***Priyanka Gupta (15/03/2025)***

**Unify Application**

**Introduction**

* Unify Application is a **no-code application builder** for enterprise-grade applications.
* Enables rapid development using a **drag-and-drop interface**.
* Allows non-programmers to contribute to software development.

**Types of Supported Applications**

1. **Web Applications**
   * Accessible via browsers on any device.
   * Fully responsive to desktops, tablets, and mobiles.
   * No downloads required; always up-to-date.
2. **Mobile Native Applications**
   * Optimized for iOS and Android.
   * Utilizes device-specific features (camera, GPS, push notifications).
   * Provides a **highly responsive** and **intuitive** mobile experience.

**Key Building Blocks of an Application**

1. **UI Components** (Prebuilt elements for application interfaces)
   * **Layout Components**: Structure and organize the interface.
   * **Rich Components**: Advanced features like tables and charts.
   * **Basic Components**: Simple elements like buttons and text fields.
   * **Repeatable Components**: Dynamic elements for lists and grids.
2. **Canvas**
   * The central workspace for designing applications.
   * Drag-and-drop functionality for easy customization.
3. **Data Sources**
   * Fetch and manage data from **external apps** or **UnifyApps' internal database**.
   * Enables seamless integration with multiple data sources.

**Connect to Data Source**

**Add Data Source**

**Introduction**

* Data sources enable integration with external applications and databases.
* Essential for data retrieval, analytics, and app functionality.

**Compatible Data Sources**

1. **Storage by Unify Apps**
   * Fetch data directly from Unify Objects.
   * Ensures efficient retrieval and usage within applications.
2. **Analytics by Unify Apps**
   * Perform analytics and generate reports from stored data.
   * Ideal for business insights and querying.
3. **Direct Application Connectors**
   * Connect to third-party applications seamlessly.
   * No additional processing required for data integration.
4. **Callable by Automations**
   * Fetch data from multiple sources and apply logic within automation scripts.
   * Provides flexibility for complex data processing.

**Steps to Add Data Sources**

1. **Access the Data Sources Panel**
   * Navigate to the **'Data Sources'** section in UnifyApps.
   * Click the **'+'** button to add a new source.
2. **Select an App**
   * Browse or search for the required data source.
3. **Define Actions and Queries**
   * Set up analytics or SQL queries for data retrieval.
4. **Configure the Data Source**
   * Enter API keys, authentication tokens, or credentials.
5. **Save and Test**
   * Save settings and test the connection for accuracy.

**Storage by Unify Apps**

**Overview**

* **Storage by Unify Apps** is a connector that allows data fetching and manipulation from **Unify Apps Objects**.
* Enables direct integration of stored data into **low-code applications**.

**Adding "Storage" Data Source**

1. **Navigate to the "Data" section** (left panel of the application).
2. **Click the "+" button** to add a new data source.
3. **Search & select "Storage by Unify Apps"** from available options.
4. **Link the data source** to enable data access as **data pills** in the application.

**Configuring the Connector**

* Select an **action** to define how data will be managed.

**Common Actions**

1. **Count Records** – Retrieves total number of records.
2. **Create Record** – Adds new data based on user actions.
3. **Delete Records** – Removes records from storage.
4. **Fetch Records** – Retrieves data into the application.
5. **Get Records by IDs** – Fetches specific records using unique IDs.
6. **Update Record by ID** – Modifies or creates a record based on ID.
7. **Update Records by Query** – Updates multiple records matching specific criteria.

**Advanced Actions**

1. **Aggregate Metadata** – Summarizes and analyzes stored data.
2. **Export Records** – Extracts data in CSV, XLS, or XLSX formats.
3. **Import Records** – Bulk adds data from external files.
4. **Semantic Search Records** – Enables natural language-based searching.
5. **Share Records** – Grants access to records for users/teams.

**Defining Input for Actions**

* **Select Object**: Choose the object from which to fetch data.
* **Search Object**: Define search parameters (Field + Value).
* **Conditional Counting**:
  + **Where**: Set conditions for filtering records.
  + **Field + Operator + Value**: Define filtering logic.
* **Event Handlers**: Configure responses for success and failure scenarios.
* **Reviewing Output**: Retrieved data appears as **data pills** in the application.

**Mapping Data to UI Components**

* Data fetched via Storage by Unify Apps can be mapped to **UI components and logic flows**.

**Analytics By Unify Apps**

**Overview**

* **Analytics by Unify Apps** enables running queries and aggregation functions on data stored in **Unify Objects** or **Standard Platform Objects**.
* Helps in creating **Reporting Dashboards** within low-code applications.

**Adding "Analytics" Data Source**

1. **Navigate to the "Data" section** (left panel of the application).
2. **Click the "+" button** to add a new data source.
3. **Search & select "Analytics by Unify Apps"** from the available options.
4. **Integrate with Unify Apps Objects** to analyze stored data.

**Configuring Analytics by Unify Apps**

* **Choose an action** for querying and analyzing data.

**Common Actions**

1. **Analytics Query** – Runs a query on reports.
2. **Execute Analytics SQL Query** – Allows writing custom SQL queries on Analytics Databases.

**Advanced Actions**

1. **Aggregate Metadata** – Provides metadata for filtering, sorting, and searching reports.
2. **Export Records** – Extracts data in CSV, XLS, or XLSX formats.

**Defining Inputs for Analytics Queries**

* **Select Group**: Choose a group where the report is stored (**Storage Group** or **Platform Group**).
* **Select Base Report**: Pick the specific report for querying.
* **Define Projections**:
  + **Field Name**: Select a field for aggregation.
  + **Aggregation Function**: Use standard or custom aggregation (SUM, AVG, COUNT, etc.).
  + **Alias**: Rename the field in the output.
  + **Percentage Change**: Option to calculate field's percentage change.
  + **Custom Projection**: Allows advanced query customizations.
* **Conditional Counting**:
  + **Where**: Apply filters for fetching specific records.
  + **Field + Operator + Value**: Define filter conditions.
* **Search Object**: Case-insensitive search across multiple fields.

**Reviewing Output & Mapping Data**

* Data is available as **data pills** for easy integration with UI components.

**Using Analytics Node in Stat Cards**

1. **Add Analytics Data Source**: Use "Analytics Query" action and select an object.
2. **Configure Projections**: Choose correct fields and aggregation functions.
3. **Use Aggregation Functions**:
   * **Group Function**: Returns a list of all field values.
   * **Count Function**: Returns the count of records.
   * **Group + Count Combination**: Provides unique values and their record counts.
4. **Map Output to Stat Card**: Select the **data pill** for display.

**Callable by Unify Apps**

**Overview**

* **Callable** acts as a data source that triggers an **automation** and returns its output.
* Used for **complex data integrations** from multiple sources.

**Adding Callable as a Data Source**

1. **Open "Data Source"** section (left panel of the app).
2. **Click the "+" button** to add a new data source.
3. **Search for "Callable"** and select it.

**Configuring the Callable**

* After adding Callable, select **“Call Automation”** to trigger an automation.

**Defining Input for the Callable**

1. **Select Automation** – Choose the automation to execute.
2. **Enter Parameters** – Provide required input values or **map data pills** as inputs.

**Reviewing Output of the Callable**

* Check the **output schema** to identify data pills.
* Run a **preview test** to verify automation output.

**Utilizing the Output**

* Callable's **data pills** can be mapped to UI components and logic flows.
* Output can be **directly used** in different parts of your application.

**Data Source Settings**

**Overview**

* Query execution in web and mobile apps can be **automatic** or **manual**.
* Optimizing execution settings improves **performance, efficiency, and user experience**.

**Run Behavior Options**

|  |  |
| --- | --- |
| **Run Behavior** | **Description** |
| **Automatic** | Executes when its associated block becomes visible in the UI. |
| **Manual** | Runs only when triggered by a user action (e.g., clicking a button). |

**Running a Query Automatically**

* Queries execute **without user input** (e.g., on page load or data changes).
* **Steps:** *Go to "Run Behavior" in Input Tab → Select "Automatic"*.

**Running a Query Manually**

* Ideal for **large datasets** to avoid **performance issues or rate limits**.
* **Steps:** *Go to "Run Behavior" in Input Tab → Select "Manual"*.

**Additional Run Behavior Options**

1. **Running on Page Load**
   * Ensures **latest data loads immediately**.
   * *Enable via Advanced Tab → Select "Run this query on page load".*
2. **Running Periodically**
   * Useful for **real-time data updates** (e.g., every 10 seconds).
   * *Enable via Advanced Tab → Select "Run this query periodically" → Set interval (ms).*
3. **Refreshing on Window Focus**
   * Ensures users always see the latest data when returning to the app.
4. **Preventing Query Execution**
   * Queries can be disabled until **specific conditions are met** (e.g., *run only if user count >10*).
   * Option to display a **custom error message** when a query is blocked.

**Testing Data Sources**

**Overview**

* Ensures **data accuracy, integrity, structure, and business logic**.
* Helps verify that the data source returns the **expected format**.

**Run Preview**

* Fetches data dynamically to **validate schema and structure**.
* Confirms that **returned data matches expected format** (e.g., JSON structure).

**Output Section**

* Displays **JSON-formatted output** for the data source.
* Verifies API responses, including:
  + **Schema consistency across environments.**
  + **Presence and correctness of all required fields.**
  + **Testing schema changes before deployment.**

**Validating Business Logic**

* Ensures data follows **business rules** (e.g., *price cannot be negative*).
* Validates **calculations** (e.g., *totals, averages*).

**Error Handling**

* Displays **error codes and messages** when a configuration issue exists.
* Helps in **debugging and restructuring** data sources.

**JS Variables and Functions**

**Variables**

* **Purpose:** Store and manage data dynamically.
* **Usage:** Define conditions, manipulate UI components, and enable interactivity.

**Data Types**

1. **Number** – Integers & floats.
2. **String** – Text values (single/double quotes, backticks).
3. **Boolean** – true / false.
4. **Object** – Key-value pairs, can include arrays and functions.
5. **Array** – Ordered list of values.

**Variable Operations**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Applicable Types** | **Description** |
| Set value | All | Assigns a new value. |
| Add to beginning | Array, String | Inserts an element at the start. |
| Add to end | Array, String | Appends an element. |
| Remove from first | Array | Deletes the first element. |
| Remove from last | Array | Deletes the last element. |
| Toggle value | Boolean | Flips true ↔ false. |
| Increment by | Number | Increases value. |
| Decrement by | Number | Decreases value. |
| Merge properties | Object | Combines properties into an object. |

**Functions**

* **Purpose:** Execute reusable blocks of JavaScript code.
* **Usage:** Process inputs, manipulate data, and return results.

**Use Case Example**

**Fetching Current Date** (Formatted as MM-DD-YYYY)

* Define a JavaScript function.
* Use data pills as input.
* Return a formatted date string for UI mapping.

**Defining Layout in Your App Interface**

**What is Layout?**

Layout determines how visual elements are arranged in your app's UI. It helps in structuring components like buttons, text, and images to enhance usability and design consistency.

**Main Layout Components**

**1.Stack**

* A flexible UI component that **organizes elements** either **vertically or horizontally**.
* **Automatically manages spacing** and alignment between child components.
* Can **contain an indefinite number** of elements.

**Best for:** Dynamic content, lists, repeated elements.

**2. Container**

* A **rigid layout component** used to group UI elements.
* Can hold a **fixed number of components**.
* Helps in **structuring content** within a confined area.

**Best for:** Section-based layouts, specific UI groupings.

**Container Component in UI Design**

**What is a Container?**

A **Container** is a layout component used to **group and organize UI elements** into a structured and rigid format. It helps maintain consistency and improves visual hierarchy within your application.

**Primary Use Cases**

**Grouping Related Components**

* Organize **form fields, buttons, labels** into a single structured section.

**Responsive Web Layouts**

* Adapt column layouts **based on screen size** (e.g., 4-column on desktop, 2-column on mobile).

**Consistent Styling**

* Apply **uniform padding, margins, and styling** to all child components.

**Configuring a Container**

**Choosing a Layout**

* Select from various **column structures**:
  + **Full width, 1:1, 1:2, 2:1**
  + **Up to 6 columns** for precise content distribution.

**Adding Components**

* Drag and drop elements **(text, images, charts, etc.)** inside the container.
* **Nesting** is supported: A **stack or another container** can be placed inside a container for **complex layouts**.

**Styling the Container**

* Adjust spacing and alignment using **gap, padding, and margin settings**.
* Use the **‘+’ button** inside the appearance tab to access **height, width, and other styling properties**.
* Apply a **theme** for consistent design.

**Key Styling Properties:**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| **Padding** | Adds space inside the container around its content. |
| **Height** | Controls the vertical size of the container. |
| **Min-Height** | Ensures the container doesn’t shrink below a set value. |
| **Max-Height** | Prevents the container from growing too tall. |
| **Gap** | Defines spacing between child elements. |
| **Margin** | Creates space around the container for better separation. |

**Stack Component in UI Design**

**What is a Stack?**

A **Stack** is a flexible UI component that **groups and organizes elements** either **vertically (Column)** or **horizontally (Row)**. It automatically manages **spacing and alignment**, making it an essential tool for structured UI design.

**Creating a Stack with Child Elements**

**Step 1: Select the Stack**

* Choose the **Stack component** from the component panel.
* Click on it within your layout.

**Step 2: Add Components**

* In the **hierarchy panel**, click **"Add Component"** next to the Stack.
* Select and insert **text, images, buttons, or other UI components**.

**Step 3: Organize Child Elements**  
Adjust **layout settings** to control how child components are arranged:

|  |  |
| --- | --- |
| **Setting** | **Description** |
| **Direction** | Arrange child elements **vertically (Column) or horizontally (Row)**. |
| **Align Items** | Aligns child components **along the cross-axis** (e.g., **Start, Center, End, Stretch**). |
| **Justify Content** | Aligns child components **along the main axis** (e.g., **Start, Center, Space Between**). |

**Styling Your Stack**

**Key Styling Properties:**

|  |  |
| --- | --- |
| **Property** | **Description** |
| **Gap** | Sets space between child components (**none, small, medium, large**). |
| **Padding** | Adds space inside the Stack’s border (**none, small, medium, large**). |
| **Margin** | Adds outer space around the Stack (**none, small, medium, large**). |
| **Flex Wrap** | Controls if components **wrap onto multiple lines** or stay on one. |
| **Overflow** | Defines how overflowing content is handled (**visible, hidden, scroll**). |
| **Theme** | Choose between **Light and Dark mode** styling. |

**Mapping Data to Interface Components in UnifyApps**

**Overview**

In **data-rich applications**, mapping **data sources** to **UI components** is essential. This is achieved through **Data Pills**, which dynamically fetch and display real-time data in components like **tables, charts, and cards**.

**Introduction to Data Pills**

**Data Pills** represent dynamic data points from various sources. They can be inserted into:  
**Text fields**  
**Labels**  
**Input boxes**  
**Tables, charts, and cards**

Clicking on an input field, label, or value field opens the **Input Data Pills** popup for easy selection.

**Fetching Data Sources Output**

Each **data source** (APIs, databases, third-party connectors) generates **data pills**, which can be mapped into UI components.

**Common Data Sources & Their Uses:**

* **Storage by Unify Apps** → Fetch data from stored objects
* **Analytics by Unify Apps** → Query and visualize data
* **Application Connectors** → Integrate with external platforms
* **Callable by Unify Apps** → Automate data retrieval

Example: If your API returns **customer feedback scores**, you can use a **data pill** to display the **average CSAT** in a chart.

**Fetching Component State Data**

Every UI component (e.g., tables, buttons) generates **state-based data pills**, representing their real-time state.

Example: A **Table Component** has a **Selected Row** data pill, which helps display specific details when a row is clicked.

**Fetching Page Inputs**

Page Input **Data Pills** store **query parameters** passed to the page.

Example: If you have a **Bank Account Details Page**, you can pass an **Account Number** as a parameter to dynamically display the correct account details.

**Fetching Page Variables**

**Page Variables** store **dynamic data** and can be updated based on user actions.

Example: A **Button Click Counter** updates a **click count variable**, which can be displayed using a **Page Variable Data Pill**.

**Combine Multiple Data Pills**: Use formulas (e.g., concatenation) to merge pills into a single field.  
Example: **Full Address = Street + City + ZIP Code**

**Fetching Logged-In User Details**

The **Logged-In User Data Pills** fetch user-specific details, such as:

|  |  |  |
| --- | --- | --- |
| **Pill Category** | **Pill Name** | **Description** |
| **Logged In User** | **Name** | Full name of the logged-in user. |
| **Logged In User** | **Username** | Username of the logged-in user. |
| **Logged In User** | **Email** | Email ID of the logged-in user. |
| **Logged In User** | **Custom Attributes** | Additional user-defined properties. |

**Use Case:** Personalize greetings with Hello, [Logged in User Name].

**Fetching Page URL Details**

The **Location Data Pills** fetch URL-related details for **navigation and filtering** purposes.

|  |  |  |
| --- | --- | --- |
| **Pill Category** | **Pill Name** | **Description** |
| **Location** | **Href** | Full URL of the current page. |
| **Location** | **Hostname** | Domain of the current page. |
| **Location** | **Pathname** | Path section of the URL. |
| **Location** | **Search** | Query parameters in the URL. |
| **Location** | **Hash** | The anchor section of the URL. |

**Use Case:** Auto-filter content based on URL parameters (?category=Electronics).

### *****Priyanka Gupta (17/03/2025)*****

### ****How Unify AI Agents Work?****

Unify AI Agents function as **intelligent digital assistants** that help businesses by **answering questions and solving problems** using a combination of:  
 **Language models** (AI brain)  
 **Knowledge processing** (smart information retrieval)  
 **RAG pipeline** (accurate response generation)

These agents ensure **quick, consistent, and accurate** responses by instantly accessing business knowledge.

## ****Core Components of Unify AI Agents****

### ****1.Language Model Foundation (AI Brain)****

**Large Language Models (LLMs)** power AI agents by:  
**Understanding** natural language  
**Generating** context-aware responses  
**Adapting** to different communication styles  
**Learning** from interactions over time

**Example:** If a customer asks about a product, the AI understands the intent and provides a precise answer.

### ****2.Knowledge Processing System****

AI Agents use a **vector-based knowledge system** to store and retrieve information efficiently.

#### ****Vector Processing**** (Transforming text into AI-friendly data)

#### Converts information into **mathematical representations** Ensures **precise matching** of questions with relevant data Captures **context & relationships** between topics

#### **Example:** Company policies are converted into vector format for ****quick lookup****.

#### ****Vector Database**** (AI’s organized memory)

#### Stores **structured knowledge** Maintains **connections** between related topics Updates **dynamically** with new information

#### **Example:** Stores **customer history** for personalized support.

### ****3.RAG Pipeline (Retrieval-Augmented Generation)****

Ensures AI provides **accurate, context-rich answers** by fetching and enhancing relevant data.

#### ****Retrieval Phase**** (Finding the right info)

#### Converts **questions** into a searchable format **Searches** the knowledge base Identifies & **ranks** relevant information

#### **Example:** Finding **specific leave policies** when an employee asks about vacation rules.

#### ****Augmentation Phase**** (Adding context)

#### Combines **user query** with retrieved information Adds **background details** for better understanding

#### **Example:** Merging **current & past policies** for a **comprehensive** HR response.

#### ****Generation Phase**** (Creating the response)

#### **Formulates** well-structured, accurate answers Maintains **clarity, consistency, and tone** **Formats** responses for easy reading

#### **Example:** Generating a **detailed policy explanation** with examples & updates.

## ****Why Unify AI Agents?****

**Faster & smarter** than traditional software  
**Real-time knowledge retrieval** for accurate answers  
**Continuously improves** with every interaction  
**Highly adaptable** to business needs

These AI Agents **outperform traditional chatbots** by combining **deep language understanding, advanced retrieval, and contextual response generation** – making them **powerful tools** for modern businesses!

### *****Pre-Processing in AI Agent Workflow*****

**Pre-processing** is a crucial step in AI Agent response generation, helping optimize **query understanding, retrieval, and ranking** before final response formation. It ensures that AI retrieves **the most relevant** information from stored knowledge.

## ****AI Agent Response Generation Workflow****

The complete process consists of three layers:

1.**Knowledge Indexing**

* Processes and stores documents in a vector database

2.**Query Processing & Retrieval** (Pre-Processing)

* **Query Rephrasing**
* **Chunk Retrieval**
* **Ranking/Reordering**

3.**Response Generation**

* **Answer Formation**
* **Response Delivery**

**Pre-Processing sits at the core of this system, refining queries and ensuring accurate data retrieval.**

## ****Pre-Processing Stages****

### ****1.Query Rephrasing****

* Converts **user queries** into more structured, meaningful formats
* Uses **LLMs** (e.g., GPT-4, Claude 3.5 Sonnet) for context-based rephrasing.
* Helps match **queries** with the best knowledge sources

**Example:**  
**User Query:** "What's our WFH policy?"  
**Rephrased Query:** "What are the current company policies and guidelines regarding working from home?"

**How to Configure?**  
Choose a **Query Rephrasing Model** from available options  
Define **custom prompts** for query refinement

### ****2.Chunk Retrieval****

* Extracts **relevant knowledge sections** from the vector store
* Uses **vector similarity search** to fetch information
* You can **define** how many chunks to retrieve (Default: 50)

**Example:**  
For a **WFH policy query**, AI retrieves:  
Complete remote work policy **documentation**  
Management **approval protocols**  
**Time tracking** and accountability guidelines  
Related **HR procedures**

**How to Configure?**  
Define **chunk retrieval limit** based on data complexity  
Higher values improve search but may slow performance

### ****3.Ranking Chunks****

* **Prioritizes** retrieved chunks based on **relevance scores**
* Uses **ranking models** to order information logically
* Ensures **most useful content** appears first

**Example:** (For a Password Reset query)

|  |  |  |
| --- | --- | --- |
| **Content Type** | **Relevance Score** | **Priority** |
| Password Reset Procedure | **0.95** | **High** |
| Account Security Guidelines | **0.82** | **Medium** |
| Password Requirements | **0.78** | **Medium** |
| General Account Information | **0.45** | **Low** |

**How to Configure?**  
Select a **Ranking Model** to prioritize results  
Adjust parameters for **better search accuracy**

## ****Why Pre-Processing is Essential?****

**Enhances query accuracy** for AI-generated responses   
**Fetches the most relevant** knowledge chunks   
**Optimizes response speed** while maintaining accuracy   
**Improves ranking** of retrieved information for clear answers

By fine-tuning **Query Rephrasing, Chunk Retrieval, and Ranking**, AI Agents deliver **smarter, faster, and more precise** responses tailored to user needs.