**Test Plan Approach and**

**LLD – Login & Registration**

IBM Corporation

**Authors**

|  |  |
| --- | --- |
| **Name** | **Role in IBM Project** |
| Pankaj Hukre | Application Developer |
|  |  |

**Document History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Document Revision Description** | **Document Author** |
| 15-Jul-2024 | 0.1 | Draft | Pankaj Hukre |

# **Introduction**

This document aims at defining the LLD for Base Journey “Login & Registration” for the business banking application.

# **Scope**

Following Sub Journeys are in scope.

* + Individual user registration
  + Sole proprietor registration
  + Added user registration
  + Login of Sole prop/ Individual user using mobile & MPIN
  + Login of Sole prop/ Individual/ Added user using user id, corp id & password

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr#** | **Application** | **User Type** | **User Journey** | **DBP provides** |
| 1 | MSME Mobile App | Business Banking Users / Corporate Customers | Login & Registration |  |
| 2 | Assisted Portal | Bank Staffs | Login & Registration |  |

# **System Context Diagram**

# **A diagram of a computer Description automatically generated with medium confidence**

# **Component Diagram**

# **A screenshot of a computer Description automatically generated**

# **Assumptions**

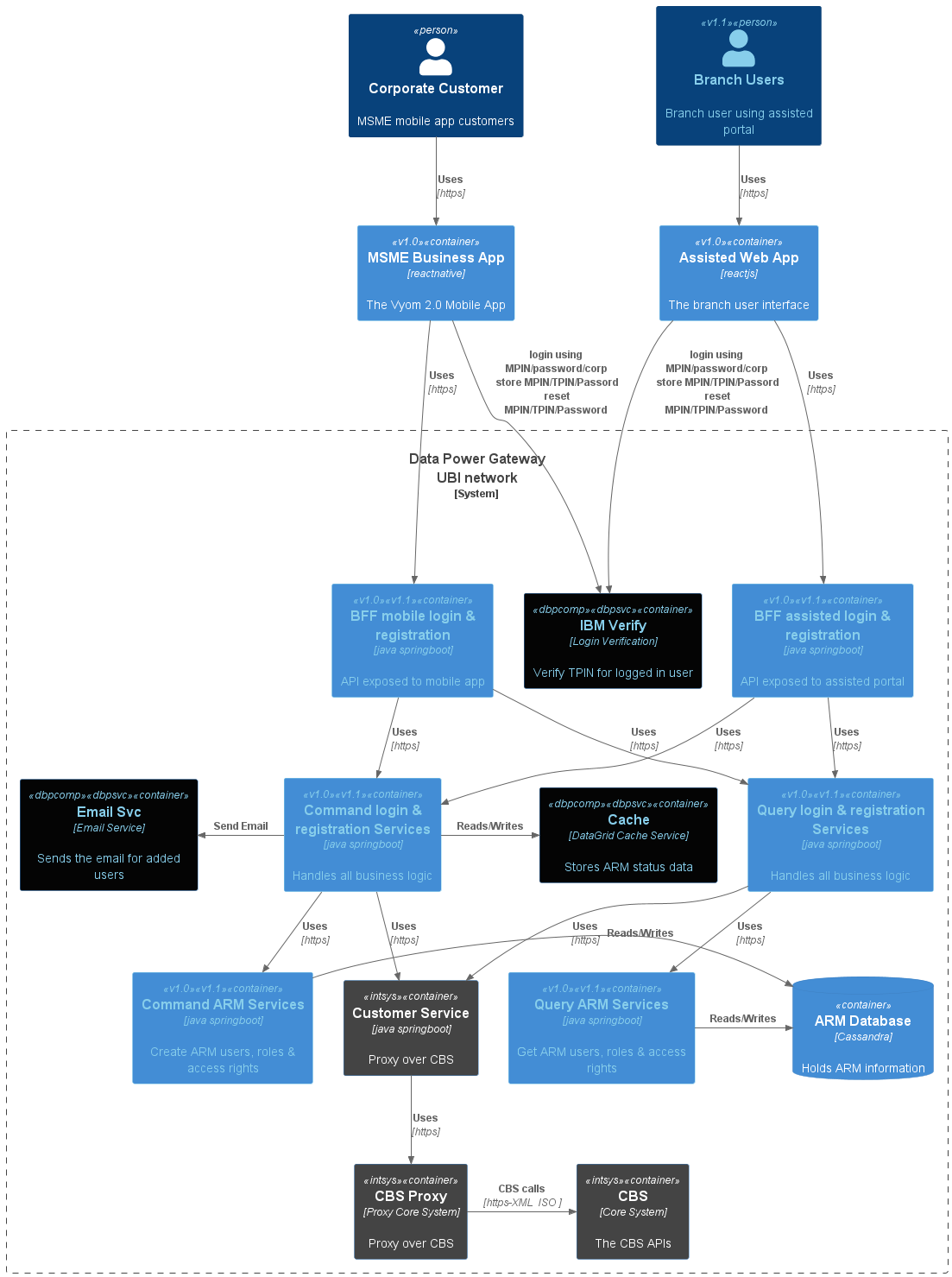
1. Corporate customers will only use Vyom MSME Mobile app for business banking.
2. Assisted portals are only for B2E users and accessible only by bank/branch staff.
3. Individual, Sole Proprietor and Customer Admin (in case of non-individual) already got their user-id created and mapped to their retail CIF in DBP and are logged in. Also, for Sole Proprietor and Customer Administration (in case of non-individual), their user-id needs to be mapped to their corresponding User Group in DBP.
4. Exhaustive list of modules and their features grouped under financial and non-financial should be provided by McKinsey and onboarded in DBP master database. [To be confirmed]

# **Process Flow**

Use Case A: Individual/Sole prop registration



# **System Context Diagram**



# **Component Diagram**

A screenshot of a computer

Description automatically generated

**Sequence Diagrams**

1. Individual / Sole prop registration

1. Non-Individual registration
2. Individual with m
3. Forgot PIN



1. Change PIN



1. Expired PIN



1. Forgot TPIN



1. Forgot password



1. Login with Corp id

****

1. Login with mobile & pin

**REST APIs**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sl. No.** | **API** | **URL** | **Request** | **Response** | **Comments** |
|  |  |  |  |  |  |

**API Swagger:**

**For reference, attached is the excel sheet of APIs:**

|  |  |
| --- | --- |
| List of APIs |  |

|  |  |
| --- | --- |
| Domain |  |
| User Domain |  |
| Customer Domain |  |
| Account Domain |  |

**Domain Entity and Domain Validations (Business Validations)**

**Domain Events**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

**Error Message:**

|  |  |
| --- | --- |
| Error Code | Error Message |
| CUST\_NOT\_FOUND | Scheme is not initialized or empty |
| CUST\_NOT\_INIT | Customer is not available or invalid |
|  |  |

**Application Properties**

|  |  |  |  |
| --- | --- | --- | --- |
| Property Name | Short Description | Property |  |
|  |  |  |  |
|  |  |  |  |

1. **Domain and Event Models**

**Entities:**

An entity is an object with a unique identity that persists over time. For instance, in a banking application, customers and accounts would qualify as entities. Each entity possesses a distinct identifier within the system, facilitating retrieval or lookup. However, this identifier may not necessarily be directly exposed to users and can take the form of a GUID or a primary key in a database. Additionally, an entity's identity may extend across multiple bounded contexts and endure beyond the application's lifecycle. For example, bank account numbers or government-issued IDs remain independent of any specific application's duration. Furthermore, the attributes associated with an entity may undergo changes over time, such as a person's name or address, while still representing the same individual. Entities are capable of holding references to other entities.

**Value objects:**

Unlike entities, a value object lacks identity and is solely defined by the values of its attributes. These objects are immutable, meaning any updates necessitate the creation of a new instance to replace the previous one. While value objects can incorporate methods encapsulating domain logic, these methods should not produce side effects on the object's state. Common examples of value objects include colors, dates and times, and currency values.

**Aggregates:**

An aggregate delineates a consistency boundary around one or more entities. Within an aggregate, precisely one entity serves as the root, facilitating lookup through its identifier. Other entities within the aggregate are regarded as children of the root and are referenced through pointers from the root. Aggregates serve the purpose of modeling transactional invariants. Real-world scenarios often entail intricate webs of relationships—customers create orders, orders contain products, and products have suppliers. In such cases, ensuring consistency across multiple related objects poses a challenge. Aggregates address this challenge by providing a means to manage and enforce transactional invariants, thus guaranteeing consistency within the domain.

1. **Deployment Model**

**Application Container (Module-based Application):**

Embracing an application container model facilitates the deployment of modularized applications. By encapsulating various modules within containers, each component can be managed independently, promoting scalability, flexibility, and ease of maintenance. This approach enables seamless deployment and orchestration of application components, facilitating rapid development and deployment cycles.

**Topics (Kafka Integration):**

Leveraging Kafka for topic management enhances communication and data streaming within the application ecosystem. Topics serve as communication channels that enable producers to publish messages and consumers to subscribe and process them. By utilizing Kafka's robust messaging system, applications can achieve real-time data processing, fault tolerance, and scalability. This integration empowers applications to efficiently handle large volumes of data and enables event-driven architectures, fostering responsiveness and agility in application development and deployment.

1. **DevSecOps Considerations and Configurations**

DevOps is a software development approach that emphasizes collaboration, integration, automation, and continuous delivery. In this context:

**Package (Maven Integration):**

Utilizing Maven for packaging and building allows for efficient management of project dependencies and the creation of deployable artifacts. Maven streamlines the build process by automating tasks such as compilation, testing, and packaging, enhancing the speed and reliability of software delivery.

**Container Image (Building):**

Building container images enables the encapsulation of applications and their dependencies into portable, scalable units. Through this process, developers can create consistent environments across various platforms, making deployment more predictable and manageable.

**Tekton Triggers (Webhook Integration):**

Integrating Tekton triggers via webhooks facilitates automation in the CI/CD pipeline. Webhooks provide a mechanism for triggering Tekton pipelines based on specific events, such as code commits or pull requests. This automation streamlines the development workflow, allowing for faster feedback loops and smoother delivery cycles.

**ArgoCD Configuration (Pull-Based Mechanism):**

Implementing ArgoCD with a pull-based mechanism simplifies and automates the deployment of applications in Kubernetes environments. ArgoCD continuously monitors a Git repository for changes to application configurations and automatically synchronizes the desired state with the actual state of the cluster. This approach ensures consistency and reliability in application deployment while minimizing manual intervention.