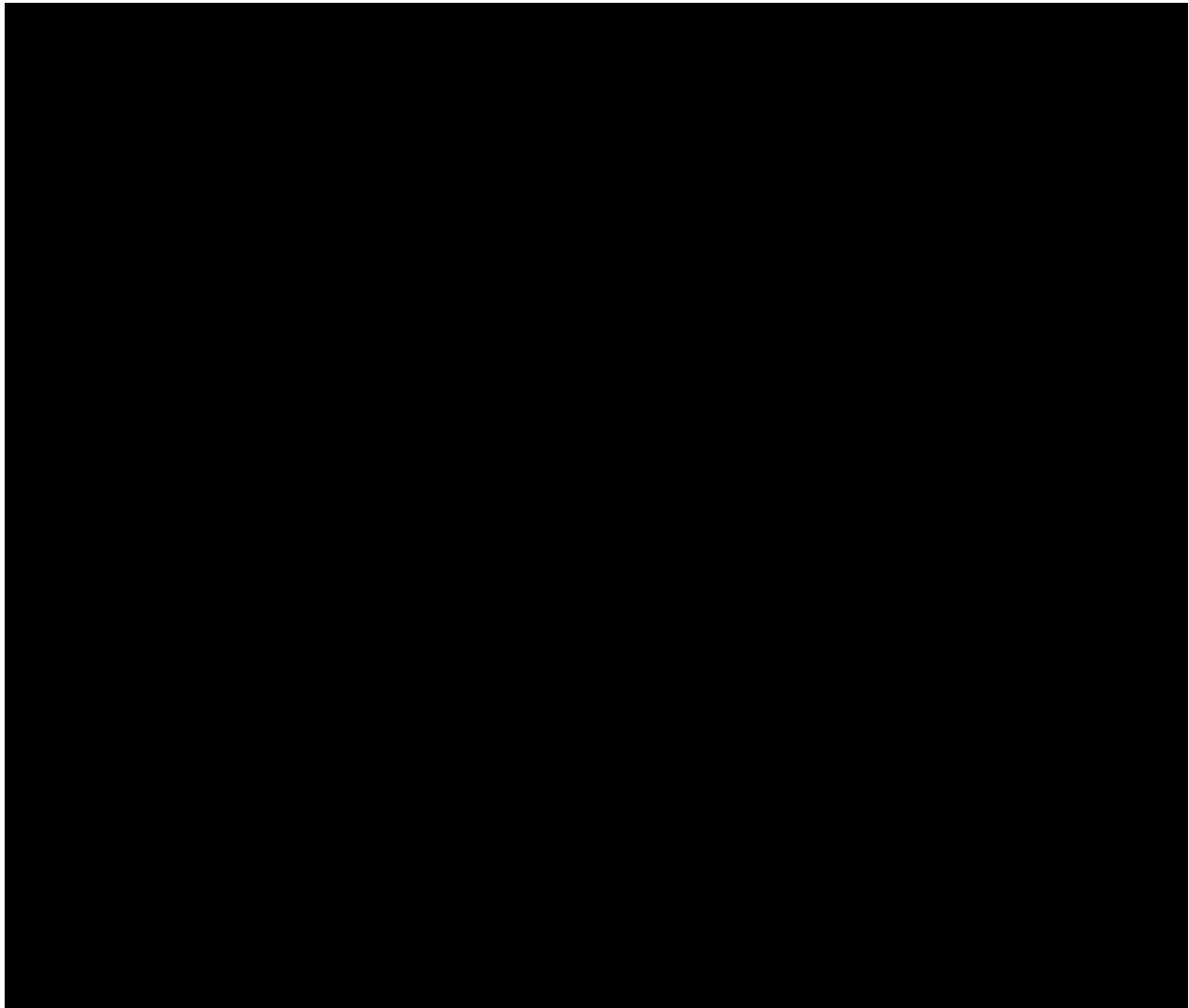


Update My Data

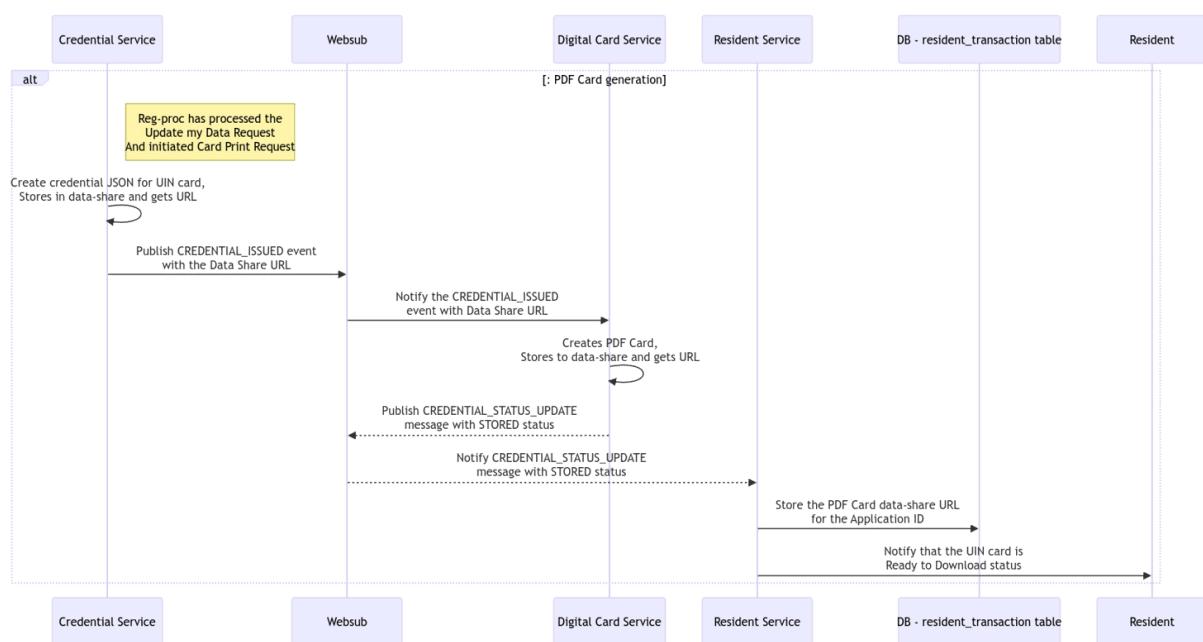
- Fetch existing AID in progress



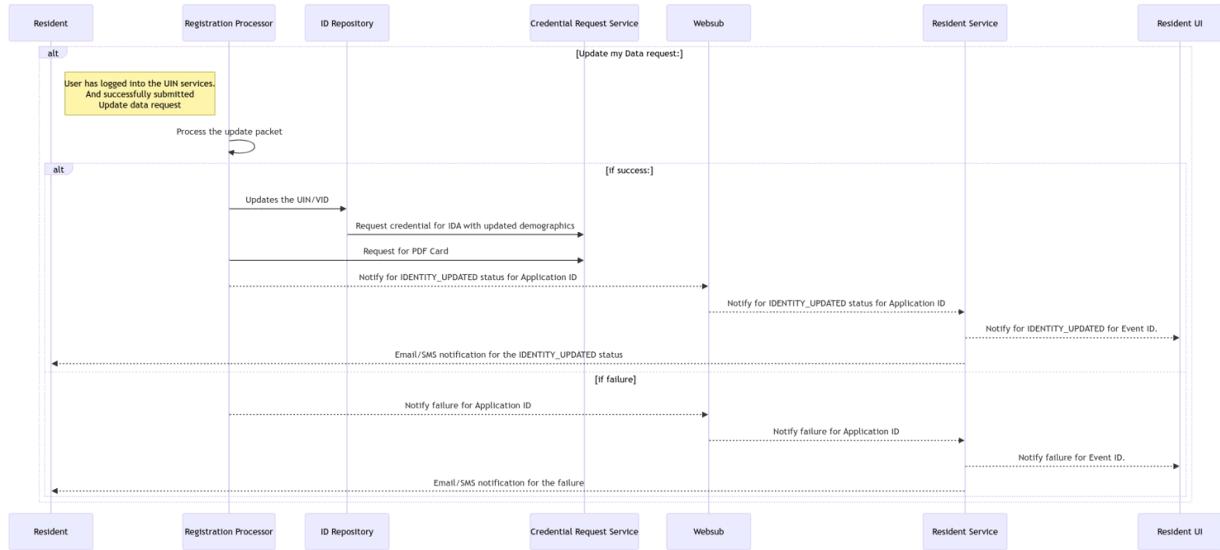
- **Submits the update request**



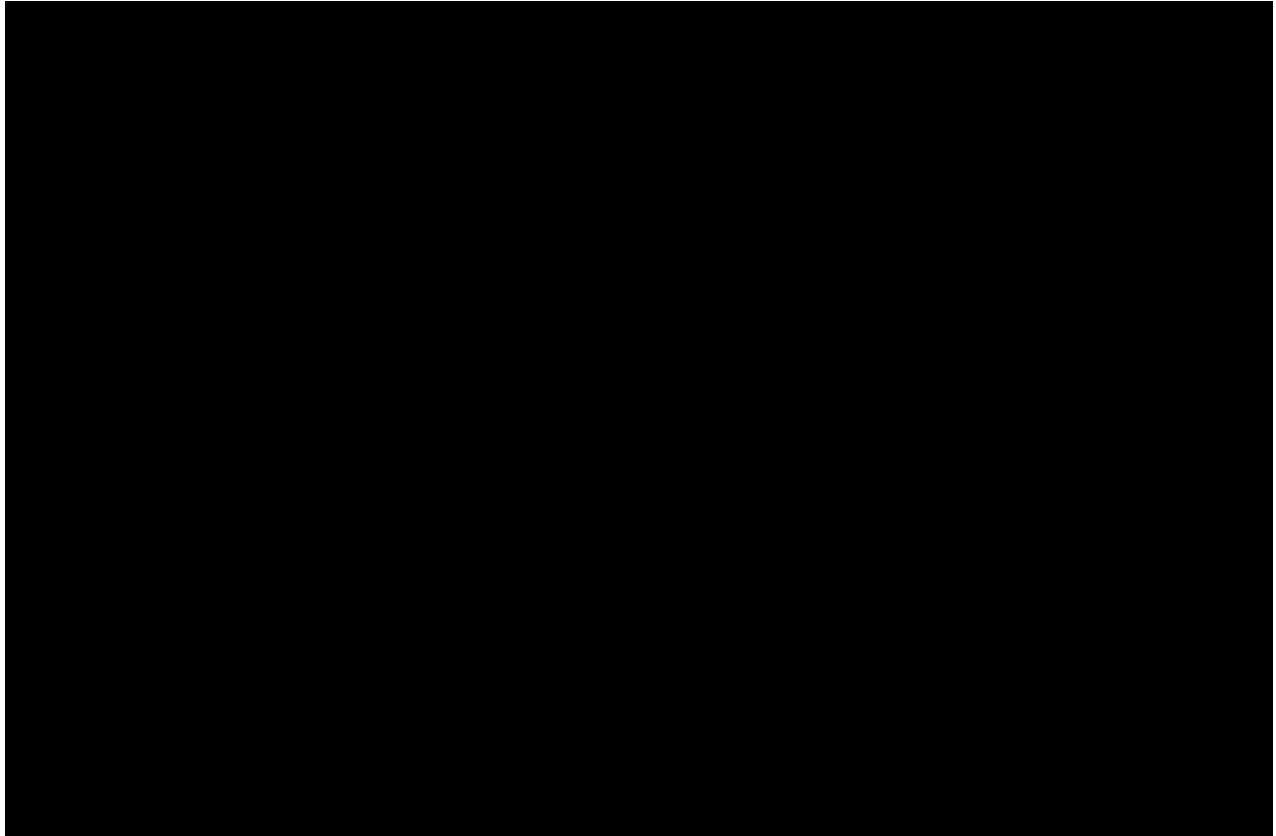
- **Notifies that the UIN card is ready to Download status**



- Notify for IDENTITY_UPDATED for the Event Id

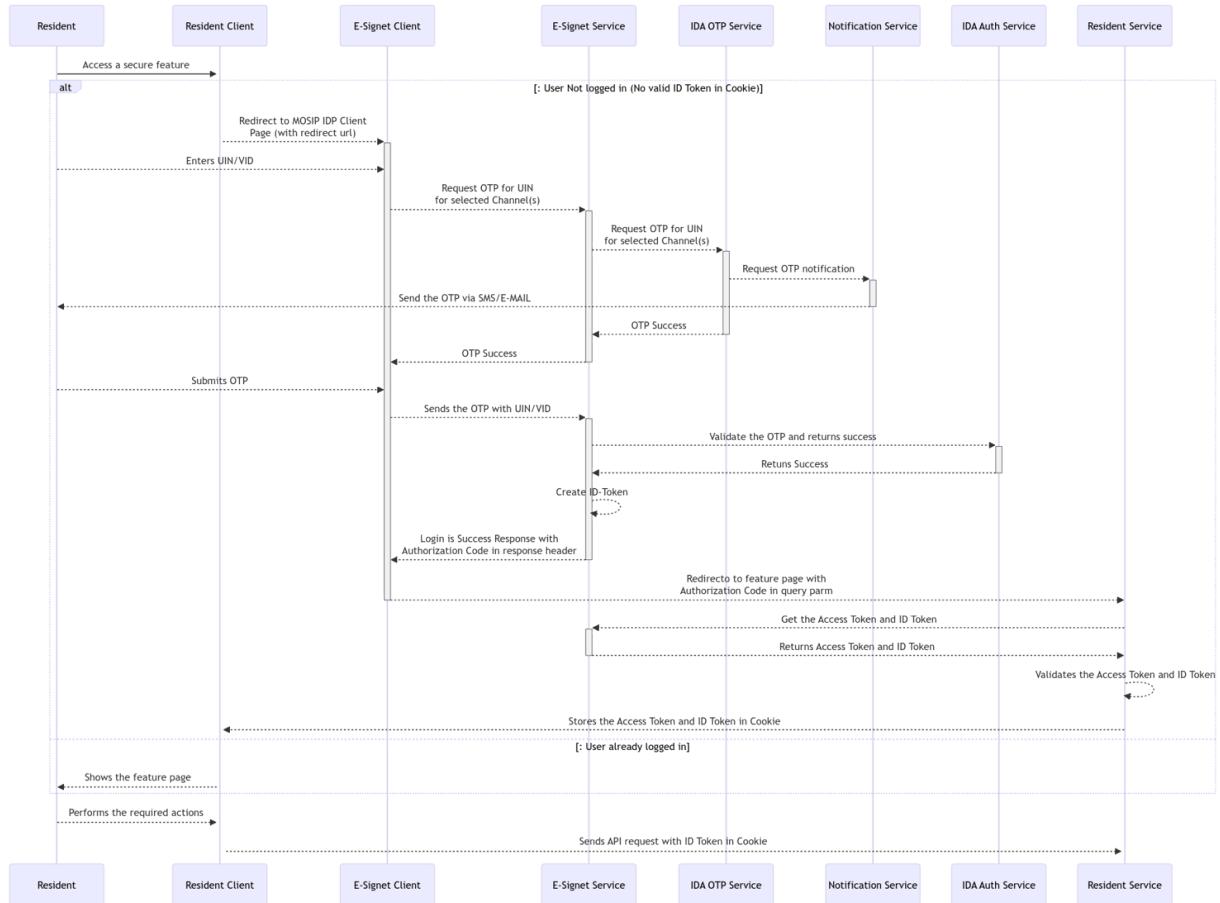


- Get the Status for the AID



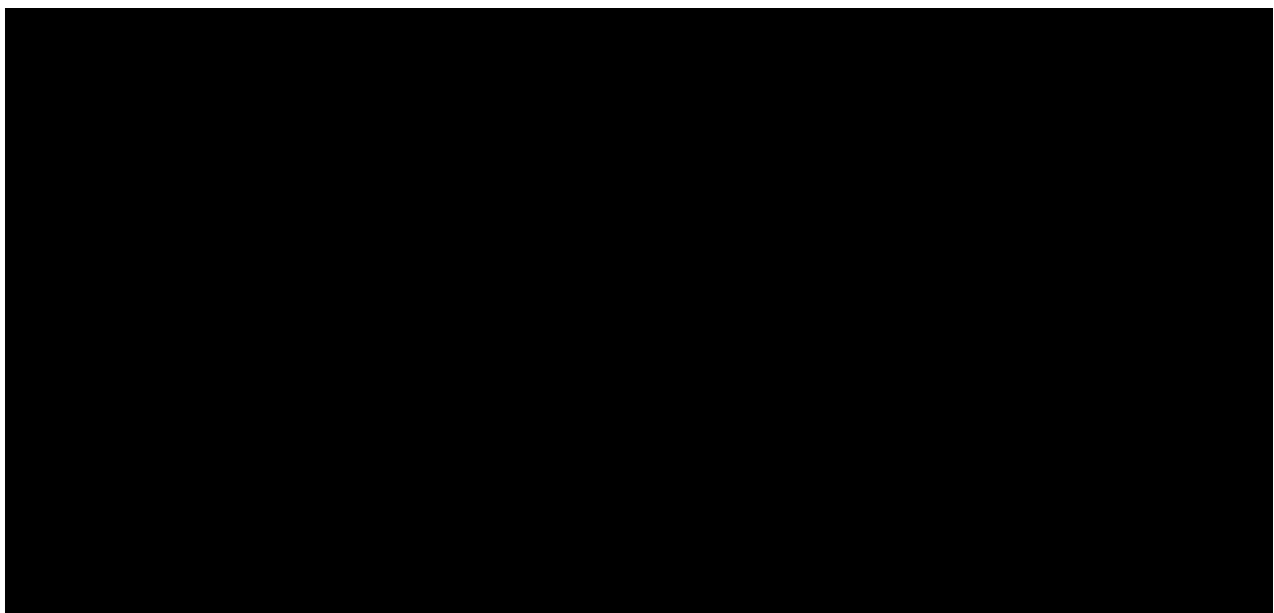
Login With eSignet (OpenID Connect)

- Validate the Access token and ID token

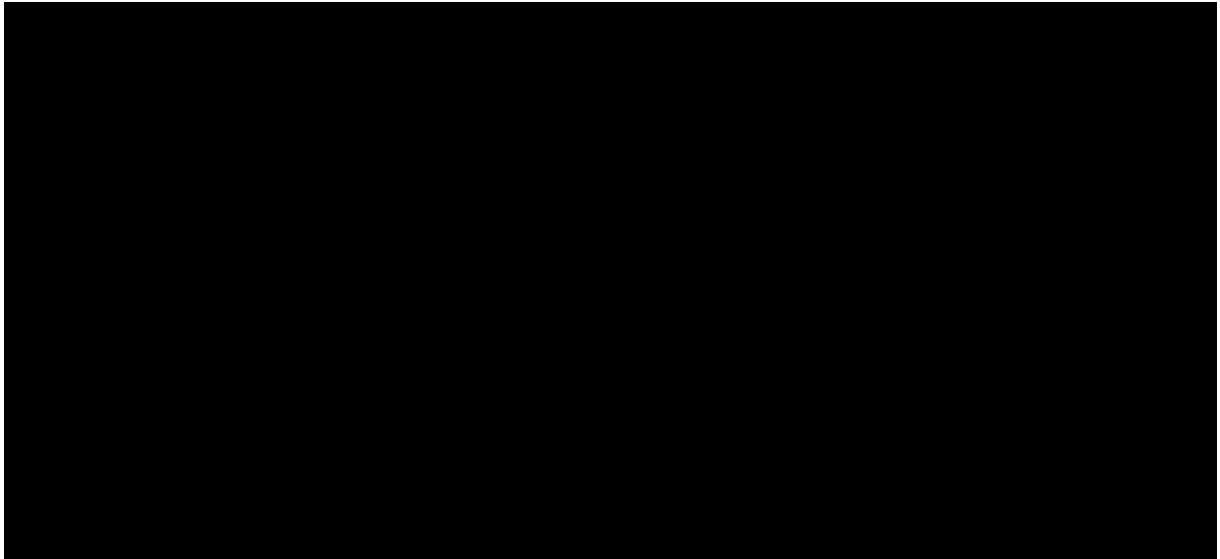


Get Information

- Fetch Supporting Documents PDF



- **Fetch registration centers**



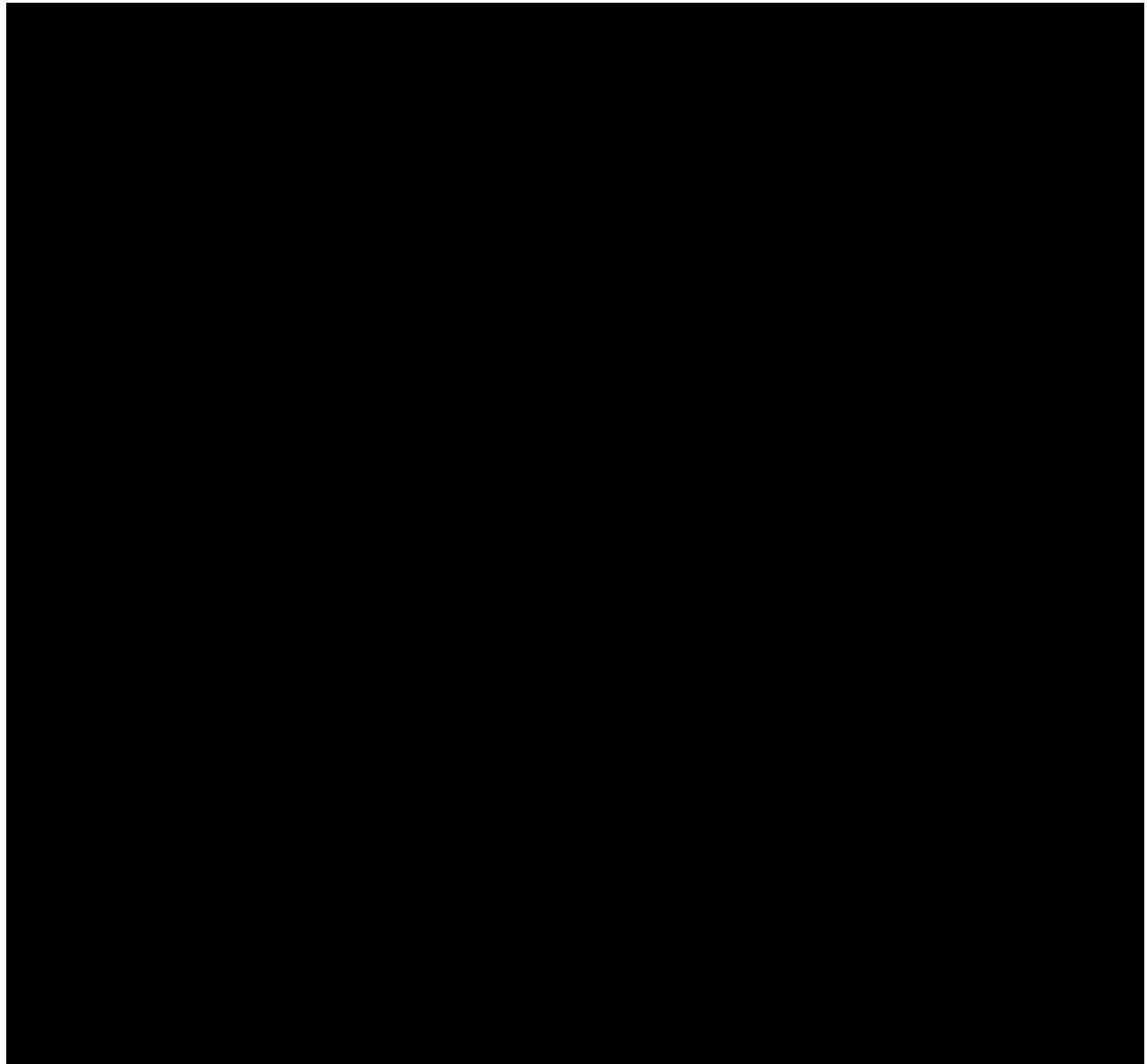
Verify Email ID/ Phone number

The residents can use this feature to verify their registered email ID or phone number.

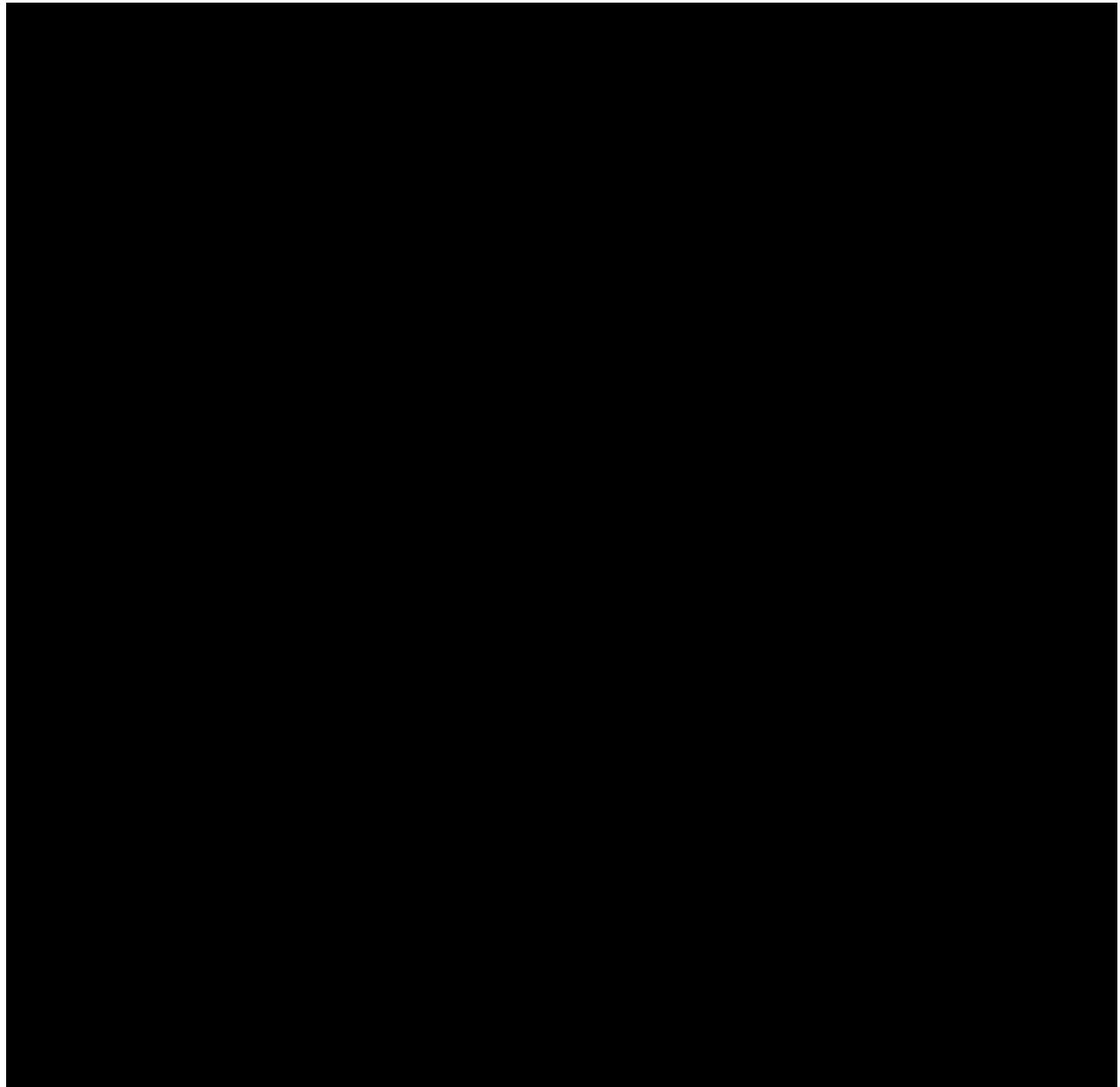
Get My UIN

The residents can use this feature for one of the following:

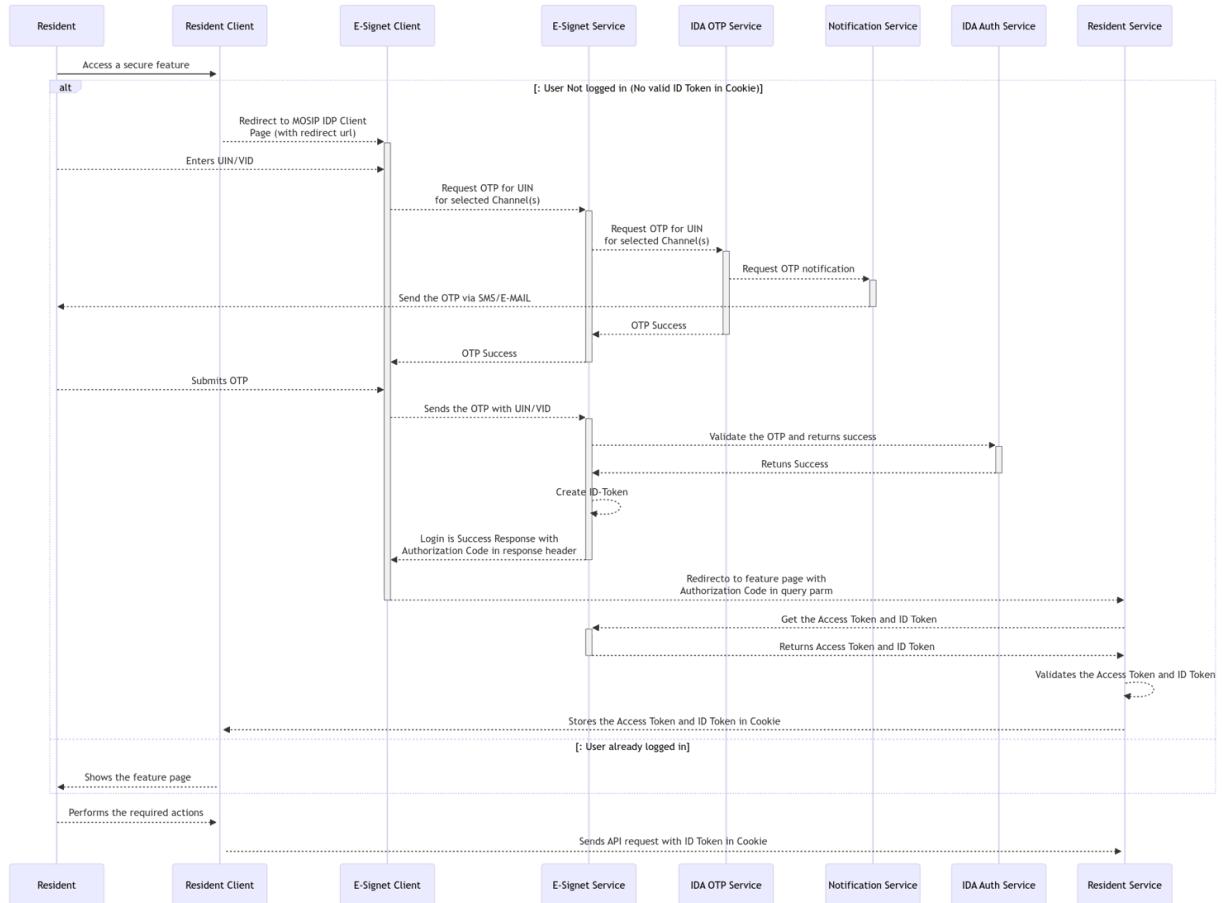
1. Download their UIN card
 2. Check the status of their Application ID (AID)
- **Get PDF card for AID if ready**



- **Check if PDF card URL is notified by Digital card service.**



- **Validates the Action token and ID token.**



Book an appointment

The residents can book an appointment for registration using the pre-registration portal. To do so, they can click on “Book an appointment” tile which will redirect them to the pre-registration portal. To know more about pre-registration portal, refer to this link [<https://docs.mosip.io/1.2.0/modules/pre-registration>]

End User Guide

The Resident Portal is a user-friendly web-based platform designed to assist residents in accessing various services associated with their Unique Identification Number (UIN). This portal offers a range of essential services such as:

1. **UIN services** using UIN/VID (through [e-Signet](#)):

- View My History
- Manage My VID
- Secure My ID
- Track My Requests
- Get Personalised Card
- Share My Data
- Logout

2. Get Information

- About Registration Centers
- List of supporting documents

3. Get My UIN (using UIN/ VID/ AID)

4. Verify email ID and/ or phone number

5. Responsive UI support- Support for the application to work seamlessly on various resolutions.

6. Book an appointment for new enrolment (via the pre-registration portal)

7. Ancillary features

- Font size
- Get Notifications (email and bell notifications)
- View profile details of the logged in user (name, photo, and last login details)

Below is the detailed explanation of each of the features mentioned above.

UIN Services

Residents can use these services to view, update, change, manage or share their data. They can also report an issue in case of a grievance.

Login

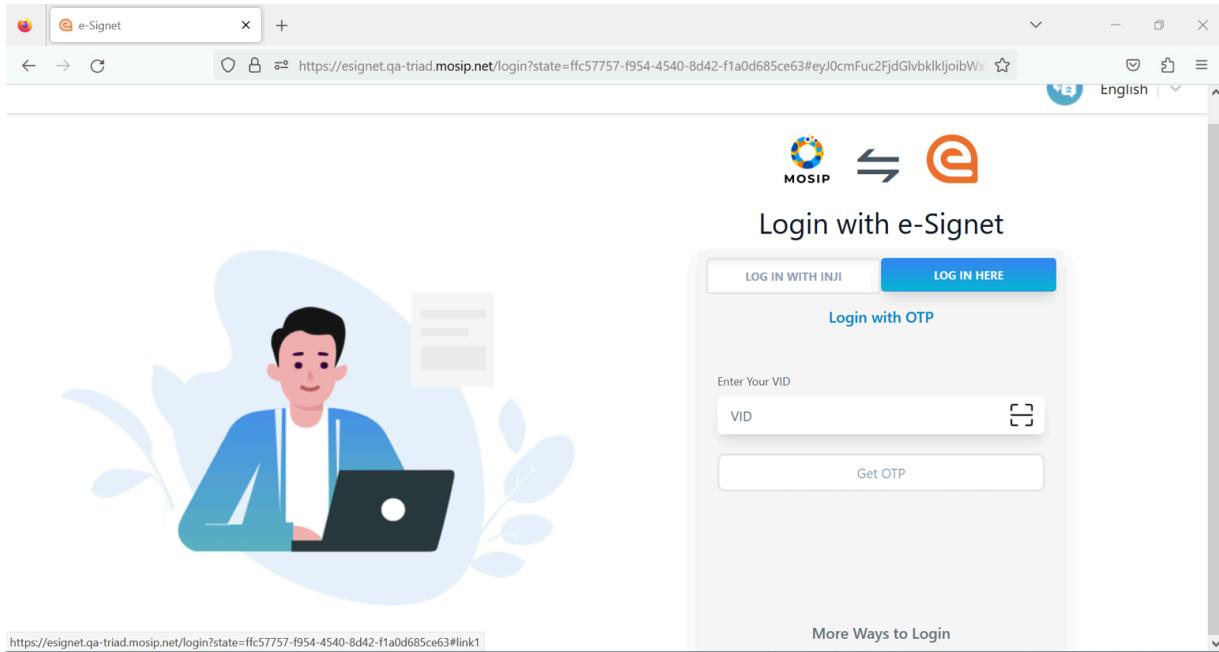
Pre-requisites: To login into the Resident Portal, the resident should have their unique virtual ID (VID) or Unique Identification Number (UIN) and also have access to the registered email ID/ phone number to be able to receive the OTP.

1. Resident accesses the Resident Portal dashboard page.

2. Resident clicks on **UIN Services**.

The login screen appears and the resident can choose one of the options to log in.

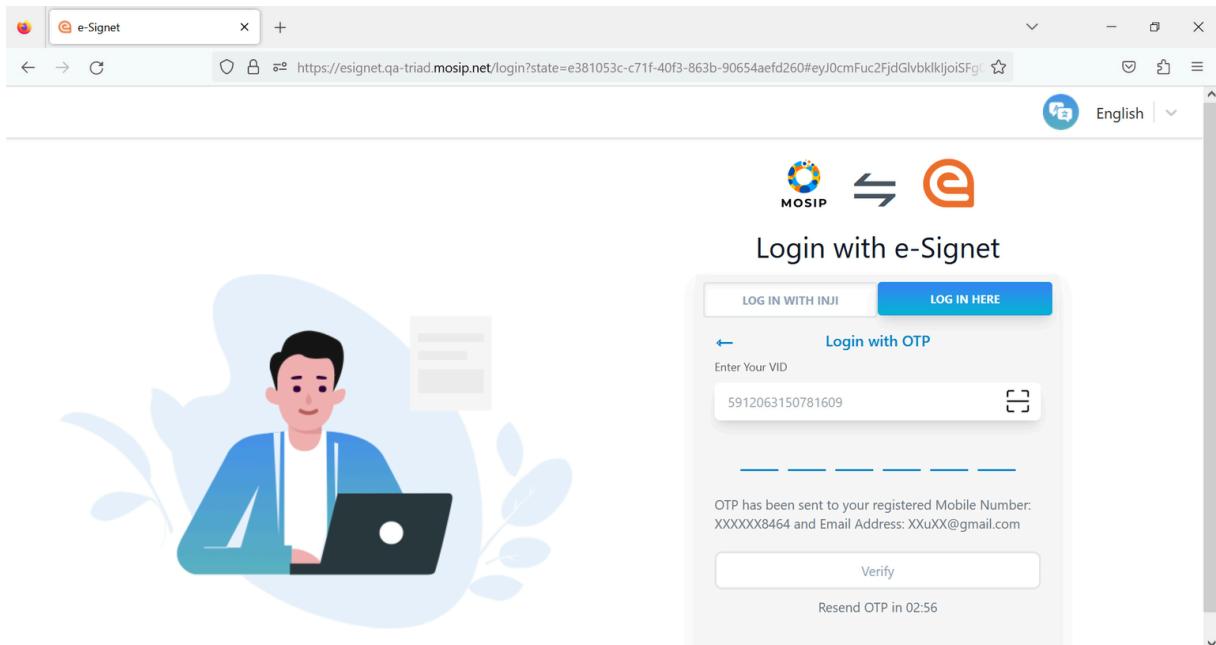
- To login with OTP authentication, the resident clicks on **Log in here > More ways to login > Login with OTP**.



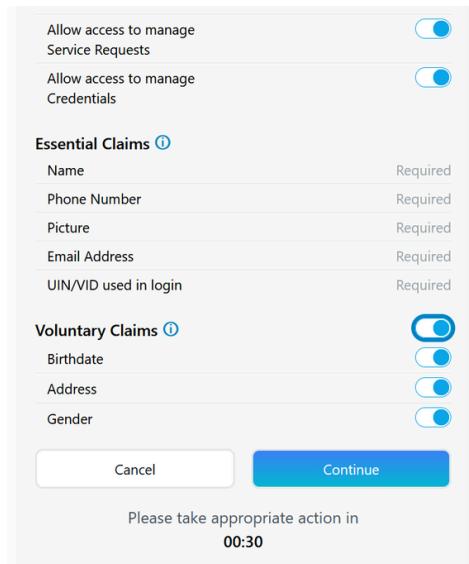
- Resident needs to enter valid VID in the **Enter Your VID** text field and check the box **I'm not a robot**.

- Next, the resident clicks on the **Get OTP** button.

- The resident receives the OTP on the registered phone number and email ID.
- The resident needs to enter the valid OTP received within stipulated time and click the **Verify** button.



6. The resident is then navigated to the Consent page. On this page, the **Essential** and **Voluntary** claims are displayed. Additionally, they will also have to allow access to their data in **Authorize Scope** section to avail various services. Based on the consent provided by the resident, the services will be made available for modification.
7. The resident has the choice to select from the list of Voluntary claims while the Essential claims are mandatory.



8. The resident should now click the **Continue** button. The system navigates the resident to the UIN Services menu page from where they can avail various services.

Note: Consent page will be available only for first time login.

View My History

The residents can view the history of all the transactions associated with their logged-in UIN/ VID. They can also view their details and if any unaccounted entry is found, a report can be raised against the same.

The residents can perform the following:

1. **Search:** The residents can enter an Event ID to search a particular event.
2. **Filter based on date (From date and To date):** The Residents can put a "from" and "to" date in order to get the list of all the events performed in the chosen date range.
3. **Filter based on status (Success/ In Progress/ Failure):** The Residents can filter based on the status of the event. E.g.: If they want to view all "In Progress" events, they can choose the status as "In Progress". Additionally, they can also select any combination of the above three options.
4. **Filter based on History Type (Authentication, ID Management, Data Update, Data Share, Service Requests):** The Residents can filter based on the type of event. Additionally, they can also choose any combination of the above five options.
 - **Authentication Request:** This includes all the authentication and e-KYC requests.
 - **ID Management Request:** This includes the below services:
 - Manage My VID (Generate/Revoke VID)
 - Verify phone number/email ID
 - Secure My ID (Lock/unlock various authentication types)
 - **Data Update Request:** This includes the below services:
 - Update my UIN (demographic data and contact data)

- **Data Share Request:** This includes the below services:
 - Share with a partner
 - **Service Request:** This includes the below services:
 - Download configured card
 - Physical card
 - Get my UIN
 - Book an appointment (lost UIN, Update UIN, Pre-registration, other)
5. **Go button:** Residents can click on the **Go** button once they are done selecting all the required filters.
 6. **Download the PDF of the results:** The residents can download the PDF version of the search result populated.
 7. Clicking on the accordion/ the caret of a particular event, the following options will appear:
 - a. **View Details:** The residents can view the details about an event by clicking on **View Details**. They will be redirected to **Track My Request** page with pre-filled EID where they can see further details about the event.
 - b. **Pin Event to the top:** The residents can pin the events to the top of the list based on their preference. Currently, this is configured for up to 3 events but it can be customized as per country's requirements. Also, the resident can unpin the pinned events by clicking **Unpin from Top**.
 - c. **Report a grievance:** The residents can report a grievance in case of fraud or for any event not initiated by them. On clicking **Report an Issue**, the resident will be redirected to the **Grievance Redressal Form** page where they will see a set of pre-filled data as well as a set of data to be filled.
 - Pre-filled data:
 - Name
 - Event ID (EID)
 - Registered Email ID
 - Registered Mobile Number
 - Data to be filled:
 - Alternate Email ID
 - Alternate Mobile Number
 - Comments

Once the event is completed, a message is displayed containing the grievance tracking ID.

Below are the images with different filters on this page.

Manage My VID

On clicking **Manage My VID**, the resident will be taken to a page where they can view details of the existing VIDs, generate new VID, revoke existing VID or download a VID card.

The following types of VIDs can be seen based on the VID policy:

- Perpetual VID
- Temporary VID
- One-time VID

Note: The resident can get to know about the features of a particular VID by hovering over the "i" symbol.

The residents can perform the following:

1. **Create a new VID :** The residents can click on the **Create** button against any of the VID type selected. They can click on **Yes** to proceed. Once the event is completed, a message is displayed containing the Event ID along with a link to track the service.

2. **Revoke an existing VID:** The residents can click on the **Delete icon** to revoke an existing VID. They can click on **Yes** to proceed. Once the event is completed, a

message is displayed containing the Event ID along with a link to track the service.

3. Download a VID card:

- a. The residents can click on the **Download icon** to initiate the download process. They can click on **Download** to proceed. Once the event is completed, a message is displayed containing the Event ID, a link to track the service and the password combination.
- b. Once the card is ready to download, they will receive a notification for the same under the **bell icon** displayed on the top right corner of the screen and as an Email notification.
- c. On clicking on the notification, the resident will be taken to **Track My Request** page with pre-filled EID.

- d. On this screen, the resident will be able to download the card by clicking on **Download My VID card** button on the bottom left corner of the screen.
- e. The downloaded card will be a password protected PDF. The residents can view the downloaded VID card by entering the password combination displayed on the screen.

4. **View VID number:** All the VID numbers will be masked by default. The residents can view the unmasked version of VID by clicking on eye icon next to the VID number.

Secure My ID

On clicking **Secure My ID**, the residents can view the status of all the authentication types. They can choose to lock or unlock authentication types like the following:

1. Email authentication
2. Mobile authentication
3. Demographic authentication
4. Fingerprint authentication

5. Iris authentication
6. Face authentication

The residents can perform the following,

View the current status of authentication types: The **lock icon** on each card indicates the current status of the authentication type. E.g.: If the lock is open, the authentication type is unlocked which means the residents can authenticate themselves using that particular authentication type and vice versa.

Lock/ unlock the authentication types: To lock/ unlock a particular authentication type, the residents can click on lock/ unlock button. Once the preferences of each authentication type is selected, the residents can click on **Submit** to save the changes and click **Yes** to proceed. Once the event is completed, a message is displayed containing the Event ID along with a link to track the service.

Track My Requests

On clicking `Track My Requests`, the residents can track the status of an EID associated with the logged-in UIN/ VID. They can also view and download the detailed information about the entered EID like:

1. Event ID- This is the unique ID provided against each event

2. Event Type- This is the feature that is being availed. E.g.: Lock/unlock authentication types
3. Event Status- This is the status of the event which can hold values like Success, Failure or In-Progress
4. Individual ID- This is the type of individual ID that was used to login. E.g.: VID or UIN
5. Summary- This is the detailed description of the event.
6. Timestamp- This is the time when the event occurred.
7. Authentication Mode- This is the authentication mode used to login. E.g.: OTP or Biometric or QR code
8. Partner Logo- This is the logo of the registered partner.
9. Partner Name- This is the name of the registered partner.
10. Attribute List- This is the list of attributes shared with the registered partner.
11. Purpose- This is the purpose of sharing data with the registered partner as input by the resident.

The resident can reach `Track My Requests` page by the following ways:

1. `UIN services > View history > Click on the event tile > View details`
2. By clicking the bell icon
3. `UIN services > Track My Requests`

Note:

- Residents can download their updated UIN /VID card.
- **Report a grievance:** The residents can report a grievance in case of fraud or for any event not initiated by them. On clicking `Report an Issue`, the resident will be redirected to the `Grievance Redressal Form` page where they will see a set of pre-filled data as well as a set of data to be filled.

Get Personalised Card

On clicking `Get Personalised Card`, the residents can select the data to be added to their credential. They can preview the chosen data and download it. To be able to download the card, residents should select at least 3 attributes from the list mentioned below:

- Name- Name of the resident. They can choose the format in which they want the name to be displayed.
- Date of Birth- Date of birth of the resident. They can choose the format in which they want the date of birth to be displayed.
- UIN- Unique Identification Number. They can choose to mask or unmask the UIN.
- Perpetual VID- Perpetual Virtual ID. They can choose to mask or unmask the VID.
- Phone number- Registered phone number of the resident. They can choose to mask or unmask the phone number.
- Email ID- Registered email ID of the resident. They can choose to mask or unmask the email ID.
- Address- Address of the resident. They can choose the format in which they want the address to be displayed.
- Gender
- Photo

These details can be previewed as and when the attributes are chosen.

Once the event is completed, a message is displayed containing the Event ID along with a link to track the service.

Share My Data

On clicking **Share My Data**, the residents can choose the data to be shared with any of the registered partners to avail various third party services.

To share the data, residents should select at least 3 attributes from the list mentioned below:

- Name- Name of the resident. They can choose the format in which they want the name to be displayed.

- Date of Birth- Date of birth of the resident. They can choose the format in which they want the date of birth to be displayed.
- UIN- Unique Identification Number. They can choose to mask or unmask the UIN.
- Perpetual VID- Perpetual Virtual ID. They can choose to mask or unmask the VID.
- Phone number- Registered phone number of the resident. They can choose to mask or unmask the phone number.
- Email ID- Registered email ID of the resident. They can choose to mask or unmask the email ID.
- Address- Address of the resident. They can choose the format in which they want the address to be displayed.
- Gender
- Photo

These details can be previewed as and when the attributes are chosen.

Additionally, the residents have to:

1. Select the partner with whom they want to share their data from a dropdown list of registered partners.
2. Enter the purpose of sharing the data with the registered partner.
3. On clicking the **Share** button, the resident will have to provide consent to share their data with the external partner.
4. Once the event is completed, a message is displayed containing the Event ID along with a link to track the service.

Menu Bar

The Resident Portal menu bar contains the following:

1. Font Size- Residents can alter the size of the font based on their preferences.
2. Language- Residents can select the language of preference.
3. Bell icon Notification- Residents can view the notifications of all the asynchronous events in chronological order.
4. Profile Icon- Residents can view the following:
 - Name of the logged in user
 - Photo of the logged in user
 - Last login details
 - Logout option

Book an Appointment

The residents can book an appointment for registration using the pre-registration portal. To do so, they can click on `Book an appointment` tile which will redirect them to the pre-registration portal. To know more about pre-registration portal, refer to this link [Pre-registration](#).

Verify email ID/ phone number

The residents can use this feature to verify their registered email ID or phone number.

Steps to verify email ID/ phone number:

1. Resident clicks either on Verify email ID or Verify phone number option
2. Enter the UIN/VID.
3. Select I'm not a robot against the captcha and click on Send OTP .
4. Resident enters the OTP received on the requested channel and clicks on Submit .

Based on the scenario, any of the below three messages will be displayed:

- a. **Email ID/ phone number successfully verified:** On successful verification, a message is displayed on the screen saying that the phone number/ email ID has been successfully verified.
- b. **Email ID/ phone number was already verified:** If the verification has been previously completed, a message is displayed saying the email ID/ phone number was already verified.
- c. **Email ID/ phone number does not exist:** If there is no email ID/ phone number linked to the UIN/VID, a message is displayed saying no email ID/ phone number was found associated to this UIN/VID.

Get My UIN

The residents can use this feature for one of the following:

1. Download their UIN card
2. Check the status of their Application ID (AID)

Steps to download the UIN:

1. Resident clicks on Get My UIN
2. Enter the AID/UIN/VID.
3. Select I'm not a robot against the captcha and click on Send OTP .
4. Resident enters the OTP received on the registered email ID/ phone number and clicks on Submit .
5. The default PDF of UIN card will be downloaded and a success message is seen stating that the UIN has been successfully downloaded.

Steps to check the status of the AID:

Note: If the UIN is not ready, then the AID is used to get status else UIN card will be downloaded using AID too.

1. Resident clicks on `Get My UIN`.
2. Enter the AID.
3. Select `I'm not a robot` against the captcha and click on `Send OTP`.
4. Resident enters the OTP received on the registered email ID/ phone number and clicks on `Submit`.
5. The status of the AID will be shown.

Get Information

List of supporting documents

Residents can view the list of supported documents in the PDF format and download the same. Also, some sample documents are available for reference.

List of Registration Centers

Residents can search for Registration Centres on the basis of below two mechanisms:

Nearby centers: The resident will be asked to allow permission for location access in order to enable the system to suggest the nearest Registration Centres.

Manually look for centers: If the Resident wants to manually look for a center, they can do so by choosing a level in location hierarchy from the drop-down (e.g.: Region, Province, Postal Code) and entering the value against the same.

They can also download the PDF version of the result displayed on the screen for reference.

Update My Data

1. Identity data

- Name
- DOB
- Gender
- POI document

2. Address

- Full address
- POA document

3. Contact Information

- Phone number
- Email ID

4. Preferred Language

5. **Identity Data:** Residents can update their identity data like Name, Date of Birth, Gender certain number of times (number of times the identity data can be updated is configurable). **To update the Identity Data, the residents will have to do the following:**

- a. Go to "Update My Data"

b. Click on "Identity" tab

c. Enter the new Name/ new Date of Birth/ new Gender in preferred language.

- d. The resident will then have to choose the type of document from drop-down.
- e. Upload a valid supporting document as Proof of Identity to back their change in identity request.

f. Once the document is uploaded, the "Preview" button will be made clickable

g. The Residents will then be taken to the preview screen where they can view the updated data and the uploaded supporting document which they can modify if required. When the resident is satisfied with all the data entered, he can go ahead and submit the data update request by clicking on "Update" button.

h. The Resident will then have to accept the terms and conditions and click on "Submit" button to submit the data update request.

i. Once the event is completed, a message will be displayed containing the Event ID along with a link to track the service.

j. A bell icon and an email notification will be triggered using which the residents can view the status of the application.

k. Once the update is successful, the card can be downloaded with new data by clicking on the particular notification.

- ① Note: The Resident can update the Proof of Identity document itself with no change in data.

6. **Address:** Residents can update their partial address or full address on the basis of their requirement any number of times. **To update the Address, the residents will have to do the following:**

- a. Go to "Update My Data"

b. Click on "Address" tab

c. Enter the new address.

d. The resident will then have to choose the type of document from drop-down.

- e. Upload a valid supporting document as Proof of Address to support their change in address request.
- f. Once the document is uploaded, the preview button will be enabled.

g. The Residents will be taken to the preview screen where they can view the updated data and the uploaded supporting document which they can modify if required by clicking on the pencil icon.

h. When the resident is satisfied with all the data entered, they can go ahead and submit the data update request by clicking on “update”.

- i. The Resident will also have to accept the terms and conditions in order to proceed and click on "Submit" button to proceed.
- j. Once the event is successful, a message will be displayed consisting of the Event ID along with a link to track the service.

k. A bell icon and an email notification will be triggered using which the residents can view the status of the application.

I. Once the update is successful, the card can be downloaded with new data by clicking on the particular notification.

- ⓘ Note: Additionally, the Resident can also update the Proof of Address document itself with no change in data.

7. Contact Data: Residents can update their existing email ID and phone number.
To update the Contact Data, the residents will have to do the following:

a . Go to "Update My Data".

- b. Click on "Contact Data" tab.
- c. Enter the new email ID or Phone number (whichever needs to be updated).

d. The Resident will receive an OTP over their new email ID/ Phone number and thereby entering the OTP received on the new email ID/ phone number.

e. Once the event is completed, a message will be displayed consisting of the Event ID along with a link to track the service.

f. A bell icon and an email notification will be triggered using which the residents can view the status of the application.

g. Once the update is successful, the card can be downloaded with new data by clicking on the particular notification.

8. **Notification Language Preference:** Residents can update the language in which all the notifications are being sent to them. The residents can change the notification language as many times as they want to. **To update the Notification Language, the residents will have to do the following:**
 - a. Go to "Update My Data"

b. Go to "Language Preference" tab

c. Click on the "New Notification Language" drop-down.

d. Choose the new Notification Language and click in "Submit" button.

- e. On clicking on "Submit", a message will be displayed consisting of the Event ID along with a link to track the service.

Multi-Lingual Support:

The Residents can view the entire portal in the language that they prefer using the language change option on the top right corner of the screen. On choosing any

language, all the labels/ texts/ success or error messages, PDF downloads will be displayed in the chosen language.

To change the language, the residents will have to do the following:

1. Click on the language option from the header menu.
2. On clicking the language option, a drop-down will open that will have the list of languages in which the Resident Portal can be rendered in.
3. On choosing any language, the screen will be refreshed and the entire portal will be rendered in the chosen language.
4. Menu option in English.
5. Menu option in French

6. Menu option in Arabic

- ⓘ **Note:** Only one update request can be raised at a time. A second update request can only be initiated when there are no existing requests in progress. If the user still wants to request another data update, they can discard the request in progress (only the requests that are in "Draft" stage in ID Repository can be discarded) and then raise a new data update request.

Collab Guide

The Resident Portal Guide for Collab Environment provides a comprehensive introduction to the services available on Resident Portal, as well as a guide on how to navigate its features.

Resident Portal is a user-friendly web-based platform designed to assist residents in accessing services related to their Unique Identification Number (UIN). This portal allows residents with a UIN to conveniently manage their UIN, protect their identity, track requests, and obtain personalized cards. For a more detailed overview of the features offered, please visit [here](#).

This Resident Portal Demo Setup guide serves as a tool to demonstrate the impressive capabilities of MOSIP's system. Let's embark on this journey together to explore the potential of Resident Portal.

Please note that for developers setting up Resident Portal locally, refer to the [Developers Guide](#).

Pre-requisites

- Accessing Resident Portal in Collab environment does not require any complex setup.
- All you need is a UIN or a VID and you are good to go!
- To get your UIN credentials for Collab environment follow the below steps:
 - The provision of a UIN (Unique Identification Number) as a demo credential will enable you to have a firsthand experience of the Resident Portal's capabilities and explore its various features.
 - Please complete the [form](#) provided and submit it. Once received, we will generate a demo credential for you to use in order to login to the Resident Portal.

Note: Please use 111111 as the OTP, for any OTP-based feature in the Collab environment.

Step-by-Step Process

Step 1: Access Resident Portal

Visit the following URL to access Resident Portal in Collab environment: [Resident Portal](#)

Step 2: Login

The recommended method for accessing the Resident Portal in the Collab environment is to use a valid UIN (Unique Identification Number), which can be obtained by completing the form provided. This is one of the options available for logging in and accessing UIN-related services on the Resident Portal.

There are multiple methods for logging in to access UIN services on the Resident Portal:

1. **Login with Inji:** To download and use the Inji application, click [here](#).
2. **Login with Biometrics:** To set up a biometrics device on your system, click [here](#).
3. **Login with OTP (One-Time Password):** Enter UIN/VID received from our end and use 111111 as OTP, for login to Resident Portal in Collab environment.

Step 3: Explore the features of Resident Portal

- To access registration related information, click [here](#).
- Go to `Get Information` section.
 - Click `Registration Centers` to get the list of Registration Centers near you or to look for Registration Centers based on location hierarchy.
 - Click `Supporting Documents` to get the list of documents that Resident Portal supports.
- To download the UIN card or to know the status of your AID, click [here](#).
 - Next, go to `Get My UIN` section.
- To book an appointment for a new enrolment, click [here](#).

- Go to [Book an Appointment](#) section.
- Explore the steps of Pre-registration Portal to book an appointment.
- To verify your email ID or phone number, click [here](#).
 - For **Email verification**- Use the OTP received on your email, as provided at the time of UIN creation.
 - For **Phone number verification** - Since our mobile services are temporarily unavailable, please use the email services.
- To access all the features of Resident portal in MOSIP, click [here](#) to login.
- For more information, refer to our [end user guide](#).

Additional Resources

To get more information about Resident Portal, click [here](#).

You will be able to explore and get a better understanding of Resident Portal configurations, UI, deployment, and OIDC client configuration to mention a few.

By adhering to these instructions, you will acquire the necessary knowledge to effortlessly configure the Resident Portal and proficiently utilize all of its functionalities.

Get in Touch

If you require any assistance or encounter any issues during the testing and integration process, kindly reach out to us through the support mechanism provided below.

- Navigate to [Community](#).
- Provide a detailed description about the support you require or provide detailed information about the issue you have encountered, including steps to reproduce, error messages, logs and any other relevant details.

Thank you. Wishing you a pleasant experience!

Deploy

Deployment Guide

This guide contains all the information required for successful deployment and running of Resident Portal. It includes information about the Database and template scripts, seed data, roles, OIDC client setup, etc.

DB scripts

Resident Service DB Scripts to be run: [DB scripts](#)

Master-data Template Scripts

The master-data templates required for the Resident portal are added to the [template](#) and [template type](#) DML excel files in [mosip/mosip-data](#) repository. These scripts need to be applied to the corresponding table.

Keycloak Roles

`mosip-resident-client` needs to have below roles in keycloak:

- `RESIDENT`
- `SUBSCRIBE_AUTH_TYPE_STATUS_UPDATE_ACK_GENERAL`
- `SUBSCRIBE_AUTHENTICATION_TRANSACTION_STATUS_GENERAL`
- `SUBSCRIBE_CREDENTIAL_STATUS_UPDATE_GENERAL`
- `SUBSCRIBE_REGISTRATION_PROCESSOR_WORKFLOW_COMPLETED_EVENT_GENERAL`
- `CREDENTIAL_REQUEST`

Resident OIDC Client setup

Here is the document which explains how `resident-oidc` partner is onboarded through [partner-onboarder after deployment](#).

For more details on how to configure the Resident OIDC client, refer [here](#).

Configuration Guide

Overview

The following guide outlines some important properties that can be customized for a given installation. Please note that this list is not exhaustive but serves as a checklist for reviewing properties that are likely to differ from the default settings. For a complete list of properties, refer to the files listed below.

Configuration files

Resident Service uses the following configuration files:

```
application-default.properties  
resident-default.properties  
resident-ui-share-credential-schema.json  
resident-ui-update-demographics-schema.json  
resident-ui-personalized-card-schema.json
```

Resident Services dependent modules configuration files

ⓘ Changes are done only in the below config files for dependent modules.

id-repository-default.properties

id-authentication-default.properties

mosip-vid-policy.json

registration-default.properties

Changes done in the MOSIP config repo

Please go to the [Files changes](#) section and refer to removed and added properties.

Database

Properties used for configuring the database.

```
mosip.resident.database.hostname=postgres-postgresql.postgres
mosip.resident.database.port=5432
javax.persistence.jdbc.driver=org.postgresql.Driver
javax.persistence.jdbc.url=jdbc:postgresql://${mosip.resident.database.ho
javax.persistence.jdbc.user=residentuser
javax.persistence.jdbc.password=*****
```

Token generation

```
resident.appid=resident
resident.clientId=*****
resident.secretKey=*****
token.request.issuerUrl=${mosip.keycloak.issuerUrl}
```

Logger related properties

URL pattern for logging filter. For example, "/callback/*". *Defaults to "/*".*

```
logging.level.root=INFO
logging.level.io.mosip.resident.batch=INFO
logging.level.io.mosip.resident.filter=DEBUG
resident.logging.filter.enabled=false
resident.logging.filter.url.pattern=/*
```

Rest template logger filter

This will print the request details such as URL, headers, and body for debugging purposes. The default is false.

```
logging.level.io.mosip.resident.config.LoggingInterceptor=DEBUG  
resident.rest.template.logging.interceptor.filter.enabled=false
```

DB calls logger filter

This will print the repository method calls for debugging purposes. The default is false.

```
logging.level.io.mosip.resident.aspect.DatabaseLoggingAspect=DEBUG  
resident.db.logging.aspect.enabled=false
```

Micrometer metrics for DB response time & rest template API call response time

```
resident.db.metrics.aspect.enabled=true  
resident.rest.template.metrics.interceptor.filter.enabled=true
```

Websub topic subscription and WebsubCallbackRequestDecoratorFilter

```
subscriptions-delay-on-startup_millisecs=120000  
re-subscription-interval-in-seconds=43200  
resident.websub.request.decorator.filter.enabled=true
```

Partner details

```
mosip.ida.partner.type=*****  
ida.online-verification-partner-id=*****  
idrepo-dummy-online-verification-partner-id=*****  
resident.share-credential.partner.type=*****  
resident.authentication-request.partner.type=*****  
resident.order-physical-card.partner.type=*****
```

DB properties to skip automatic table creation in startup

```
hibernate.show_sql=false  
hibernate.hbm2ddl.auto=none  
hibernate.temp.use_jdbc_metadata_defaults=false  
hibernate.jdbc.lob.non_contextual_creation = true  
spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults=false
```

Allowed Authentication types and default unlock duration

These are the authentication types allowed for a resident and default unlock duration.

```
auth.types.allowed=otp-email,otp-phone,demo,bio-FINGER,bio-IRIS,bio-FACE  
resident.auth-type.default.unlock.duration.seconds=100
```

Template type codes for allowed Auth-type list (auth.types.allowed)-

Templates type codes for authentication types

```
resident.otp-email.template.property.attribute.list=mosip.otp-email.template  
resident.otp-phone.template.property.attribute.list=mosip.otp-phone.template  
resident.demo.template.property.attribute.list=mosip.demo.template.property.attribute.list  
resident.bio-FINGER.template.property.attribute.list=mosip.bio-finger.template  
resident.bio-IRIS.template.property.attribute.list=mosip.bio-iris.template  
resident.bio-FACE.template.property.attribute.list=mosip.bio-face.template
```

Template type codes for Auth-type status-

Templates type codes for authentication types status

```
resident.UNLOCKED.template.property.attribute.list=mosip.unlocked.template.  
resident.LOCKED.template.property.attribute.list=mosip.locked.template.pr
```

Validation properties

Below are the properties used for validation purpose.

```
mosip.id.validation.identity.phone=^([6-9]{1})([0-9]{9})$  
mosip.id.validation.identity.email=^[\w-]+\.\w+@\w+\.\w+$  
resident.grievance-redressal.alt-email.chars.limit=128  
resident.grievance-redressal.alt-phone.chars.limit=64  
resident.grievance-redressal.comments.chars.limit=1024  
resident.share-credential.purpose.chars.limit=1024  
mosip.resident.eid.length=16  
mosip.resident.eventid.searchtext.length=${mosip.resident.eid.length}  
resident.message.allowed.special.char.regex=^[A-Za-z0-9 .,-]+$  
resident.purpose.allowed.special.char.regex=^[A-Za-z0-9 .,-]+$  
resident.id.allowed.special.char.regex=^[0-9]+$  
resident.document.validation.transaction-id.regex=^[0-9]{10}$  
resident.document.validation.document-id.regex=^[A-Za-z0-9-]{20},{$  
resident.validation.is-numeric.regex=^[0-9]+$  
resident.otp.validation.transaction-id.regex=^[0-9]{10}$  
resident.validation.event-id.regex=^[0-9]${mosip.resident.eid.length}$$
```

Security

```
mosip.security.csrf-enable:false  
mosip.security.secure-cookie:false
```

Keycloak authentication client

```
token.request.appid=resident  
token.request.clientId=*****  
token.request.secretKey=*****
```

Keycloak authentication allowed audience

```
auth.server.admin.allowed.audience=mosip-resident-client,mosip-reg-client
```

Mapping Identity json to map with the applicant id json

```
registration.processor.identityjson=identity-mapping.json
```

Machine creation and search configs

Properties used for machine specification and center

```
resident.update-uin.machine-name-prefix = resident_machine_
resident.update-uin.machine-spec-id = RESIDENT-1
resident.update-uin.machine-zone-code = MOR
resident.center.id=10007
resident.machine.id=10045
```

Auth Adapter rest template authentication configs

```
mosip.iam.adapter.appid=resident
mosip.iam.adapter.clientid=*****
mosip.iam.adapter.clientsecret=*****
```

Exclusion list of URL patterns that should not be part of authentication and authorization

Property used to define the endpoints that should not be part of authentication.

```
mosip.service.end-points=/**/req/otp,/**/proxy/**/*,/**/validate-otp,/**/
```

Configuration for google re-captcha

```
mosip.resident.captcha.enable=true  
mosip.resident.captcha.id.validate=mosip.resident.captcha.id.validate  
mosip.resident.captcha.sitekey=*****  
mosip.resident.captcha.secretkey=*****  
mosip.resident.captcha.resource.url=http://resident-captcha.resident/reside  
mosip.resident.captcha.recaptcha.verify.url=https://www.google.com/recaptch
```

Comma separated values of property keys to be exposed to UI

This property is used to define the keys of the properties to be exposed to UI.

```
resident.ui.propertyKeys=mosip.mandatory-languages,mosip.optional-languag
```

Auth allowed urls

```
auth.allowed.urls=https://${mosip.resident.host}/,https://${mosip.residen
```

MOSIP eSignet config

When enabling MOSIP eSignet comment Mock Keycloak config, vise versa.

```
mosip.iam.module.clientID=*****  
mosip.iam.module.clientsecret=  
mosip.iam.base.url=https://${mosip.esignet.host}/v1/esignet  
mosip.iam.authorization_endpoint=https://${mosip.esignet.host}/authorize  
mosip.iam.token_endpoint=${mosip.iam.base.url}/oauth/v2/token  
mosip.iam.userinfo_endpoint=${mosip.iam.base.url}/oidc/userinfo  
mosip.iam.certs_endpoint=${mosip.iam.base.url}/oauth/.well-known/jwks.json  
auth.server.admin.issuer.uri=${mosip.iam.base.url}  
auth.server.admin.issuer.domain.validate=true  
auth.server.admin.oidc userinfo.url=${mosip.iam.userinfo_endpoint}  
mosip.iam.module.token.endpoint.private-key-jwt.auth.enabled=true  
mosip.iam.module.token.endpoint.private-key-jwt.expiry.seconds=7200  
mosip.resident.oidc userinfo.jwt.signed=true
```

Auth Adapter ValidateTokenHelper

This property will directly apply the certs URL without the need for constructing the path from the issuer URL. This is useful to keep a different certs URL for integrating with MOSIP IDP for offline token validation.

```
auth.server.admin.oidc.certs.url=${mosip.iam.certs_endpoint}  
mosip.iam.logout.offline=true  
auth.server.admin.validate.url=  
mosip.resident.oidc.userinfo.jwt.verify.enabled=false
```

Resident login configuration for eSignet

```
mosip.iam.module.redirecturi=${mosip.api.internal.url}/resident/v1/login-  
mosip.iam.module.login_flow.name=authorization_code  
mosip.iam.module.login_flow.scope=openid profile Manage-Identity-Data Man  
mosip.iam.module.login_flow.claims={"userinfo":{"name":{"essential":true}}}  
mosip.iam.module.login_flow.response_type=code  
mosip.iam.module.admin_realm_id=mosip
```

User-info claim attributes

Used in open-id-connect based login with UIN/VID in MOSIP-IDP

```
mosip.resident.identity.claim.individual-id=individual_id  
mosip.resident.identity.claim.ida-token=ida_token
```

Scopes

Used for login purposes

```

mosip.scope.resident.getinputattributevalues=Manage-Identity-Data
mosip.scope.resident.patchrevokevid=Manage-VID
mosip.scope.resident.postgeneratevid=Manage-VID
mosip.scope.resident.getvids=Manage-VID
mosip.scope.resident.getAuthTransactions=Manage-Service-Requests
mosip.scope.resident.postAuthTypeUnlock=Manage-Authentication
mosip.scope.resident.postAuthTypeStatus=Manage-Authentication
mosip.scope.resident.getAuthLockStatus=Manage-Authentication
mosip.scope.resident.patchUpdateUin=Manage-Identity-Data
mosip.scope.resident.getServiceAuthHistoryRoles=Manage-Service-Requests
mosip.scope.resident.postSendPhysicalCard=Manage-Credentials
mosip.scope.resident.getUnreadServiceList=Manage-Service-Requests
mosip.scope.resident.getNotificationCount=
mosip.scope.resident.getNotificationClick=Manage-Service-Requests
mosip.scope.resident.getupdatedttimes=Manage-Service-Requests
mosip.scope.resident.postRequestDownloadPersonalizedCard=Manage-Credentials
mosip.scope.resident.postRequestShareCredWithPartner=Manage-Credentials
mosip.scope.resident.postUnPinStatus=Manage-Service-Requests
mosip.scope.resident.postPinStatus=Manage-Service-Requests
mosip.scope.resident.getDownloadCard=Manage-Credentials
mosip.scope.resident.postPersonalizedCard=Manage-Credentials
mosip.scope.resident.get0rderRedirect=Manage-Credentials

```

Key manager encryption/decryption configuration

Properties used to define application and reference id.

```

APPLICATION_Id=RESIDENT
PARTNER_REFERENCE_Id=mpartner-default-resident
mosip.resident.keymanager.application-name=RESIDENT
mosip.resident.keymanager.reference-id=resident_document
mosip.datashare.application.id=PARTNER
mosip.datashare.reference.id=mparter-default-euin
mosip.resident.oidc.keymanager.reference.id=IDP_USER_INFO
mosip.resident.sign.pdf.application.id=KERNEL
mosip.resident.sign.pdf.reference.id=SIGN

```

Object Store configuration

To configure the 'Object Store Configuration', update the 'Object Store URL' and other properties as below:

object.store.s3.url=

```
mosip.resident.object.store.account-name=resident
mosip.resident.object.store.bucket-name=resident
mosip.resident.object.store.adapter-name=s3Adapter
object.store.s3.use.account.as.bucketname=true
object.store.s3.accesskey=*****
object.store.s3.secretkey=*****
object.store.s3.url=http://minio.minio:9000
object.store.s3.region=${s3.region}
object.store.s3.readlimit=10000000
```

Virus Scanner configuration

Property used whether to enable virus scanner flag

```
mosip.resident.virus-scanner.enabled=true
```

VID Policy URL

Property used to get the vid policy json

```
mosip.resident.vid-policy-url=${config.server.file.storage.uri}mosip-vid-
```

Resident UI Schema JSON file

Property used to get the UI schema json

```
resident-ui-schema-file-name-prefix=resident-ui
resident-ui-schema-file-url=${config.server.file.storage.uri}${resident-u
resident-ui-schema-file-source-prefix=url:${resident-ui-schema-file-url}}
```

Identity Mapping JSON file

Property used to get the identity mapping json

```
identity-mapping-file-name=identity-mapping.json  
identity-mapping-file-url=${config.server.file.storage.uri}${identity-map-  
identity-mapping-file-source=url:${identity-mapping-file-url}}
```

Credential Data format MVEL file name

This property is used to get the data format from MVEL file

```
resident-data-format-mvel-file-name=credentialdata.mvel  
resident-data-format-mvel-file-url=${config.server.file.storage.uri}${res-  
resident-data-format-mvel-file-source=url:${resident-data-format-mvel-file-}
```

WebSub Topic and callback properties for auth-type status event

Below websub properties used for authentication type status event

```
resident.websub.authtype-status.secret=*****  
resident.websub.authtype-status.topic=AUTH_TYPE_STATUS_UPDATE_ACK  
resident.websub.callback.authtype-status.relative.url=${server.servlet.co-  
resident.websub.callback.authtype-status.url=${mosip.api.internal.url}${r-}
```

WebSub Topic and callback properties for auth-transaction status event

Below websub properties used for authentication transaction status event

```
resident.websub.authTransaction-status.secret=*****  
resident.websub.authTransaction-status.topic=AUTHENTICATION_TRANSACTION_S-  
resident.websub.callback.authTransaction-status.relative.url=${server.ser-  
resident.websub.callback.authTransaction-status.url=${mosip.api.internal..}
```

WebSub Topic and callback properties for credential status event

Below websub properties used for credential status event

```
resident.websub.credential-status.secret=*****
resident.websub.credential-status.topic=CREDENTIAL_STATUS_UPDATE
resident.websub.callback.credential-status.relative.url=${server.servlet.}
resident.websub.callback.credential-status.url=${mosip.api.internal.url}$
```

WebSub Topic and callback properties for regproc complete workflow event

Below websub properties used for regproc complete workflow event

```
resident.websub.regproc.workflow.complete.secret=*****
mosip.regproc.workflow.complete.topic=REGISTRATION_PROCESSOR_WORKFLOW_COM
resident.websub.callback.regproc.workflow.complete.relative.url=${server.}
resident.websub.callback.regproc.workflow.complete.url=${mosip.api.internal}
```

TokenId generator

```
mosip.kernel.tokenid.uin.salt=*****
mosip.kernel.tokenid.partnercode.salt=*****
```

Mask functions

Properties used to get the data format from MVEL file.

```
resident.email.mask.function=maskEmail
resident.phone.mask.function=maskPhone
resident.data.mask.function=convertToMaskData
```

Batch job configuration for credential status update

```
mosip.resident.update.service.status.job.enabled=false  
mosip.resident.update.service.status.job.initial-delay=60000  
#Interval for checking the credential status for async requests. Note, th  
mosip.resident.update.service.status.job.interval.millisecs=600000
```

Template type codes for email subject

```
resident.template.email.subject.request-received.DOWNLOAD_PERSONALIZED_CARD=download-personalized-card
resident.template.email.subject.success.DOWNLOAD_PERSONALIZED_CARD=cust-a
resident.template.email.subject.failure.DOWNLOAD_PERSONALIZED_CARD=cust-a

resident.template.email.subject.request-received.ORDER_PHYSICAL_CARD=order-a-physical-card
resident.template.email.subject.success.ORDER_PHYSICAL_CARD=order-a-physical-card
resident.template.email.subject.failure.ORDER_PHYSICAL_CARD=order-a-physical-card

resident.template.email.subject.request-received.SHARE_CRED_WITH_PARTNER=share-cred-with-partner
resident.template.email.subject.success.SHARE_CRED_WITH_PARTNER=share-cred-with-partner
resident.template.email.subject.failure.SHARE_CRED_WITH_PARTNER=share-cred-with-partner

resident.template.email.subject.request-received.AUTH_TYPE_LOCK_UNLOCK=lock-unlock
resident.template.email.subject.success.AUTH_TYPE_LOCK_UNLOCK=lock-unlock
resident.template.email.subject.failure.AUTH_TYPE_LOCK_UNLOCK=lock-unlock

resident.template.email.subject.request-received.UPDATE_MY_UIN=update-demo-data
resident.template.email.subject.success.UPDATE_MY_UIN=update-demo-data-success
resident.template.email.subject.failure.UPDATE_MY_UIN=update-demo-data-failure
resident.template.email.subject.regproc-success.UPDATE_MY_UIN=update-demo-data-success
resident.template.email.subject.regproc-failure.UPDATE_MY_UIN=update-demo-data-failure
resident.template.email.subject.cancelled.UPDATE_MY_UIN=update-demo-data-cancelled

resident.template.email.subject.request-received.GENERATE_VID=gen-or-revoke-vid
resident.template.email.subject.success.GENERATE_VID=gen-or-revoke-vid-success
resident.template.email.subject.failure.GENERATE_VID=gen-or-revoke-vid-failure

resident.template.email.subject.request-received.REVOKE_VID=gen-or-revoke-vid
resident.template.email.subject.success.REVOKE_VID=gen-or-revoke-vid-success
resident.template.email.subject.failure.REVOKE_VID=gen-or-revoke-vid-failure

resident.template.email.subject.request-received.VID_CARD_DOWNLOAD=vid-card-download
resident.template.email.subject.success.VID_CARD_DOWNLOAD=vid-card-download-success
resident.template.email.subject.failure.VID_CARD_DOWNLOAD=vid-card-download-failure

resident.template.email.subject.request-received.GET_MY_ID=get-my-uin-card
resident.template.email.subject.success.GET_MY_ID=get-my-uin-card-success
resident.template.email.subject.failure.GET_MY_ID=get-my-uin-card-failure

resident.template.email.subject.request-received.VALIDATE_OTP=verify-my-phone-email
resident.template.email.subject.success.VALIDATE_OTP=verify-my-phone-email-success
resident.template.email.subject.failure.VALIDATE_OTP=verify-my-phone-email-failure

resident.template.email.subject.success.SEND_OTP=receive-otp-mail-subject
```

Template type codes for email content

```
resident.template.email.content.request-received.DOWNLOAD_PERSONALIZED_CARD=download-personalized-card
resident.template.email.content.success.DOWNLOAD_PERSONALIZED_CARD=cust-a
resident.template.email.content.failure.DOWNLOAD_PERSONALIZED_CARD=cust-a

resident.template.email.content.request-received.ORDER_PHYSICAL_CARD=order-a-physical-card
resident.template.email.content.success.ORDER_PHYSICAL_CARD=order-a-physical-card
resident.template.email.content.failure.ORDER_PHYSICAL_CARD=order-a-physical-card

resident.template.email.content.request-received.SHARE_CRED_WITH_PARTNER=share-cred-with-partner
resident.template.email.content.success.SHARE_CRED_WITH_PARTNER=share-cred-with-partner
resident.template.email.content.failure.SHARE_CRED_WITH_PARTNER=share-cred-with-partner

resident.template.email.content.request-received.AUTH_TYPE_LOCK_UNLOCK=lock-unlock
resident.template.email.content.success.AUTH_TYPE_LOCK_UNLOCK=lock-unlock
resident.template.email.content.failure.AUTH_TYPE_LOCK_UNLOCK=lock-unlock

resident.template.email.content.request-received.UPDATE_MY_UIN=update-demo-data
resident.template.email.content.success.UPDATE_MY_UIN=update-demo-data-success
resident.template.email.content.failure.UPDATE_MY_UIN=update-demo-data-failure
resident.template.email.content.regproc-success.UPDATE_MY_UIN=update-demo-data-success
resident.template.email.content.regproc-failure.UPDATE_MY_UIN=update-demo-data-failure
resident.template.email.content.cancelled.UPDATE_MY_UIN=update-demo-data-cancelled

resident.template.email.content.request-received.GENERATE_VID=gen-or-revoke-vid
resident.template.email.content.success.GENERATE_VID=gen-or-revoke-vid-success
resident.template.email.content.failure.GENERATE_VID=gen-or-revoke-vid-failure

resident.template.email.content.request-received.REVOKE_VID=gen-or-revoke-vid
resident.template.email.content.success.REVOKE_VID=gen-or-revoke-vid-success
resident.template.email.content.failure.REVOKE_VID=gen-or-revoke-vid-failure

resident.template.email.content.request-received.VID_CARD_DOWNLOAD=vid-card-download
resident.template.email.content.success.VID_CARD_DOWNLOAD=vid-card-download-success
resident.template.email.content.failure.VID_CARD_DOWNLOAD=vid-card-download-failure

resident.template.email.content.request-received.GET_MY_ID=get-my-uin-card
resident.template.email.content.success.GET_MY_ID=get-my-uin-card-success
resident.template.email.content.failure.GET_MY_ID=get-my-uin-card-failure

resident.template.email.content.request-received.VALIDATE_OTP=verify-my-phone-email
resident.template.email.content.success.VALIDATE_OTP=verify-my-phone-email-success
resident.template.email.content.failure.VALIDATE_OTP=verify-my-phone-email-failure

resident.template.email.content.success.SEND_OTP=receive-otp-mail-content
```

Template type codes for SMS content

```
resident.template.sms.request-received.DOWNLOAD_PERSONALIZED_CARD=cust-and-down-my  
resident.template.sms.success.DOWNLOAD_PERSONALIZED_CARD=cust-and-down-my  
resident.template.sms.failure.DOWNLOAD_PERSONALIZED_CARD=cust-and-down-my  
  
resident.template.sms.request-received.ORDER_PHYSICAL_CARD=order-a-physical-card  
resident.template.sms.success.ORDER_PHYSICAL_CARD=order-a-physical-card-success_SMS  
resident.template.sms.failure.ORDER_PHYSICAL_CARD=order-a-physical-card-failure_SMS  
  
resident.template.sms.request-received.SHARE_CRED_WITH_PARTNER=share-cred-with-partner  
resident.template.sms.success.SHARE_CRED_WITH_PARTNER=share-cred-with-partner-success_SMS  
resident.template.sms.failure.SHARE_CRED_WITH_PARTNER=share-cred-with-partner-failure_SMS  
  
resident.template.sms.request-received.AUTH_TYPE_LOCK_UNLOCK=lock-unlock-auth  
resident.template.sms.success.AUTH_TYPE_LOCK_UNLOCK=lock-unlock-auth-success_SMS  
resident.template.sms.failure.AUTH_TYPE_LOCK_UNLOCK=lock-unlock-auth-failure_SMS  
  
resident.template.sms.request-received.UPDATE_MY_UIN=update-demo-data-request  
resident.template.sms.success.UPDATE_MY_UIN=update-demo-data-success_SMS  
resident.template.sms.failure.UPDATE_MY_UIN=update-demo-data-failure_SMS  
resident.template.sms.regproc-success.UPDATE_MY_UIN=update-demo-data-regproc-success_SMS  
resident.template.sms.regproc-failure.UPDATE_MY_UIN=update-demo-data-regproc-failure_SMS  
resident.template.sms.cancelled.UPDATE_MY_UIN=update-demo-data-discarded-SMS  
  
resident.template.sms.request-received.GENERATE_VID=gen-or-revoke-vid-request  
resident.template.sms.success.GENERATE_VID=gen-or-revoke-vid-success_SMS  
resident.template.sms.failure.GENERATE_VID=gen-or-revoke-vid-failure_SMS  
  
resident.template.sms.request-received.REVOKE_VID=gen-or-revoke-vid-request  
resident.template.sms.success.REVOKE_VID=gen-or-revoke-vid-success_SMS  
resident.template.sms.failure.REVOKE_VID=gen-or-revoke-vid-failure_SMS  
  
resident.template.sms.request-received.VID_CARD_DOWNLOAD=vid-card-download-request  
resident.template.sms.success.VID_CARD_DOWNLOAD=vid-card-download-success_SMS  
resident.template.sms.failure.VID_CARD_DOWNLOAD=vid-card-download-failure_SMS  
  
resident.template.sms.request-received.GET_MY_ID=get-my-uin-card-request  
resident.template.sms.success.GET_MY_ID=get-my-uin-card-success_SMS  
resident.template.sms.failure.GET_MY_ID=get-my-uin-card-failure_SMS  
  
resident.template.sms.request-received.VALIDATE_OTP=verify-my-phone-email  
resident.template.sms.success.VALIDATE_OTP=verify-my-phone-email-success_SMS  
resident.template.sms.failure.VALIDATE_OTP=verify-my-phone-email-failure_SMS  
  
resident.template.sms.success.SEND_OTP=receive-otp
```

Template type codes for purpose (success) content

```
resident.template.purpose.success.DOWNLOAD_PERSONALIZED_CARD=cust-and-down  
resident.template.purpose.success.ORDER_PHYSICAL_CARD=order-a-physical-ca  
resident.template.purpose.success.SHARE_CRED_WITH_PARTNER=share-cred-with  
resident.template.purpose.success.AUTH_TYPE_LOCK_UNLOCK=lock-unlock-auth-t  
resident.template.purpose.success.UPDATE_MY_UIN=update-demo-data-positive  
resident.template.purpose.success.GENERATE_VID=gen-or-revoke-vid-positive  
resident.template.purpose.success.REVOKE_VID=gen-or-revoke-vid-positive-p  
resident.template.purpose.success.GET_MY_ID=get-my-uin-card-positive-purpos  
resident.template.purpose.success.VALIDATE_OTP=verify-my-phone-email-positive  
resident.template.purpose.success.VID_CARD_DOWNLOAD=vid-card-download-posi
```

Template type codes for purpose (in-progress/failure) content

```
resident.template.purpose.failure.DOWNLOAD_PERSONALIZED_CARD=cust-and-down  
resident.template.purpose.failure.ORDER_PHYSICAL_CARD=order-a-physical-ca  
resident.template.purpose.failure.SHARE_CRED_WITH_PARTNER=share-cred-with  
resident.template.purpose.failure.AUTH_TYPE_LOCK_UNLOCK=lock-unlock-auth-t  
resident.template.purpose.failure.UPDATE_MY_UIN=update-demo-data-negative  
resident.template.purpose.failure.GENERATE_VID=gen-or-revoke-vid-negative  
resident.template.purpose.failure.REVOKE_VID=gen-or-revoke-vid-negative-p  
resident.template.purpose.failure.GET_MY_ID=get-my-uin-card-negative-purpos  
resident.template.purpose.failure.VALIDATE_OTP=verify-my-phone-email-negative  
resident.template.purpose.failure.VID_CARD_DOWNLOAD=vid-card-download-negative  
resident.template.purpose.failure.AUTHENTICATION_REQUEST=authentication-request
```

Template type codes for purpose (Cancelled) content

```
resident.template.purpose.cancelled.UPDATE_MY_UIN=update-demo-data-cancel
```

Template type codes for purpose(Identity updated) content

```
resident.template.purpose.regproc-success.UPDATE_MY_UIN=update-demo-data-
```

Template type codes for summary (success) content

```
resident.template.summary.success.DOWNLOAD_PERSONALIZED_CARD=cust-and-download
resident.template.summary.success.ORDER_PHYSICAL_CARD=order-a-physical-card
resident.template.summary.success.SHARE_CRED_WITH_PARTNER=share-cred-with-partner
resident.template.summary.success.AUTH_TYPE_LOCK_UNLOCK=lock-unlock-authentication-type
resident.template.summary.success.UPDATE_MY_UIN=update-demo-data-successfully
resident.template.summary.success.GENERATE_VID=gen-or-revoke-vid-successfully
resident.template.summary.success.REVOKE_VID=gen-or-revoke-vid-successfully
resident.template.summary.success.GET_MY_ID=get-my-uin-card-successfully
resident.template.summary.success.VALIDATE OTP=verify-my-phone-email-successfully
resident.template.summary.success.VID_CARD_DOWNLOAD=vid-card-download-successfully
resident.template.summary.success.AUTHENTICATION_REQUEST=authentication-request
```

Template type code for summary (cancelled) content

```
resident.template.summary.cancelled.UPDATE_MY_UIN=update-demo-data-cancelled
```

Template type code for summary (regproc-success) content

```
resident.template.summary.regproc-success.UPDATE_MY_UIN=update-demo-data-successfully
```

Template type codes for acknowledgement PDFs

```
resident.template.ack.share-cred-with-partner=acknowledgement-share-cred-with-partner
resident.template.ack.manage-my-vid=acknowledgement-manage-my-vid
resident.template.ack.order-a-physical-card=acknowledgement-order-a-physical-card
resident.template.ack.download-a-personalized-card=acknowledgement-download-a-personalized-card
resident.template.ack.update-demographic-data=acknowledgement-update-demographic-data
resident.template.ack.verify-email-id-or-phone-number=acknowledgement-verify-email-id-or-phone-number
resident.template.ack.secure-my-id=acknowledgement-secure-my-id
resident.template.ack.authentication.request=acknowledgment-authentication-request
resident.template.ack.get.my.id=acknowledgment-get-my-id
resident.template.ack.vid.card.download=acknowledgment-vid-card-download
```

Template type codes for supporting documents, service history, registration centers and vid card

```
resident.template.support-docs-list=supporting-docs-list
mosip.resident.service.history.template.type.code=service-history-type
resident.template.registration.centers.list=registration-centers-list
mosip.resident.vid.card.template.property=vid-card-type
```

Template required properties

```
resident.template.date.pattern=dd-MM-yyyy
resident.template.time.pattern=HH:mm:ss
resident.ui.track-service-request-url=https://{$mosip.resident.host}/#/ui/track-service-request
```

View history filters

```
resident.view.history.serviceType.filters=ALL,AUTHENTICATION_REQUEST,SERVICE_REQUEST
resident.view.history.status.filters=ALL,SUCCESS,IN_PROGRESS,FAILED,CANCELED
```

Maximum data to download in a PDF

```
resident.service-history.download.max.count=100
resident.registration-centers.download.max.count=100
```

The Registration centers will be searched based on the distance value in meters from the Geo location identified

```
resident.nearby.centers.distance.meters=2000
```

Page size in Bell Icon Notification list and view history

```
resident.notifications.default.page.size=100  
resident.view-history.default.page.size=10
```

Token related config

```
auth.validate.id-token=true  
idToken=id_token  
auth.token.header=Authorization  
mosip.resident.access_token.auth_mode.claim-name=acr  
mosip.resident.oidc.id_token.ida_token.claim-name=sub  
mosip.resident.oidc.auth_token.expiry.claim-name=exp  
mosip.resident.oidc userinfo.encryption.enabled=false  
mosip.client.assertion.reference.id=  
mosip.include.payload=true  
mosip.include.certificate=true  
mosip.include.cert.hash=false
```

Rectangle coordinates for PDF signed data

```
mosip.resident.service.uincard.lowerleftx=73  
mosip.resident.service.uincard.lowerlefty=100  
mosip.resident.service.uincard.upperrightx=300  
mosip.resident.service.uincard.upperrighty=300  
mosip.resident.service.uincard.signature.reason="Digitally Signed"
```

File name for the downloaded PDFs

```
mosip.resident.download.registration.centre.file.name.convention=RegistrationCentre  
mosip.resident.download.supporting.document.file.name.convention=SupportingDocument  
mosip.resident.download.personalized.card.naming.convention=PersonalisedCard  
mosip.resident.ack.manage_my_vid.name.convention=Ack_{featureName}_{eventId}_{timestamp}  
mosip.resident.ack.secure_my_id.name.convention=Ack_{featureName}_{eventId}_{timestamp}  
mosip.resident.ack.personalised_card.name.convention=Ack_{featureName}_{eventId}_{timestamp}  
mosip.resident.ack.update_my_data.name.convention=Ack_{featureName}_{eventId}_{timestamp}  
mosip.resident.ack.share_credential.name.convention=Ack_{featureName}_{eventId}_{timestamp}  
mosip.resident.ack.order_physical_card.name.convention=Ack_{featureName}_{eventId}_{timestamp}  
mosip.resident.ack.name.convention=Ack_{featureName}_{eventId}_{timestamp}  
mosip.resident.uin.card.name.convention=UIN_{eventId}_{timestamp}  
mosip.resident.vid.card.name.convention=VID_{eventId}_{timestamp}  
mosip.resident.download.service.history.file.name.convention=View_history  
mosip.resident.download.nearest.registration.centre.file.name.convention=NearestCentre  
mosip.resident.download.card.naming.convention=Get_my_UIN_{timestamp}
```

Credential request config for sharing credential to partner

```
mosip.resident.request.credential.credentialType=vercred  
mosip.resident.request.credential.credentialType=euin  
mosip.resident.request.credential.isEncrypt=true  
mosip.resident.request.credential.encryption.key=*****  
  
mosip.digital.card.credential.type=PDFCard  
mosip.credential.issuer=*****
```

Claim names

```
mosip.resident.name.token.claim-name=name  
mosip.resident.photo.token.claim-photo=picture  
mosip.resident.individual.id.claim.name=individual_id  
mosip.resident.email.token.claim-email=email  
mosip.resident.phone.token.claim-phone=phone_number
```

Value based properties

```
otpChannel.email=email  
otpChannel.mobile=phone  
mosip.idrepo.vid.reactive-status=ACTIVE  
resident.dateofbirth.pattern=yyyy/MM/dd  
mosip.resident.photo.attribute.name=photo  
mosip.resident.order.card.payment.enabled=true  
resident.update.preferred.language.by.name=true  
resident.documents.category=individualBiometrics  
mosip.resident.schema.attribute-name=attributeName  
mosip.resident.applicant.name.property=applicantName  
mosip.resident.authentication.mode.property=authenticationMode  
resident.attribute.names.without.documents.required=preferredLanguage,ema  
resident.additional.identity.attribute.to.fetch=UIN,email,phone,dateOfBirth
```

OTP Flooding

Configure Time limit for OTP Flooding scenario (in minutes).

```
otp.request.flooding.duration=1  
otp.request.flooding.max-count=10
```

Maximum file size and types for uploading document

```
mosip.max.file.upload.size.in.bytes=2306867  
mosip.allowed.extension=pdf,jpeg,png,jpg
```

Reg-proc packet status codes

```
resident.success.packet-status-code.list=PROCESSED,SUCCESS,UIN_GENERATED  
resident.in-progress.packet-status-code.list=PROCESSING,REREGISTER,RESEND  
resident.failure.packet-status-code.list=REJECTED,FAILED,REPROCESS_FAILED
```

Reg-proc packet transaction type codes

```
resident.REQUEST RECEIVED.packet-transaction-type-code.list=PACKET RECEIVED
resident.VALIDATION_STAGE.packet-transaction-type-code.list=CMD_VALIDATION
resident.VERIFICATION_STAGE.packet-transaction-type-code.list=DEMOGRAPHIC
resident.UIN_GENERATION_STAGE.packet-transaction-type-code.list=UIN_GENERATION
resident.CARD_READY_TO_DOWNLOAD.packet-transaction-type-code.list=PRINT_SHEET
```

Sequence order of reg-proc transaction type codes

sequence-order=Request received, Validation stage, Verification stage, Uin generation, Card ready to download

Synchronous events

```
resident.request.success.status.list.AUTHENTICATION_REQUEST=AUTHENTICATION  
resident.request.failed.status.list.AUTHENTICATION_REQUEST=AUTHENTICATION_FAILED  
resident.request.cancelled.status.list.AUTHENTICATION_REQUEST=AUTHENTICATION_FAILED  
  
resident.request.new.status.list.DOWNLOAD_PERSONALIZED_CARD=NEW  
resident.batchjob.process.success.status.list.DOWNLOAD_PERSONALIZED_CARD=NEW  
resident.request.failed.status.list.DOWNLOAD_PERSONALIZED_CARD=FAILED  
resident.request.cancelled.status.list.DOWNLOAD_PERSONALIZED_CARD=FAILED  
  
resident.request.new.status.list.GET_MY_ID=NEW  
resident.request.in-progress.status.list.GET_MY_ID=OTP_REQUESTED  
resident.request.success.status.list.GET_MY_ID=CARD_DOWNLOADED,OTP_VERIFIED  
resident.request.failed.status.list.GET_MY_ID=FAILED  
resident.request.cancelled.status.list.GET_MY_ID=FAILED  
  
resident.request.new.status.list.BOOK_AN_APPOINTMENT=  
resident.request.success.status.list.BOOK_AN_APPOINTMENT=  
resident.request.failed.status.list.BOOK_AN_APPOINTMENT=  
resident.request.cancelled.status.list.BOOK_AN_APPOINTMENT=  
  
resident.request.new.status.list.GENERATE_VID=NEW  
resident.request.success.status.list.GENERATE_VID=VID_GENERATED  
resident.request.failed.status.list.GENERATE_VID=FAILED  
resident.request.cancelled.status.list.GENERATE_VID=FAILED  
  
resident.request.new.status.list.REVOKE_VID=NEW  
resident.request.success.status.list.REVOKE_VID=VID_REVOKED  
resident.request.failed.status.list.REVOKE_VID=FAILED  
resident.request.cancelled.status.list.REVOKE_VID=FAILED  
  
resident.request.new.status.list.SEND OTP=  
resident.request.success.status.list.SEND OTP=  
resident.request.failed.status.list.SEND OTP=  
resident.request.cancelled.status.list.SEND OTP=  
  
resident.request.new.status.list.VALIDATE OTP=OTP REQUESTED  
resident.request.success.status.list.VALIDATE OTP=OTP VERIFIED  
resident.request.failed.status.list.VALIDATE OTP=OTP VERIFICATION FAILED  
resident.request.cancelled.status.list.VALIDATE OTP=FAILED  
  
resident.request.new.status.list.DEFAULT=  
resident.request.success.status.list.DEFAULT=  
resident.request.failed.status.list.DEFAULT=  
resident.request.cancelled.status.list.DEFAULT=
```

Asyc Request Types

```
resident.async.request.types=VID_CARD_DOWNLOAD,ORDER_PHYSICAL_CARD,SHARE_CRED,...
```

Asynchronous events

```
resident.request.new.status.list.SHARE_CRED_WITH_PARTNER=NEW
resident.request.in-progress.status.list.SHARE_CRED_WITH_PARTNER=ISSUED
resident.request.success.status.list.SHARE_CRED_WITH_PARTNER=RECEIVED,DATA_UPLOADED
resident.request.failed.status.list.SHARE_CRED_WITH_PARTNER=FAILED
resident.request.notification.status.list.SHARE_CRED_WITH_PARTNER=FAILED,NOTIFICATION_SENT

resident.request.new.status.list.ORDER_PHYSICAL_CARD=NEW
resident.request.in-progress.status.list.ORDER_PHYSICAL_CARD=PAYOUTMENT_CONFIRMED
resident.request.success.status.list.ORDER_PHYSICAL_CARD=CARD_DELIVERED
resident.request.failed.status.list.ORDER_PHYSICAL_CARD=FAILED,PAYOUTMENT_FAILED
resident.request.notification.status.list.ORDER_PHYSICAL_CARD=PAYOUTMENT_CONFIRMED,NOTIFICATION_SENT

resident.request.new.status.list.UPDATE_MY_UIN=NEW
resident.request.in-progress.status.list.UPDATE_MY_UIN=PROCESSING,PAUSED,RETRYING
resident.request.success.status.list.UPDATE_MY_UIN=PROCESSED,DATA_UPDATED
resident.request.failed.status.list.UPDATE_MY_UIN=FAILED,REJECTED,REPROCESSED
resident.request.notification.status.list.UPDATE_MY_UIN=PROCESSED,DATA_UPDATED,NOTIFICATION_SENT

resident.request.new.status.list.AUTH_TYPE_LOCK_UNLOCK=NEW
resident.request.in-progress.status.list.AUTH_TYPE_LOCK_UNLOCK=LOCKED
resident.request.success.status.list.AUTH_TYPE_LOCK_UNLOCK=COMPLETED
resident.request.failed.status.list.AUTH_TYPE_LOCK_UNLOCK=FAILED
resident.request.notification.status.list.AUTH_TYPE_LOCK_UNLOCK=COMPLETED

resident.request.new.status.list.VID_CARD_DOWNLOAD=NEW
resident.request.in-progress.status.list.VID_CARD_DOWNLOAD=ISSUED
resident.request.success.status.list.VID_CARD_DOWNLOAD=STORED,CARD_READY_TO_USE
resident.request.failed.status.list.VID_CARD_DOWNLOAD=FAILED
resident.request.notification.status.list.VID_CARD_DOWNLOAD=STORED,CARD_READY_TO_USE,NOTIFICATION_SENT
```

Attributes name based template type codes

Define property name in below format-

resident.<attribute name>.template.property.attribute.list

```
resident.PHONE.template.property.attribute.list=mosip.phone.template.prop  
resident.EMAIL.template.property.attribute.list=mosip.email.template.prop  
resident.GENERATE_VID.template.property.attribute.list=mosip.generated.tem  
resident.REVOKE_VID.template.property.attribute.list=mosip.revoked.templa
```

Template type codes for event status code

```
resident.event.status.SUCCESS.template.property=mosip.event.status.succes  
resident.event.status.FAILED.template.property=mosip.event.status.failed.  
resident.event.status.IN_PROGRESS.template.property=mosip.event.status.in  
resident.event.status.CANCELED.template.property=mosip.event.status.cance
```

Template type codes for event types

Define property name in below format-

```
resident.event.type.<eventType>.template.property
```

```
resident.event.type.AUTHENTICATION_REQUEST.template.property=mosip.event.  
resident.event.type.SHARE_CRED_WITH_PARTNER.template.property=mosip.event  
resident.event.type.DOWNLOAD_PERSONALIZED_CARD.template.property=mosip.eve  
resident.event.type.ORDER_PHYSICAL_CARD.template.property=mosip.event.type  
resident.event.type.GET_MY_ID.template.property=mosip.event.type.GET_MY_ID  
resident.event.type.UPDATE_MY_UIN.template.property=mosip.event.type.UPDA  
resident.event.type.GENERATE_VID.template.property=mosip.event.type.GENER  
resident.event.type.REVOKE_VID.template.property=mosip.event.type.REVOKE_  
resident.event.type.AUTH_TYPE_LOCK_UNLOCK.template.property=mosip.event.t  
resident.event.type.VID_CARD_DOWNLOAD.template.property=mosip.event.type.  
resident.event.type.SEND OTP.template.property=mosip.event.type.SEND OTP  
resident.event.type.VALIDATE OTP.template.property=mosip.event.type.VALID  
resident.event.type.DEFAULT.template.property=mosip.event.type.DEFAULT
```

Template type codes for service types

Define property name in below format-

```
resident.service-type.<serviceType>.template.property
```

```
resident.service-type.AUTHENTICATION_REQUEST.template.property=mosip.servi  
resident.service-type.SERVICE_REQUEST.template.property=mosip.service.type  
resident.service-type.DATA_UPDATE_REQUEST.template.property=mosip.service.  
resident.service-type.ID_MANAGEMENT_REQUEST.template.property=mosip.service.  
resident.service-type.DATA_SHARE_REQUEST.template.property=mosip.service.  
resident.service-type.ASYNC.template.property=mosip.service.type.ASYNC
```

Template type codes for id-authentication request types description

Define property name in below format-

```
resident.id-auth.request-type.<authTypeCode>.<statusCode>.descr
```

```
resident.id-auth.request-type.OTP-REQUEST.SUCCESS.descr=mosip.ida.auth-re  
resident.id-auth.request-type.OTP-AUTH.SUCCESS.descr=mosip.ida.auth-reque  
resident.id-auth.request-type.DEMO-AUTH.SUCCESS.descr=mosip.ida.auth-reque  
resident.id-auth.request-type.FINGERPRINT-AUTH.SUCCESS.descr=mosip.ida.auth  
resident.id-auth.request-type.IRIS-AUTH.SUCCESS.descr=mosip.ida.auth-reque  
resident.id-auth.request-type.FACE-AUTH.SUCCESS.descr=mosip.ida.auth-reque  
resident.id-auth.request-type.STATIC-PIN-AUTH.SUCCESS.descr=mosip.ida.auth  
resident.id-auth.request-type.STATIC-PIN-STORAGE.SUCCESS.descr=mosip.ida.a  
resident.id-auth.request-type.EKYC-AUTH.SUCCESS.descr=mosip.ida.auth-reque  
resident.id-auth.request-type.KYC-AUTH.SUCCESS.descr=mosip.ida.auth-reque  
resident.id-auth.request-type.KYC-EXCHANGE.SUCCESS.descr=mosip.ida.auth-re  
resident.id-auth.request-type.IDENTITY-KEY-BINDING.SUCCESS.descr=mosip.id  
resident.id-auth.request-type.TOKEN-REQUEST.SUCCESS.descr=mosip.ida.auth-req  
resident.id-auth.request-type.TOKEN-AUTH.SUCCESS.descr=mosip.ida.auth-reque  
resident.id-auth.request-type.UNKNOWN.SUCCESS.descr=mosip.ida.auth-reques  
resident.id-auth.request-type.OTP-REQUEST FAILED.descr=mosip.ida.auth-requ  
resident.id-auth.request-type.OTP-AUTH FAILED.descr=mosip.ida.auth-reques  
resident.id-auth.request-type.DEMO-AUTH FAILED.descr=mosip.ida.auth-reques  
resident.id-auth.request-type.FINGERPRINT-AUTH FAILED.descr=mosip.ida.auth  
resident.id-auth.request-type.IRIS-AUTH FAILED.descr=mosip.ida.auth-reques  
resident.id-auth.request-type.FACE-AUTH FAILED.descr=mosip.ida.auth-reques  
resident.id-auth.request-type.STATIC-PIN-AUTH FAILED.descr=mosip.ida.auth  
resident.id-auth.request-type.STATIC-PIN-STORAGE FAILED.descr=mosip.ida.a  
resident.id-auth.request-type.EKYC-AUTH FAILED.descr=mosip.ida.auth-reques  
resident.id-auth.request-type.KYC-AUTH FAILED.descr=mosip.ida.auth-reques  
resident.id-auth.request-type.KYC-EXCHANGE FAILED.descr=mosip.ida.auth-re  
resident.id-auth.request-type.IDENTITY-KEY-BINDING FAILED.descr=mosip.id  
resident.id-auth.request-type.TOKEN-REQUEST FAILED.descr=mosip.ida.auth-req  
resident.id-auth.request-type.TOKEN-AUTH FAILED.descr=mosip.ida.auth-reque  
resident.id-auth.request-type.UNKNOWN FAILED.descr=mosip.ida.auth-request
```

Template type codes for authentication modes (authTypeCode)

Define property name in below format-

```
resident.auth-type-code.<authTypeCode>.code
```

```
resident.auth-type-code.OTP-REQUEST.code=mosip.auth-type-code.OTP-REQUEST  
resident.auth-type-code.OTP-AUTH.code=mosip.auth-type-code.OTP-AUTH  
resident.auth-type-code.DEMO-AUTH.code=mosip.auth-type-code.DEMO-AUTH  
resident.auth-type-code.FINGERPRINT-AUTH.code=mosip.auth-type-code.FINGERPRINT-AUTH  
resident.auth-type-code.IRIS-AUTH.code=mosip.auth-type-code.IRIS-AUTH  
resident.auth-type-code.FACE-AUTH.code=mosip.auth-type-code.FACE-AUTH  
resident.auth-type-code.STATIC-PIN-AUTH.code=mosip.auth-type-code.STATIC-PIN-AUTH  
resident.auth-type-code.STATIC-PIN-STORAGE.code=mosip.auth-type-code.STATIC-PIN-STORAGE  
resident.auth-type-code.EKYC-AUTH.code=mosip.auth-type-code.EKYC-AUTH  
resident.auth-type-code.KYC-AUTH.code=mosip.auth-type-code.KYC-AUTH  
resident.auth-type-code.KYC-EXCHANGE.code=mosip.auth-type-code.KYC-EXCHANGE  
resident.auth-type-code.IDENTITY-KEY-BINDING.code=mosip.auth-type-code.IDENTITY-KEY-BINDING  
resident.auth-type-code.TOKEN-REQUEST.code=mosip.auth-type-code.TOKEN-REQUEST  
resident.auth-type-code.TOKEN-AUTH.code=mosip.auth-type-code.TOKEN-AUTH  
resident.auth-type-code.PWD.code=mosip.auth-type-code.PWD  
resident.auth-type-code.PIN.code=mosip.auth-type-code.PIN  
resident.auth-type-code.OTP.code=mosip.auth-type-code.OTP  
resident.auth-type-code.Wallet.code=mosip.auth-type-code.Wallet  
resident.auth-type-code.L1-bio-device.code=mosip.auth-type-code.L1-bio-device
```

Flag to retrieve UIN or VID

Below property will retrieve VID when requested. Default is false so, UIN will be retrieved. Endpoints using below property- /individualId/otp, /aid/status.

```
resident.flag.use-vid-only=true
```

Class name of the referenceValidator

Commenting or removing this property will disable reference validator.

```
mosip.kernel.idobjectvalidator.referenceValidator=io.mosip.kernel.idobjec
```

Request time validation

For validating request time as per before & after time limit (in seconds) in contact-details/update API.

```
resident.future.time.limit=60  
resident.past.time.limit=60
```

Date time formatting styles

The `java.time.format.FormatStyle` enum to use for date time formatting based on locale. Allowed values with examples are:

- FULL ('Tuesday, April 12, 1952 AD' or '3:30:42pm PST')
- LONG('January 12, 1952')
- MEDIUM ('Jan 12, 1952')
- SHORT ('12.13.52' or '3:30pm')

Current value is MEDIUM. For more details refer to the enum.

```
resident.date.time.formmatting.style=MEDIUM  
resident.date.time.replace.special.chars={" ":"_", ",":", ":" :".",""}
```

Cache expiration time (in milliseconds)

```
resident.cache.expiry.time.millisec.templateCache=86400000
resident.cache.expiry.time.millisec.partnerCache=86400000
resident.cache.expiry.time.millisec.getValidDocumentByLangCode=86400000
resident.cache.expiry.time.millisec.getLocationHierarchyLevelByLangCode=86400000
resident.cache.expiry.time.millisec.getImmediateChildrenByLocCodeAndLangCode=86400000
resident.cache.expiry.time.millisec.getLocationDetailsByLocCodeAndLangCode=86400000
resident.cache.expiry.time.millisec.getCoordinateSpecificRegistrationCenter=86400000
resident.cache.expiry.time.millisec.getApplicantValidDocument=86400000
resident.cache.expiry.time.millisec.getRegistrationCentersByHierarchyLevel=86400000
resident.cache.expiry.time.millisec.getRegistrationCenterByHierarchyLevel=86400000
resident.cache.expiry.time.millisec.getRegistrationCenterWorkingDays=86400000
resident.cache.expiry.time.millisec.getLatestIdSchema=86400000
resident.cache.expiry.time.millisec.getGenderCodeByGenderTypeAndLangCode=86400000
resident.cache.expiry.time.millisec.getDocumentTypesByDocumentCategoryAndLangCode=86400000
resident.cache.expiry.time.millisec.getDynamicFieldBasedOnLangCodeAndFieldCategory=86400000
resident.cache.expiry.time.millisec.getCenterDetails=86400000
resident.cache.expiry.time.millisec.getImmediateChildrenByLocCode=86400000
resident.cache.expiry.time.millisec.getLocationHierarchyLevels=86400000
resident.cache.expiry.time.millisec.getAllDynamicFieldByName=86400000
resident.cache.expiry.time.millisec.getNameValueFromIdentityMapping=86400000
```

Separators

Usage: resident.attribute.separator.<attribute>=<separator string>

```
resident.attribute.separator.fullAddress=,
```

Thread properties

Async thread for audit calls. Limit the number of async threads created in Resident services. This count is divided into 4 thread groups configured in 'io.mosip.resident.config.Config' class.

```
mosip.resident.async-core-pool-size=100
mosip.resident.async-max-pool-size=100
```

Logo property

This property is used in all downloaded PDF files.

```
mosip.pdf.header.logo.url=https://mosip.io/images/mosipn-logo.png
```

Map (zoom in & out)

These properties are used in reg-center feature for map zoom in & out.

```
mosip.resident.zoom=14  
mosip.resident.maxZoom=18  
mosip.resident.minZoom=5
```

Transliteration workaround properties

Transliteration work around property since eng to fra directly is not supported in icu4j. This can be added for any other unsupported language also.

For example, resident-transliteration-workaround-for-<fromLanguageCode>-<toLanguageCode> = fromLanguageCode-intermediateLanguageCode-toLanguageCode.

For this, intermediate language code transliteration should work in both ways.

```
resident-transliteration-workaround-for-eng-fra=eng-hin,hin-fra  
resident-transliteration-workaround-for-eng-spa=eng-hin,hin-spa
```

Reg-processer-credential-partner-policy-url

This is a policy url to fetch delimiter to download card after updating uin.

```
mosip.resident.reg-processer-credential-partner-policy-url=${config.serve
```

The request IDs used in Resident REST APIs

```
mosip.resident.api.id.otp.request=mosip.identity.otp.internal
mosip.resident.api.id.auth=mosip.identity.auth.internal
auth.internal.id=mosip.identity.auth.internal
mosip.registration.processor.print.id=mosip.registration.print
vid.create.id=mosip.vid.create
resident.vid.id=mosip.resident.vid
resident.vid.id.generate=mosip.resident.vid.generate
resident.vid.policy.id=mosip.resident.vid.policy
resident.vid.get.id=mosip.resident.vid.get
auth.type.status.id=mosip.identity.authtype.status.update
resident.authlock.id=mosip.resident.authlock
resident.checkstatus.id=mosip.resident.checkstatus
resident.euin.id=mosip.resident.euin
resident.printuin.id=mosip.resident.printuin
resident.uin.id=mosip.resident.uin
resident.rid.id=mosip.resident.rid
resident.updateuin.id=mosip.resident.updateuin
resident.authunlock.id=mosip.resident.authunlock
resident.authhistory.id=mosip.resident.authhistory
resident.authLockStatusUpdateV2.id=mosip.resident.auth.lock.unlock
resident.service.history.id=mosip.service.history.get
resident.document.upload.id=mosip.resident.document.upload
resident.document.get.id=mosip.resident.document.get
resident.document.list.id=mosip.resident.document.list
resident.service.pin.status.id=mosip.resident.pin.status
resident.service.unpin.status.id=mosip.resident.unpin.status
resident.document.delete.id=mosip.resident.document.delete
resident.contact.details.update.id=mosip.resident.contact.details.update
resident.contact.details.send.otp.id=mosip.resident.contact.details.send.otp
mosip.resident.service.status.check.id=mosip.registration.external.status
resident.service.unreadnotificationlist.id=mosip.resident.service.history
resident.service.event.id=mosip.resident.event.status
resident.identity.info.id=mosip.resident.identity.info
resident.share.credential.id=mosip.resident.share.credential
vid.revoke.id=mosip.vid.update
resident.revokevid.id=mosip.resident.vidstatus
mosip.resident.revokevid.id=mosip.resident.vid.revoke
mosip.resident.grievance.ticket.request.id=mosip.resident.grievance.ticket.request
resident.channel.verification.status.id=mosip.resident.channel.verification.status
resident.event.ack.download.id=mosip.resident.event.ack.download
resident.download.card.eventid.id =mosip.resident.download.card.eventid
mosip.resident.request.vid.card.id=mosip.resident.request.vid.card
mosip.credential.request.service.id=mosip.credential.request.service.id
mosip.resident.checkstatus.individualid.id=mosip.resident.check-stage-status
mosip.resident.download.personalized.card.id=mosip.resident.download.personalized.card
mosip.resident.transliteration.transliterate.id=mosip.resident.transliteration.transliterate
resident.ui.properties.id=resident.ui.properties
mosip.resident.identity.auth.internal.id=mosip.identity.auth.internal
```

```
mosip.resident.user.profile.id=mosip.resident.profile  
resident.download.reg.centers.list.id=mosip.resident.download.reg.centers  
resident.download.nearest.reg.centers.id=mosip.resident.download.nearest.  
resident.download.supporting.documents.id=mosip.resident.download.support  
resident.send.card.id=mosip.resident.send.card  
resident.pinned.eventid.id=mosip.resident.pinned.eventid  
resident.unpinned.eventid.id=mosip.resident.unpinned.eventid  
resident.auth.proxy.partners.id=mosip.resident.auth.proxy.partners  
resident.events.eventid.id=mosip.resident.events.eventid  
resident.notification.id=mosip.resident.notification.get  
resident.profile.id=mosip.resident.profile.get  
resident.notification.click.id=mosip.resident.notification.click  
mosip.credential.store.id=mosip.credential.store  
resident.vids.id=mosip.resident.vids.get  
mosip.resident.download.uin.card=mosip.resident.download.uin.card  
mosip.registration.processor.registration.sync.id=mosip.registration.sync  
id.repo.update=mosip.id.update  
mosip.resident.get.pending.drafts=mosip.resident.get.pending.drafts  
mosip.resident.discard.pending.drafts=mosip.resident.discard.pending.draf
```

The request versions used in Resident REST APIs

```
mosip.resident.api.version.otp.request=1.0
mosip.resident.api.version.auth=1.0
auth.internal.version=1.0
mosip.registration.processor.application.version=1.0
mosip.resident.create.vid.version=v1
resident.vid.version=v1
resident.vid.version.new=1.0
resident.revokevid.version=v1
resident.revokevid.version.new=1.0
resident.version.new=1.0
resident.checkstatus.version=v1
resident.authLockStatusUpdateV2.version=1.0
resident.service.history.version=1.0
resident.document.get.version=1.0
resident.document.list.version=1.0
resident.service.pin.status.version=v1
resident.service.unpin.status.version=v1
resident.document.delete.version=1.0
mosip.resident.service.status.check.version=1.0
resident.service.event.version=1.0
resident.identity.info.version=1.0
resident.share.credential.version=1.0
mosip.resident.request.response.version=1.0
mosip.resident.grievance.ticket.request.version=1.0
resident.channel.verification.status.version=1.0
resident.event.ack.download.version=1.0
resident.download.card.eventid.version=1.0
mosip.resident.request.vid.card.version=1.0
mosip.credential.request.service.version=1.0
mosip.resident.checkstatus.individualid.version=1.0
resident.ui.properties.version=1.0
mosip.resident.get.pending.drafts.version=1.0
mosip.resident.discard.pending.drafts.version=1.0
```

Auth Services API calls

```
IDA_INTERNAL=${mosip.ida.internal.url}/idauthentication/v1/internal
INTERNALAUTH=${IDA_INTERNAL}/auth
INTERNALAUTHTRANSACTIONS=${IDA_INTERNAL}/authTransactions
KERNELENCRYPTONSERVICE=${IDA_INTERNAL}/getCertificate
OTP_GEN_URL=${IDA_INTERNAL}/otp
KERNELAUTHMANAGER=${mosip.kernel.authmanager.url}/v1/authmanager/authenti
```

Credential Req & service calls

```
CREDENTIAL_STATUS_URL=${mosip.idrepo.credrequest.generator.url}/v1/credential/status  
CREDENTIAL_REQ_URL=${mosip.idrepo.credrequest.generator.url}/v1/credential/request  
CREDENTIAL_CANCELREQ_URL=${mosip.idrepo.credrequest.generator.url}/v1/credential/cancelrequest  
CREDENTIAL_TYPES_URL=${mosip.idrepo. credential.service.url}/v1/credential/types
```

IdRepo identity Service calls

```
IDREPO_IDENTITY=${mosip.idrepo.identity.url}/idrepository/v1/identity  
IDREPOSITORY=${IDREPO_IDENTITY}/  
IDREPOGETIDBYUIN=${IDREPO_IDENTITY}/idvid  
IDREPOGETIDBYRID=${IDREPO_IDENTITY}/idvid  
IDREPO_IDENTITY_URL=${IDREPO_IDENTITY}/idvid/{id}  
GET_RID_BY_INDIVIDUAL_ID=${IDREPO_IDENTITY}/rid/{individualId}  
IDREPO_IDENTITY_UPDATE_COUNT=${IDREPO_IDENTITY}/{individualId}/update-count  
AUTHTYPESTATUSUPDATE=${IDREPO_IDENTITY}/authtypes/status  
IDREPO_IDENTITY_GET_DRAFT_UIN=${IDREPO_IDENTITY}/draft/uin/{UIN}  
IDREPO_IDENTITY_DISCARD_DRAFT=${IDREPO_IDENTITY}/draft/discard/
```

IdRepo vid Service calls

```
IDREPO_VID=${mosip.idrepo.vid.url}/idrepository/v1/vid  
CREATEVID=${IDREPO_VID}  
GETUINBYVID=${IDREPO_VID}  
IDAUTHCREATEVID=${IDREPO_VID}  
IDAUTHREVOKEVID=${IDREPO_VID}  
RETRIEVE_VIDS=${IDREPO_VID}/uin/
```

Key manager API calls

```

KEYMANAGER=${mosip.kernel.keymanager.url}/v1/keymanager
ENCRYPTURL=${KEYMANAGER}/encrypt
DECRYPT_API_URL=${KEYMANAGER}/decrypt
mosip.resident.keymanager.encrypt-uri=${KEYMANAGER}/encrypt
mosip.resident.keymanager.decrypt-uri=${KEYMANAGER}/decrypt
PACKETSIGNPUBLICKEY=${KEYMANAGER}/tpmsigning/publickey
mosip.keymanager.jwt.sign.end.point=${KEYMANAGER}/jwtSign
PDFSIGN=${KEYMANAGER}/pdf/sign

```

Master Data api calls

```

MASTER=${mosip.kernel.masterdata.url}/v1/masterdata
TEMPLATES=${MASTER}/templates
MACHINEDETAILS=${MASTER}/machines
MACHINESEARCH=${MASTER}/machines/search
MACHINECREATE=${MASTER}/machines
CENTERDETAILS=${MASTER}/registrationcenters
VALID_DOCUMENT_BY_LANGCODE_URL=${MASTER}/validdocuments/{langCode}
LOCATION_HIERARCHY_LEVEL_BY_LANGCODE_URL=${MASTER}/locationHierarchyLevel
LOCATION_HIERARCHY=${MASTER}/locationHierarchyLevels
IMMEDIATE_CHILDREN_BY_LOCATIONCODE_AND_LANGCODE_URL=${MASTER}/locations/immediatechildren
LOCATION_INFO_BY_LOCCODE_AND_LANGCODE_URL=${MASTER}/locations/info/{locationCode}
IMMEDIATE_CHILDREN_BY_LOCATION_CODE=${MASTER}/locations/immediatechildren
REGISTRATION_CENTER_FOR_LOCATION_CODE_URL=${MASTER}/registrationcenters/{registrationCenter}
REGISTRATION_CENTER_BY_LOCATION_TYPE_AND_SEARCH_TEXT_PAGINATED_URL=${MASTER}/registrationcenters
COORDINATE_SPECIFIC_REGISTRATION_CENTERS_URL=${MASTER}/getcoordinatespecific
APPLICANT_VALID_DOCUMENT_URL=${MASTER}/applicanttype/{applicantId}/language
WORKING_DAYS_BY_REGISTRATION_ID=${MASTER}/workingdays/{registrationCenter}
LATEST_ID_SCHEMA_URL =${MASTER}/idschema/latest
TEMPLATES_BY_LANGCODE_AND_TEMPLATETYPECODE_URL=${MASTER}/templates/{langCode}
DYNAMIC_FIELD_BASED_ON_LANG_CODE_AND_FIELD_NAME=${MASTER}/dynamicfields/{fieldName}
DYNAMIC_FIELD_BASED_ON_FIELD_NAME=${MASTER}/dynamicfields/{fieldName}
DOCUMENT_TYPE_BY_DOCUMENT_CATEGORY_AND_LANG_CODE=${MASTER}/documenttypes/

```

Notification service

```

SMSNOTIFIER=${mosip.kernel.notification.url}/v1/notifier/sms/send
EMAILNOTIFIER=${mosip.kernel.notification.url}/v1/notifier/email/send
resident.notification.emails=mosiptestuser@gmail.com
resident.notification.message=Notification has been sent to the provided email

```

Partner manager service URLs

```
PMS_PARTNER_MANAGER=${mosip.pms.partnermanager.url}/v1/partnermanager
POLICY_REQ_URL=${PMS_PARTNER_MANAGER}/partners/{partnerId}/credentialtype
PARTNER_API_URL=${PMS_PARTNER_MANAGER}/partners
PARTNER_DETAILS_NEW_URL=${PMS_PARTNER_MANAGER}/partners/v2
mosip.pms.pmp.partner.rest.uri=${PMS_PARTNER_MANAGER}/partners?partnerType
```

Reg-proc service calls

```
REGPROCPRINT=http://regproc-group7.regproc/registrationprocessor/v1/print
SYNCSERVICE=${mosip.regproc.status.service.url}/registrationprocessor/v1/
PACKETRECEIVER=${mosip.packet.receiver.url}/registrationprocessor/v1/packe
GET_RID_STATUS=${mosip.regproc.transaction.service.url}/registrationproce
REGISTRATIONSTATUSSEARCH=${mosip.regproc.status.service.url}/registration
```

Resident API calls

```
mosip.service-context=${server.servlet.context-path}
RESIDENT_SERVICE=${mosip.resident.url}${mosip.service-context}
RESIDENT_REQ_CREDENTIAL_URL=${RESIDENT_SERVICE}/req/credential/status/
GET_ORDER_STATUS_URL=${RESIDENT_SERVICE}/mock/print-partner/check-order-s
mosip.resident.download-card.url=${mosip.api.public.url}${mosip.service-c
mosip.resident.grievance.url=${mosip.api.public.url}${mosip.service-cont
```

Other service calls

```
MIDSCHHEMAURL=${mosip.kernel.syncdata.url}/v1/syncdata/latestidschema
DIGITAL_CARD_STATUS_URL=${mosip.digitalcard.service.url}/v1/digitalcard/
RIDGENERATION=${mosip.kernel.ridgenerator.url}/v1/ridgenerator/generate/r
otp-generate.rest.uri=${mosip.kernel.otpmanager.url}/v1/otpmanager/otp/ge
mosip.resident.service.mock.pdf.url=https://uidai.gov.in/images/New_eAadh
mosip.kernel.masterdata.audit-url=${mosip.kernel.auditmanager.url}/v1/aud
```

Identity Mapping json

Below config is used to get identity mapping and get remaining update count for the Identity Attributes .

This is used in Resident in Update UIN feature to show remaining update count for the Identity Attribute.

```
{  
    "identity": {  
        "IDSchemaVersion": {  
            "value": "IDSchemaVersion"  
        },  
        "selectedHandles": {  
            "value": "selectedHandles"  
        },  
        "name": {  
            "value": "fullName"  
        },  
        "gender": {  
            "value": "gender"  
        },  
        "dob": {  
            "value": "dateOfBirth"  
        },  
        "age": {  
            "value": "age"  
        },  
        "introducerRID": {  
            "value": "introducerRID"  
        },  
        "introducerUIN": {  
            "value": "introducerUIN"  
        },  
        "introducerVID": {  
            "value": "introducerVID"  
        },  
        "introducerName": {  
            "value": "introducerName"  
        },  
        "phone": {  
            "value": "phone"  
        },  
        "phoneNumber": {  
            "value": "phone"  
        },  
        "email": {  
            "value": "email"  
        },  
        "emailId": {  
            "value": "email"  
        },  
        "uin": {  
            "value": "UIN"  
        },  
        "vid": {  
    }
```

```
        "value": "VID"
    },
    "individualBiometrics": {
        "value": "individualBiometrics"
    },
    "introducerBiometrics": {
        "value": "introducerBiometrics"
    },
    "individualAuthBiometrics": {
        "value": "individualAuthBiometrics"
    },
    "officerBiometricFileName": {
        "value": "officerBiometricFileName"
    },
    "supervisorBiometricFileName": {
        "value": "supervisorBiometricFileName"
    },
    "residenceStatus": {
        "value": "residenceStatus"
    },
    "preferredLanguage": {
        "value": "preferredLang"
    },
    "locationHierarchyForProfiling": {
        "value": "zone, postalCode"
    },
    "addressLine1": {
        "value": "addressLine1"
    },
    "addressLine2": {
        "value": "addressLine2"
    },
    "addressLine3": {
        "value": "addressLine3"
    },
    "location1": {
        "value": "city"
    },
    "location2": {
        "value": "region"
    },
    "location3": {
        "value": "province"
    },
    "postalCode": {
        "value": "postalCode"
    },
    "location4": {
```

```
        "value": "zone"
    },
    "fullAddress": {
        "value": "addressLine1,addressLine2,addressLine3,city,region,",
    },
    "bestTwoFingers": {
        "value": "bestTwoFingers"
    },
    "birthdate": {
        "value": "dateOfBirth"
    },
    "picture": {
        "value": "face"
    },
    "phone_number": {
        "value": "phone"
    },
    "address": {
        "value": "addressLine1,addressLine2,addressLine3,city,region,",
    },

    "individual_id": {
        "value": "individual_id"
    },
    "attributes": {
        "value": "fln,ad1,ad2,ad3,cit,reg,pro,poc,cph,em,ph,gen,dob"
    },
    "street_address": {
        "value": "addressLine1,addressLine2,addressLine3"
    },
    "locality": {
        "value": "city"
    },
    "region": {
        "value": "region"
    },
    "postal_code": {
        "value": "postalCode"
    },
    "country": {
        "value": "province"
    },
    "password": {
        "value": "password"
    }
},
"metaInfo": {
    "value": "metaInfo"
}
```

```
{  
    "audits": {  
        "value": "audits"  
    },  
    "documents": {  
        "poa": {  
            "value": "proofOfAddress"  
        },  
        "poi": {  
            "value": "proofOfIdentity"  
        },  
        "por": {  
            "value": "proofOfRelationship"  
        },  
        "pob": {  
            "value": "proofOfDateOfBirth"  
        },  
        "poe": {  
            "value": "proofOfException"  
        }  
    },  
    "attributeUpdateCountLimit": {  
        "fullName": 2,  
        "gender": 4,  
        "dateOfBirth": 3  
    }  
}
```

Mvel File config

This file contains Mvel method definitions for masking attributes, getting passwords, and formatting attributes.

This is used in Resident for downloading PDF cards and for [masking attributes](#) in the share credential feature and personalize card feature.

Configuring Resident OIDC Client

Below are the steps to create the Resident OIDC client as standard steps in DevOps after e-Signet and Resident deployment.

Pre-requisites

1. Have a user created in keycloak with the below roles as needed for the Authorization token in the API requests:
 - i. ZONAL_ADMIN,
 - ii. PARTNER_ADMIN,
 - iii. POLICY_MANAGER,
 - iv. MISP_PARTNER,
 - v. PMS_ADMIN
2. Authenticating user to take the token and use it in all APIs invoked in further steps:
 - **Swagger URL** - <https://api-internal.dev2.mosip.net/v1/authmanager/swagger-ui/index.html?configUrl=/v1/authmanager/v3/api-docs/swagger-config#/authmanager/getAllAuthTokens>
 - **Request Body:**

```
{  
    "id": "string",  
    "version": "string",  
    "requesttime": "2023-01-04T11:49:29.007Z",  
    "metadata": {},  
    "request": {  
        "userName": "balaji",  
        "password": "mosip123",  
        "appId": "partner",  
        "clientId": "mosip-pms-client",  
        "clientSecret": "*****"  
    }  
}
```

Step-by-step process to create and configure the Resident OIDC client

I. Create Auth Partner for Resident OIDC Client

Step 1: Creating a policy group for resident OIDC Client

- Note: Since policymanager service swagger does not work, you can use postman for APIs in it.
- POST - `https://api-internal.dev2.mosip.net/v1/policymanager/policies/group/new`

Request Body:

```
{  
    "id": "string",  
    "version": "string",  
    "requesttime": "2023-01-04T11:52:19.734Z",  
    "metadata": {},  
    "request": {  
        "name": "Resident OIDC Client Policy Group",  
        "desc": "Resident OIDC Client Policy Group"  
    }  
}
```

- Make note of the `policyGroupId` from the response.

Step 2: Creating a policy for Resident OIDC client

POST - `https://api-internal.dev2.mosip.net/v1/policymanager/policies`

Request Body:

```
{  
    "id": "",  
    "metadata": null,  
    "request": {  
        "policyId": "resident-oidc-client-policy",  
        "name": "resident-oidc-client-policy",  
        "desc": "Resident OIDC Client Policy",  
        "policies": {  
            "authTokenType": "policy",  
            "allowedKycAttributes": [{  
                "attributeName": "fullName"  
, {  
                "attributeName": "gender"  
, {  
                "attributeName": "residenceStatus"  
, {  
                "attributeName": "dateOfBirth"  
, {  
                "attributeName": "photo"  
, {  
                "attributeName": "phone"  
, {  
                "attributeName": "email"  
, {  
                "attributeName": "individual_id"  
}],  
            "allowedAuthTypes": [{  
                "authSubType": "IRIS",  
                "authType": "bio",  
                "mandatory": false  
, {  
                "authSubType": "FINGER",  
                "authType": "bio",  
                "mandatory": false  
, {  
                "authSubType": "FACE",  
                "authType": "bio",  
                "mandatory": false  
, {  
                "authSubType": "",  
                "authType": "otp",  
                "mandatory": false  
, {  
                "authSubType": "",  
                "authType": "otp-request",  
                "mandatory": false  
, {  
                "authSubType": "",  
            }]}]
```

```

        "authType": "kyc",
        "mandatory": false
    }, {
        "authSubType": "",
        "authType": "demo",
        "mandatory": false
    }, {
        "authSubType": "",
        "authType": "kycauth",
        "mandatory": false
    }, {
        "authSubType": "",
        "authType": "kyceexchange",
        "mandatory": false
    }
],
},
"policyGroupName": "Resident OIDC Client Policy Group",
"policyType": "Auth",
"version": "1.1"
},
"version": "1.0",
"requesttime": "2022-12-29T13:12:28.479Z"
}

```

Step 3: Publishing policy

POST - <https://api-internal.dev2.mosip.net/v1/policymanager/policies/{{policyId}}/group/{{policyGroupId}}/publish>

Path params: * `policyId` - resident-oidc-client-policy * `policyGroupId` - from previous response

Step 4: Resident OIDC Client Partner self registration

Swagger URL: <https://api-internal.dev2.mosip.net/v1/partnermanager/swagger-ui/index.html?configUrl=/v1/partnermanager/v3/api-docs/swagger-config#/partner-service-controller/partnerSelfRegistration>

Request Body:

```
{  
    "id": "string",  
    "version": "string",  
    "requesttime": "2023-01-04T12:01:37.001Z",  
    "metadata": {},  
    "request": {  
        "partnerId": "resident-oidc-client-partner",  
        "policyGroup": "Resident OIDC Client Policy Group",  
        "organizationName": "IITB",  
        "address": "bangalore",  
        "contactNumber": "6483839992",  
        "emailId": "resident-oidc-client-partner1@mosip.com",  
        "partnerType": "Auth_Partner",  
        "langCode": "eng"  
    }  
}
```

Step 5: Upload ROOT Certificate as CA certificate

i. Get certificate from keymanager with below parameters:

- **Swagger URL:** <https://api-internal.dev2.mosip.net/v1/keymanager/swagger-ui/index.html?configUrl=/v1/keymanager/v3/api-docs/swagger-config#/keymanager/getCertificate>
- `AppID: "ROOT", refID: ""`

ii. Uploaded it as CA certificate:

- **Swagger URL -** <https://api-internal.dev2.mosip.net/v1/partnermanager/swagger-ui/index.html?configUrl=/v1/partnermanager/v3/api-docs/swagger-config#/partner-service-controller/uploadCACertificate>

Request Body (Example only):

```
{
  "id": "string",
  "version": "string",
  "requesttime": "2023-01-04T12:05:36.167Z",
  "metadata": {},
  "request": {
    "certificateData": "-----BEGIN CERTIFICATE-----\nMIIDpzCCAo+gAwIBAgII\n",
    "partnerDomain": "Auth"
  }
}
```

Step 6: Upload RESIDENT certificate as CA certificate

i. Get certificate from keymanager with below parameters:

- **Swagger URL:** <https://api-internal.dev2.mosip.net/v1/keymanager/swagger-ui/index.html?configUrl=/v1/keymanager/v3/api-docs/swagger-config#/keymanager/getCertificate>
- `AppID: "RESIDENT", refID: ""`

ii. Uploaded it as CA certificate:

Swagger URL - <https://api-internal.dev2.mosip.net/v1/partnermanager/swagger-ui/index.html?configUrl=/v1/partnermanager/v3/api-docs/swagger-config#/partner-service-controller/uploadCACertificate>

Request Body (Example only):

```
{
  "id": "string",
  "version": "string",
  "requesttime": "2023-01-04T12:05:36.167Z",
  "metadata": {},
  "request": {
    "certificateData": "-----BEGIN CERTIFICATE-----\nMIIDpzCCAo+gAwIBAgII\n",
    "partnerDomain": "Auth"
  }
}
```

Step 7: Upload `RESIDENT : IDP_USER_INFO` certificate as Partner certificate

i. Get certificate from keymanager with below parameters:

- **Swagger URL:** <https://api-internal.dev2.mosip.net/v1/keymanager/swagger-ui/index.html?configUrl=/v1/keymanager/v3/api-docs/swagger-config#/keymanager/getCertificate>
- `AppID: "RESIDENT", refID: "IDP_USER_INFO"`

ii. Uploaded it as Partner certificate:

Swagger URL - <https://api-internal.dev2.mosip.net/v1/partnermanager/swagger-ui/index.html?configUrl=/v1/partnermanager/v3/api-docs/swagger-config#/partner-service-controller/uploadPartnerCertificate>

Request Body (Example only):

```
{
  "id": "string",
  "version": "string",
  "requesttime": "2023-01-04T12:08:21.575Z",
  "metadata": {},
  "request": {
    "partnerId": "resident-oidc-client-partner",
    "certificateData": "-----BEGIN CERTIFICATE-----\nMIIDwjCCAqggAwIBAgII\n",
    "partnerDomain": "Auth"
  }
}
```

Step 8: Create policy Mapping request:

- **Swagger URL:** <https://api-internal.dev2.mosip.net/v1/partnermanager/swagger-ui/index.html?configUrl=/v1/partnermanager/v3/api-docs/swagger-config#/partner-service-controller/mapPolicyToPartner>
- Path param:
 - `partnerId` : resident-oidc-client-partner

Request Body:

```
{  
  "id": "string",  
  "version": "string",  
  "requesttime": "2023-01-04T13:18:11.206Z",  
  "metadata": {},  
  "request": {  
    "policyName": "resident-oidc-client-policy",  
    "useCaseDescription": "resident-oidc-client-policy"  
  }  
}
```

Output:

```
{  
  "id": "string",  
  "version": "string",  
  "responsetime": "2023-01-04T12:10:57.353Z",  
  "metadata": null,  
  "response": {  
    "mappingkey": "834602",  
    "message": "Policy mapping request submitted successfully."  
  },  
  "errors": []  
}
```

Make note of the `mappingKey`.

Step 9: Approve policy mapping:

Swagger URL - <https://api-internal.dev2.mosip.net/v1/partnermanager/partners/policy/{{mapping key}}>

Note: This mapping key will be returned as an output from policy mapping request.

Request Body:

```
{  
  "id": "string",  
  "version": "string",  
  "requesttime": "2023-01-04T12:13:15.114Z",  
  "metadata": {},  
  "request": {  
    "status": "APPROVED"  
  }  
}
```

II. Creating Resident OIDC Client

Step 1: Prepare the RESIDENT JWKS public key JSON.

i. Get certificate from keymanager with below parameters

- **Swagger URL:** <https://api-internal.dev2.mosip.net/v1/keymanager/swagger-ui/index.html?configUrl=/v1/keymanager/v3/api-docs/swagger-config#/keymanager/getCertificate>
- `AppID: "RESIDENT", refID: ""`

ii. Store the certificate as `resident-oidc.cer` file. Make sure to replace chars with line breaks and save it*

iii. Get the KeyID of the above certificate using Get All Certificates API

- **Swagger URL:** <https://api-internal.dev2.mosip.net/v1/keymanager/swagger-ui/index.html?configUrl=/v1/keymanager/v3/api-docs/swagger-config#/keymanager/getAllCertificates>
- `AppID: "RESIDENT", refID: ""`

From the response get the `keyId`. This will be the `kid` attribute in the OIDC client creation step.

iv. Get JWKS public key JSON from certificate

Use the `certpem2jwksjson.jar` with below command to get the JWKS of that.
(Attached the Java code of that for creating automated step of this)

```
java -jar certpem2jwksjson.jar resident-oidc.cer
```

In the console, the JSON text of the public key of the certificate will be printed.
Copy it.

v. Correct the `kid` in JWKS public key JSON

In the JSON public key, replace the `kid` value with the `keyId` in the earlier step.

Step 2: Create the OIDC client in PMS

Swagger URL: <https://api-internal.dev2.mosip.net/v1/partnermanager/swagger-ui/index.html?configUrl=/v1/partnermanager/v3/api-docs/swagger-config#/client-management-controller/createClient>

In the request body, make sure to replace the below attributes:

1. `publicKey` - the JWKS public key JSON from earlier step
2. `logoUri` - Correct hostname for the Resident UI
3. `redirectUris` - Correct the hostname for Resident Service

Request Body (Example only):

```
{  
    "id": "string",  
    "version": "string",  
    "requesttime": "2023-01-04T12:15:14.854Z",  
    "metadata": {},  
    "request": {  
        "name": "resident-oidc-client",  
        "policyName": "resident-oidc-client-policy",  
        "publicKey": {  
            "kty": "RSA",  
            "e": "AQAB",  
            "use": "sig",  
            "kid": "RbW-bNIIihYlr2GwVyqIgshHHFxe2pIkvdTp_Iedmic",  
            "alg": "RS256",  
            "n": "AMROKZuU_9xeybzmZdRHLCpJqh1ThfHtEf_Vbbm11Tpfdno0-eoYga-"  
        },  
        "authPartnerId": "resident-oidc-client-partner",  
        "logoUri": "https://resident.dev2.mosip.net/assets/logo.png",  
        "redirectUris": [  
            "https://api-internal.dev2.mosip.net/resident/v1/login-redirect"  
        ],  
        "grantTypes": [  
            "authorization_code"  
        ],  
        "clientAuthMethods": [  
            "private_key_jwt"  
        ]  
    }  
}
```

The response will contain the Resident OIDC client ID in `clientId` attribute.

Step 3: Configure the Resident OIDC client in `resident-default.properties`.

Configure the above obtained Resident OIDC client ID `resident-default.properties` with property name `mosip.iam.module.clientID`.

Note: This will need a restart of the resident service if it is already deployed.

Browsers Supported

Resident Portal is currently compatible and certified with the following list of browsers:

SL No	Browser	Version
1.	Chrome	124.0.6367.208
2.	Firefox	125.0.1
3.	Edge	124.0.2478.51

Support Systems

Administration

Overview

The MOSIP platform is configured via the Admin application. This application can be accessed only by a privileged group of administration personnel. The admin module provides the following functions:

1. Management of resources via CRUD operations:
 - a. Zone
 - b. Centers (registration centers)
 - c. Device
 - d. Machine
 - e. Users (Admin, registration staff)
2. Registration administration
 - a. Packet status
 - b. Retrieve lost RID
 - c. Resume RID

Administrative zones

- Administrative zones are virtual boundaries which a country can define to better manage their resources that are used during registrations. These resources includes Centers, Users, Machines and Devices. These zones can be defined in a hierarchical fashion and a country can allocate resources to such zones based on their requirements.
- Resources under each zone is managed by a *Zonal Admin*. This is done by assigning an Administrative zone to the Zonal Admin during user creation.
- These Zonal Admins can exist at any zonal hierarchy. (For e.g, a Zonal Admin can directly be mapped to the whole country as a Zone or can be mapped to a significantly smaller zone such as a city). Thus, these resources when mapped to an Administrative zone can only be managed by the Admin of that zone.

Activate/deactivate/decommission resources

What is deactivation of a resource?

Deactivation refers to a reversible action in which the `isActive` value for a resource in database is set to "False". This means that the resource will not be available for use unless and until it is activated later through the admin portal as required by the country.

What is decommissioning a resource?

- Decommission refers to a permanent/irreversible action of the resource. This also automatically deactivates it and the `isDeleted` value for the resource is set to "True".
- The primary difference being that a decommissioned resource cannot be brought into commission again as decommission refers to a permanent shutdown.
- Also, in cases where a center has some resources mapped to it (e.g. machines, devices or users), the portal will not allow the admin to decommission such a center.

Note- Activation/Deactivation/Decommission of a center in one language will be applied to the same center created in all the languages.

Services

1. [Admin Service](#)
2. [Hotlist Service](#)
3. [kernel Masterdata Service](#)
4. [Kernel Syncdata Service](#)

Frontend- Admin portal

Reference implementation of the Admin portal is available in [admin-ui](#) repository.

To know more, refer to the [Admin portal user guide](#).

Developer Guide

To know more about the setup, read [Admin Services Developer's Guide](#).

API

Refer [API Documentation](#).

Source code

[Github repo](#).

Develop

Admin Services Developers Guide

Overview

The admin application is a web-based application used by a privileged group of administrative personnel to manage various master data. The various resources that can be managed by an administrator are:

- Center (Registration centers)
- Device
- Machine
- User (Admin, Registration staff)

Along with resource and data management, the admin can generate master keys, check registration status, retrieve lost RIDs, and resume processing of paused packets.

- **Masterdata Service** exposes API to perform CRUD operations on masterdata through Admin service.
- **Hotlist Service** provides functionality to block/unblock any IDs with option of expiry. This hotlisted information will also be published to MOSIP_HOTLIST WebSub topic.
- **Sync Data Service** can be accessed only by the privileged group of admin personnel and enables default configurations and seed data to be setup when the MOSIP platform gets initialized.

The admin module has four services:

- Admin service
- Kernel Masterdata service
- Kernel Syncdata service
- Hotlist service

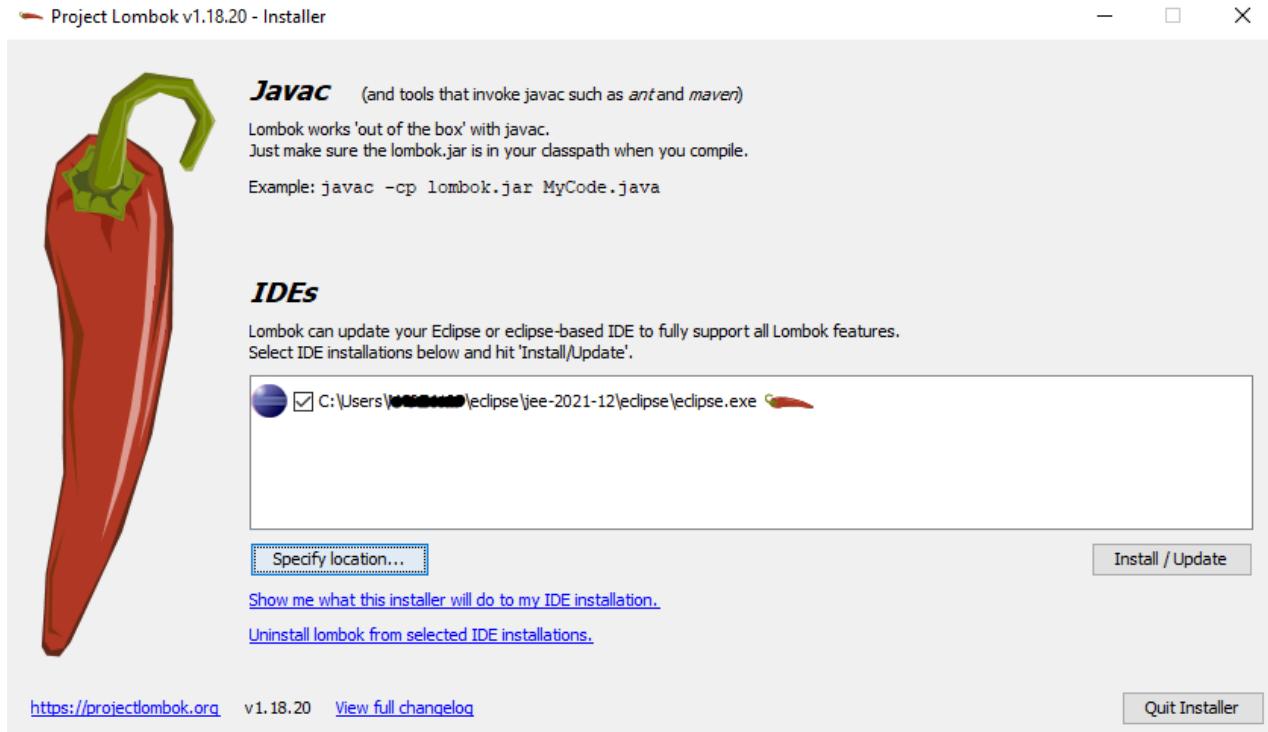
The documentation here will guide you through the pre-requisites required for the developer' setup.

Software setup

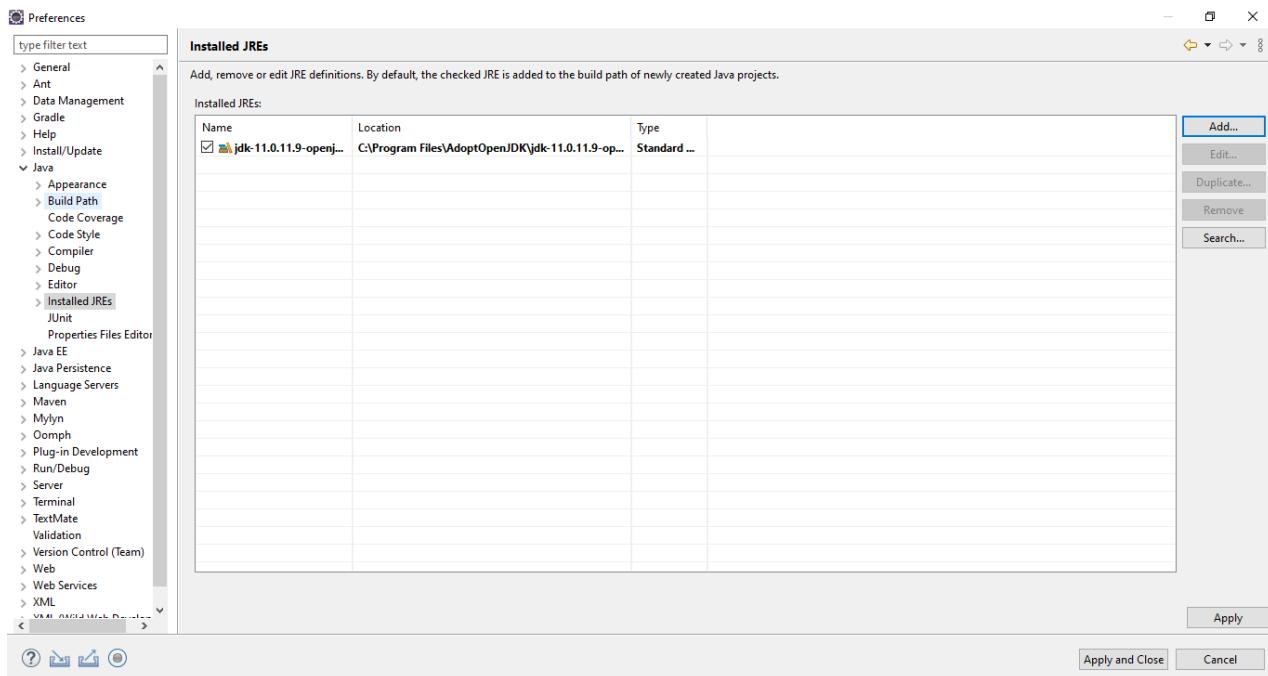
1. JDK 11
2. Any IDE (like Eclipse, IntelliJ IDEA)
3. Apache Maven (zip folder)
4. pgAdmin
5. Postman
6. Git/GitHub Desktop
7. Notepad++ (optional)
8. lombok.jar (file)
9. settings.xml

Follow the steps below to set up Admin Services on your local system:

1. Download [lombok.jar](#) and [settings.xml](#).
2. Unzip Apache Maven and move the unzipped folder in `C:\Program Files` and `settings.xml` to `conf` folder `C:\Program Files\apache-maven-3.8.4\conf`.
3. Install Eclipse, open the `lombok.jar` file and wait for some time until it completes the scan for Eclipse IDE and then click `Install/ Update`.



1. Check the Eclipse installation folder `C:\Users\userName\eclipse\jee-2021-12\eclipse` to see if the `lombok.jar` is added. By doing this, you don't have to add the dependency of `lombok` in your `pom.xml` file separately as it is auto-configured by Eclipse.
2. Configure the JDK (Standard VM) with your Eclipse by traversing through `Preferences → Java → Installed JREs`.

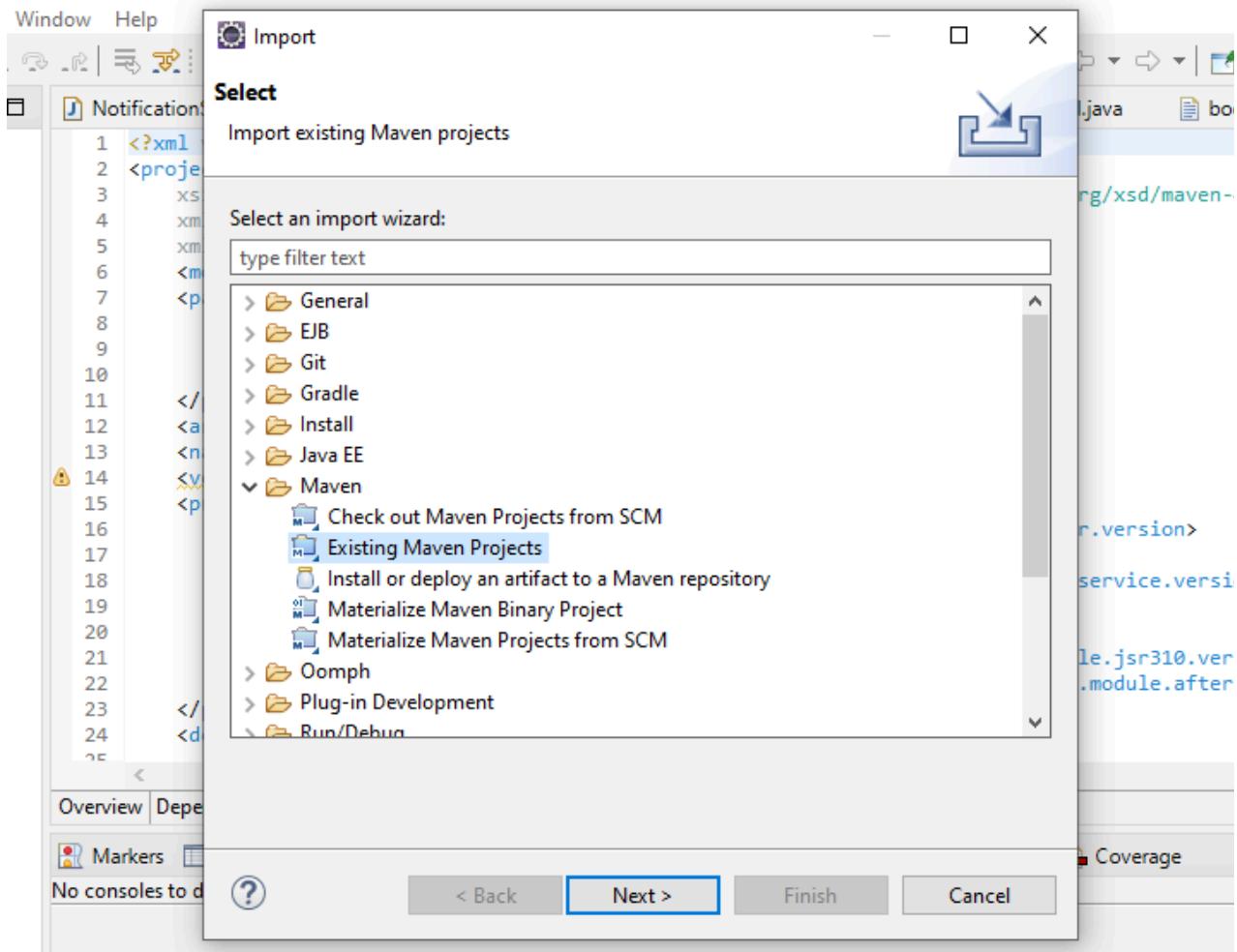


Code setup

For the code setup, clone [admin-services](#) repository and follow the guidelines mentioned in the [Code Contributions](#).

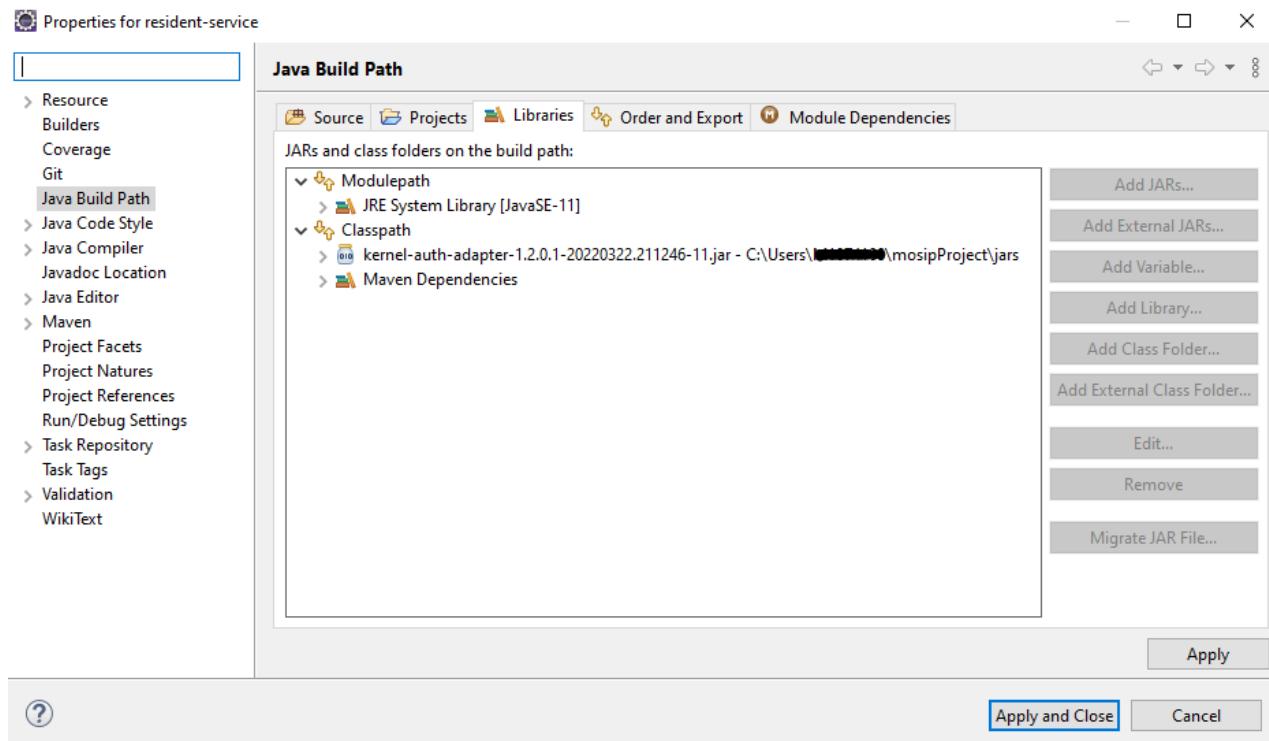
Importing and building of a project

1. Open the project folder where `pom.xml` is present.
2. Open command prompt from the same folder.
3. Run the command `mvn clean install -Dgpg.skip=true -DskipTests=true` to build the project and wait for the build to complete successfully.
4. After building of a project, open Eclipse and select `Import Projects → Maven → Existing Maven Projects → Next → Browse to project directory → Finish`.
5. After successful importing of project, update the project by right-click on `Project → Maven → Update Project`.



Environment setup

1. For the environment setup, you need an external JAR that is available [here](#) with different versions. (E.g.: You can download `kernel-auth-adapter.jar` and add to project `Libraries → Classpath → Add External JARs → Select Downloaded JAR → Add → Apply and Close`).



1. Clone [admin-services repository](#).
2. Any changes in the properties for Masterdata and Admin services should be done in `application-local1.properties` file.
3. By default the Admin-services is connected to dev environment.
4. To run the specific service from IDE, `Open IDE -> Specific service -> src/main/java/io.mosip.specific service -> Right click on the file and select run as Java Application.`

For example, to run the Admin service, `open IDE -> admin-service -> src/main/java/io.mosip.admin -> Right click on AdminBootApplication.java and select run as Java Application.`

1. To run the specific service from Command Prompt, Open Project folder → open command prompt from same folder → Execute `java -jar target/specific-service-1.2.0.jar`.

For example, to run the admin service, Open Project folder → open command prompt from same folder → Execute `java -jar target/admin-service-1.2.0-SNAPSHOT.jar`.

The service should now be up and running.

Admin services API

- For API documentation, refer [here](#).
- The APIs can be tested with the help of **Swagger-UI** and **Postman**.
- Swagger is an interface description language for describing restful APIs expressed using JSON. You can access Swagger-UI of admin-services for dev-environment from:

Admin service- `http://dev.mosip.net/v1/admin/swagger-ui/index.html?configUrl=/v1/admin/v3/api-docs/swagger-config#/` and localhost from `http://localhost:8098/v1/admin/swagger-ui/index.html?configUrl=/v1/admin/v3/api-docs/swagger-config#/.`

Masterdata- `http://dev.mosip.net/v1/masterdata/swagger-ui/index.html?configUrl=/v1/masterdata/v3/api-docs/swagger-config#/` and localhost from `http://localhost:8086/v1/masterdata/swagger-ui/index.html?configUrl=/v1/masterdata/v3/api-docs/swagger-config#/.`

Syncdata- `http://dev.mosip.net/v1/syncdata/swagger-ui/index.html?configUrl=/v1/syncdata/v3/api-docs/swagger-config#/` and localhost from `http://localhost:8089/v1/syncdata/swagger-ui/index.html?configUrl=/v1/syncdata/v3/api-docs/swagger-config#/.`

Hotlist- `http://dev.mosip.net/v1/hotlist/swagger-ui/index.html?configUrl=/v1/hotlist/v3/api-docs/swagger-config#/` and localhost from `http://localhost:8095/v1/hotlist/swagger-ui/index.html?configUrl=/v1/hotlist/v3/api-docs/swagger-config#/`

- Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs—faster. It is widely used tool for API testing.
- Create an environment as shown in the image below.

This environment is created for dev. Give the variable name as `url` and set both the values as `https://dev.mosip.net`.

- In the similar way, create another environment for localhost as shown below.

This environment is created for localhost. Give the variable name as `url` and set both the values as `http://localhost:8099`.

Test

Admin Portal User Guide

Overview

An admin application is a web-based application used by a privileged group of administrative personnel to manage various master data sets. The various resources that an Admin can manage are:

1. Center (Registration centers)
2. Device
3. Machine
4. Users (Admin, Registration staff)

Along with the resource and data management, the admin can generate master keys, check registration status, retrieve lost RID, and resume processing of paused packets. To start using the Admin portal, an admin user must be assigned to a zone.

To learn more, refer to the video below!

Session 1

LTS 1.2 - Admin UI Part1 03-02-2022



First Admin user

1. Setup of hierarchical zones
2. Create Admin roles in KeyCloak
3. Create the first admin user in KeyCloak and assign the "GLOBAL_ADMIN" role

i **Note:** On the login of the first admin user, user zone mapping is handled automatically.

The above is done automatically as part of the [default sandbox installation](#).

Login

1. Select the preferred language.
2. Login with Keycloak credentials.

Actions

1. Map the other users(admins/registration operators/supervisors) to their respective zones
2. Create centers and assign the users to a particular center
3. **Highly recommended:** Ensure to revoke the first super user's zone mapping and role after the first user actions are completed.

Admin roles and their default accessibility matrix

- GLOBAL_ADMIN
- ZONAL_ADMIN
- REGISTRATION_ADMIN
- MASTERDATA_ADMIN
- KEY_MAKER

GLOBAL_ADMIN	ZONAL_ADMIN	REGISTRATION_ADMIN	MASTERDATA_ADMIN	KEY_MAKER
Centers	Devices	Packet Status	Devices	GenerateMasterKey
User Zone Mapping	Machines	Pause/Resume RID	Machines	GenerateCSR
All Master Data	User Zone Mapping	Retrieve Lost RID	All Master Data	GetCertificate
Masterdata Bulk Upload	User Center Mapping	Packet Bulk Upload	Masterdata Bulk Upload	UploadCertificate
	All Master Data			UploadOtherDomainCertificate
	Masterdata Bulk Upload			

Center

- This portal allows an Admin to view, create, edit, activate, deactivate and decommission registration centers.
- An Admin can manage only centers under their [administrative zones](#).

The administrator can filter the list of registration centers based on parameters like ***Center name, Center type, Status, and Location code.***

- The system does not fetch the details of decommissioned registration centers but only active and inactive centers are displayed.
- If the admin does not find a center, they can click the *Center not available in logged in language* button. Clicking on this button displays the list of centers that are already created in other languages. On selecting a particular center, the information will be auto-populated in the Create page and be made available to the admin for modifications.

- Language specific fields can be modified to create a center with the currently logged in language.

Create center

- A center is created with multiple attributes and is mapped to the administrative zone that it belongs to.
- A center can only be mapped to the configured location hierarchy level.
- While defining centers, an admin can also define the working days of the week for a center and any exceptional holidays that might be applicable for a particular center.

Update center

- An admin can update a center even after it has been created. The updates can include adding the details that were missed during the creation of the center or changing the details of a center as required.
- To update, click the **Edit** option from the Actions menu against a center name.

- ⓘ **Note-** Updates made to language specific fields updates data only for that language in the database while updates made to non-language dependent fields updates data against all the language entries for that center.

Activate/deactivate/decommission center

- Select the **Deactivate/Decommission** option from the Actions menu against the center.
- Activation/Deactivation/Decommission of a center in one language will be applied to the same center created in all the languages.

To know more, refer [Activate/deactivate/decommission resources](#)

Devices

- Using this portal, an admin can manage the devices a country will use for registering residents like devices used for bio-metric capture (Fingerprint, Iris, Web camera, etc.), printers, and scanners.
- This portal allows an Admin to view, create, edit, activate, deactivate, and decommission registration centers.
- The admin portal allows an admin to view the list of all the devices available in the jurisdiction of their administrative zone.
- The system does not fetch the details of decommissioned devices but only the active and inactive devices.

 **Note:**

- The device entity is language agnostic (independent of languages).
- The data collected about Devices is used only for book keeping, i.e., the MOSIP system does not use this data for any validation.

The Admin can filter the list of Registration centers based on parameters like *Device Name, Mac Address, Serial Number, Status, Map Status, Device Type, and Device Spec ID.*

Create devices

A Device can be created with multiple attributes and be mapped to the Administrative Zone it belongs to.

Update devices

- An admin can update missing information or change device details even after it is created.
- To update, click the **Edit** option from the Actions menu against a device.

Activate/deactivate/decommission device

Select the **Deactivate/Decommission** option from the Actions menu against the device.

Map/un-map/re-map the device to a center

- Admin portal allows an Admin to map/un-map each device to a center.
- This mapping specifies as to which center the device will be used in.
- A device can only be mapped to a center that belongs under the device's Administrative Zone.
- To do so, select the device and choose a **Center Name** from the dropdown.

Machines

- Admin portal allows an administrator to manage the machines a country will use for registering residents.
- This portal allows an Admin to view, create, edit, activate, deactivate and decommission machines.
- The admin portal allows an admin to view the list of all the machines available in the jurisdiction of their administrative zone.
- The system does not fetch the details of decommissioned machines but only shows the active and inactive machines.

 **Note:** Machine entities are also language agnostic.

The administrator can filter the list of machines based on parameters like *Machine name, Mac address, Serial number, Status, and Machine type*.

Create machines

- A machine can be created with attributes like *Machine ID, machine name, MAC address, serial number, machine spec ID, and administrative zone* the machine belongs to.
- A machine needs to be mapped to an administrative zone.

Update machines

- An admin can update missing details or make changes to machine details even after it is created.
- To update, click the **Edit** option from the Actions menu against a machine.

Note- Updates made to language specific fields update data only for that language in the database while updates made to non-language dependent fields updates data against all the language entries for that center.

Activate/deactivate/decommission machine

An admin can deactivate or decommission a machine through the admin portal.

Map/un-map/re-map machine to a center

- Admin portal allows an Admin to map/un-map each machine to a center.
- This mapping specifies as to which center the machine will be used in.

- A machine can only be mapped to a center that belongs under the machine's Administrative Zone.
- To do so, select the machine and choose a **Center Name** from the dropdown.

Users

- MOSIP uses Keycloak as an IAM (Identity access management tool) for managing Users. These users are internal users of MOSIP including Registration Officers, Registration Supervisors, Zonal Admins, Global Admins, etc.
- using this portal, an Admin can map the users to a zone and a center.

User Zone Mapping

- Once a user is created in Keycloak, they need to be mapped to a zone to access specific information available in that zone.
- Admin portal allows an admin to map users to a zone. This mapping specifies which zone the user will belong to.
- A user can only be mapped to a zone that belongs under the user's Administrative Zone.
- A user can later be unmapped from the zone in case a user needs to be moved to another zone. In such cases, the user will later need to be mapped to the new zone. The below image displays the list of users that are mapped to a zone.

Map/Un-map/re-map user to a zone

To map a user to a zone,

1. Click Resources→ User Zone mapping
2. Click **+Map Zone**
3. Select the *User Name, and Administrative Zone* from the dropdown.

4. Click **Save**.

To re-map a user to a zone,

1. Click Resources→ User Zone mapping
2. Select **Remap** from the Actions menu against the mapped user.
3. Update the *User Name/ Administrative Zone* from the dropdown.
4. Click **Save**.

(i) Note- If the center is already mapped, the admin needs to unmap the center to remap the zone.

User Center Mapping

- Once the user is mapped to a zone, they will be listed in the screen below. Now, the user will be mapped to a center to be able to manage their assigned center.
- Admin portal allows an admin to map users to a center. This mapping specifies as to which center the user will be used in.
- A user can only be mapped to a center that belongs under the user's Administrative Zone.
- A user can later be unmapped from the Center in cases where a User needs to be moved to another Center. In such cases, the user will later need to be mapped to the new center. In case the user is required to be mapped to a Registration center outside the Administrative Zonal restriction, the Administrative Zone of the user must be changed.

Map/un-map/re-map user to a registration center

To map a user to a center,

1. Click Resources→ User Center Mapping
2. Select **Map** from the Actions menu against the mapped user.
3. Select the **Center Name** from the dropdown against the User Name, Administrative Zone.
4. Click **Save**.

Search and dropdowns

- To get the results starting with a specific character/ set of characters, prepend that specific character/set of characters with `asterisk` symbol.
- Similarly to get the results ending with a specific character/ set of characters, append that specific character/ set of characters with `asterisk`.
- For the results containing a specific character/ set of characters, prepend and append that specific character/ set of characters with `asterisk`.

Below is the image illustrating the same.

Packet status (based on RID)

- A Registration packet generated in the Registration client is sent to the Registration Processor for further processing and UIN generation.
- Using this Portal, A Registration Admin can view the status of a packet by entering the RID of the packet.
- The packet status will contain all the stages the packet has passed through along with the last stage the packet is in.
- In case the packet has not been processed or is marked for *Re-Send/Re-Register*, the admin will be able to view specific comments indicating the reason for that particular status.

Pause/Resume RID

- The Registration Admin has the privilege to view the registration packets that are in a paused state.
- Admin can use this portal to resume or reject paused packets. They would have 3 options:
 - Resume processing (from where it was paused)
 - Resume from the beginning
 - Reject

Once processing of a packet is resumed, it will be removed from this list

Retrieve lost RID

- The Registration Admin can use this feature to retrieve lost RID.
- For instance, if the resident did not provide any valid email and/or phone number and has lost the RID slip received during the registration, to find their RID details, the resident contact the MOSIP helpline and share details such as name, center name, registration date, and postal code to the admin, who will use the lost RID feature and try to retrieve the RID number.

A few filters may be applied to retrieve the RID.

Note: This feature is currently under development.

Master Data

- Admin portal allows an Admin to manage the Masterdata applicable for a country.
- These data include a list of Genders, a list of Holidays, Templates, Center Types, Machine Types, etc.

To know more, refer to the [Masterdata guide](#).

Bulk upload

- If a country decides to upload the data through the .csv files, they could use this feature to upload the existing data into the MOSIP platform.
- The listing screen displays the uploaded data transaction information.
- As the information inside .csv files may be huge, it would go through the batch job to process the information and store it in the tables. Also, it may take time to get a unique transaction ID against a particular action.

Master Data

To upload Master data using the Admin portal,

1. Go to Bulk Upload > Master Data
2. On the master data dashboard, click **Upload Data**.
3. Select the operation (insert/update/delete)

4. Select the table name into which the data needs to be uploaded.
 5. Click **Choose file** to select the data and click **Upload**
- To view the format for inserting data in a particular table, click on the Download icon.
 - A CSV file gets downloaded in which the first row represents the column names and the rest of the rows are the data that will be inserted into the table(sample).
 - From the 1.2.0.1-B2 version, apart from the comma, other special characters (i.e., '|';'\$etc.) can also be used as a separator in the CSV file used for masterdata bulk upload. This can be done by updating the property `mosip.admin.batch.line.delimiter` with the same special character.

Note: While editing CSV files, it is recommended to keep track of the Date format and Time format to be the same as the acceptable formats. The acceptable Date format is *YYYY-MM-DD* and the acceptable Time format is *HH:MM:SS*. Any other Date and Time formats in CSV files will result in a `(DataType Mismatch Error)`.

Packets

To upload packets using the Admin portal,

1. Go to Bulk Upload > Packets
2. On the packet upload dashboard, click **Upload Packet**.
3. Select the following from the dropdown:

- Center name
- Source (currently displays Registration Client)
- Process (New, Update UIN, Lost, Biometric correction)
- Supervisor status (Approved/Rejected)

These details are important if the packet needs to be synced before upload.

4. Click **Choose file** to select the packets and click **Upload**.

How is the packet upload performed with or without the DATA_READ role?

LoggedIn User Role	Packet Sync	Packet Upload
With DATA_READ	Yes	only after successful sync
Without DATA_READ	No	Yes

For uploading the packets through the Admin portal, ensure that the packets are available in the machine or the external hard disk connected from where the Admin Portal is being used.

Key Manager

With the help of this feature, the Admin user can generate and manage the keys required in MOSIP.

GenerateMasterKey

- The logged in user with **KEY_MAKER** role will have access to view and generate the master key in the Admin portal.
- Using this option, the logged in user will be able to generate only the [Root](#) key and [Module](#) master key. To generate the key, the user has to select the Application ID from the options available in the dropdown, leave the Reference ID blank for the [Root](#) and [Module](#) master key, and provide other certificate attributes to be used at the time of generation of the certificate for the key.

- These certificate attributes in the portal are optional, if not provided, default values configured in the Key Manager service will be used.
- For the Kernel signature key (which is considered the master key and stored in [HSM](#)), a Reference ID needs to be provided and the value has to be [SIGN](#).
- **The force** flag option is available in key generation. The logged in user can select the option value **True** to force the invalidation existing key and generate a new key in [Key Manager](#) service.
- The logged in user has to select the return object after the generation of the key.
- The user can select either *Certificate* or *CSR (Certificate Signing Request)*. The key will be generated only when the key is not available in [Key Manager](#) service otherwise already generated key certificate will be returned for the generation request.

GenerateCSR

- *CSR (certificate signing request)* is required when there is a need to procure a valid certificate from a valid CA.
- *GenerateCSR* option can be used to request for a CSR and this option will be visible to all the users who log in to the Admin portal.

- The logged in user can request for generation of CSR for any key generated in [Key Manager](#) service.
- The user has to provide the Application ID and Reference ID to get a CSR.
- A new key will be auto-generated in case the key does not exist and the already existing key has expired for the Module Encryption keys.
- Whereas, for [Module](#) master key or [Root](#) key, a new key will not get auto-generated in case the key does not exist, but the new key will get auto-generated if the key exists and has expired. The current valid key will always be used to generate a CSR.

GetCertificate

- The user can get a certificate for all the keys generated in Keymanager and any partner certificates uploaded in Keymanager service for partner data sharing purposes.
- The *GetCertificate* option is visible to all the users who log in to the Admin portal.
- The user has to provide the Application ID and Reference ID to get a certificate.
- A new key will be auto generated in case the key does not exist and the already existing key has expired for Module encryption keys.

- Whereas, for [Module](#) master key or [Root](#) key, a new key will not get auto-generated in case the key does not exist, but a new key will get auto-generated if the key exists and has expired. For the partner certificate, a new key will not be generated in the Key Manager service.
- Only current valid certificates will be returned when the user requests a certificate.

UploadCertificate

- The logged in user can use this option to update the certificate for all the keys generated in the [Key Manager service](#).
- This option is used in scenarios where a valid CA certificate has been procured for a key available in the [Key Manager service](#).

UploadOtherDomainCertificate

- The logged in user can use this option to upload a partner certificate in [Key Manager service](#).
- Partner certificates will be used in the Key Manager service to encrypt any sharable data using the partner certificate required in datashare from MOSIP to any partner.
- Partner certificates can also be used in the Key Manager service for signature verification purposes.