JASF Software

BC Analyzer
Software Architecture Document

Version <1.6>

Revision History

Date	Version	Description	Author
14/09/25	1.0	The document was created	Samuel Duran
14/09/25	1.1	Added architectural representation	Flora Rodríguez
15/09/25	1.2	Added architectural goals and constraints	Flora Rodríguez
16/09/25	1.3	Added process view	Flora Rodríguez
16/09/25	1.4	Added deployment view and logical view	José Julián Brenes
16/09/25	1.5	Added implementation view, its overview, layers and the quality of the architecture	Aaron Moncada
16/09/25	1.6	Added introduction, the use-case realizations and the use-case view	Samuel Duran

Table of Contents

Software Architecture Document

1. Introduction

This Software Architecture Document (SAD) presents the architectural design for BC Analyzer. This document establishes the overall architectural framework and serves as the primary reference for all stakeholders involved in the system's development, implementation, and maintenance.

The document is structured to provide multiple architectural viewpoints that collectively describe the system's design, ensuring that different stakeholders can understand the aspects most relevant to their roles. It captures the key architectural decisions, rationale, and constraints that shape the system's design.

This document is intended for software architects, developers, project managers, quality assurance teams, and other stakeholders who need to understand the system's architectural foundation.

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

This document will provide information about the representation of the architecture that have been chosen by the team, specifying the views that have been made to represent the system´s structure. It also provides the goals and constraints that the system has, including the objectives that are meant to be reached after it has been finished and also including the restrictions that the system has, what can´t be reached and what isn´t a goal affordable for the project.

This document also identifies a number of stakeholders for the project, these stakeholders are the Executives and the Management Team which are expected to have a role of generating executive´s reports and KPIs, basically they have the intention of use the system as a way to obtain information about the productivity of the bank marketing so they can improve it and also have the opportunity to lead their business to make different finance recommendations to their clients.

The second group of stakeholders identified are the Data Analysts, they are supposed to use the system as a information source, they will apply filters so they can obtain information in different points of view, they will also explore segments of information and exporting information so they can create their own conclusions from the data analyzed and let the Management Team and Executives know this conclusions so they can make decision based on this on this information.

Finally, the last stakeholder identified are the technical team, they are responsible for the development of the system; and the integration and the application´s maintenance, this includes the data loading and the users and profiles managements

1.2 Scope

This Software Architecture Document aims to provide a comprehensive understanding of what the System will be worked on, basically this document is based on how the system will be made, the decisions that will be taken in terms of development.

This document is expected to make an establishment of the way the system will be made, explaining the decisions taken by the team, this includes the technologies, the architectures, the views, goals and representations; so this way the team will be working on the same line and the same design, making a certain way of contract between the team members.

1.3 Definitions, Acronyms, and Abbreviations

4+1 View Model: An architectural framework that describes software architecture using five concurrent views: Use Case, Logical, Process, Deployment, and Implementation views.

Architectural Coverage: The extent to which use cases or scenarios exercise and test different architectural elements of the system.

Component Diagram: A UML diagram that shows the organization and dependencies among software components.

Class Diagram: A UML diagram that represents the static structure of a system by showing classes, their attributes, methods, and relationships.

Activity Diagram: A UML diagram that illustrates the flow of activities, decisions, and concurrent processes within a system.

Deployment Diagram: A UML diagram that shows the physical deployment of software components on hardware nodes and their interconnections.

Implementation View: An architectural view that describes the overall structure of the implementation model and the decomposition of software into layers and subsystems.

Logical View: An architectural view that shows the static structure, main entities, and business services that organize the system's functionality.

Process View: An architectural view that describes the system's decomposition into processes and illustrates concurrency and synchronization aspects.

Use Case View: An architectural view that captures the interaction between stakeholders and the system through use cases and scenarios.

Deployment View: An architectural view that illustrates the distribution of system components across the technological infrastructure.

API: Application Programming Interface - A set of protocols and tools for building software applications that defines how components should interact.

Backend: The server-side of an application that handles data processing, database interactions, and business logic.

CSV: Comma Separated Values - A file format that stores tabular data in plain text where each line represents a data record.

Express.js: A minimal and flexible Node.js web application framework that provides features for building web and mobile applications.

Frontend: The client-side of an application that provides the user interface and user experience.

MongoDB: A document-oriented NoSQL database that stores data in flexible, JSON-like documents.

NoSQL: A category of database management systems that do not use the traditional relational database model and SQL query language.

React.js: A JavaScript library for building user interfaces, particularly single-page applications with dynamic and interactive components.

Webstack: The collection of software technologies used together to create and run web applications, including frontend, backend, and database technologies.

Dashboard: A visual interface that displays key information, metrics, and data in an organized and easily digestible format.

KPIs: Key Performance Indicators - Measurable values that demonstrate how effectively an organization is achieving key business objectives.

Recommendation Engine: A system that analyzes data to suggest relevant products, services, or actions to users based on patterns and preferences.

Horizontal Scaling: The practice of adding more servers or nodes to a system to handle increased load, rather than upgrading existing hardware.

Lightweight Processes: Single threads of control that require minimal system resources and can be created and managed efficiently.

Heavyweight Processes: Groupings of lightweight processes that require more system resources and typically involve inter-process communication.

Scalability: The ability of a system to handle increased workload by adding resources without compromising performance.

1.4 References

NA

1.5 Overview

The rest of the document contains the Architectural Representation which describes the architecture's system and the ways it will be represented. The next point is the Architectural Goals and Constraints which includes the objectives and the special constraints that affects the system. Then comes the Use-Case View that includes a list of the use cases or scenarios. Then is the Logical View which describes the architecturally significant parts of the design model. Next is the Process View, the Deployment View, the Implementation View that are all defined in the Glossary and finally the Quality is a description of how the software architecture contributes to all capabilities of the system.

2. Architectural Representation

The software architecture of the Recommendation Engine and Banking Campaign Dashboard, BC Analyzer, is described through a set of views that allow the system to be represented from different perspectives, following the 4+1 view model.

- **Use Case View:** developed from the requested functional requirements, using use case diagrams and user story narratives, to capture the interaction of stakeholders with the system.
- **Logical View:** represented through class and logical component diagrams, showing the static structure, main entities, and business services that organize the system's functionality.
- **Process View:** developed from user stories, using activity diagrams that describe the flow of actions, decisions, and process concurrency during execution.
- **Deployment View (Physical view):** described through deployment diagrams, illustrating the distribution of system components on the technological infrastructure, including clients, servers, and connections.
- **Implementation View:** Represented with component diagrams that show the modular organization of the software, highlighting the separation between frontend, backend, data services, and security modules.

3. Architectural Goals and Constraints

This section describes the key architectural objectives and identified constraints that will guide the system design. The objectives are derived from the project's non-functional requirements and strategic goals to ensure that the architecture supports the required scalability, security, and usability. Constraints, on the other hand, reflect real-world restrictions imposed by the project context, such as data source and technical considerations, which influence design and implementation decisions.

Goals of Architecture

- **Usability and user experience:** The interface must be intuitive and responsive, with dynamic filters and interactive visualizations that allow for agile and dynamic exploration of the data.
- **Scalability:** The architecture must be designed with decoupled components to facilitate the incorporation of new data sources, financial products, and functionalities in future versions.
- **Maintainability:** The code must be modular, adequately documented, and follow standards that facilitate its evolution and maintenance.
- **Availability:** The application must be accessible via the web from multiple platforms and devices, ensuring continuous service.
- **Data integrity and quality:** The system must include validation mechanisms during data loading and generate data quality reports.

Constraints of Architecture

- **Initial Data Source:** The data comes from CSV/Excel files, so the architecture is optimized for reading, validating, and transforming these types of sources.
- **Technologies:** The webstack is designed to use React.js as the frontend, Express.js as the backend, and MongoDB as the database, which determines the data persistence model and its integration. The architecture must run on Windows environments, although portability to Linux environments is desirable in future versions.
- **Performance and Storage:** The architecture must be able to efficiently handle a dataset of approximately 45,211 banking campaign records, ensuring real-time queries and visualizations. In the event of an increase in volume, the system must scale horizontally to maintain performance.
- **Regulations and Security:** The system must comply with global and local data protection regulations, as it handles sensitive customer banking information. Encryption and access control measures must be implemented in subsequent versions.

4. Use-Case View

UC-01	Carga de Datos
Versión	1
Autor	Samuel Durán
Tipo	Primario
Dependencias	Ninguna
Actores	Equipo Técnico
Descripción	Este caso de uso inicia cuando el usuario decide cargar del dataset, ya sea CSV u otra fuente, que tiene la información de las campañas de marketing bancarios al sistema

Curso Normal de Eventos

Actor	Sistema
1, Inicia cuando el usuario entra al sistema y se dirige al módulo Carga de Datos	2, El sistema muestra una pantalla con la opción de cargar datos
3, El usuario carga los datos ya sea en formato CSV o Excel	4, El sistema valida que el formato del archivo sea válido
	5, El sistema valida la estructura de datos que contenga las columnas requeridas
	6, El sistema valida los tipo de datos sea el correcto
	7, El sistema identifica registros nulos o

	inconsistentes
	8, El sistema ingresa los datos a la base de datos
	9, El sistema genera el reporte de datos con porcentajes de calidad de los datos

Curso Alterno
Paso 4, El formato del archivo es inválido, el sistema manda un mensaje de error, el curso de eventos se devuelve al paso 3
Paso 5, La estructura de los datos es incorrecta, el sistema informa del error, el curso de eventos se devuelvo al paso 3
Paso 6, El tipo de datos es incorrecto, el sistema informa del error, el curso de eventos se devuelve al paso 3
Paso 8, Se genera un error en la base de datos, el sistema notifica del error en la base de datos, el flujo de eventos se devuelve al paso 3

UC-02	Filtros interactivos
Versión	1
Autor	Samuel Durán
Tipo	Primario
Dependencias	UC-01
Actores	Analista de Datos
Descripción	Este caso de uso inicia cuando el usuario ingresa al apartado de filtros y aplica alguno sobre uno o varios atributos

Curso Normal de Eventos

Actor	Sistema
1, Inicia cuando el usuario está en la página principal y se dirige al módulo de “visualizar datos”	2, El sistema muestra una pantalla con todos los filtros aplicables a atributos de edad, educación, préstamos, canal de contacto y mes
3, El usuario escoge el filtro que desea aplicar	4, El sistema despliega las opciones de valores disponibles para el filtro escogido
5, El usuario escoge el valor deseado	6, El sistema despliega la información con el filtro aplicado

	7, El sistema actualiza en tiempo real los gráficos de KPIs, la Tabla de Datos y los Contadores de métricas
8, El usuario escoge mas filtros	9, El sistema actualiza los datos con todos los filtros aplicados

Curso Alterno
Paso 9, El sistema detecta que no hay información con los filtros escogidos, el sistema manda un mensaje de “sin resultados, el flujo de eventos continúa en el paso 5

UC-08	Generar Reportes
Versión	1
Autor	Samuel Durán
Tipo	Primario
Dependencias	UC-03, UC-04, UC-05, UC-06
Actores	Directivo/Gerencia
Descripción	Este caso de uso inicia cuando el usuario desea generar un reporte descargable con los KPIs y filtros aplicados

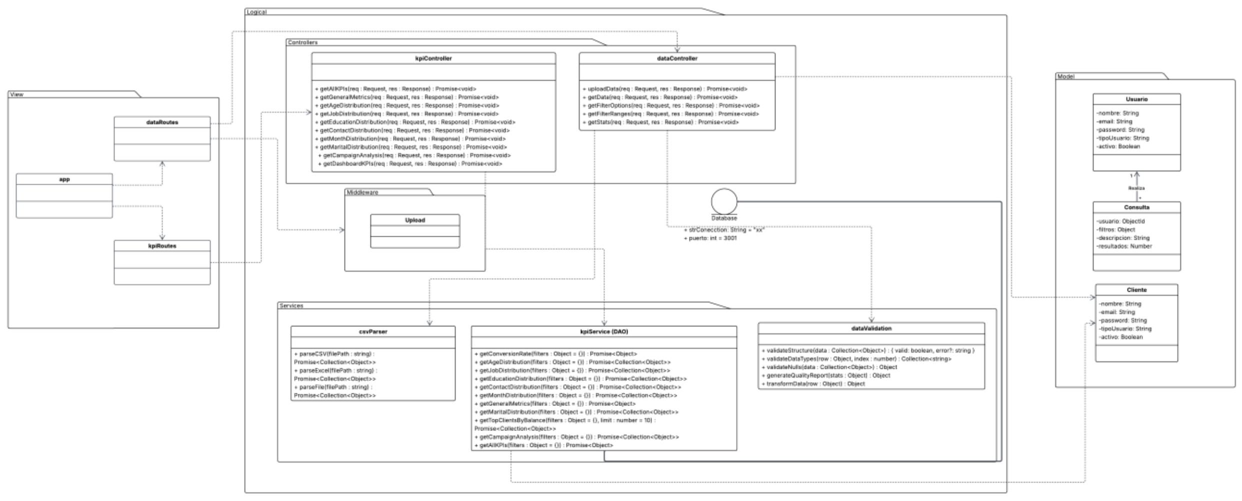
Curso Normal de Eventos

Actor	Sistema
1. Inicia cuando el usuario accede al módulo "Generar Reportes"	2. El sistema muestra las opciones de formato disponibles (PDF, Excel)
3. El usuario selecciona el formato del reporte	4. El sistema muestra las secciones disponibles para incluir en el reporte
5. El usuario selecciona los KPIs y gráficos a incluir	6. El sistema valida que los datos seleccionados estén disponibles

7. El usuario confirma la generación del reporte	8. El sistema genera el reporte con los filtros actualmente aplicados
	9. El sistema incluye todos los KPIs y gráficos seleccionados
	10. El sistema descarga el archivo generado

Curso Alterno
Paso 6: Algunos datos seleccionados no están disponibles. El sistema muestra qué elementos no pueden incluirse y permite continuar sin ellos. El flujo continúa en el paso 7. **Paso 8:** Error en la generación del reporte. El sistema muestra el mensaje "error al generar el reporte, intente nuevamente". El flujo regresa al paso 7.

5. Logical View


View
app
dataRoutes
kpiRoutes
Logical
Controllers
kpiController
+ getAllKPIs(req : Request, res : Response) : Promise<void>
+ getGeneralMetrics(req : Request, res : Response) : Promise<void>
+ getAgeDistribution(req : Request, res : Response) : Promise<void>
+ getJobDistribution(req : Request, res : Response) : Promise<void>
+ getEducationDistribution(req : Request, res : Response) : Promise<void>
+ getContactDistribution(req : Request, res : Response) : Promise<void>
+ getMonthDistribution(req : Request, res : Response) : Promise<void>
+ getMaritalDistribution(req : Request, res : Response) : Promise<void>
+ getCampaignAnalysis(req : Request, res : Response) : Promise<void>
+ getDashboardKPIs(req : Request, res : Response) : Promise<void>
dataController
+ uploadData(req : Request, res : Response) : Promise<void>
+ getData(req : Request, res : Response) : Promise<void>
+ getFilterOptions(req : Request, res : Response) : Promise<void>
+ getFilterRanges(req : Request, res : Response) : Promise<void>
+ getStats(req : Request, res : Response) : Promise<void>
Middleware
Upload
Database
+ strConecction: String = "xx"
+ puerto: int = 3001
Services
csvParser
+ parseCSV(filePath : string) : Promise<Collection<Object>>
+ parseExcel(filePath : string) : Promise<Collection<Object>>
+ parseFile(filePath : string) : Promise<Collection<Object>>
kpiService (DAO)
+ getConversionRate(filters : Object = {}) : Promise<Object>
+ getAgeDistribution(filters : Object = {}) : Promise<Collection<Object>>
+ getJobDistribution(filters : Object = {}) : Promise<Collection<Object>>
+ getEducationDistribution(filters : Object = {}) : Promise<Collection<Object>>
+ getContactDistribution(filters : Object = {}) : Promise<Collection<Object>>
+ getMonthDistribution(filters : Object = {}) : Promise<Collection<Object>>
+ getGeneralMetrics(filters : Object = {}) : Promise<Object>
+ getMaritalDistribution(filters : Object = {}) : Promise<Collection<Object>>
+ getTopClientsByBalance(filters : Object = {}, limit : number = 10) : Promise<Collection<Object>>
+ getCampaignAnalysis(filters : Object = {}) : Promise<Collection<Object>>
+ getAllKPIs(filters : Object = {}) : Promise<Object>
dataValidation
+ validateStructure(data : Collection<Object>) : { valid: boolean, error?: string }
+ validateDataTypes(row : Object, index : number) : Collection<string>
+ validateNulls(data : Collection<Object>) : Object
+ generateQualityReport(stats : Object) : Object
+ transformData(row : Object) : Object
Model
Usuario
-nombre: String
-email: String
-password: String
-tipoUsuario: String
-activo: Boolean
1
Realiza
*
Consulta
-usuario: ObjectId
-filtros: Object
-descripcion: String
-resultados: Number
Cliente
-nombre: String
-email: String
-password: String
-tipoUsuario: String
-activo: Boolean

Link: Vista Lógica

5.1 Overview

In this project, the Logical View is divided into three layers: View, Logical, and Model.

- The View layer contains the user interface (webPage), which manages the interaction with the end user.
- The Logical layer includes the controller and modules for filtering, reporting, KPI visualization, and user/query management. It represents the system's core business logic.
- The Model layer defines the data entities (Usuario, Cliente, Consulta, TipoUsuario) that represent the application's domain and are closely tied to the database structure.

This layered decomposition allows clear separation of concerns and simplifies system maintenance and scalability.

5.2 Architecturally Significant Design Packages

1. View Package
    - webPage: Represents the user interface component through which users interact with the system. It communicates with the Controller in the Logical layer.
2. Logical Package
    - Controller: Manages requests from the View, coordinates interactions between components, and orchestrates data processing.
        - Responsibilities: Logging data, generating graphs, applying filters, creating tables, and producing reports.
    - KPIgrapher: Provides methods to generate KPI metrics such as contact numbers, client distribution, conversion rate, campaign rate, age rate, and distribution channels.
    - Filter: Manages filtering logic with support for custom filters, saving, and applying them to datasets.
    - TableCreator: Responsible for creating tables from data collections, applying filters, setting page sizes, and exporting tables in different document formats.
    - ReportGenerator: Produces reports from processed data.
    - ModuloUsuarios / GestorUsuarios: Handles user validation and CRUD operations for users.
    - ModuloConsultas / GestorConsultas: Manages queries within the system with CRUD functionality.
    - DAO and iConnection: Encapsulate data access logic and database connection details, including connection string and port.
3. Model Package (Data Layer)
    - Usuario: Represents a system user, with attributes id, nombre, and tipoUsuario.
    - TipoUsuario (Enumeration): Defines user roles such as DIRECTIVO, ANALISTA_DATOS, and EQUIPO_TECNICO.
    - Cliente: Represents a client entity with attributes id, age, and job.

Consulta: Represents a query performed by a user, with attribute consulta.

5.3 Use-Case Realizations

UC-01 Data Loading Realization

Participating Architectural Elements:

- **Presentation Layer:**

FileUploadComponent, ProgressIndicator, ErrorDisplay

- **Logic Layer**:

CSVParserService, DataValidationService, QualityReportGenerator, DataTransformationEngine

- **Persistence Layer**:

DatabaseInterface, MongoDBAdapter, DataValidationRepository

Interaction Flow:

1. **FileUploadComponent** receives CSV/Excel file from Technical Team user

2. **CSVParserService** parses the file structure and extracts raw data

3. **DataValidationService** performs three-stage validation:

   - File format validation

   - Column structure validation

   - Data type validation

4. **DataTransformationEngine** processes valid records and identifies inconsistencies

5. **DatabaseInterface** manages MongoDB insertion operations with transaction control

6. **QualityReportGenerator** calculates data quality metrics and generates reports

7. Results propagate back through the component hierarchy to update the UI

Architectural Significance:

This realization exercises the complete three-layer architecture and demonstrates critical architectural decisions:

- **Error handling strategy** across all layers with graceful degradation
- **Transaction management** for data integrity with large datasets (45,000+ records)
- **Memory management** for processing large files without system overload
- **Modular validation** allowing independent testing and maintenance of validation rules

UC-02 Interactive Filters Realization

Participating Architectural Elements:

- **Presentation Layer**:

FilterComponent, MetricDisplay, TabularView, GraphicsRenderer

- **Logic Layer**:

FilterEngine, QueryBuilder, KPICalculator, RealTimeProcessor

- **Persistence Layer:**

QueryInterface, MongoDBQueryOptimizer, IndexManager

Interaction Flow:

1. **FilterComponent** captures user filter selections (age, education, loans, contact channel, month)

2. **QueryBuilder** constructs optimized MongoDB queries with proper indexing

3. **QueryInterface** executes queries against the 45,000+ record dataset

4. **RealTimeProcessor** manages concurrent data processing for multiple UI components

5. **KPICalculator** recomputes metrics based on filtered data
6. **GraphicsRenderer** updates visualizations in real-time
7. **TabularView** refreshes data display with pagination support

Architectural Significance:

This realization demonstrates critical performance and scalability decisions:

- **Real-time processing architecture** supporting sub-second response times
- **Query optimization strategy** using MongoDB indexing for large datasets
- **Concurrent processing** allowing multiple filter operations without blocking UI
- **Component decoupling** enabling independent updates of graphics, tables, and counters
- **State management** maintaining filter consistency across multiple UI components

UC-08 Report Generation Realization

Participating Architectural Elements:

- **Presentation Layer**:

ReportConfigInterface, FormatSelector, DownloadManager

- **Logic Layer**:

ReportEngine, KPIAggregator, FilterStateManager, DocumentGenerator

- **Persistence Layer**:

DataExtractionService, ReportTemplateRepository

Interaction Flow:

1. **ReportConfigInterface** captures user report requirements (format, KPIs, sections)
2. **FilterStateManager** captures current filter state for report context
3. **KPIAggregator** consolidates data from multiple dependent use cases (UC-03, UC-04, UC-05, UC-06)
4. **DataExtractionService** retrieves filtered datasets matching current user context
5. **ReportEngine** orchestrates document generation based on selected format (PDF/Excel)
6. **DocumentGenerator** creates formatted output with graphics and data tables
7. **DownloadManager** handles file delivery to user with proper error handling

Architectural Significance:

This realization demonstrates complex architectural integration:

- **Cross-component integration** aggregating functionality from multiple use cases
- **State preservation** maintaining filter context throughout report generation

- **Resource management** handling memory-intensive operations for large reports

- **Format abstraction** supporting multiple output formats through unified interface

- **Error recovery** gracefully handling missing data or generation failures

- **Security considerations** ensuring users only access data within their permissions

Common Architectural Patterns:

All realizations demonstrate key architectural patterns:

- **Separation of Concerns**: Clear boundaries between presentation, business logic, and data access

- **Dependency Injection**: Loose coupling between components for testability and maintenance

- **Error Propagation**: Consistent error handling strategy across all architectural layers

- **Performance Optimization:** Strategic use of caching, indexing, and asynchronous processing

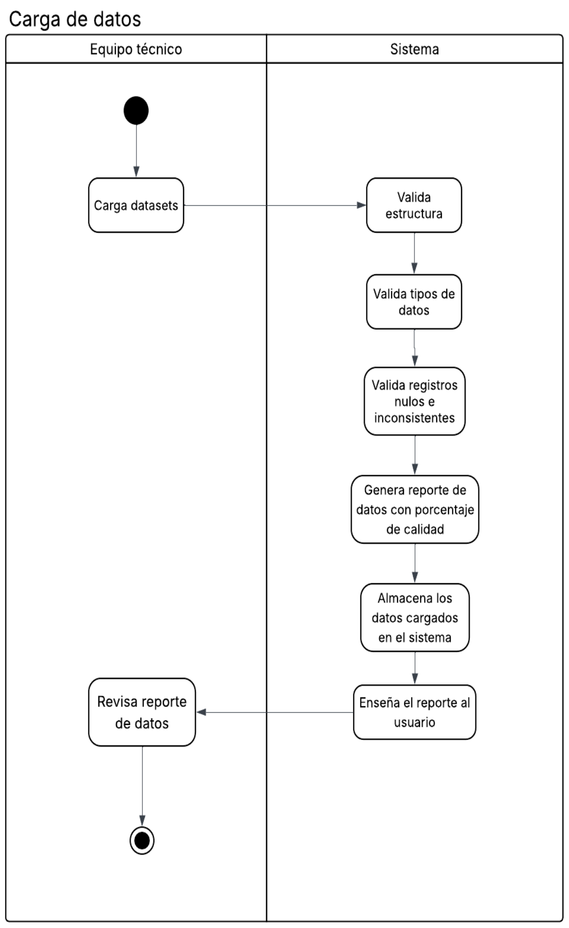
6. Process View

The system architecture is decomposed into a set of lightweight processes (individual threads of control) and heavyweight processes (groups of processes) that interact with each other to provide the functionality described in the project.

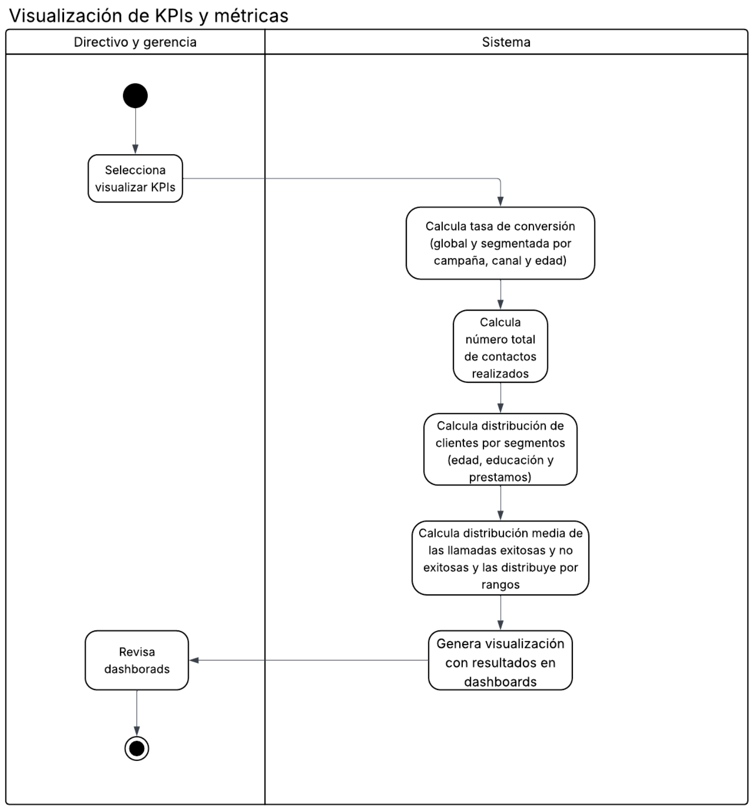
Dashboard for queries and reports:

The main processes are closely linked to queries, the KPI grapher, the table builder, and filters, all of which are responsible for user interaction with the data. The following processes are identified:

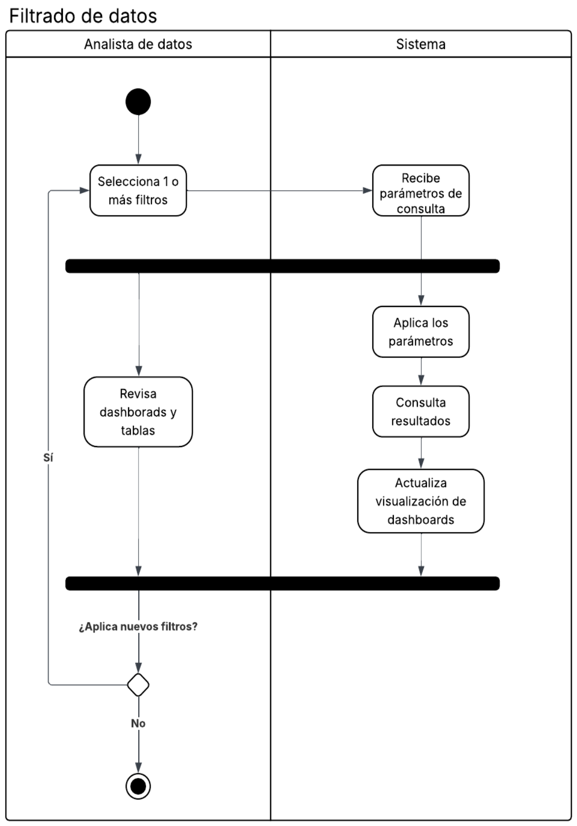
- Data loading and validation process: Executed when the user uploads a dataset (CSV/Excel). The structure is validated, inconsistent records are detected, and a data quality report is generated.


Carga de datos
Equipo técnico
Sistema
Carga datasets
Valida estructura
Valida tipos de datos
Valida registros nulos e inconsistentes
Genera reporte de datos con porcentaje de calidad
Almacena los datos cargados en el sistema
Enseña el reporte al usuario
Revisa reporte de datos

- Interactive filter application process: Allows the user to apply multiple filters simultaneously. Each modification triggers a process that recalculates KPIs and updates visualizations in real time.

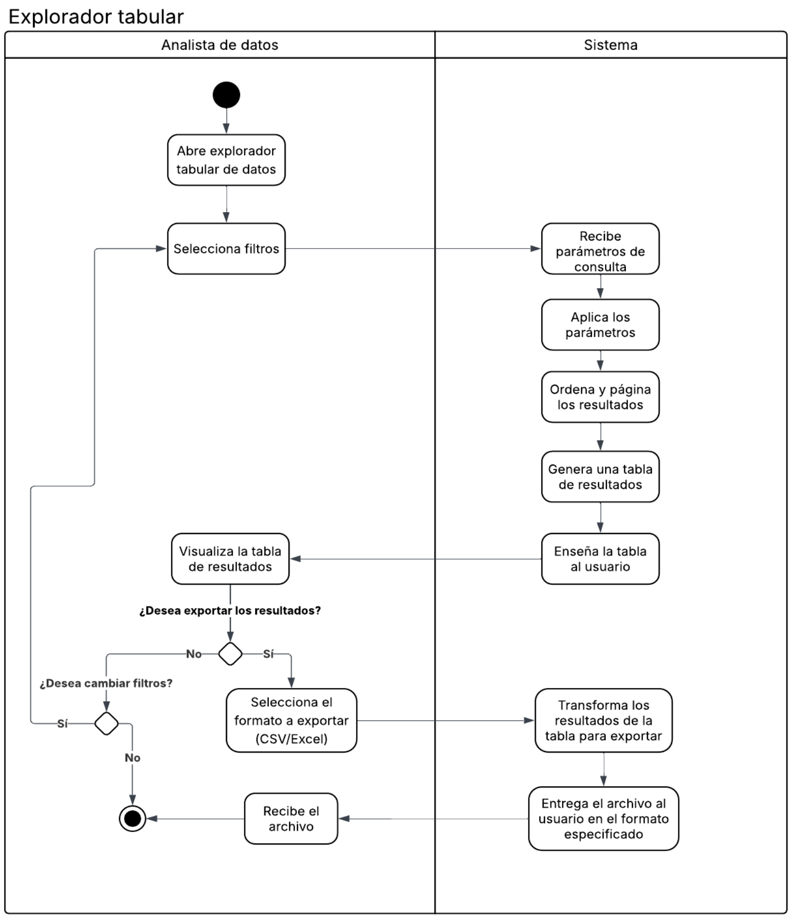


Visualización de KPIs y métricas
Directivo y gerencia
Sistema
Selecciona visualizar KPIs
Calcula tasa de conversión (global y segmentada por campaña, canal y edad)
Calcula número total de contactos realizados
Calcula distribución de clientes por segmentos (edad, educación y prestamos)
Calcula distribución media de las llamadas exitosas y no exitosas y las distribuye por rangos
Genera visualización con resultados en dashboards
Revisa dashborads

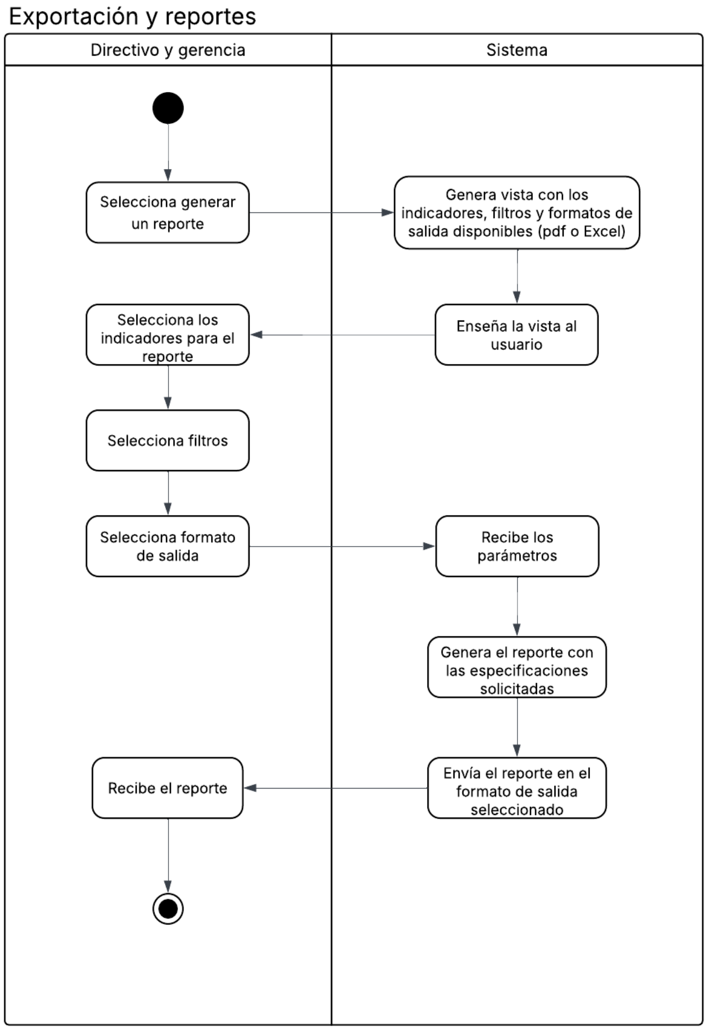


Filtrado de datos
Analista de datos
Sistema
Selecciona 1 o más filtros
Recibe parámetros de consulta
Aplica los parámetros
Consulta resultados
Actualiza visualización de dashboards
Revisa dashborads y tablas
Sí
¿Aplica nuevos filtros?
No

- Data visualization and exploration process: Keeps interactive charts and tables synchronized with the applied filters.


Explorador tabular
Analista de datos
Sistema
Abre explorador tabular de datos
Selecciona filtros
Recibe parámetros de consulta
Aplica los parámetros
Ordena y página los resultados
Genera una tabla de resultados
Enseña la tabla al usuario
Visualiza la tabla de resultados
¿Desea exportar los resultados?
No
Sí
¿Desea cambiar filtros?
Sí
No
Selecciona el formato a exportar (CSV/Excel)
Transforma los resultados de la tabla para exportar
Entrega el archivo al usuario en el formato especificado
Recibe el archivo

- Report generation and export process: Creates PDF or Excel reports that consolidate KPIs, charts, and filters used.



Exportación y reportes
Directivo y gerencia
Sistema
Selecciona generar un reporte
Genera vista con los indicadores, filtros y formatos de salida disponibles (pdf o Excel)
Enseña la vista al usuario
Selecciona los indicadores para el reporte
Selecciona filtros
Selecciona formato de salida
Recibe los parámetros
Genera el reporte con las especificaciones solicitadas
Envía el reporte en el formato de salida seleccionado
Recibe el reporte

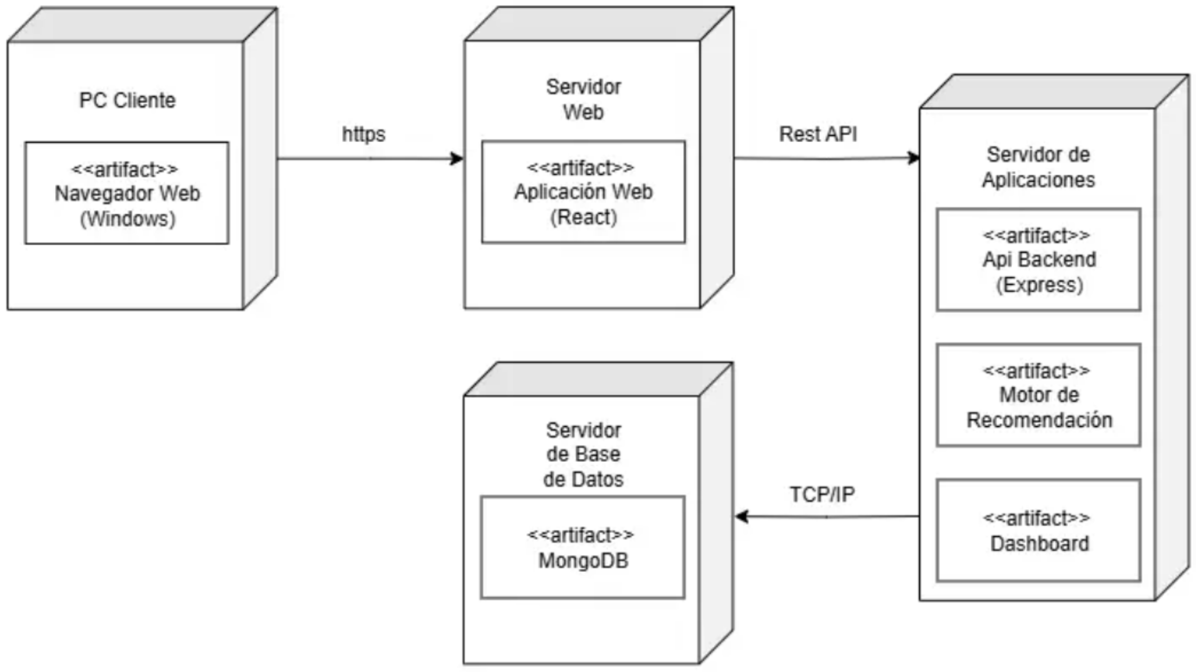
The processes in this phase communicate primarily through internal HTTP/REST calls between the UI (React.js) and the backend (Express.js).

Communication between processes:

- Frontend (React.js) - Backend (Express.js): Synchronous communication via HTTP/REST API.
- Backend (Express.js) - Database (MongoDB): Concurrent queries using the native MongoDB driver for Node.js (on which Express.js operates), with support for asynchronous operations and persistent connection management.

All processes run in a Windows environment, taking advantage of the native compatibility of Node.js (and Express.js by extension) and MongoDB with this operating system. Horizontal scalability can be achieved by replicating Express.js server instances and configuring MongoDB clusters if necessary.

7. Deployment View

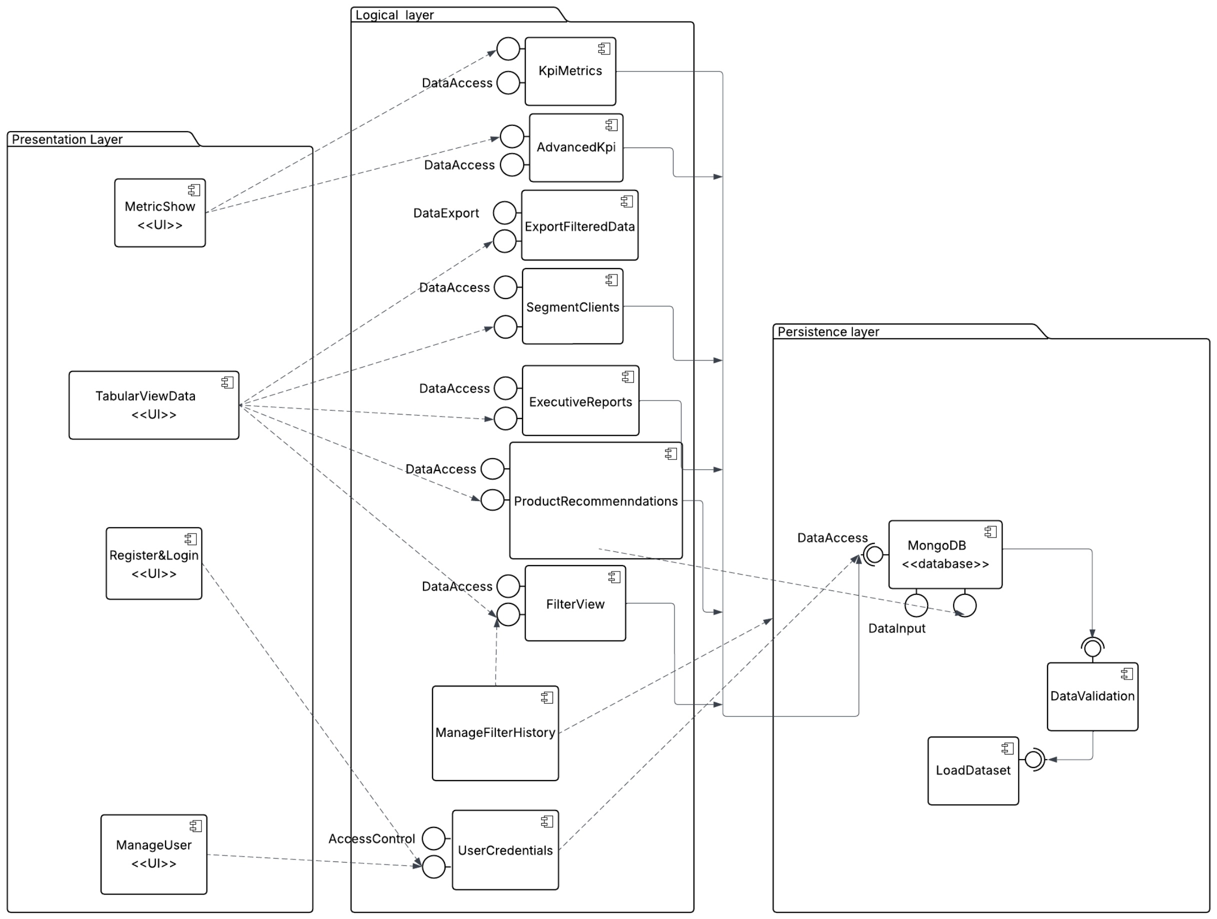

PC Cliente
<<artifact>> Navegador Web (Windows)
https
Servidor Web
<<artifact>> Aplicación Web (React)
Rest API
Servidor de Aplicaciones
<<artifact>> Api Backend (Express)
<<artifact>> Motor de Recomendación
<<artifact>> Dashboard
TCP/IP
Servidor de Base de Datos
<<artifact>> MongoDB

The deployment view, also called the physical view, is composed of four layers: the Client PC, the Web Server, the Application Server, and the Database Server.

- Client PC: Runs a Web Browser on Windows, where the user accesses the main application. The browser communicates with the Web Server over the HTTPS protocol.
- Web Server: Hosts the Web Application developed with React. It provides the frontend interface and communicates with the Application Server using REST APIs.
- Application Server: Executes the core business logic through the Backend API built with the Node.js framework Express. It also includes the Recommendation Engine and the Dashboard module.
- Database Server: Provides persistent storage through MongoDB, which communicates with the Application Server over TCP/IP.

All nodes are interconnected through standard communication protocols (HTTPS, REST, TCP/IP), ensuring secure and efficient interaction between the presentation layer, business logic, and data storage.

8. Implementation View



The implementation view for this project decomposes into 3 layers, the presentation layer, the logical and the persistence one, some notable subsystems to mention briefly could be the database which is MongoDB, and it's expected to hold all the information recovered from the csv files, other notable subsystems could be the Tabular View which is responsible for showing most of the data information in the webapp and let's it be changed by the filter views.

8.1 Overview

The presentation layer is fully responsible with the subsystems that the user can directly interact with, none of these subsystems can directly interact with the database, the logical layer involves with incapsulating all the logic between the metrics, filtering, reports and lastly, the persistence layer involves the database, offering some interfaces for accessing it and others for filling it and loading the data set.

8.2 Layers

The system is divided in these layers and subsystems:

1. Presentation layer: The UI consists of 3 different main functionalities.
   - Metric Show has the responsibility to interact and show all the graphics collected while analyzing the data.
   - Tabular View Data will show all the processed data with the applied filters and will also allow the user to make a report with the current filters.

- Manage User and Register/Login will have more emphasis on phase 2, nonetheless they are already contemplated to interact with the user credentials in the project.
- In Manage User the user will have the ability to change some of their credentials including the password, username and profile picture.
- The Login UI would be the first screen the user will see, in this interface it will be able to insert his credentials in the system to gain access to its contents.
- In the Register UI the user must be able to create a new account, some of this account's permissions will be limited until an administrator changes their role.
- A main menu is contemplated; it does not hold any relevance to the implementation diagram since its purpose is merely to navigate through the system, but it doesn't require access to the functionalities of the other modules.

2. Logical Layer:

- Filter View will alter almost immediately the tabulated view UI based only on the preferences of the user, this filters will range through different values, most of them depending on the information in the .csv file, they will also be able to be combined with each other, also is worth to notice that the graphs will also be affected by these filters.
- The KPI Metrics component will calculate all KPI and indicators, such as the conversion rate, quantity of contacts, client distribution and call's length, these will have filters as mentioned before.
- In the Tabular Data UI, the user will be able to export the data with the preferences and filters previously inserted, the program will use the Export Filter Data module to build a .csv, .xsl or pdf file depending on the user preference with the filtered data.
- The Tabular Data UI will also communicate with the Manage Filter History module to store different groups of filters, reuse them and keep a history.

  2.1 Logical Layer phase 2: These are some contemplations for the logical layer phase 2 needs:

  - A Module for automatically segmenting clients, mostly for identifying them in diverse groups.
  - More advanced KPI Metrics and different scenery simulations to predict impact and visualize different parameters, these will be useful also in the executives reports modules which will be able to make reports with these more advanced filters and functionalities.

3. Persistence Layer:

- A mongo database with the capacity to receive and output data.
- Scripts that can interact with the MongoDB and correctly insert all the .csv dataset in the database.
- Data validation for each of the entries on the .csv document.

Size and performance:

- The system and its database needs to successfully manage more than 45,211 registers, since that is the stipulated in the SRS.
- The filters and metrics must be shown and updated in real time.

9. Quality

- Extensibility: This software works with different modules, which means more metrics or filters could be added or deleted without touching the database persistence, also all documentation will be updated regularly to include all new available functionalities.
- Reliability: To avoid problems with the management of the data set, all data will be validated before entering the database.
- Portability: The use of MongoDB for the persistence approach will allow the software to run on cloud or containers if needed.
- Security and privacy: In the second phase the software will make use of different credentials per user which will allow it to protect the sensible information the software manages, also all credentials and sensible information regarding the user must be safely encrypted.
- Usability: The program will count with interactive visuals with intuitive filters that the user will be able to use for his preferences, also, all graphics and data visualizations must be responsive, enabling the user to interact almost seamlessly with these functionalities.