WKA Studio (WIP)

WKA Technologies Pvt Ltd.
SINE, IIT BOMBAY

Table of Contents

1 Introduction to WKA Studio 2
1.1 What is WKA Studio? 2
1.2 Why WKA Studio? 5
1.3 What kind of applications can be developed using WKA Studio? 5
1.4 What are the major features of WKAS and advantages? 5
1.5 Core techniques and algorithms implemented in WKA Studio 8
1.6 More about WKA Studio and WKAS/PL 8
1.7 WKAS/PL 12
1.7.1 More about WKAS/PL 13
2 Expert system technology 16
2.1 What is expert system technology? 16
2.2 How business logic is implemented using expert system 19
2.2.1 Business logic (Rule-base) 19
2.2.2 Business logic (procedural code) 21
3 Getting started with WKA Studio 24
3.1 Skill-sets required for WKA Studio 24
3.1.1 Developer 24
3.1.2 Administrator 24
3.2 WKA Studio Interfaces 24
3.3 List of WKA Studio Interfaces 25
3.4 Creating your first app 30
3.4.1 Creating app manually 30
3.4.2 Modifying simple app 33
3.5 More about WKAS-based apps 41
4 Appendix 42

1 INTRODUCTION TO WKA STUDIO

1.1 What is WKA Studio?

WKA Studio is a platform to develop, configure and manage enterprise software solutions and apps (a kind of IDE: to create, manage and test software solutions) backed by LLMs and state of the art goal driven rule-engine supported by a rule-based 4GL.

It is a DiY (Do-it-yourself) completely web-based rapid and configurable application development Platform (it is kind of web-based IDE: Integrated Development Environment) and can be hosted as PaaS on cloud. Configurable means objects used in app can be modified based or added new ones based on requirement e.g. predefined database query objects do not need to modify the core code, parameter values are automatically added at run-time. WKAS has lot of web-based UI interfaces to create and configure various objects externally which are referred and invoked in rules, code and in various other objects using high productive programming language which is part of WKAS. This reduces the writing of core code to the great extent.

The architecture is fundamentally based on state-of-the-art rule engine based expert system technology blended with effective and optimal use of 3-tier multi-tenant web-based architecture. Apps based on rule-based expert system technology can be modified on the fly, interfaces can be made simpler and intuitive, apps can be built to ask, invoke or only relevant information in the context.

Database interfaces allow to manage databases and database objects such as queries, tables, views, functions and procedures like a typical database front-ends like Heidi. It has ML front-end which helps to quickly build ML models. Most of the typical functionality required such as data management, data selection, pre-processing, model selection, creating various experiments using different data-sets, storing models into database etc. at frond-end and core ML functionality is executed using ML server backed by generic Python code.

There are many configurable components such as email, charts, upload documents, QR and bar-codes, captcha, pdf generation, data imports, data entry, management of database objects and some of these can be configured at run-time as well. It is like one-stop-shop development environment. WKAS application and its development is fundamentally different from the way conventional application development using programming languages like Java/C++ and the way applications work. WKA Studio can also be used to develop chat-bots.

Open-source software integration with WKA Studio

- Bootstrap, Easy UI & InputMask: for App UI
- Code Mirror: for Language Scripts
- IronPython: allows business logic to be written in Python and invoked in WKA Studio.
- Plotly: Visualization tool
- MongoDB: supports NoSQL databases
- PDFSharp and MigraDoc: configure and create pdf documents, convert HTML into pdf
- Quartz: for job scheduling system
- Python: For ML/NLP
- AlaSQL: Client side SQL engine

Figure 1-1 WKA Studio Internals

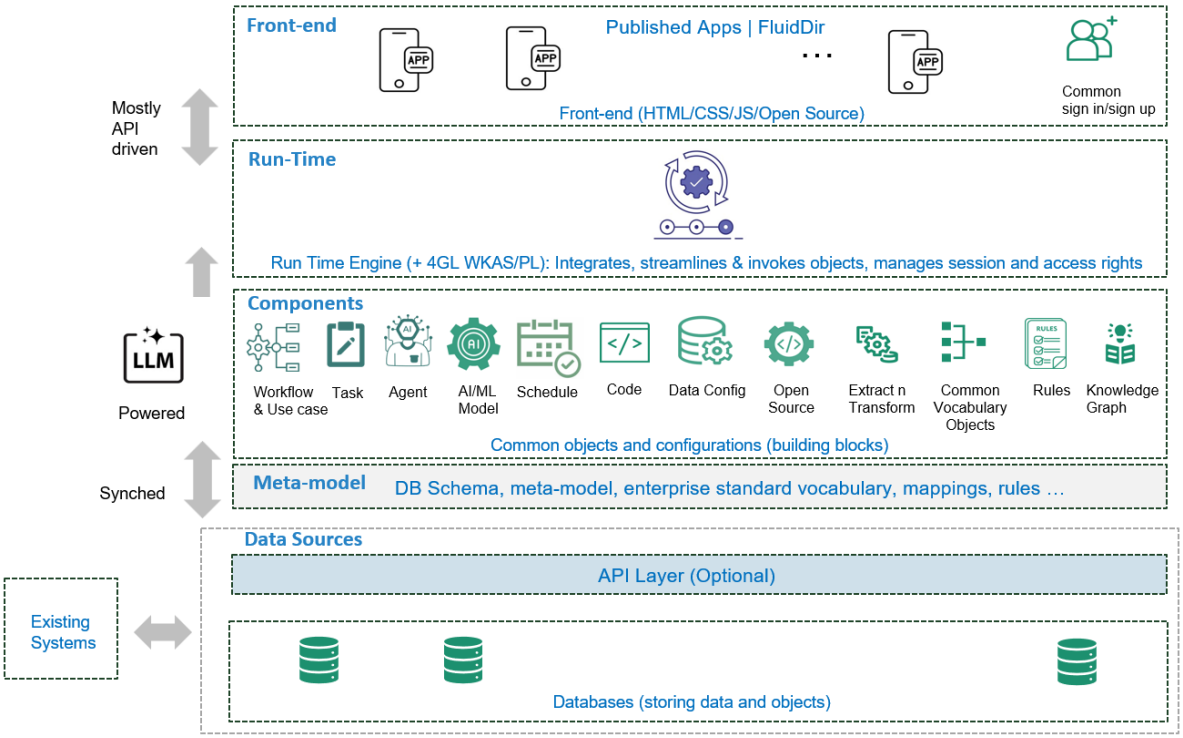


Figure 1-2 Agent framework using WKA Studio

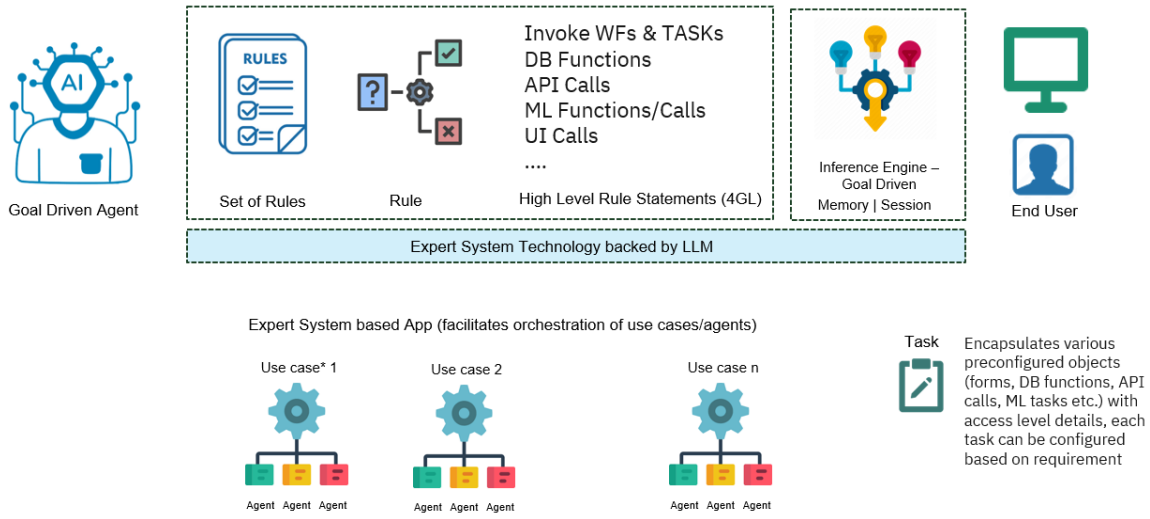


Figure 1-3 Role of GenAI in WKA Studio

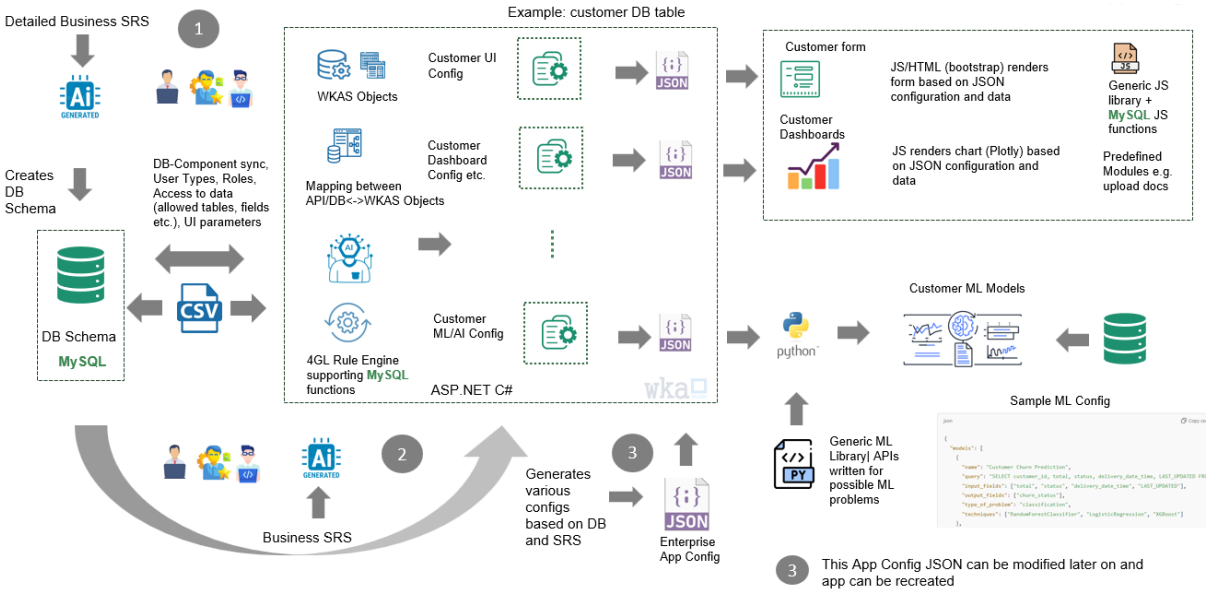
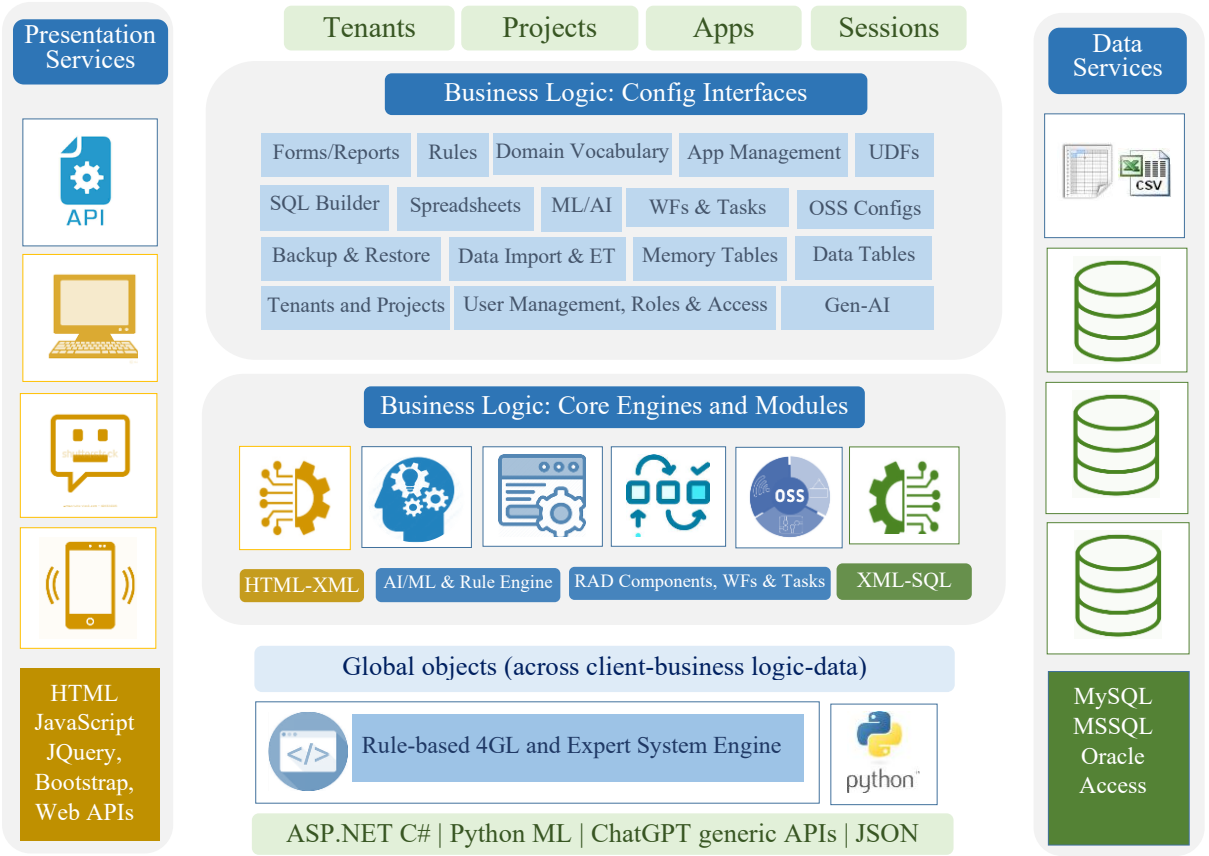


Figure 1-4 WKA Studio Technology Stack



1.2 Why WKA Studio?

- To achieve 100% automation including automation of cognitive tasks (making organization knowledge-based), no manual physical forms, no data exchange through emails, no unstructured data in spreadsheets and word documents. All data in one place centrally available in real time for effective reporting and decision making.
- To develop your own software, extend the existing ones, add any ad-hoc based requirements such as creating quick new forms to get data, surveys, questionnaires, tests or quizzes etc. whenever you need it! No need to knock our door.
- One platform can cater almost all of your business software and knowledge automation needs. We have added all ingredients and components required for a typical software. Applications can be seamlessly accessed on web, mobile (through web app) and chat-bots.

1.3 What kind of applications can be developed using WKA Studio?

Table 1.1 Application categories

Application category	Description
Business applications and enterprise systems	WKAS can be used to develop business applications including ERPs allowing clients to make whatever changes whenever they want.
Advisory	Advising or assisting end-user on various subject matters like experts in domain: legal, health, real-estate, tax, government schemes and services, farming, education and career, etc. They can be used by end-users: customers, students, citizens etc. or junior experts.
Diagnostics and troubleshooting	Helping/assisting end-user solving problems: medical diagnosis, crop-diagnosis, device specific diagnosis, IT diagnosis etc.
Knowledge-based intelligent systems	Converting existing information-based contents in the form like text books, manuals, rule-books, web-contents into structured and knowledge-based for so that end-user can drill-down or get actionable knowledge.
P&R	Enables deep personalization and recommendation at entity level (N=1)
Intelligent search/ selection and match	Replacing existing SQL search with semantic-based search, it searches contextually (semantically matching) similar/relevant contents or products.
General ML and Rule-based systems	Addressing classification, clustering, collaborative filtering, association mining, NLP and optimization kinds of problems using various technologies and Python APIs. Rule-based systems can be used to implement dynamic business logic and domain knowledge, set configurations, adhering to check-list, rule-book and compliance, etc.

1.4 What are the major features of WKAS and advantages?

Table 1.2 Major features

Feature	Advantages
Uses knowledge-based approach	Simplifies and expedites app development. Avoids complex and crowded interfaces, divides interfaces into simple and manageable ones with lot of images/icons and annotations. Responsive apps (integrated boots-

	trap)/chat-bots can be made intelligent guiding, invoking and asking only relevant inputs based on context, past experiences or requirement, and can be personalized. Makes app development more modular. Apps can be modelled using rule-based approach, the flow can be changed on-the-fly. User interfaces are configurable, application code (business logic) is separate from execution engine making it possible to modify the app's logic whenever needed
WKAS core Framework and Interfaces for knowledge-based app development	Development and configuration interfaces to build and configure UIs, database access, retrieval and updates etc. (saving substantial coding efforts). These interfaces can be extended to clients so that their IT team can maintain apps and databases (clients can have full access to their databases) on their own (DiY: Do it Yourself or MiY: Manage it Yourself and OiY: own it yourself) or support team can rectify issues and do customizations at client location itself or online. Extending existing apps or adding multiple/additional apps is easy as all these apps will be using common WKAS framework. It can help to build complete and integrated solutions like ERPs. When WKA Studio framework/platform is updated to next version, all apps running on that will have updated UIs (e.g. new features of HTML/CSS, new look-n-feel of web pages etc.) and functionality.
4GL rule-based language (WKAS/PL)	One of the first 4GL expert system (rule-based) languages supporting to build real business applications. Provides high level of functions, makes it possible to integrate various components reducing explicit programming efforts to great extent. It supports various charts and dashboards to be included in apps.
Integration of Python generic ML APIs	Templates for typical ML problems like classification, clustering, regression, recommendation, association mining, NLP have been modelled and created. Lots of functions are being added to display the Python API results, interpret them using set of rules and use them other functions of WKAS. Entire UI, data sources, training, testing etc. is managed through WKA Studio and multiple models can be tried out.
Inbuilt text and voice chat-bot interface	Knowledge-based applications can be accessed using voice and WhatApp like interfaces. This chat-bot interface can be running on remote servers and accessing WKA apps through APIs.
Common vocabulary across all computing layers	Makes it easier to develop 3-tier web apps. Once variables/data objects are defined, they can be used and accessed in web-pages using JavaScript, business logic using WKAS/PL and in database queries.
Web-based and Multi-tenant architecture	Many people can collaborate and work on the same project from anywhere at any time. Multiple clients can be served simultaneously. Reduces support and maintenance efforts.
Native support for multi-lingual app development	Develop multi-lingual apps. App can be developed in English and can be deployed in any language supported by Google/MS Translator APIs without much efforts.
Integration with external APIs and programming languages.	No need to integrate external APIs separately they can be invoked using WKAS/PL seamlessly. Already integrated APIs for sending SMSs etc. Integrate insights from analytics tools like R. Python code can be used to add functionality.

In-built PDF designer	Helps to design/configure PDF document layouts/reports. Multiple data sets/charts can be included in reports.
Customizable email interface	HTML-based interface (data accessed through web-methods) can be integrated into forms, can be invoked in restricted modes for customized emails (e.g. recipients, subjects etc. can be fixed/restricted to list etc.)
Sign up and Sign In Interfaces	In-built interfaces to sign up, sign in, forgot password, change password using email/mobile number with optional OTP authentication using mobile/email.
Quick weblinks	Quick weblinks for specific time can be created and sent to respective users to auto login and jump to a particular page through flow created through set of rules.
Database Management Interfaces	Makes it easier to create, define and connect database schemas interactively and manipulate database objects, design and test database queries, etc. Provides interfaces to manage MySQL databases. Modifications to existing data can be quickly done.
In-built charting tools supporting commonly used charts for dashboards.	Helps to show data using dash-boards. Charting utilities are integrated with query builder and interfaces to directly view data using charts fetched from database.
Data-import Interfaces	Helps to configure data imports from text files as well from other databases.
Integration with MS-Excel	Can access and populate any data from excel spreadsheet by mapping cells, ranges, columns, rows to WKA Studio variables.
Access through APIs	Apps can be integrated with other applications.
Sessions Management	User can manage their session data easily. Sessions can be stored and used for personalization and analytics purposes.
Intelligent app search	Using keywords user can quickly jump to app and any part of app instead of going through Q&As e.g. add employee would open employee app and jump to add employee form interface.
Drag-n-drop and data based app-create creation	Helps to quickly build app based on type of app functionality like rule-based apps, database apps, quiz-apps, interactive reports and matching apps (from CSV/Excel files). Apps can be created from rules stored in excel sheets. Apps can be created in two mode: 1. Using drag-n-drop interface and, 2. Create app using existing data table.
Integration with other WKAS technologies	Like case-based reasoning, genetic algorithms, deep profiling, feature weight-calculations etc. Brings in advantages of having other technologies to normal software applications. Intelligence can be incorporated in existing apps making them intelligent and smart, add analytics to apps, personalize apps etc. Other KBAI based advisory, diagnostics, personalization, intelligent search apps and other knowledge automation applications can be developed.

Quick set up on cloud	A VM can be quickly configured on cloud and entire setup can be uploaded in matter of half to one hour. This can also help to build POCs for clients and show to them.
MySQL-Open-source as back-end	Although WKAS is database agnostic, it uses MySQL as default database.
Pre-built universal database schema	Most of the ingredients and building blocks are available. Extensible ERPs can be developed quickly. Schema is designed to work across multiple players rather than specific to one organization. IT team does not need to spend time on creating database schema. Includes most of the generic entities required to model users, products, employees, customers, catalogues, contents, inventory, schedules, transactions, operations, etc. Database design has been based on generic models which can accommodate large number of variations e.g. transaction entity is developed to model any kind of transaction. Saves time on managing too many entities based on domain or function

1.5 Core techniques and algorithms implemented in WKA Studio

Table 1.3 Core algorithms and techniques implemented

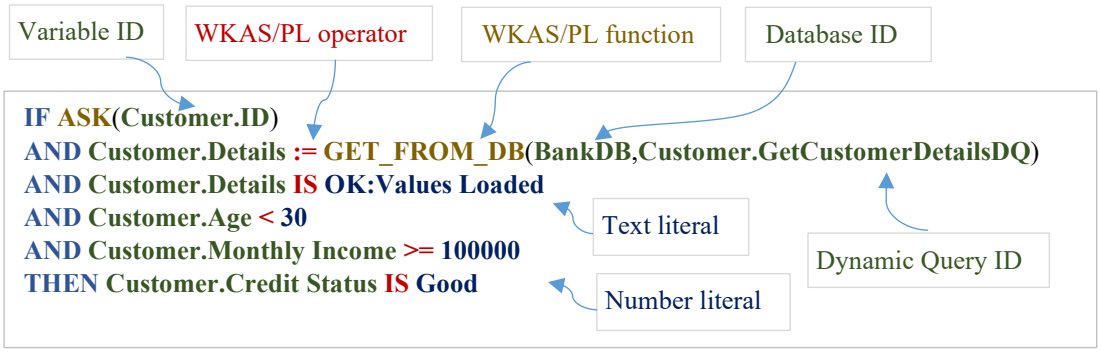
Engine/Module	**Description**
Rule-based expert system engine	Models and incorporates domain expertise. Intelligently guides and asks only relevant facts based on context and goal. This technology itself capable of addressing wide range of application categories.
Case-based reasoning engine	Stores and reuses past experiences. Uses contextual and domain knowledge to intelligently match entities on several dimensions. Models knowledge + data driven intelligence. Supports lazy machine learning. This technology itself capable of addressing wide range of application categories.
Profiling and consolidation engine	Profiles and consolidates entities on various dimensions based on business logic by pulling relevant transactions and meta-data
Feature weight calculation algorithm	Derives significance (relative importance) of parameters (affinity) for every entity based on past transactions.
Algorithm to scan and analyse text	Scans search keywords intelligently, understands the context and creates attribute-value pairs based on context
Genetic algorithm	Addresses resource optimization problems such as scheduling, routing, planning etc. using evolutionary computing.
ML Engine	Various ML problems like classification, clustering, association mining, NLP can be modelled using this engine and executed at Python/WKAS.

1.6 More about WKA Studio and WKAS/PL

In typical conventional programming languages, variables are declared inside the code and allow only data type and initial value to be set when they are declared. WKAS variables are not defined inside code but externally through global variable interface and are accessible in other modules and

interfaces. There are lots of parameters related to input data source, HTML formatting, validations can be specified and configured for every variable externally saving lot of explicit code inside the code.

Figure 1-5 Sample rule



More about WKA Studio variables.

- Each variable has large number of configurable parameters defined such as data type, possible domain values, default value, range of values, validation options such as email, number, minimum one special character, external validation JavaScript etc.

- In conventional programming, value of variable has to be asked explicitly by creating UI or through statements to read value(s) like cin>> in C++ or readline in Java before it is used in any statement or expression. WKAS variable gets value (or value is asked) at run time automatically from various sources such as HTML/aspx web-page (created automatically based on UI parameters and data types), database, report, user defined function and so on. Value of uninitialized variable is automatically asked, fetched or computed by expert system engine or external integrated scripting only when it is required (e.g. appearing in the expression/statement). For example, in above sample rule, *Customer.Monthly Income* will be asked only when *Customer.Age* is greater than 30. However, variable can be explicitly asked using ASK statement for example first statement in sample rule is asking Customer.ID because it is used to fetch customer data from database using dynamic query (dynamic queries are predefined parameterized queries): *Customer.GetCustomerDetailsDQ*. Forms, reports, parameterized queries, text and excel files, etc. can also be connected to variables to get values in groups (using forms), to display reports.

- Developer can also set lot of UI parameters such as UI control type (radio box/combo box/check box etc.), number of options per line, web-page template, JavaScript for validations, type of main buttons on the page etc.

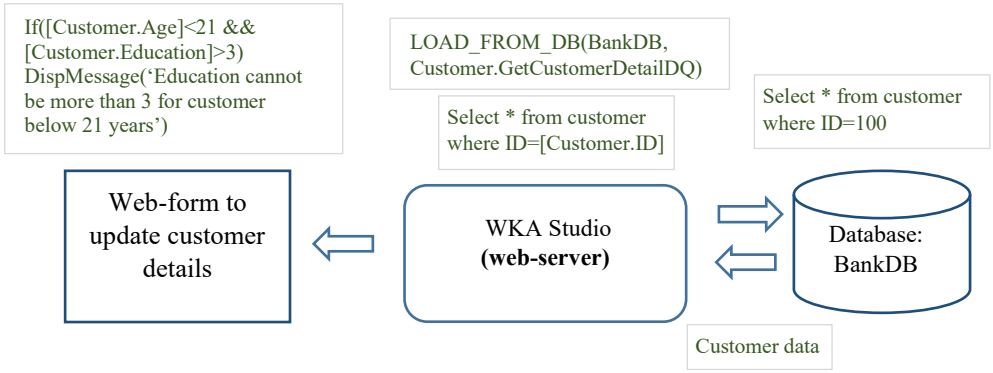
- For each variable possible value (e.g. colour), the details can be added such as image, description, inference, group etc.

- No need to write code/create Web-page to get value of variable from user. Responsive web-page is created with all in-built validations and invoked by expert system engine whenever needed.

- Variables can be used at all three tiers (client-side, web-server: business logic and to access database) by referring their names: i> in JavaScript code at front-end such as for validation, ii> in the business logic code and iii> as filters in database queries for example variable: Customer.Age can be used in JavaScript to set certain value say risk: if([Customer.Age]>60) SetValue('Customer.Risk',Higher). WKAS automatically creates and adds JavaScript by populating functions e.g. variable is replaced with GetValue('Variable Name') to get variable values at run-time. Figure 1.2 shows, how variables are used in JavaScript code and database

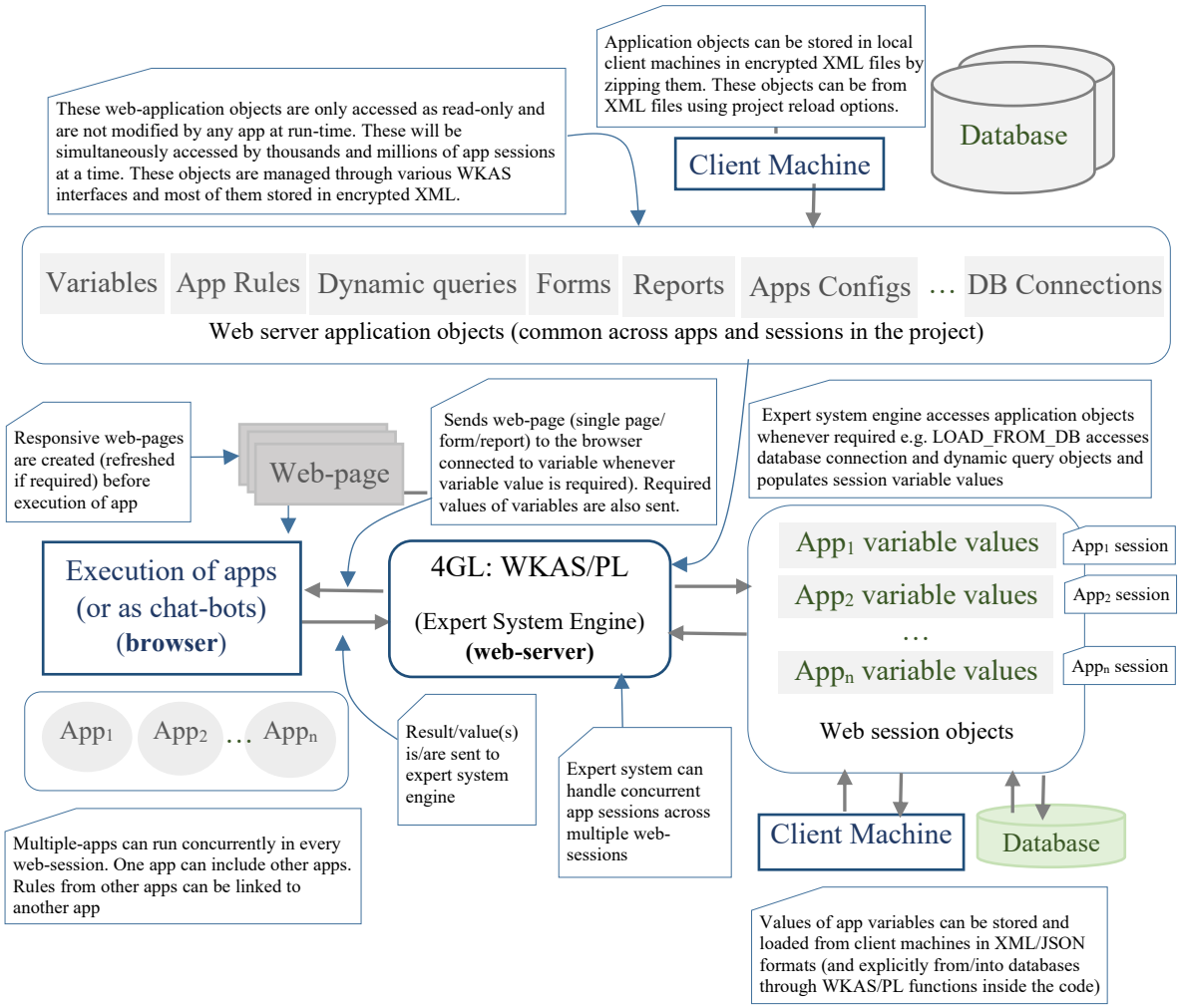
queries. Variable inside dynamic query: *Customer.GetCustomerDetailsDQ* is populated with variable value before it is sent to the database to fetch the data.

Figure 1-6 Using Variables in JavaScript and DB Queries



- Related variable names are grouped under variable group name that corresponds to logical entity which holds related information e.g. Customer, Item etc. All Customer variables such as Name, Age, ID, Monthly Income will be grouped under Customer variable group. Variables are always referred using complete name prefixed with group name they belong to, for example, Customer.Age. Note Age cannot be called variable but Customer.Age is.
- Variables are automatically created when apps are created using app create interface. For example, when Database app is created, variables are created corresponding to field names of selected database table under given variable group (may be similar to table name) as parameter during app creation.
- At run-time, app holds variable values called as session variables (or app session or session).
- Once app starts its execution, it asks, gets or computes values whenever required and holds them as long as app session is going on. Once app session gets over and app is restarted again, all variables app was using are initialised to null. However, WKA Studio expert system engine maintains the values from previous app session called as last values. These values are automatically selected as default value when input is asked through web-pages. There is an option in variable configuration to retain values across the app sessions (e.g. user ids, passwords need to kept for multiple app sessions) even across the apps.
- Variable values can be saved in files on client machines in XML/JSON format or in databases. These values can be loaded back into current session. App will not ask values which are loaded from XML file.
- App itself is started using variable typically has name same as app name. This variable is referred as goal variable.
- Lot of objects such as database connections, predefined parameterized database queries to (execute DB functionality, bring data-sets), menu lists which store possible values a variable can take, forms, reports, user defined functions etc. are configured using various interfaces of WKA Studio. These objects are classified into two broad categories called WKAS variable group objects and WKAS application group objects. Their IDs are prefixed with either variable group or application group. For example, menus are prefixed with variable groups e.g. *Customer.GenderOptions* where Customer is variable group. and forms are prefixed with application groups e.g. *Payroll.MonthlyDataEntryForm* where Payroll represents application group.

Figure 1-7 WKA Studio application variables and session management



- Each object is accessed using ID and invoked using just their IDs in code using WKAS/PL e.g. database can be accessed using logical database ID, a query can be executed using its ID and values will be populated at run-time in query before execution. All these objects may be referred in each other (for example, parameterised query can be connected to form to fetch and populate data from database in the form). This does not require lot of code inside business logic thus reducing code. Sample rule shown on page 1 includes Database ID, Dynamic query ID to fetch data from database and populate into variables using LOAD_FROM_DB function.
- Apps can upload various files into database, import data from excel files and text files at run-time. There are predefined configurations which check type of text file and excel files being imported at run-time.
- Major objects which are frequently used in app rules are:
    - Global Variable and Menus (predefined list of values)
    - Rules, User defined functions (UDFs)

- Database connections, dynamic queries (DQs), etc.
- Forms and Reports

1.7 WKAS/PL

WKAS/PL is a 4GL rule-based language to develop web-based software, mobile web-apps and chat-bots backed by expert system (classical AI) technology. Many other technologies supported by WKA Studio (WKAS) such as genetic algorithms, case-based reasoning (facilitates lazy machine learning) can be programmatically invoked using WKAS/PL. WKAS apps use WKAS/PL in rules and user defined functions to implement procedural logic. The current version supports around 310+ functions and following data types. WKAS/PL provides lots of functions to do computations and write complex business logic involving various data types e.g. supporting matrix calculations and operations.

Table 1.4 Data types supported

Text	Holds text data. Spaces can be included when text is assigned, however, when language keywords, commas, brackets etc. are used, they must be enclosed in double quotes. Student.Name:=S K Patil Student.Grad Institute:= “Indian Institute of Technology, Bombay” It can also hold XML data. If double quote appearing inside text, it has to be preceded by back-slash e.g. Customer.Statement:=“He said \”this cannot be done\””. Text datatype variables can hold XML as well as HTML data.
Number	Hold the (32 bit) signed integer data. Customer.Age:=50 Number without decimal point is treated as number. Number without decimal point is treated as number. Current version supports only 32 bit, any number more than size of 32bit converted into BigNumber.
BigNumber	Hold the (64 bit) signed integer data. Customer.Mobile:=808080808
Real	Hold the float (32bit) data. Current version does not support double but will soon be added. (64-bit double will be added soon) Student.Marks:=89.24;
Double	Store 64 bit double data. It is being implemented.
Boolean	Boolean data types hold true or false. Customer.HasJob:=true;
Compound	Holds list (array) of text, numbers, real, compound, trend, boolean, matrix etc. Customer.Assets:= [Car, Two Wheeler, Colour TV, Washing Machine]. In case values themselves contain comma, then they can be entered in double quotes or pipe | can be used as separator. Only literals and variables are supported inside array when initialised using square brackets. Expressions are not supported inside array enclosed in square brackets. Function GL can be used to convert expressions into array e.g. GL(10,10*2,SQR(10)) etc. array will be [10,20,100].

	Any literal starting with '[' and ending with ']' would be treated as compound list. Compound variable can hold different types inside. e.g. Customer.Record:=[P K Sing,40,"20,IIT Campus,Powai",[Pune,Mumbai]] e.g. Customer.Choices:=[[Mumbai,1],[Chennai,2],[Delhi,4]] Some of the compound variable can contain values with weights Student.PLKnown:=[C++=100,Java=60,Python=50]
Date	Holds time, date or time stamp depending upon initial value or Is Using Time? flag set for date variable. Student.DOB:=_0102200 Any literal starting with _ is treated as date type. Examples: _1030 would be treated as time 10:30am, _01012013 would be treated as DDMMYYYY (8 characters) format. Default formats can be changed/set using project config parameters: DefaultTimeFormat (must include colon separator HH:MN:SS), DefaultDateFormat (DD/MM/YYYY) and DefaultDateFormatLong (DD/MM/YYYY HH:MN:SS) . If length of date string is more than length of formatting string set in DefaultDateFormat then long format is used to convert text into date
Matrix	Holds matrix data Info.MatVar:={10 2 3,5 6 7} creates 2 x 3 matrix and assigns to the variable. Space (column separator) or comma (row separator) as a separator. Any literal starting with '{' and ending '}' is treated as matrix data type. However, CONVERT/READ_MATRIX function can be used to convert text into Matrix.
Trend	Holds the trend data: array of numeric values (real). Info.Sales:=|100 120 125 129 140|, space as a separator. Any literal starting with | and ending '|' is treated as trend data type. Internally trend data type is stored as matrix with one column.
Document	This holds any type of document including images. Depending upon type of document like XML, pdf, image etc. different treatment is given.
URL	This is same as Text data type except the value would be displayed as URL link when shown in the report.
Variant	Holds any data type
MTable	Memory tables, these are like data tables handling two-dimensional data. They can hold results generated by ML algorithms/APIs and results can be intelligently interpreted through expert systems. Query based data tables can also be used using various functions which support them.

1.7.1 More about WKAS/PL

- Since it follows rule-based approach, AND is used as condition/statement separator and each condition must return Boolean data type. Space is not treated as separator in WKAS/PL.

Following characters have specific meaning in WKAS/PL: comma (,), pipe (|), opening and closing brackets ([,],(,),{,}). New line or semicolon (;) is used as statement separator in user defined function to execute procedural logic.

- WKAS/PL uses lot of functions that use various objects created and managed using various WKAS interfaces. Each object has unique ID in that type. Examples are variable IDs, App IDs, DB IDs, menu IDs. These objects can be created and configured through respective interfaces. For example, new variable can be added through *Global Variables* Interface. There are quite a number of parameters which can be set for every variable created such as minimum, maximum and default values, validation script, input options like whether value is asked to the user, fetched from database or computed using some server side or JavaScript function whenever required etc. In case values to be asked to the user, web-pages are created with in-built validations and options using just a click of button, these web-pages are automatically invoked whenever inputs to variables are required. This saves lot of explicit coding of asking explicit inputs (calling database code in case, the value is to be fetched from the database), validating them, etc.

- Except few IDs like DB IDs, App IDs etc. most of the IDs are prefixed with application or variable group names e.g. *Customer.Age* where *Customer* is variable group and *Age* is variable name within variable group *Customer*, however, *Customer.Age* is called as variable name/ID in general. ID names can contain spaces and underscores e.g. *Customer.First Name* or *Customer.First_Name*. Group can be skipped for variables by using USING statement.

- Variables can be used inside database queries. For example, select $[Customer.Age] from customer where $[Customer.ID]= [Customer.ID]. $[Customer.Age] will be replaced by mapped field in table: customer and database and [Customer.Age] will be populated with value of variable *Customer.Age* at run-time.

- Database functions help to get rid of explicit database coding e.g. LOAD_FROM_DB function a> opens DB connection using configured connection string defined through *Database Connection* interface, b> executes predefined parameterized query by populating parameters run-time or direct query given with parameter placeholders in argument, c> fetches the data from the database and d> automatically populates (does the transformation of data if configured) data into respective variable objects based on mapping configuration); similarly, function UPDATE_TO_DB updates variable values into database without explicitly opening DB connection, creating update queries and executing them. Figure 1.4 shows how few lines of code is enough to load data from database, invoke data entry form and update database back to database after form is submitted.

- All functions are in CAPs and use underscore e.g. GET_FROM_DB: fetches data from database.

- Spaces are not treated as separators.

- Text literal need not be enclosed inside square brackets unless it contains keyword, function name or special characters (such as |,{},[]).

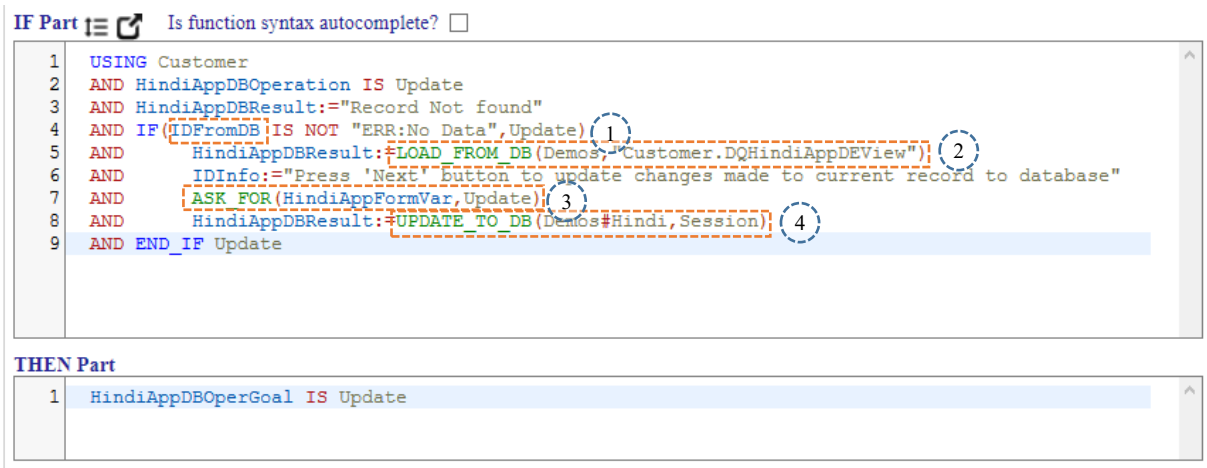
- If any ID is appearing in the statement, language parser first checks whether it is variable ID or not, in case other IDs such as menu ID has same name as variable name, it should be included in double quotes to avoid conflict e.g. GET_MENU("Customer.Education", Code, BSc) in case there exists variable named as *Customer.Education*.

- Since WKAS/PL is part of rule-based expert system technology, when working in expert system mode, execution engine asks variable value whenever needed. The functions which

does not require values to be red, variables need to be included in double quotes e.g. RESET_VAR_VALUE("Customer.Age") in case they don't need values, this statement won't ask value of variable: *Customer.Age* but will reset its value when executed.

- By default, values of variables used in the execution will be shown when goal is arrived. However, output of only specific variables can be shown by setting parameters for variable using Global Variable Interface.

Figure 1-8 Sample rule to fetch and update record



1	Variable *Customer.IDFromDB* has dynamic query connected to it, which fetches ID of customer from given ID, if query returns no value, it returns default return value set e.g. "ERR:No Data" in the current rule.
2	LOAD_FROM_DB fetches data corresponding to query named *Customer.DQHindiAppDEView* from database Demos and loads into application session (memory)
3	ASK_FOR invokes the form linked to variable *Customer.HindiAppFormVar* and sends data from session to it. This data is populated into controls by client side scripting. When data is submitted by the server, is again loaded into session and execution continues with next statement.
4	UPDATE_TO_DB updates data from session to database Demo in table Hindi.

2 EXPERT SYSTEM TECHNOLOGY

2.1 What is expert system technology?

Expert system is a classical AI (artificial intelligence) technology with focus on capturing, codifying and storing problem solving human expertise and intelligence, and applying it to solve the new problems. When deployed it works and assists like domain expert. There are lot of inherent benefits of using expert system approach to application development compared to conventional application development approach. It simplifies and facilitates development of apps in expert and smart way (with lot of scope for including heuristics)!

This approach avoids complex and crowded interfaces, divides interfaces into simple and manageable ones and hence very suitable in chat-bot and mobile app delivery. Applications are intelligent guiding and asking only relevant inputs based on current context, situation or requirement. Makes app development more modular. The user has option to go back to previous input or any save point. It can help to understand previous user experiences, reuse them in current context and adopt to that and personalize. Applications are modelled using rule-based approach, the flow can be changed on-the-fly. Business logic in the form of rules and procedural code is separate from execution engine (called as inference engine) thus makes it possible to modify the application logic whenever needed.

Figure 2-1 Sample C code: Math operations

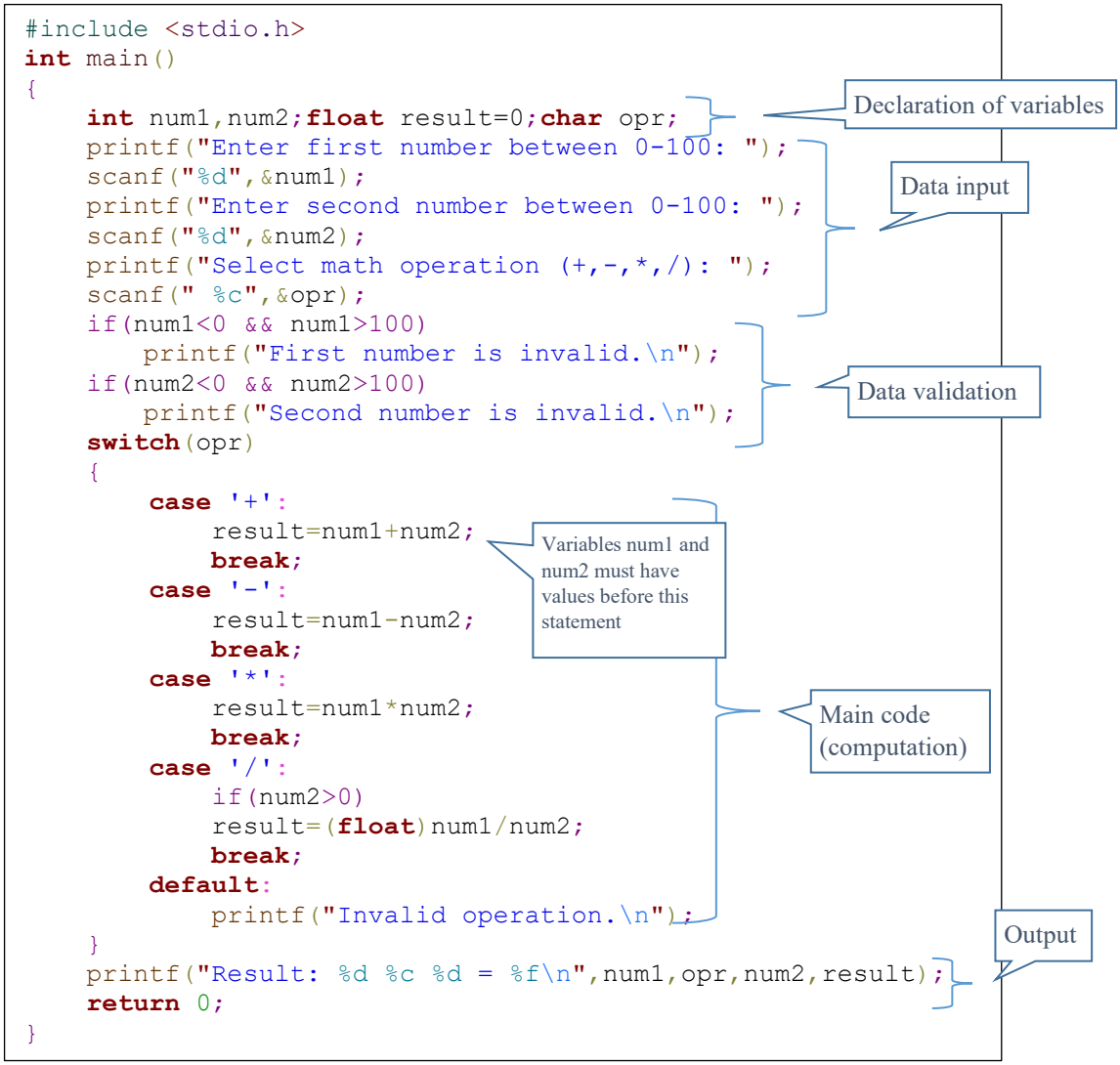
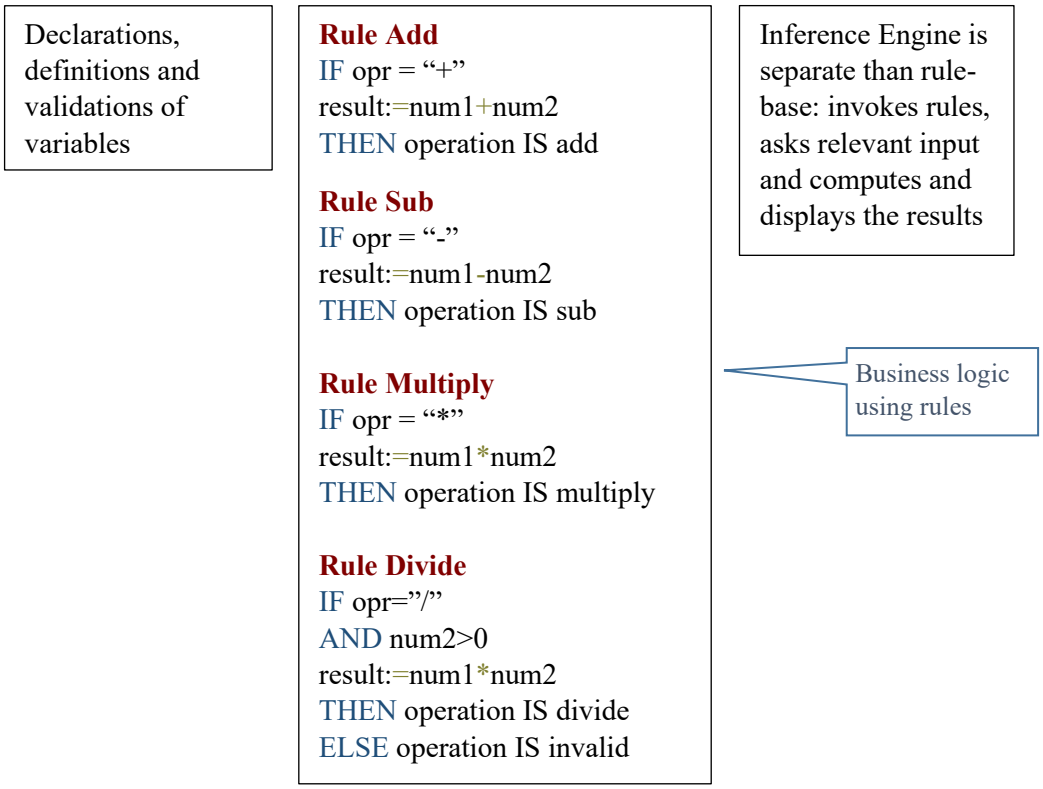


Figure 2-2 Rule-based expert system approach: math rules



Let us write and understand the code which does the basic math calculations using C language. In conventional programming language, declaration of variables, input, output, validation and computations are typically part of the same program code and it is executable unit (code is compiled and encapsulated inside), refer figure 2.1. Even slight change needs to compile code and make executable again.

However, in expert system these components are separated and no need to initialise variables as shown in figure 2.2. Variable values are (or can be provided before execution start) asked (retrieved or computed) at run-time on the fly whenever needed. In case variable values are already fed (in working memory), expert system continues to execute rule till it encounters conclusion it is looking for. This helps to jump to a particular business logic.

As discussed earlier, WKA Studio allows variable values to be retrieved from any data source such as database, computed at run-time, explicitly asked to the user when required.

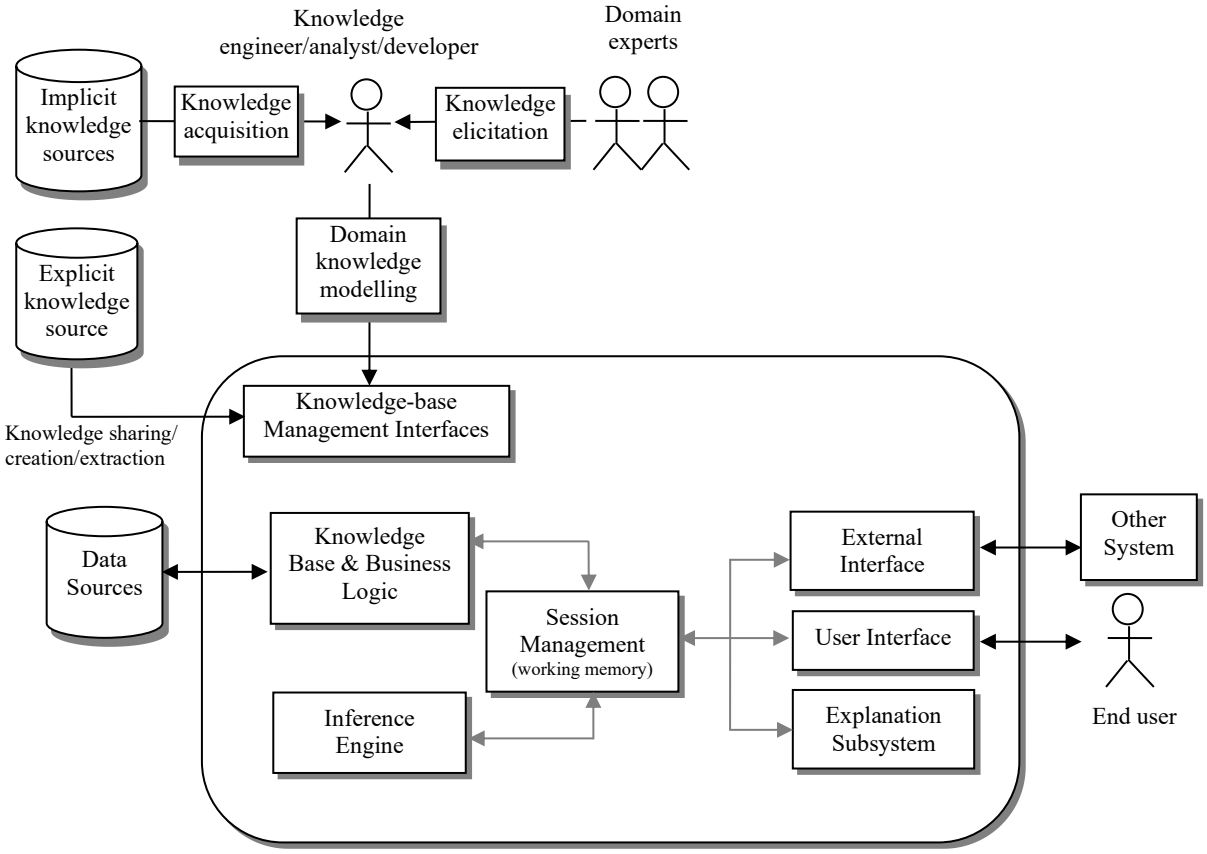
Table 2.1 Comparing conventional app development with expert system app development

Conventional App Development	**Expert System-based App Development**
Most of the time, *code is compiled* (hardcoded) and compiled code is executed. Executables cannot be changed, need source code even for minor change in business logic.	Knowledge-base (business rules and functionality) is separate from inference engine. Knowledge-base can be changed easily *on-the-fly* and Apps will automatically work with modified business logic.
Work flow is *typically hard-coded* and cannot be changed without changing source code.	Since it uses rule-based approach, work flow can be *tweaked* easily.
If business logic uses lot of rules, the sequence of rules in code must be defined	No need to worry about sequence of rules. Inference engine automatically picks up the right

based on dependency of rules e.g. consider simple 2 rule logic **Rule 1:** IF Income Status = Good AND Has Job = Yes THEN Approve Loan = Yes **Rule 2:** IF Age>=18 AND Income = Good THEN Income Status = Good. While writing code, Rule 2 must be written before Rule 1 or it can be nested within rule 2 itself. Code can be complex when number of rules grow (developers need to worry about dependency of inputs or end up with lot of nested ifs; business logic written through nested ifs cannot be reused easily). Even for adding a new rule, developer has to decide where rule should appear in the sequence.	rule based on facts (inputs provided and appearance of variables). In the example given in the left column, inference engine will pick up Rule 2 first because it understands *Income Status* is derived from Rule 1, etc. The inference engine is capable of managing *thousands of rules* and invokes only relevant ones based on current context. This makes it possible to divide business logic into smaller blocks and each block can have a rule or set of rules in it, the blocks can be invoked based on context.
Variables need to be initialized or provided inputs before appearing in statements/ expressions. e.g. in above example, values of Age, Income must and Has Job must be provided before executing the rule 2, otherwise can get *run-time error*. No working memory concept. Variable values are stored as objects whenever assigned or initialized. No built-in provision to load values of variables from automatically.	Expert system-based app *asks inputs* whenever needed when executing expressions within the rule currently being executed. For example, it will ask input Age and if it is greater than 18 then only will ask Income, etc. Thus, the app system asks only relevant inputs based on the current context, it can be even personalized and used for chat-bot kind of applications. Initial inputs can be initialised (loaded explicitly or programmatically) to jump to any part of app. The engine uses working memory where current facts are stored, it automatically picks up relevant facts based on requirement. In case, it does not find in working memory, it asks to the user explicitly. This feature can be used to directly initialize required values automatically and start app at particular option! e.g. embedding credentials and loading them when a dynamic web-link is created.
UI and UI elements are integrated with code and preconfigured especially in desktop-based systems.	*UI pages and elements are separated*, can be changed and generated on-the-fly and invoked whenever input is required.
Interfaces are often crowded with lot of inputs in one screen thus limiting their utility on smaller devices.	Typically, *less crowded*, easier to understand (use lot of images/icons/tooltips) and mostly asking only one or few relevant inputs at a time and can be easily delivered on smaller devices. The end-user has to just need to know and remember few buttons.
Typically, the code (set of statements) is executed sequentially and there is no	These apps use *heuristics incorporating human expertise*, the next statement or block of code

provision to go back to previous input. If any provision is to be made, needs to be done hard-coded e.g. in above rule, if user is asked to given inputs: Age, Income and Has Job, if user wants to correct Income after entering its value, it cannot be done.	(rules/set of rules) depends upon the current context (inputs provided)

Figure 2-3 Expert System Components



2.2 How business logic is implemented using expert system

Business logic is implemented using two basic ways

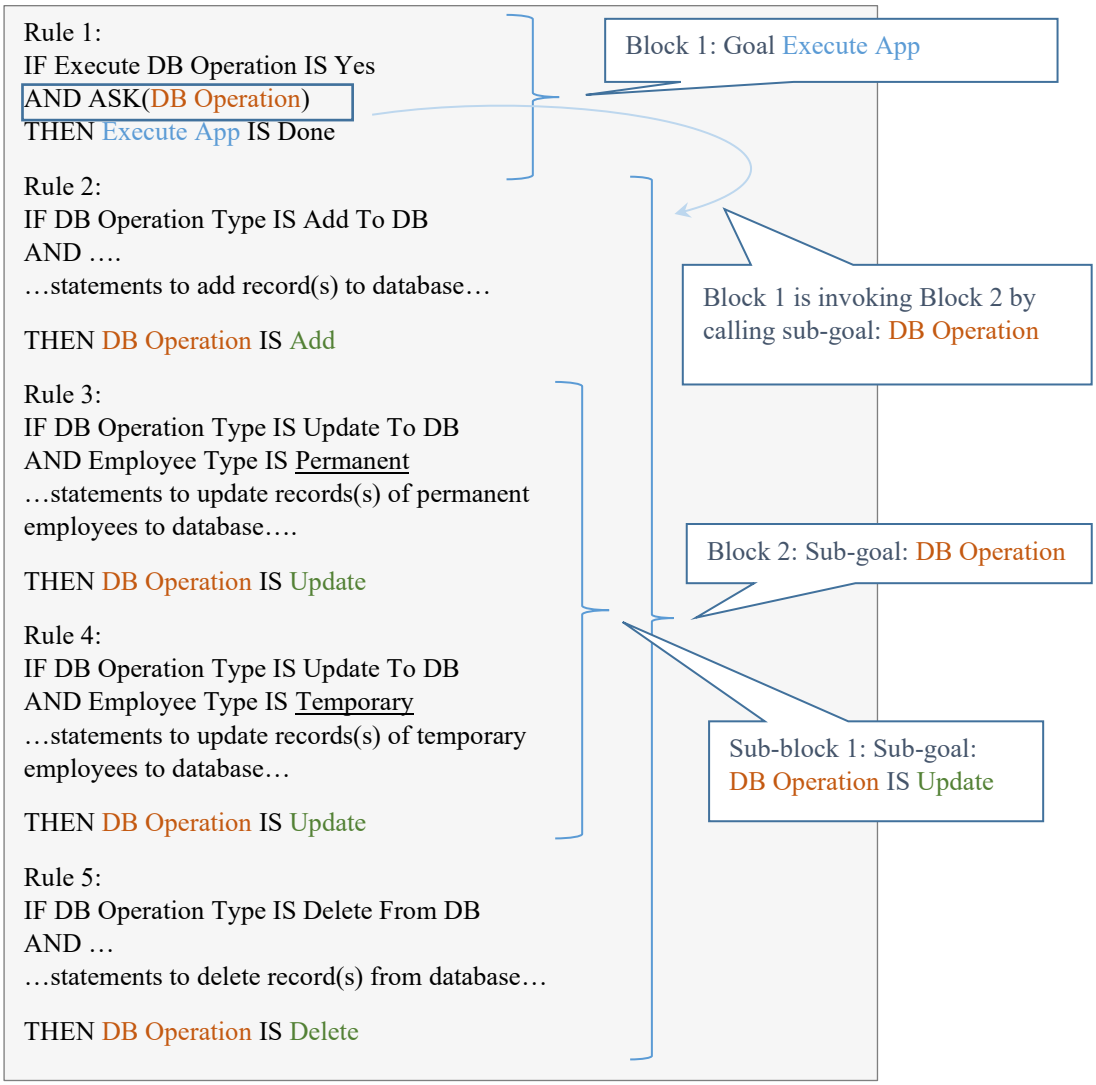
- Rule-base: contains set of rules and facts.
- Procedural logic: especially in the form of functions, procedures or APIs which are required to execute specific functionality. WKAS supports procedural logic also can be written in Python and respective database programming languages or through APIs.

2.2.1 Business logic (Rule-base)

Business logic is implemented through set of rules called as rule-base. Rules typically have format of IF conditions THEN statement ELSE statement. Rules in rule-base are divided in blocks where each block can represent specific functionality or business logic, there can be sub-blocks within block of rules. There can be hierarchical dependency on each rule or block of rules. For example, all rules which are used to perform database operations such as add, update, delete can be put in one block and all rules which update database based on different criteria can be put under one sub-block. All rules

having common variable in THEN part typically fall under one block. In expert system vocabulary, they are called goal or sub-goal variables.

Figure 2-1 Sample Rule-base



Refer to figure 1.1, there are five rules in the rule-base, block of four rules 2-5 (referred as block 2) do update database operations, rule 1 is main rule which has goal variable to get application started. Within block 2, there is sub-block of two rules which updates records to database based on criteria (one for temporary employees and another one for permanent employees).

Every expert system tool, shell or platform follows certain syntax and semantics like higher level programming languages while writing the business logic using rules. WKAS has its own language to create rules for business logic.

Figure 2-2 Sample rule and execution using backword reasoning

Rule 1

IF Applicant has Job IS Yes
AND Job Type IS Temporary
AND Experience >= 5
THEN Job Status IS Good

Conditions

Action or derive conclusion

Rule 2

IF Applicant has Job IS Yes
AND Job Type IS Permanent
THEN Job Status IS Good

Rule 3

IF Household Income >= 2000
AND Number of Dependents < 3
THEN Income Status IS Adequate

Rule 4

IF Debt to Income Ratio < 25
AND Overall Debt Liabilities IS Lower
THEN Debt Status IS Low

Rule 5

IF Job Status IS Good
AND Income Status IS Adequate
AND Debt Status IS Low
AND Credit History IS Good
THEN Approve Loan IS Yes
ELSE Approve Loan IS No

Sub-goal

Goal

Let us understand how expert system will execute rules using *backword reasoning*.

Step 1: setting the goal: in this example the goal is Approve Loan

Step 2: inference engine starts evaluating rule(s) which has (have) the goal. It will pick-up the rule 5 and will start executing it.

Step 3: engine encounters condition Job Status IS Good, it finds out Job Status IS Good is sub-goal, and is determined from rule 1 and rule 2. The engine puts rule 5 on hold and puts rule 1 & 2 rules in *conflict set*.

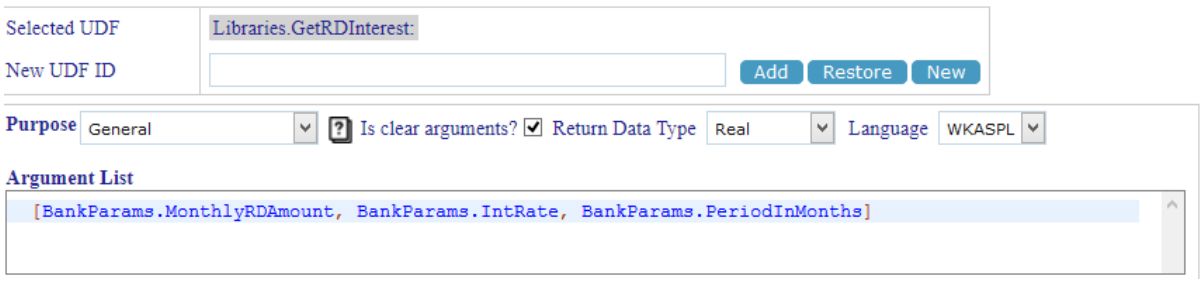
Step 4: engine starts executing first rule in conflict set means rule no 1 and starts executing it. While executing rule, it pick-ups first condition and comes across variable Applicant has Job and asks user to enter value of it. If user enters Yes, engine executes next condition and asks value of variable Job Type. If user enters Permanent, the rule 1 is discarded and second rule in conflict means rule 2 is taken for execution. All conditions of this rule are true so it *derives* Job Status IS Good. It goes back to rule 5 again.

Step 5: condition 2 of rule 5 again has sub-goal Income Status IS Adequate again suspends execution of rule 5 and pick-ups rule 3…This continues till either all conditions of rule 5 are satisfied or one of them fails. If all are satisfied, goal variable: Approve Loan value will be set to Yes otherwise it is set to No.

2.2.2 Business logic (procedural code)

In rule-based system, procedural logic can be used by writing functions or procedures using language supported by expert system tool or any external language supported. In WKA Studio, procedural code can be written in language supported by WKAS/PL, Python, database programming languages or C/C++ through DLLs.

Figure 2-3 Sample User Defined function to calculate RD interest



Home | User Defined Functions ×

```
USING BankParams
InterestAmount:=0;
IF(PeriodInMonths>=1 AND MonthlyRDAmount>=1,Go);
          InterestAmount := MonthlyRDAmount * (1 + IntRate / 400)^(1 / 3.0);
          Count:=2;
          FOR(Count IN GET_IN_LIST(2,PeriodInMonths,1),RD)
              InterestAmount := (MonthlyRDAmount+InterestAmount) * (1 + IntRate / 400)^(1 / 3.0);
              Count:=BankParams.Count+1;
          END_FOR RD;
END_IF Go;
InterestAmount:=InterestAmount-MonthlyRDAmount*PeriodInMonths;
RESET_VAR_VALUE("BankParams.Count");
RETURN InterestAmount;
```

Figure 2-4 Sample User Defined Function to calculate Term Deposit Interest

Home | Apps × | User Defined Functions ×

```
USING BankParams
InterestAmount:=0;
Temp1:=DepositAmount;
CompFreqInNumber:=C_NUMBER(GET_MENU_CODE(BankParams.CompoundingFrequency,CompoundingFreq));

COMMENT: Compute for year;
IF(PeriodInYears>=1,ForYear);
     InterestAmount:=DepositAmount*(1+(IntRate/100)/CompFreqInNumber)^(CompFreqInNumber*PeriodInYears);
     DepositAmount:=InterestAmount;
END_IF ForYear;

COMMENT: Compute for Months;
IF(PeriodInMonths>=1,ForMonth);
     InterestAmount:=DepositAmount*(1+(IntRate/100)/CompFreqInNumber)^(CompFreqInNumber*(PeriodInMonths/12.0));
     DepositAmount:=InterestAmount;
END_IF ForMonth;

COMMENT: Compute for Days;
IF(PeriodInDays>=1,ForDays);
     InterestAmount:=DepositAmount*(1+(IntRate/100)/CompFreqInNumber)^(CompFreqInNumber*(PeriodInDays/356.0));
END_IF ForDays;

InterestAmount:=InterestAmount-Temp1;
RESET_VAR_VALUE("Temp1");
RETURN InterestAmount;
```

Figure 2-5 Sample function in Python returning simple compounded interest through WKA Studio

```
import math
def GetCompoundInterest(principle,rate,time):
  rate=rate/12
  Amount = principle * (math.pow((1 + rate/100), time))
  CI = Amount - principle
  return round(CI,2)
```

This Python function can be executed using WKA Studio function: CALL_PY_FUNCTION e.g. CALL_PY_FUNCTION(P1,GetCompoundInterest,[100,10.0,12]) would return 10.47

Figure 2-6 Sample Python API returning simple compounded interest

```
import math
from flask import Flask
from flask import request
from flask import jsonify
app = Flask(__name__)
@app.route('/GetCompoundInterest', methods=["POST"])
def GetCompoundInterest():
  request_json = request.get_json()
  principle=request_json.get('principle')
  rate=request_json.get('rate')
  time=request_json.get('time')
  rate=rate/12
  Amount = principle * (math.pow((1 + rate/100), time))
  CI = Amount - principle
  return jsonify(round(CI,2))
app.run()
```

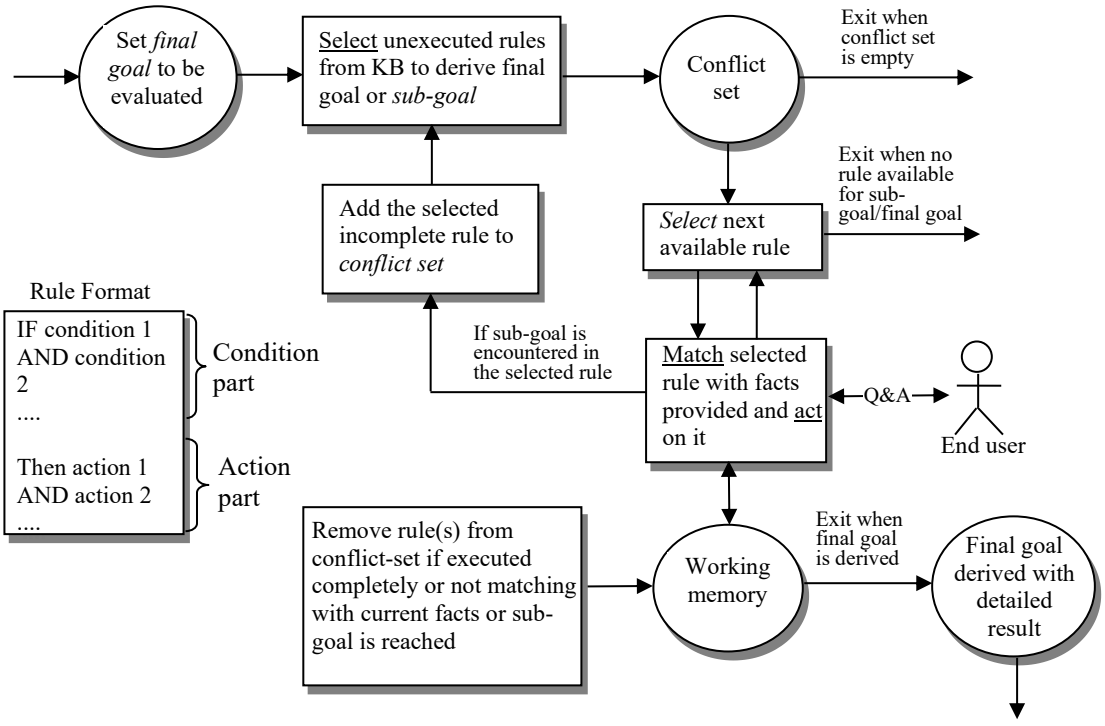
Above Python API can be executed using WKA Studio function: CALL_PY_API e.g. CALL_PY_API("http://localhost:5000","GetCompoundInterest:principle,rate,time",[100,10.0,12]) would return 10.47. If API is already configured then param names may not be required.

Figure 2-7 Sample MySQL function returning simple compounded interest through WKA Studio

```
create function GetCompoundInterest(principle int,rate float,months int) returns float
begin
 declare amount float;
 declare interest float;
 set rate=rate/12;
 set amount = principle * (pow((1 + rate/100), months));
 set interest=round(amount-principle,2);
 return interest;
end;
```

This MySQL function can be executed using CALL_DB_ROUTINE e.g. CALL_DB_ROUTINE(MySQL,GetCompoundInterest,[100,10.0,12]) would return 10.47.

Figure 2-8 How backward reasoning works?



3 GETTING STARTED WITH WKA STUDIO

3.1 Skill-sets required for WKA Studio

3.1.1 Developer

WKA Studio is developed in C#, ASP.NET, supports multiple databases, however current version supports automatic creation of schema etc. on MySQL database. Since WKA Studio is based on Expert System technology, the developer needs thorough understanding of how expert system technology works and how business logic is implemented through set of rules. Developer is expected to have fair understanding of programming concepts and implementing business logic using any typical programming language.

WKA Studio effectively uses combination of JavaScript/JQuery at front-end, combinations of WKAS/PL, Python, C#, C/C++ (through DLLs) at business layer and DBMSs at client side.

Overall

Fair understanding of WKA Studio architecture (tenants, projects, applications, WKAS sessions, app sessions), interfaces and modules

Client Side

HTML, CSS, JavaScript/JQuery, JQ plot, InputMask, Bootstrap

Business logic

WKAS/PL, Basic Python, JavaScript (some business logic can be executed at client side), C#, C++/C through DLLs.

Database

Structured query language (SQL), DB programming (like writing stored procedures (SPs))

3.1.2 Administrator

WKA Studio runs on Windows. It needs .NET 4.5, IIS and MySQL (community version will do). Administrator needs expertise about deploying and configuring ASP.NET web applications on Windows machines/servers, configuring IIS servers, installing and configuring MySQL instances, etc.

WKA Studio configuration interfaces and system tables required to run WKA Studio smoothly.

3.2 WKA Studio Interfaces

WKA Studio provides various interfaces to create various objects which are referred by IDs. Typical objects are global variables, rules, forms, reports, parameterized queries (called dynamic queries), databases and so on. Interfaces are divided into broad categories.

Domain Vocabulary

All interfaces which help in creating definitions, schema and store data frequently are clubbed in this group. It contains interfaces to create global variables, list of possible values, lookup tables, predefined ranges and generic attributes.

Core Engine Configs

These interfaces help in implementing and configuring business logic using user-defined functions and rules, configuring various algorithms, integrating various techniques and algorithms supported by WKA Studio.

Database Interfaces

Database related interfaces are included here to create and configure database connections, parameterized SQL queries and mapping between global variables and database fields; build and execute queries interactively, manage database schema (tables, fields, views, SPs, set constraints), import data from one data source to another data source (e.g. importing CSV files to MySQL Database), do data entry in database tables.

System and App Setup

Interfaces which facilitate to create app, manage files on the server, creating and setting roles and user roles, project and tenant configurations, backup and restore tenants, projects and apps, creating help documents are categories into this group.

Forms and Reports

Includes form and report designer. Drag-n-drop interface allows to develop forms and create apps.

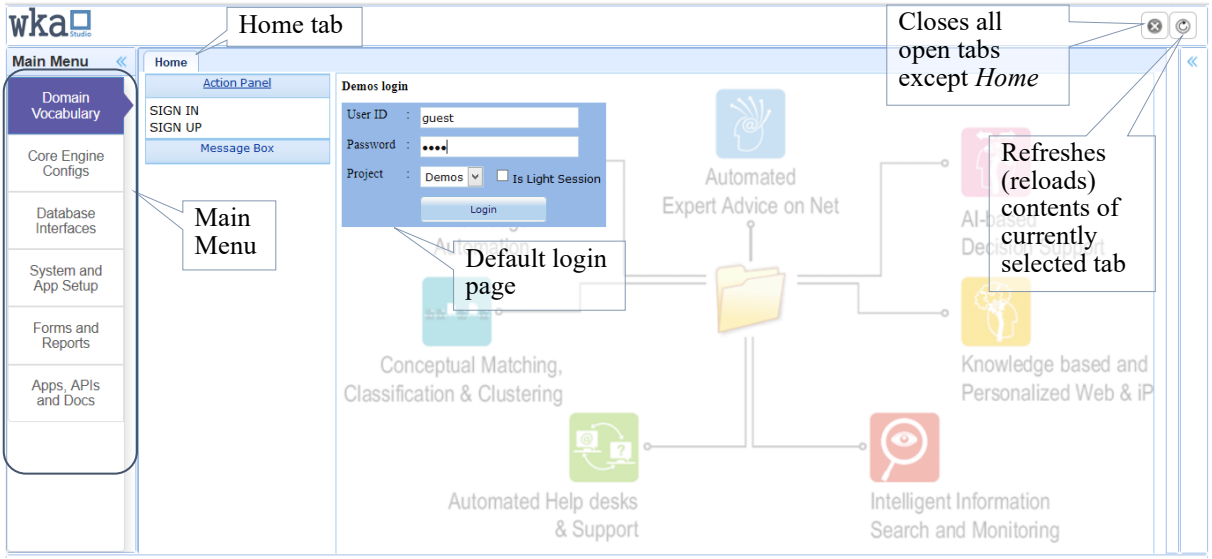
Apps, APIs and Docs

All apps in the project are listed (based on access rights to the users), APIs can be tested using APIs interface and complete WKA Studio help is available through Docs interface.

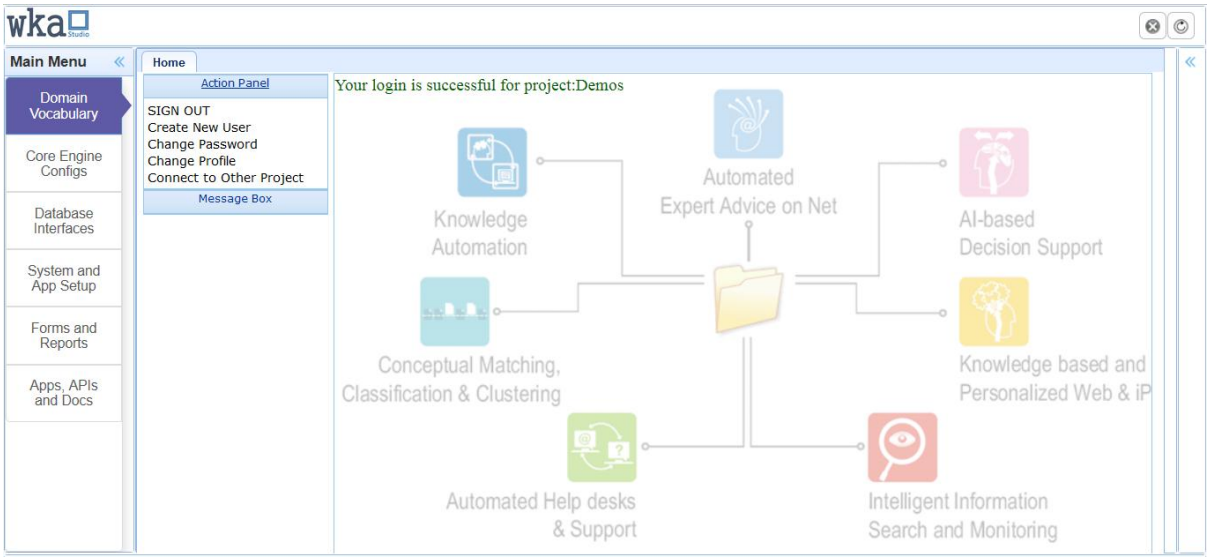
3.3 List of WKA Studio Interfaces

WKAS has many interfaces to add and update various configurations such as connecting to databases, manage database objects such as tables, views, triggers, stored procedures, predefine parameterized queries, create user defined functions, etc. Some interfaces are used at run time e.g. upload document interface which helps to upload various documents. Following is list of interfaces which are used to add and configure various objects in WKAS.

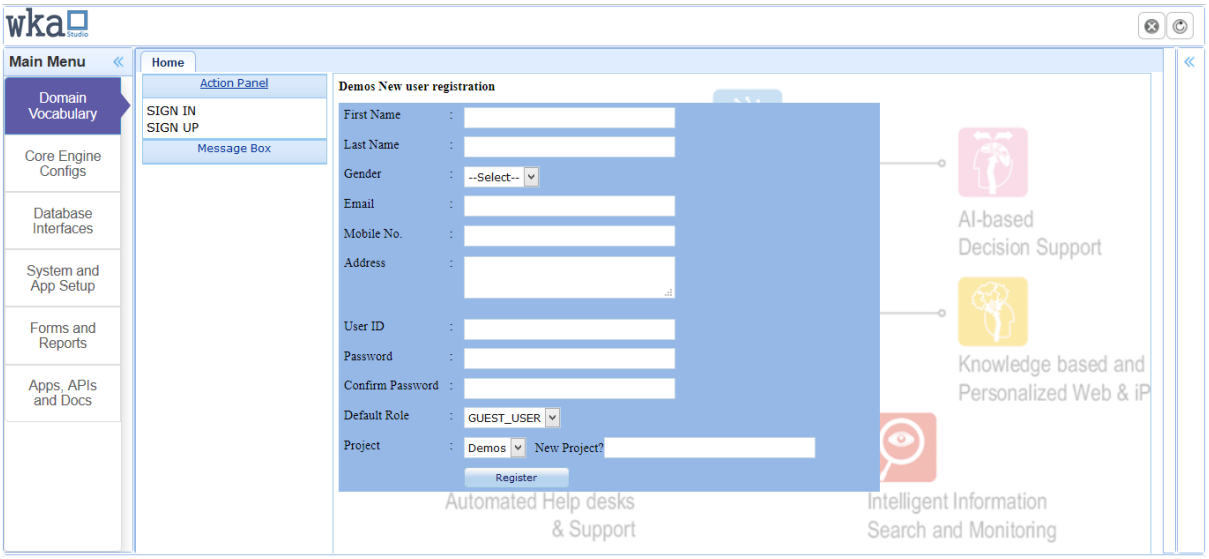
Starting interface



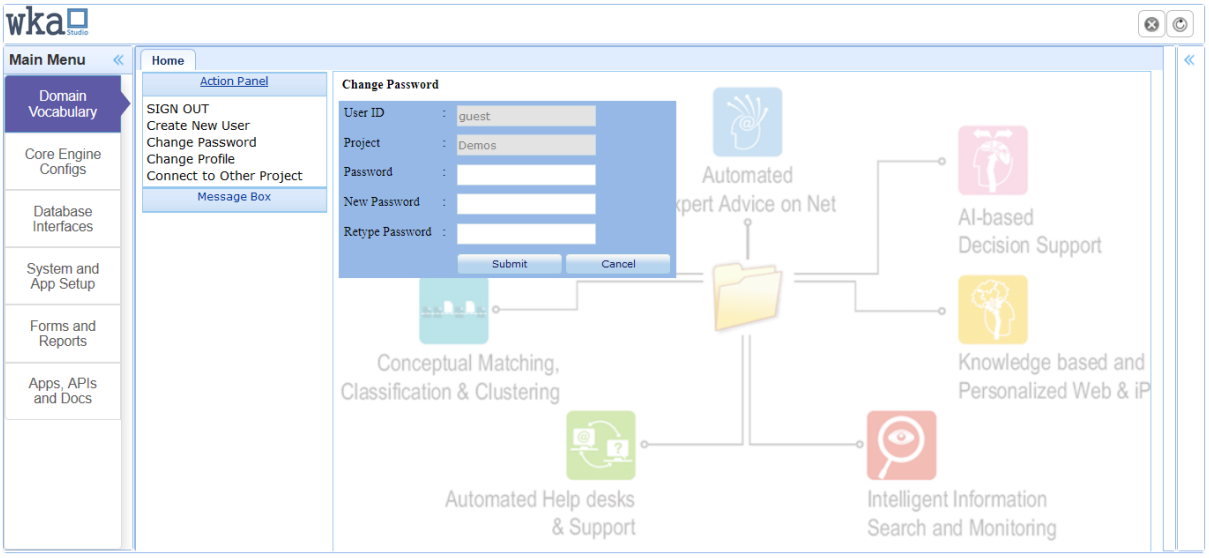
Screen after successful login



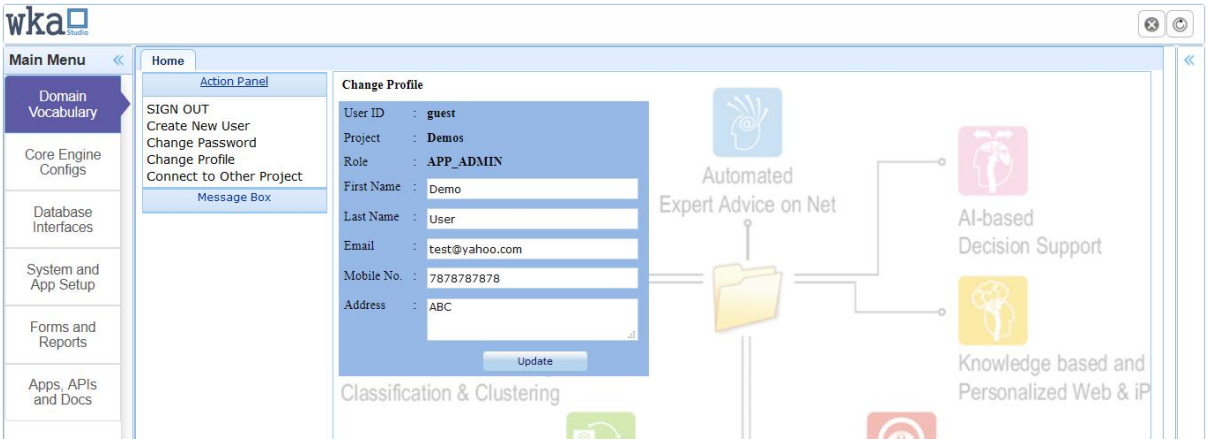
New user registration



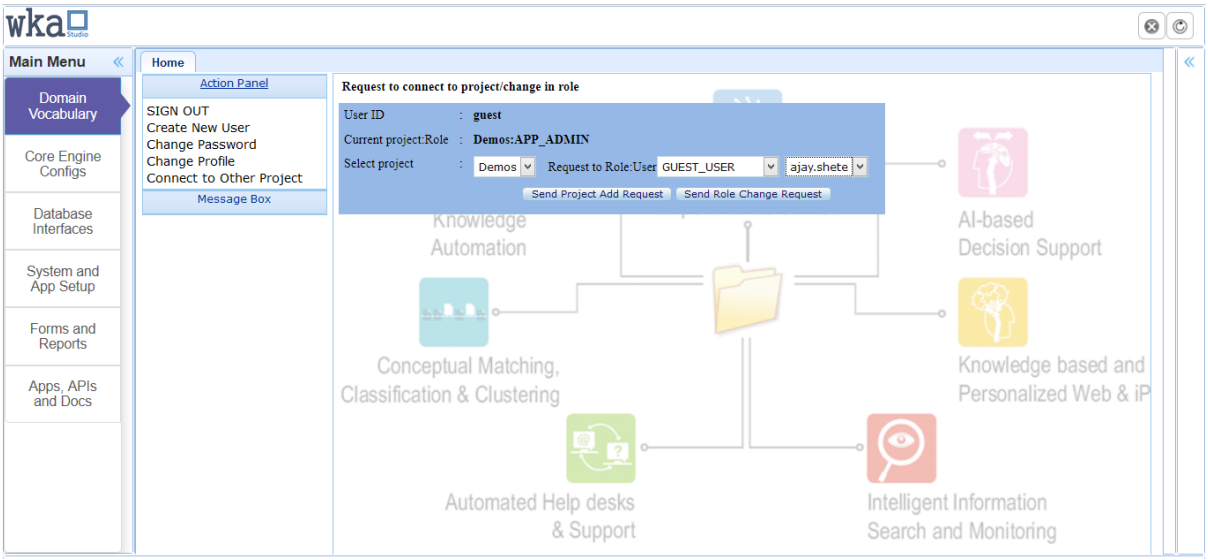
Change password



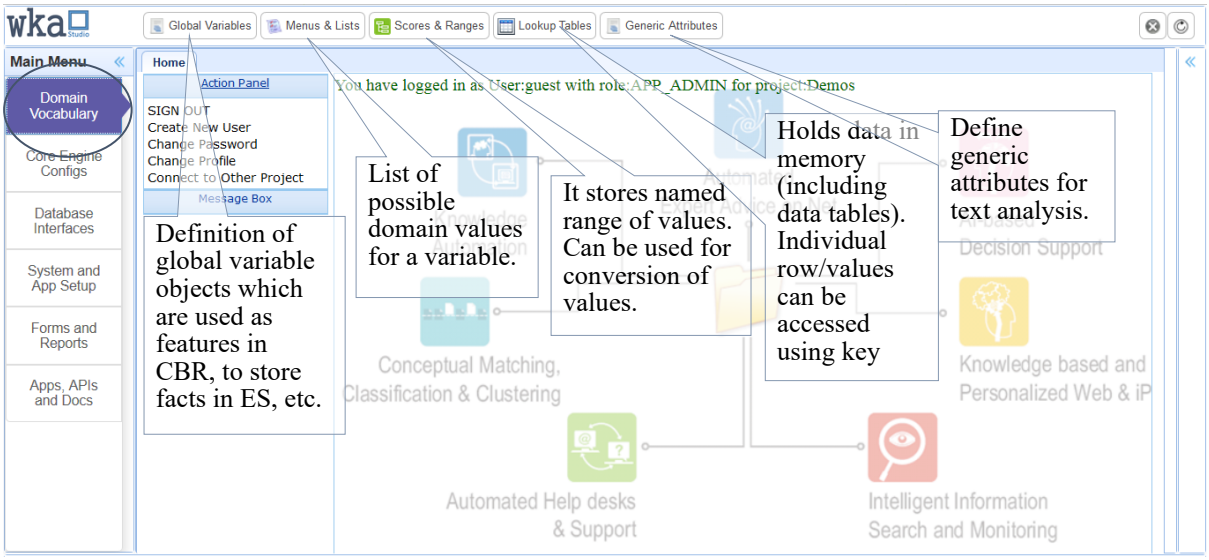
Change user profile



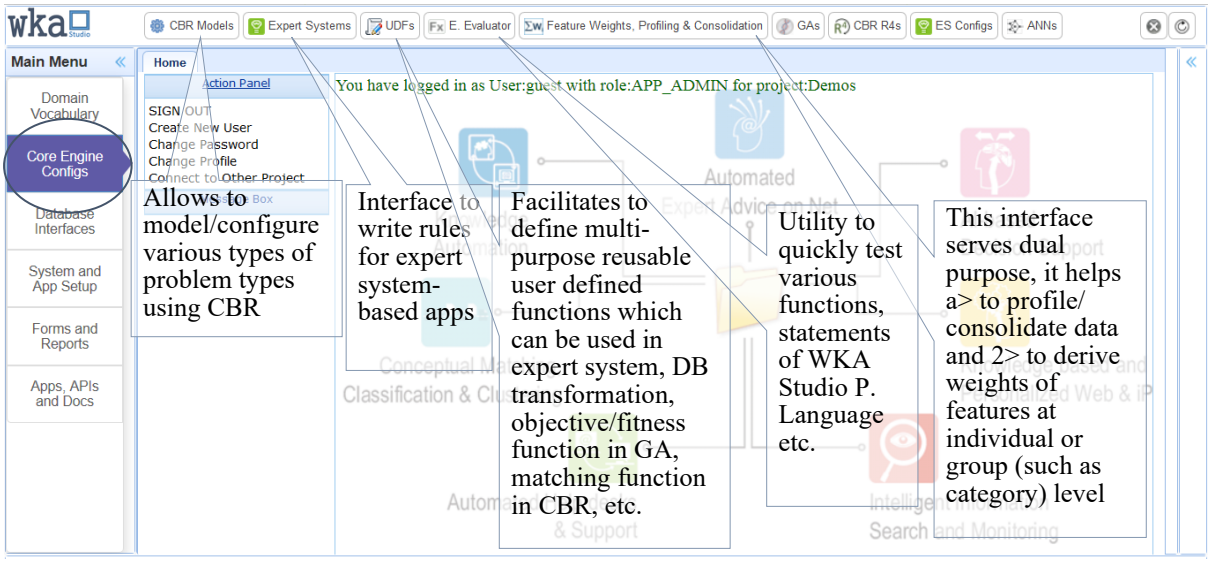
Project request/change of role request



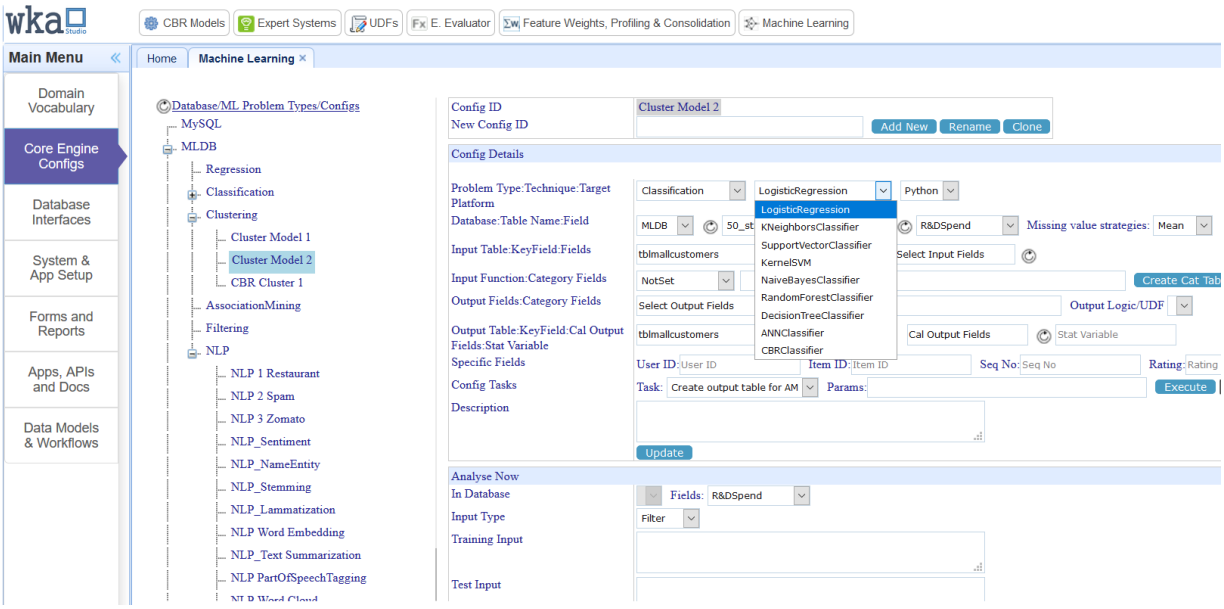
Domain Vocabulary Interfaces



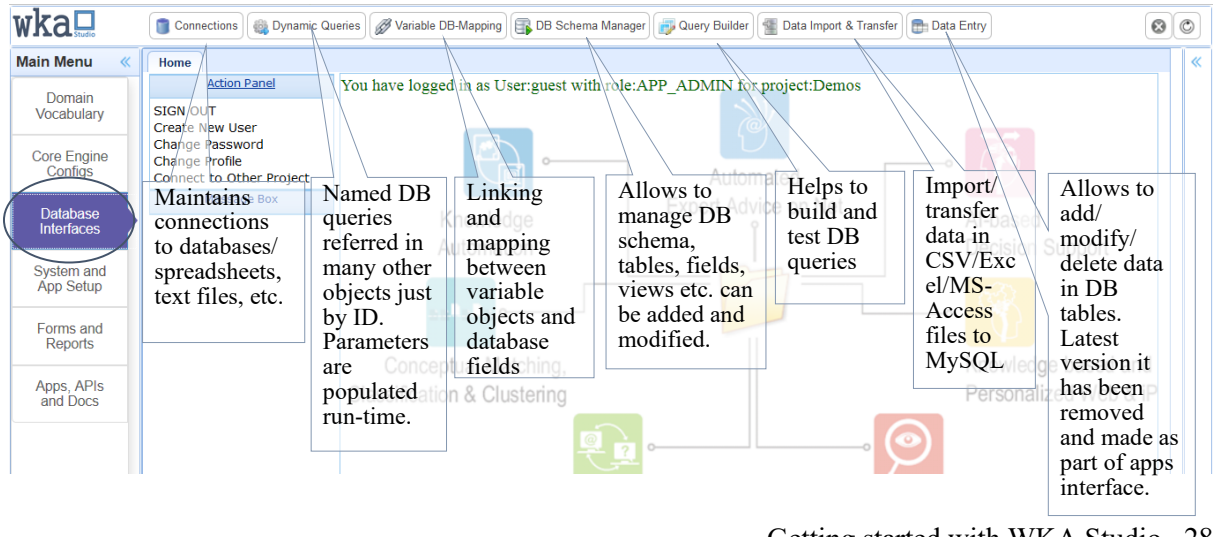
Core Engine Configuration interfaces



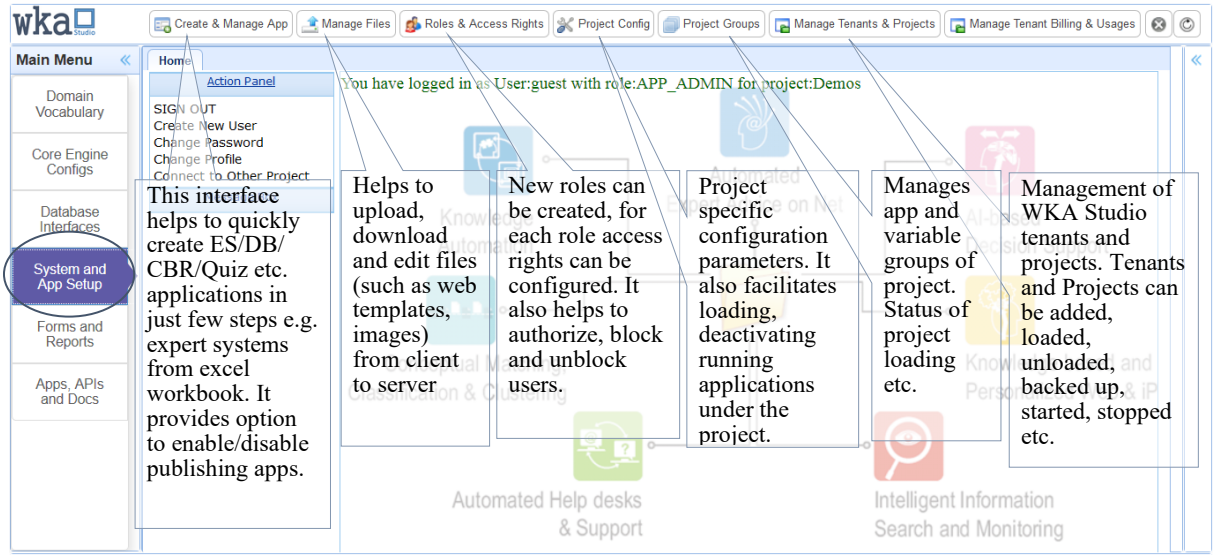
New ML Interface to model ML Proboems



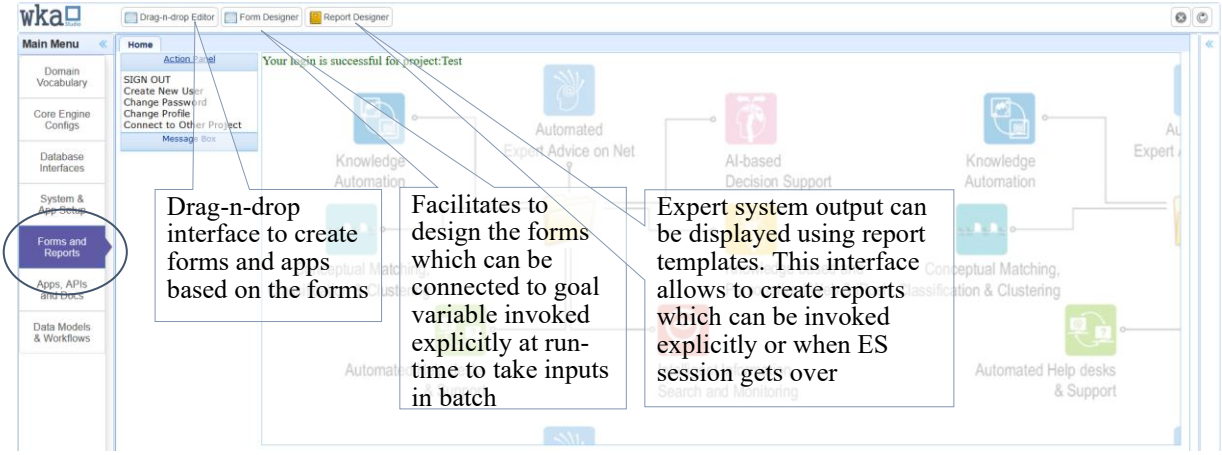
Database Interfaces



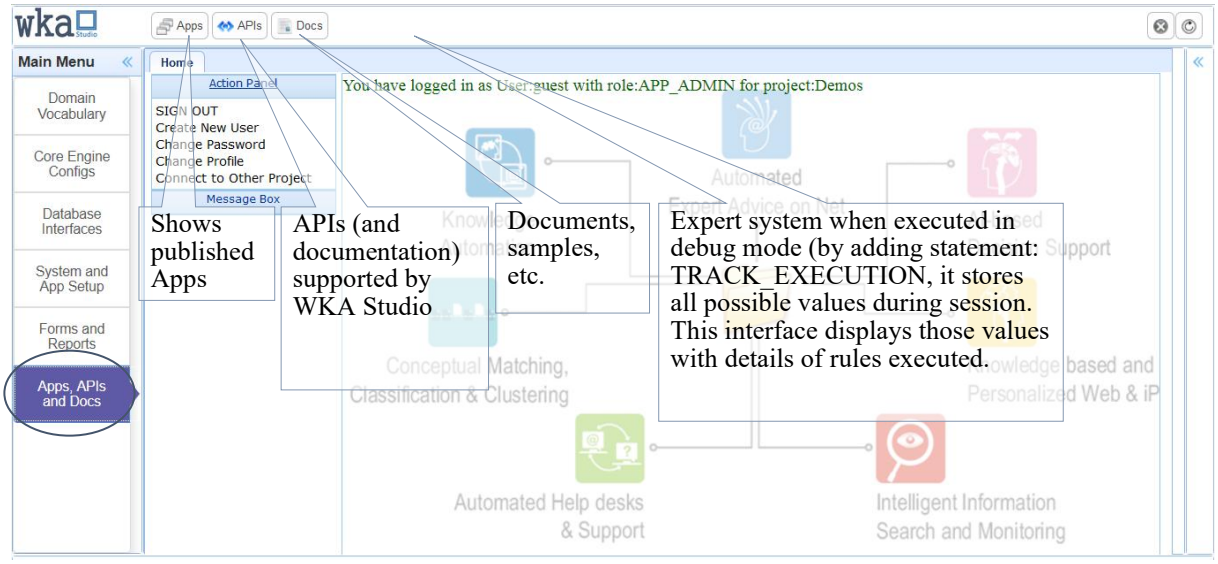
System and App Setup



Forms and Report Designer interfaces



Published Apps, APIs and Documents



3.4 Creating your first app

Simple apps can be created by two ways: 1> manually where variables and rules are added manually, 2> by writing rules in excel sheet in specified format. Specific apps like database, quiz apps can be created through app creation interface from existing database/spreadsheets. Database applications can be created from scratch using drag-n-drop interface.

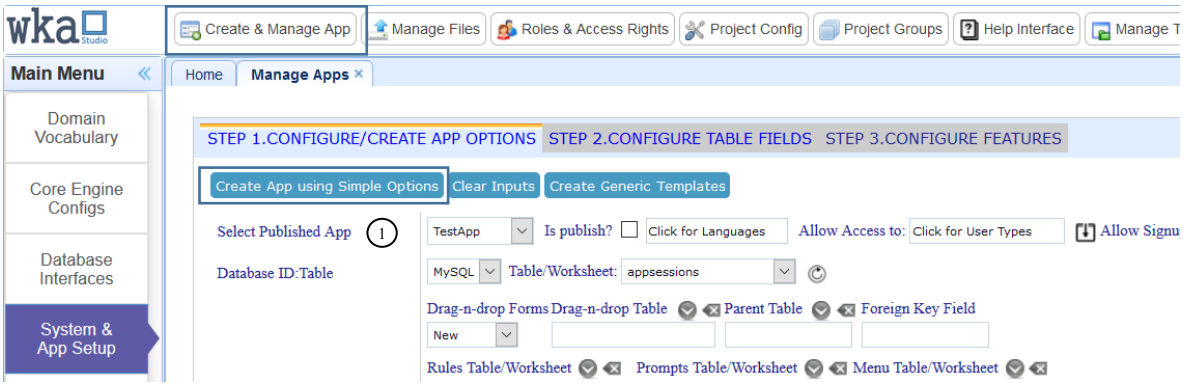
3.4.1 Creating app manually

- Create simple App using *Create & Manage Apps* interface say *MathApp* by giving application group (say MyApps) and variable group (say *MathInfo*).
- Add three input variables, *Number1*, *Number2* and *Result* in variable group say *MathInfo* as number data type using *Global Variables* Interface. Add options like min, max and default values etc. if required.
- Just add statement: *Result := Number1 + Number2* (and optionally statement to display result by default it shows output) the default rule created by using *Rules/Expert System* Interface, update the rule and click on Create Web Pages by selecting app.
- You are done! run the application through *Apps* interface

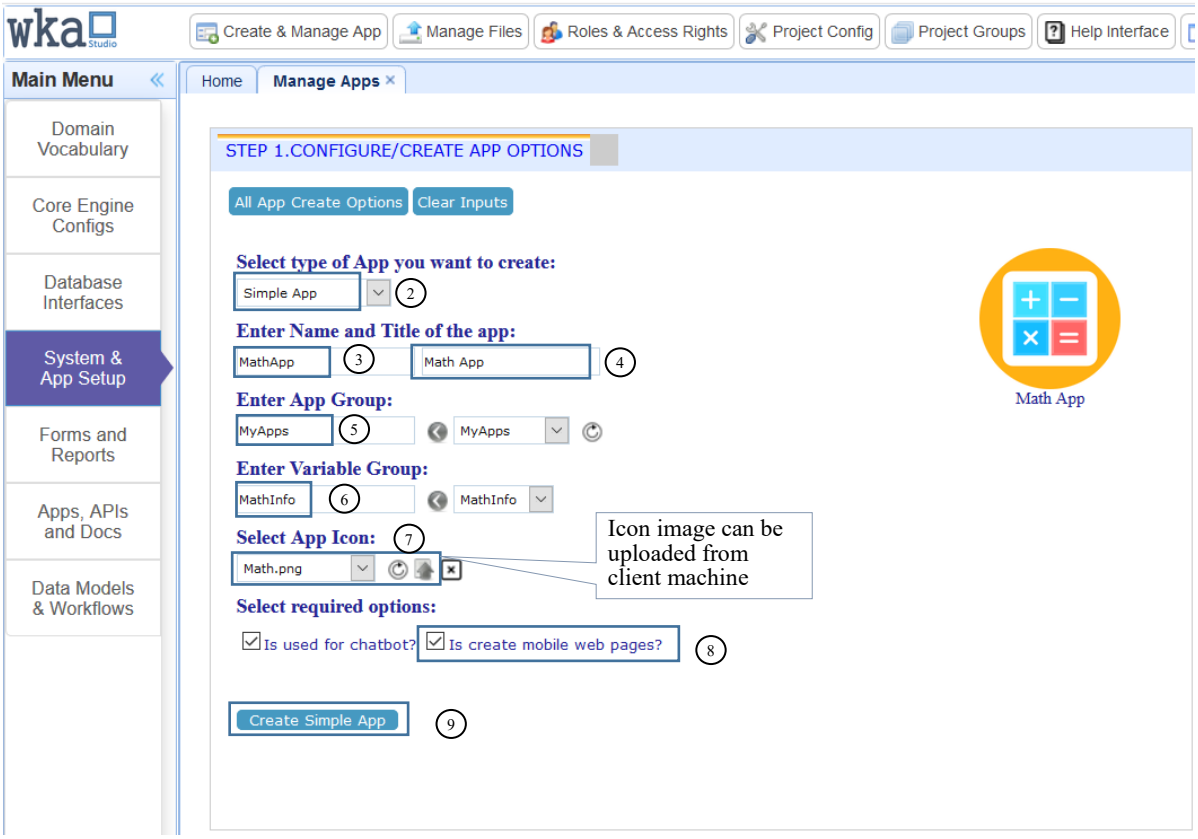
The above steps illustrate that developer just needs to write *one line of code* to create app that adds two numbers. Rest is managed through WKAS interfaces, explained in the following steps.

WKAS provides app creation interface to create apps where rules written in excel sheets can be imported. Using just few steps, all objects and entire application is created and does not to create objects explicitly (refer to tutorial section).

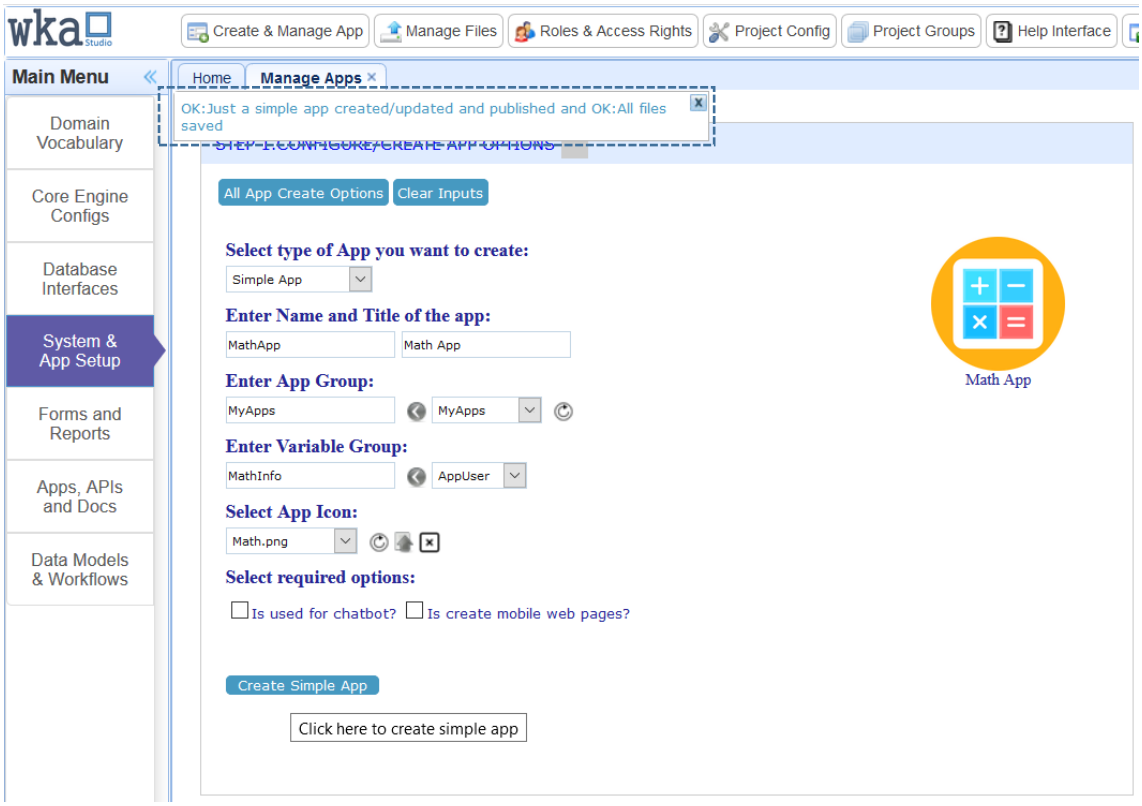
Step 1: Selecting option Create App using Simple Options (from app create interface)



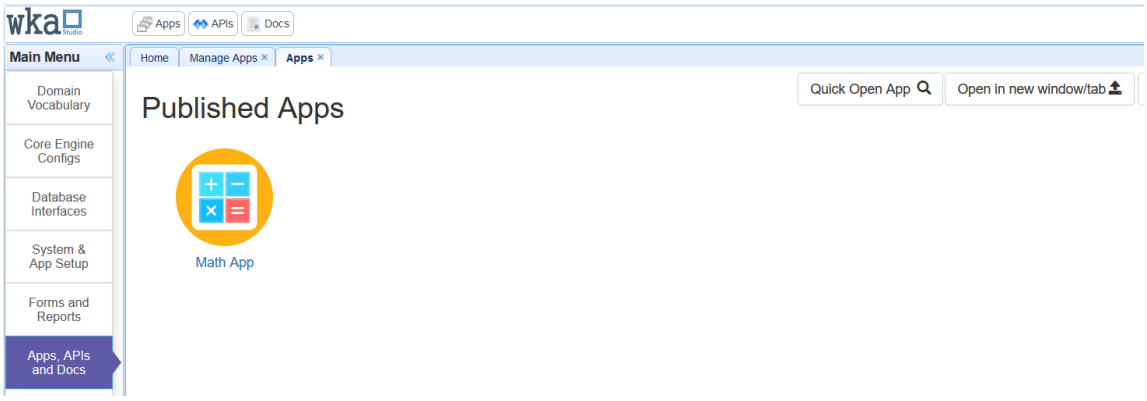
Step 2: Enter the app details and configuration

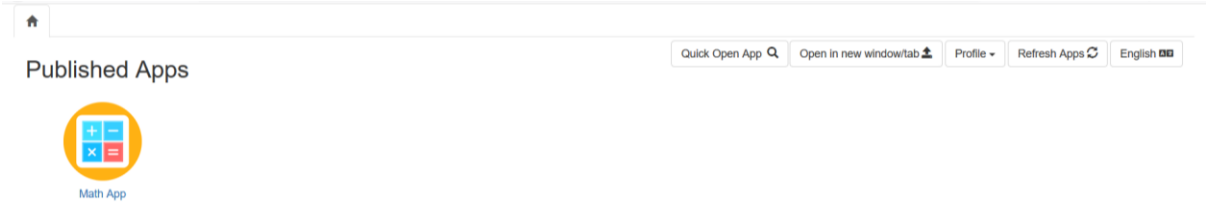


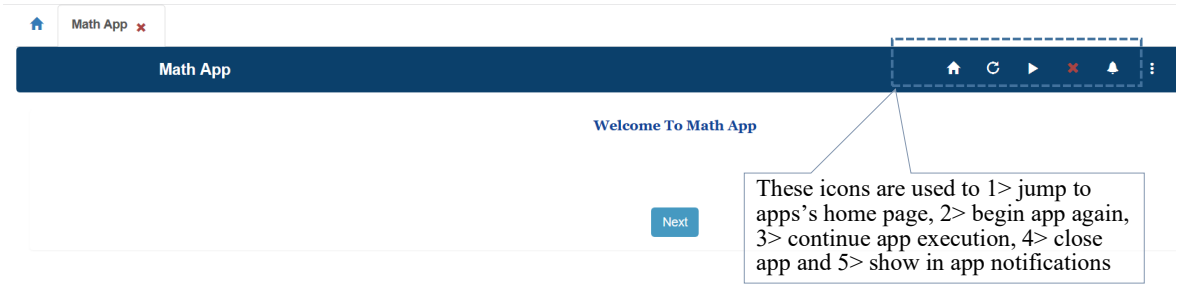
Step 3: Create app



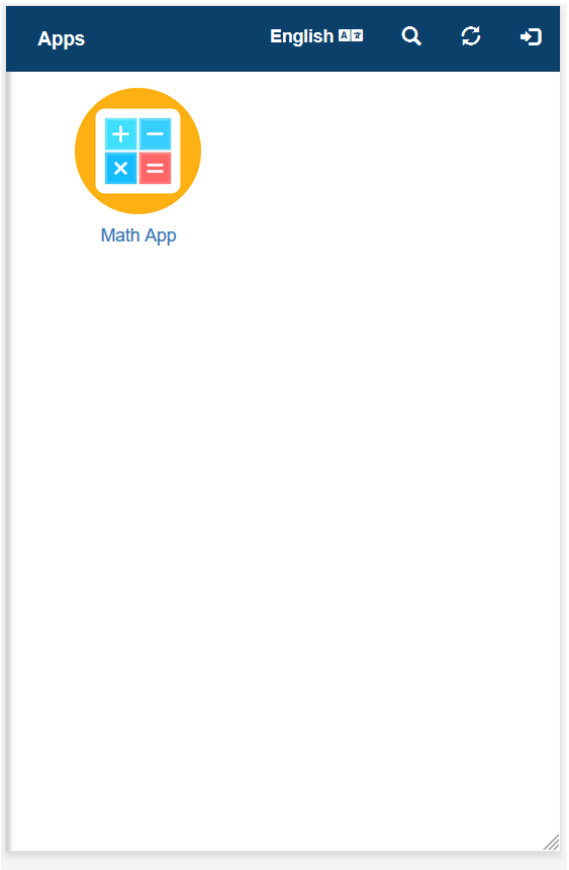
Step 4: Check and execute web app

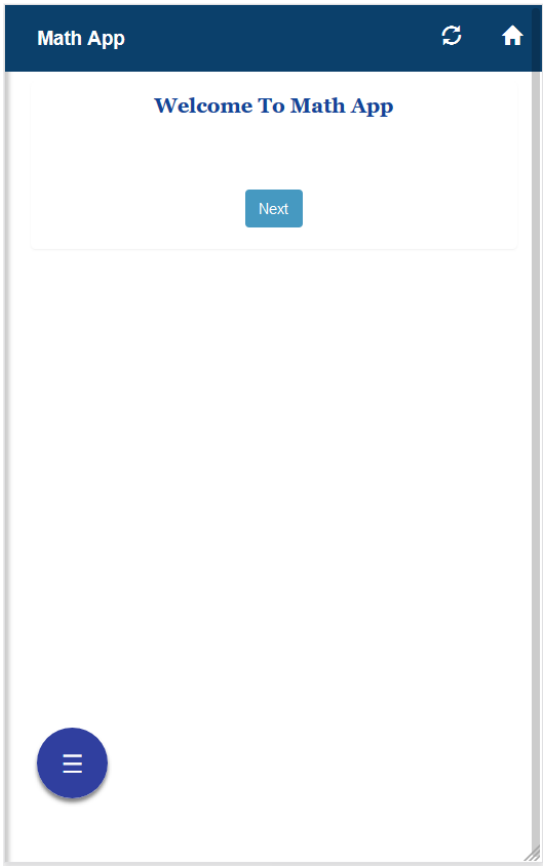






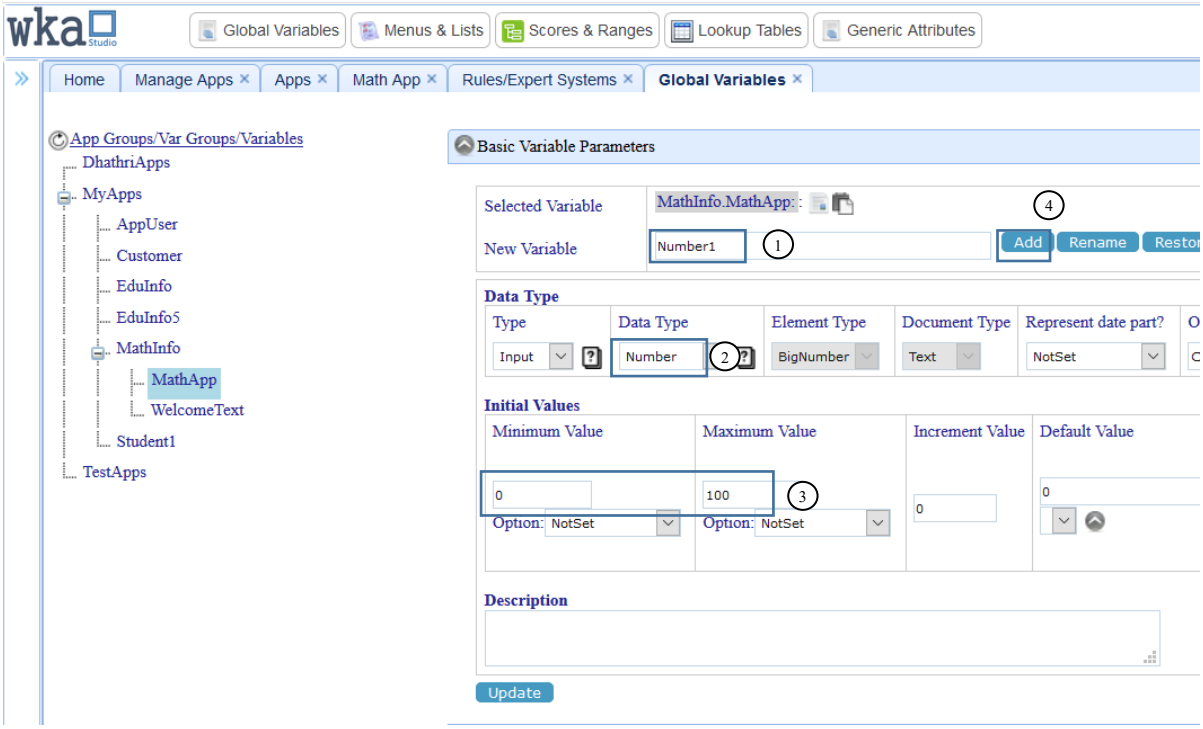
Step 6: Execute mobile app

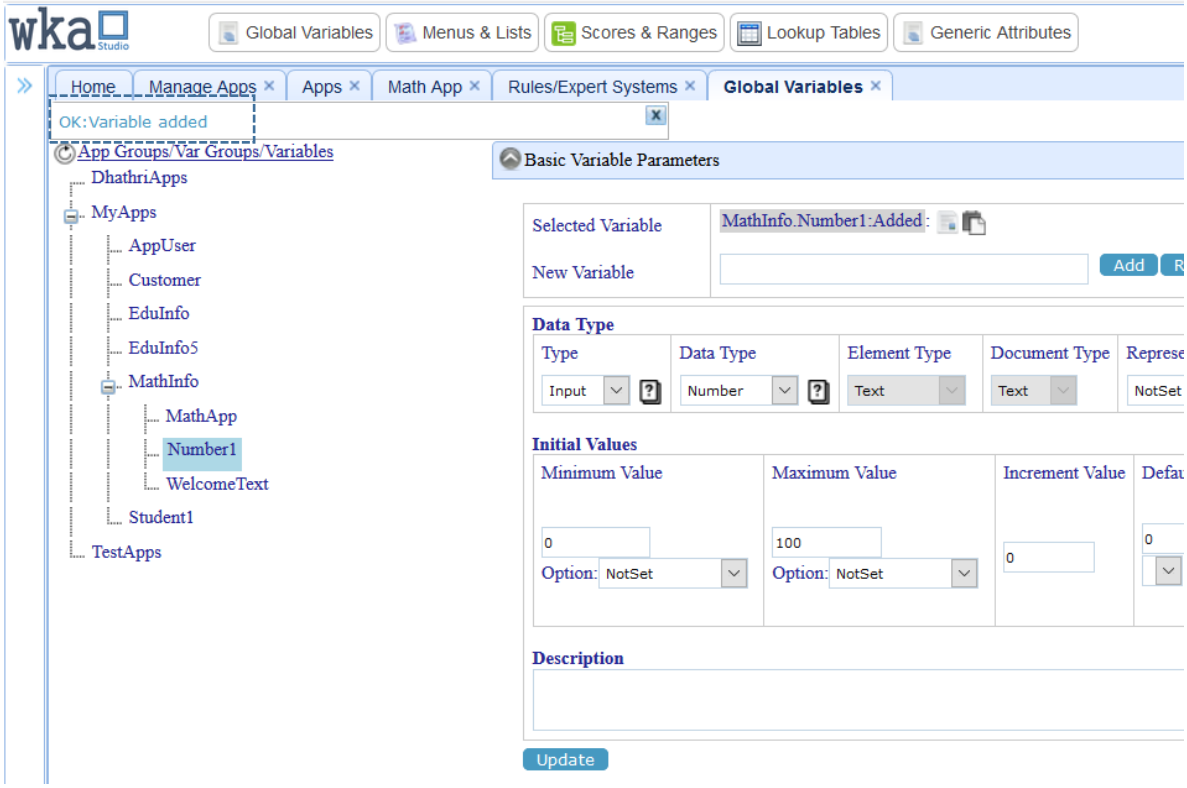


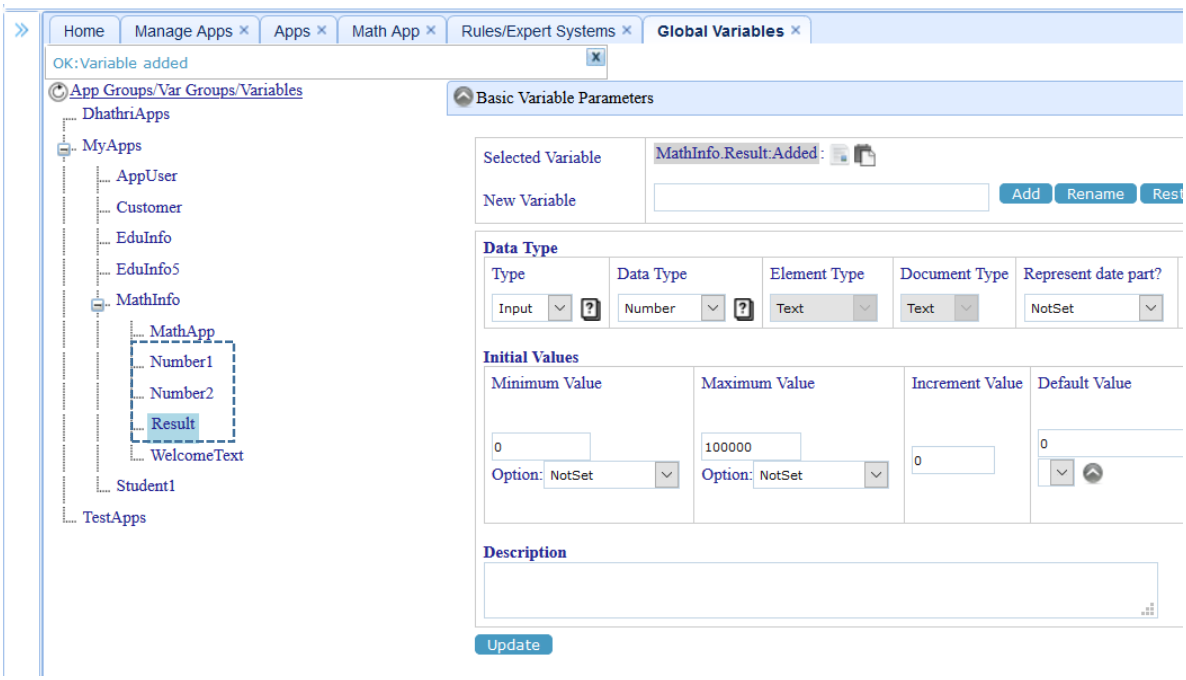


3.4.2 Modifying simple app

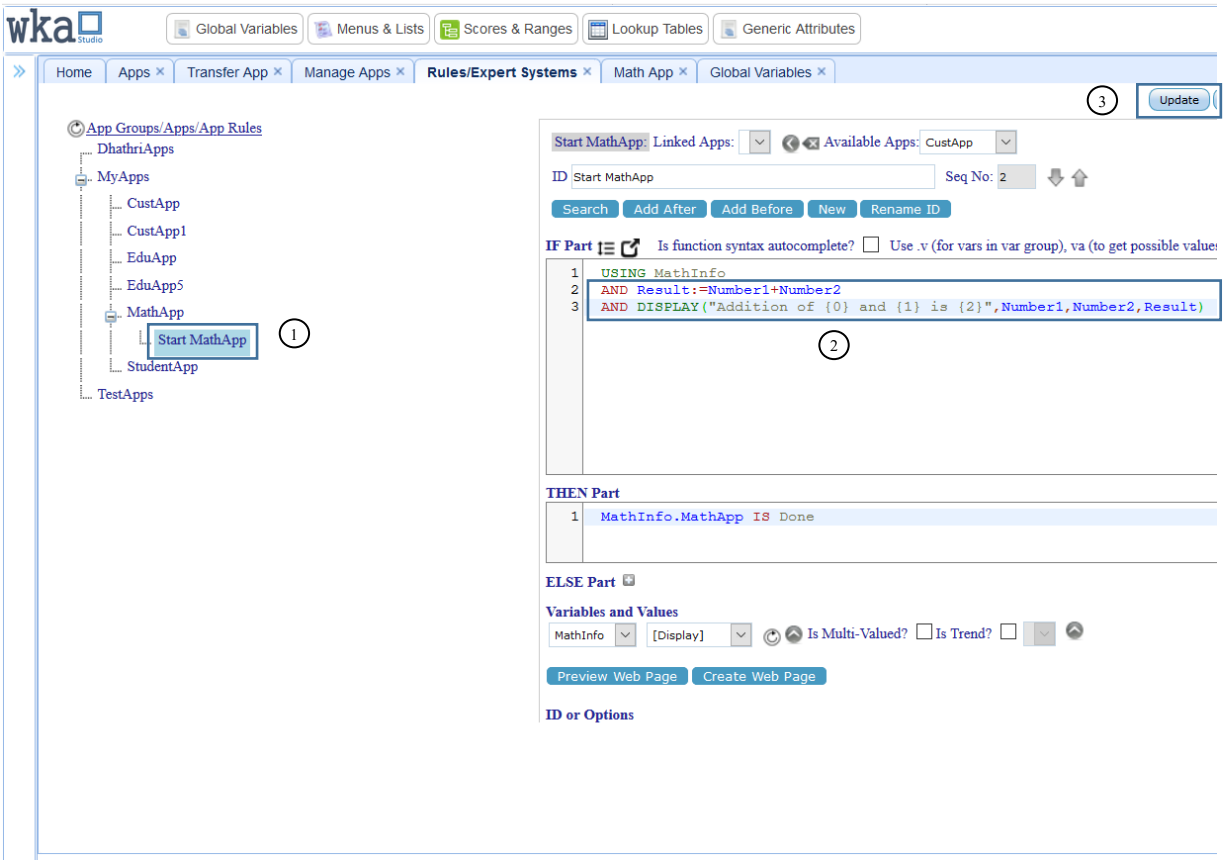
Step 1: Add variables: two numbers and result using Variable Interface

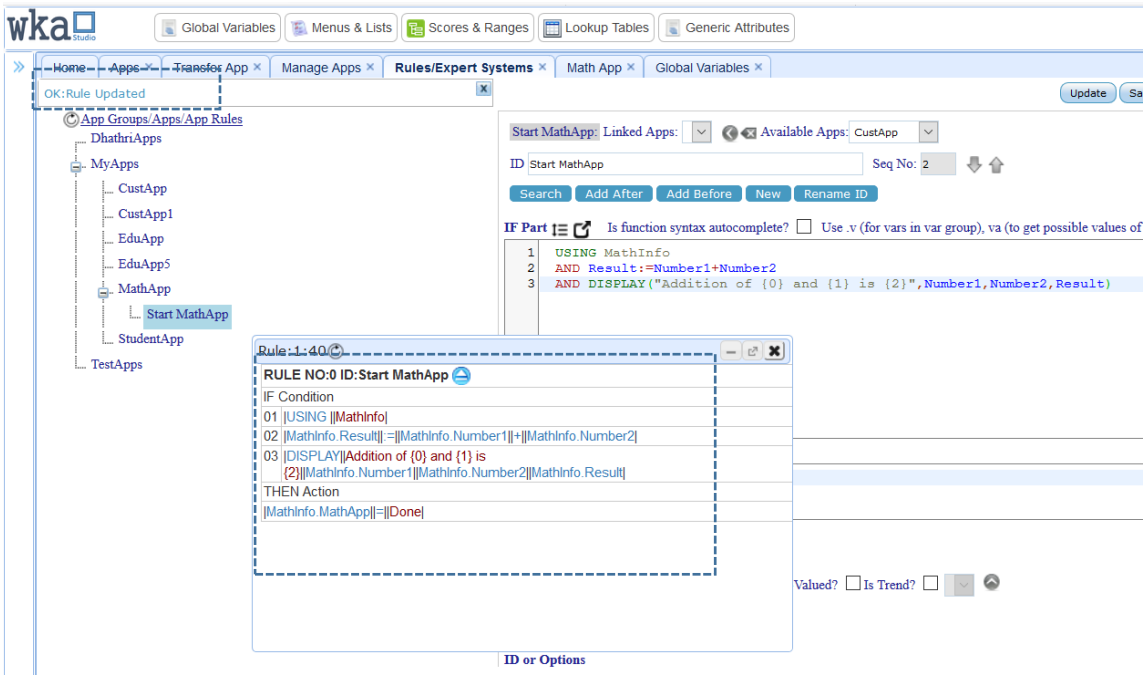

wka Studio
Global Variables
Menus & Lists
Scores & Ranges
Lookup Tables
Generic Attributes
Home
Manage Apps
Apps
Math App
Rules/Expert Systems
Global Variables
App Groups/Var Groups/Variables
DhathriApps
MyApps
AppUser
Customer
EduInfo
EduInfo5
MathInfo
MathApp
WelcomeText
Student1
TestApps
Basic Variable Parameters
Selected Variable
MathInfo.MathApp:
New Variable
Number1
1
2
3
4
Add
Rename
Data Type
Type
Data Type
Element Type
Document Type
Represent date part?
Input
Number
BigNumber
Text
NotSet
Initial Values
Minimum Value
Maximum Value
Increment Value
Default Value
0
100
Option:
NotSet
Description
Update


wka Studio
Global Variables
Menus & Lists
Scores & Ranges
Lookup Tables
Generic Attributes
Home
Manage Apps
Apps
Math App
Rules/Expert Systems
Global Variables
OK:Variable added
App Groups/Var Groups/Variables
DhathriApps
MyApps
AppUser
Customer
EduInfo
EduInfo5
MathInfo
MathApp
Number1
WelcomeText
Student1
TestApps
Basic Variable Parameters
Selected Variable
MathInfo.Number1:Added
New Variable
Add
Data Type
Type
Data Type
Element Type
Document Type
Input
Number
Text
NotSet
Initial Values
Minimum Value
Maximum Value
Increment Value
0
100
Option:
NotSet
Description
Update

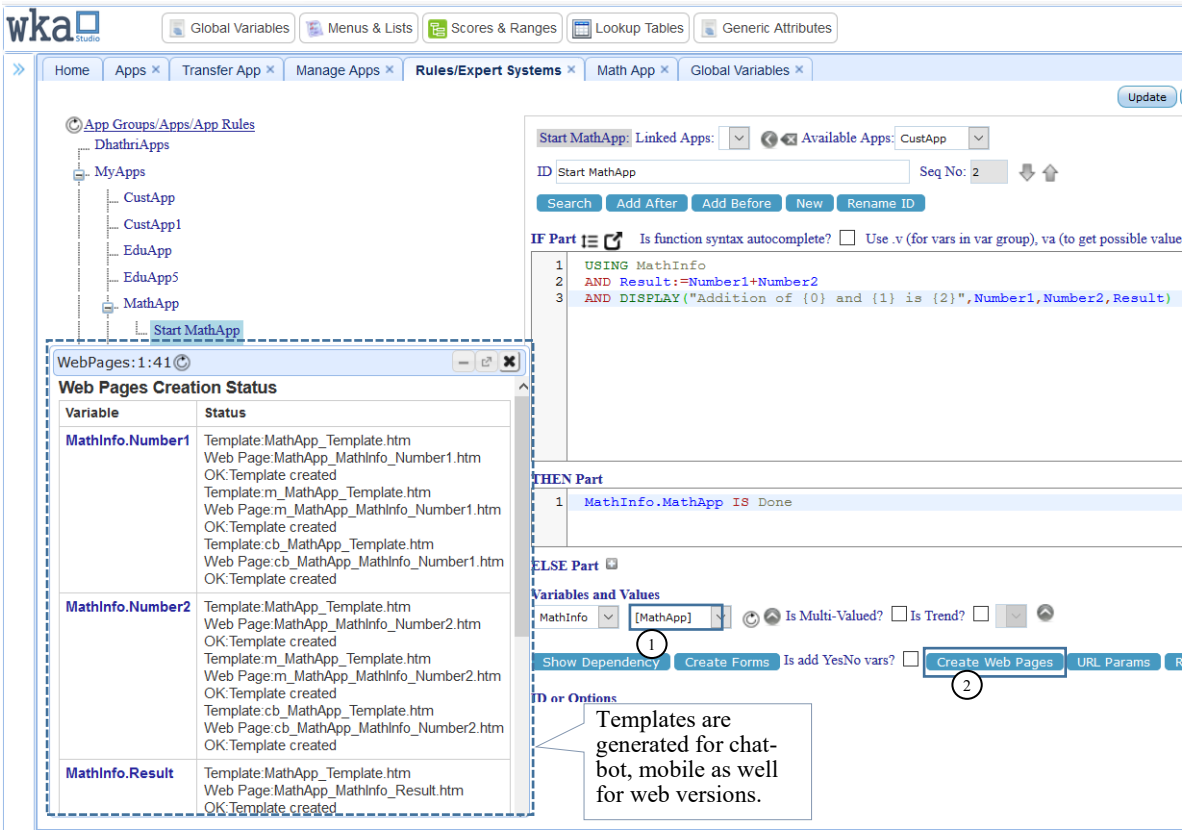


Step 2: Modify default rule, adding statement and creating Web-pages (WKAS automatically adds JavaScript validation scripts)

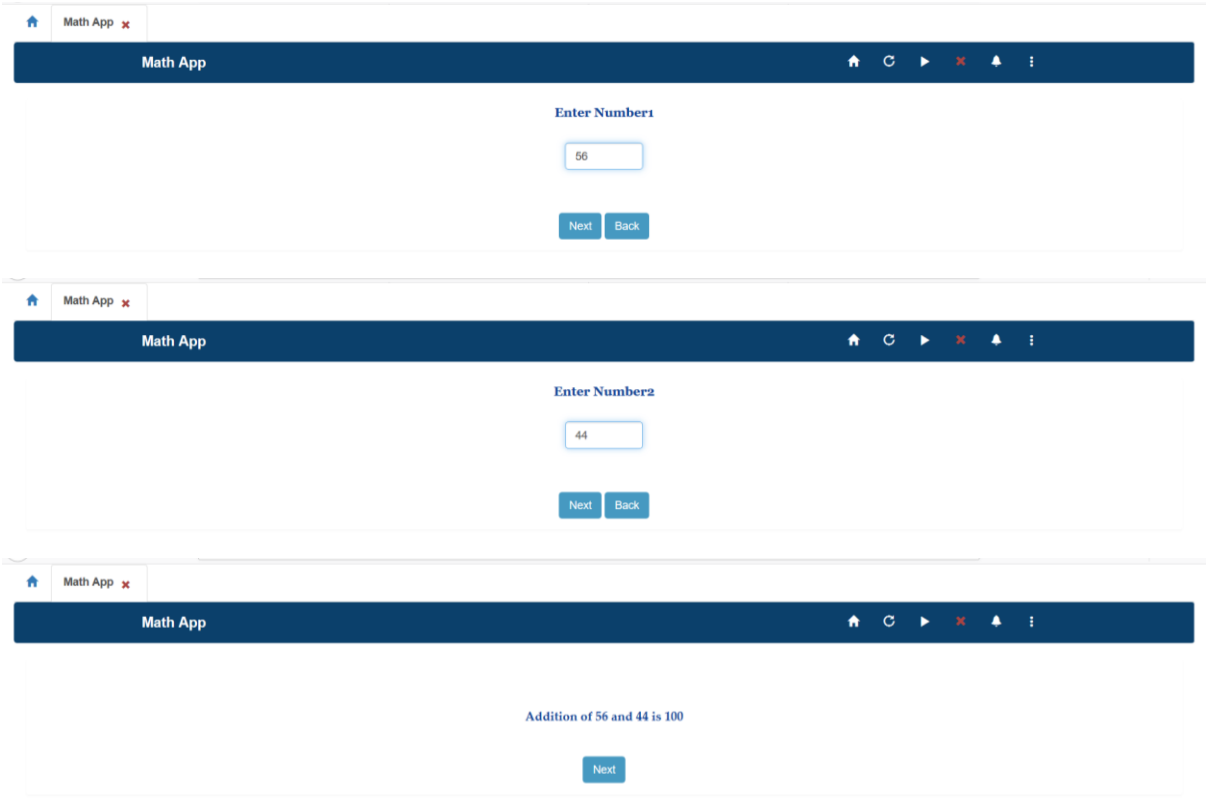




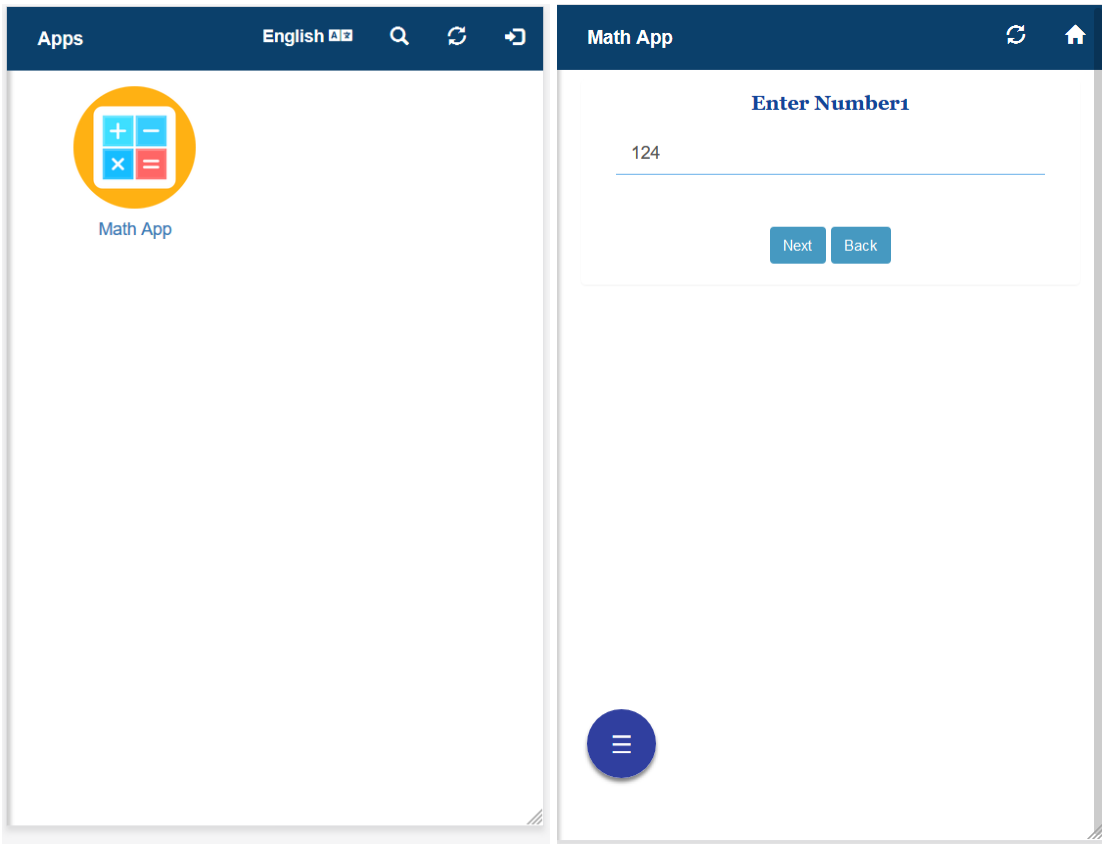
Step 3: Create html templates

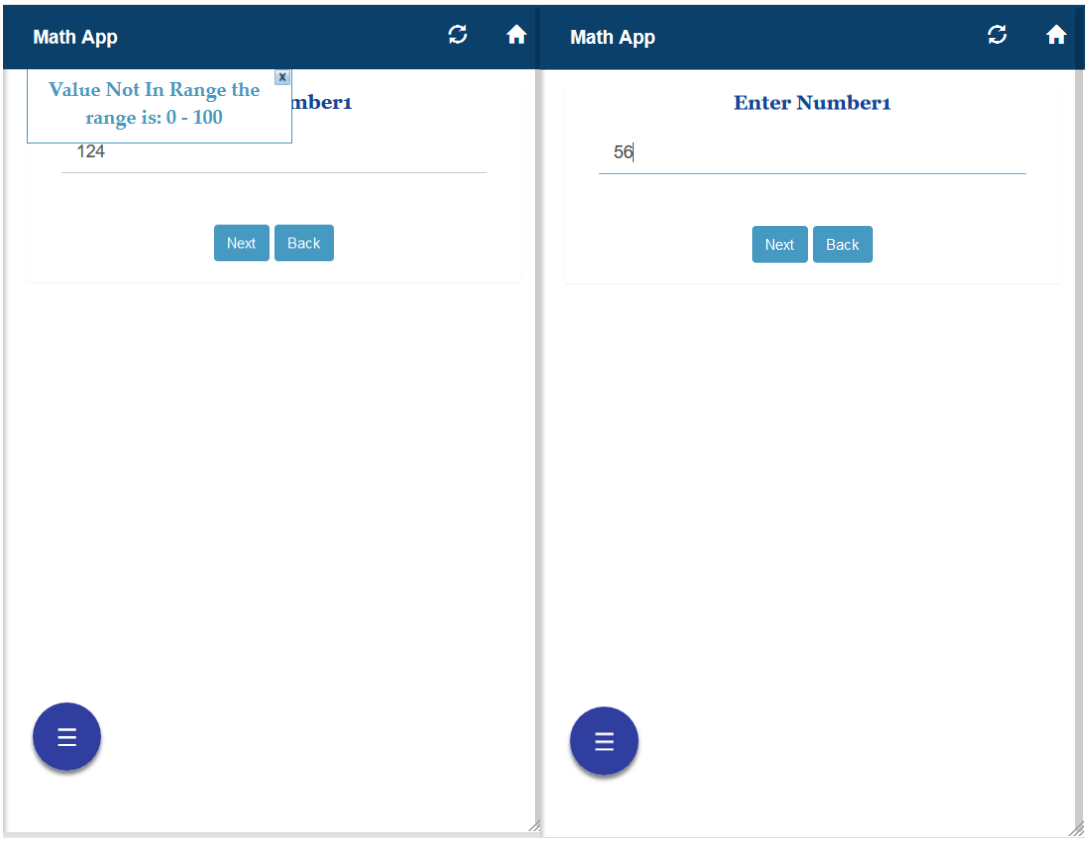


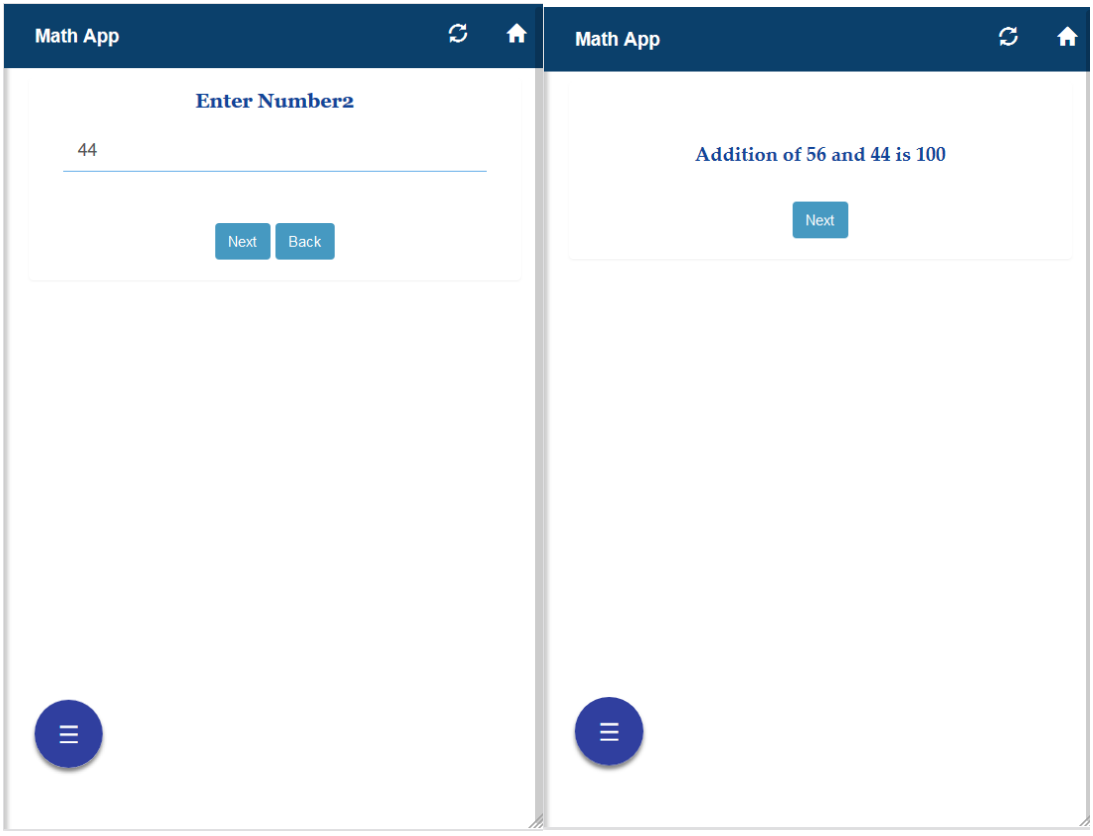
Step 4: Check Web App Execution

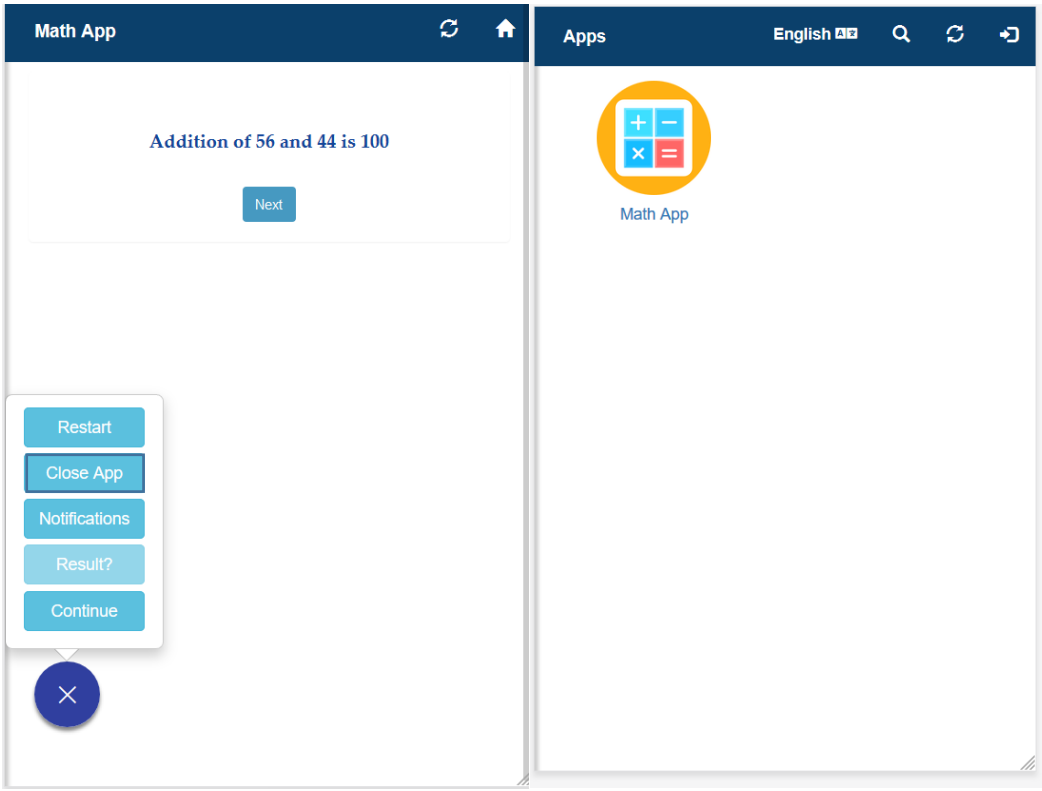


Step 5: Check Mobile App Execution

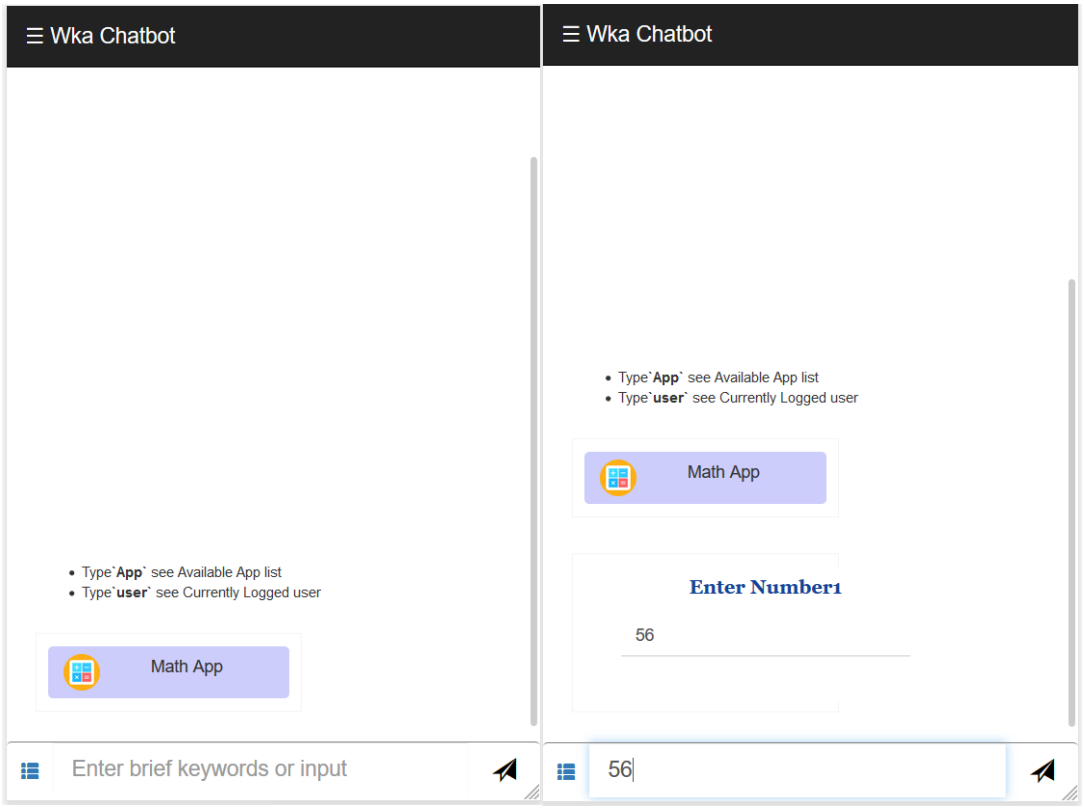


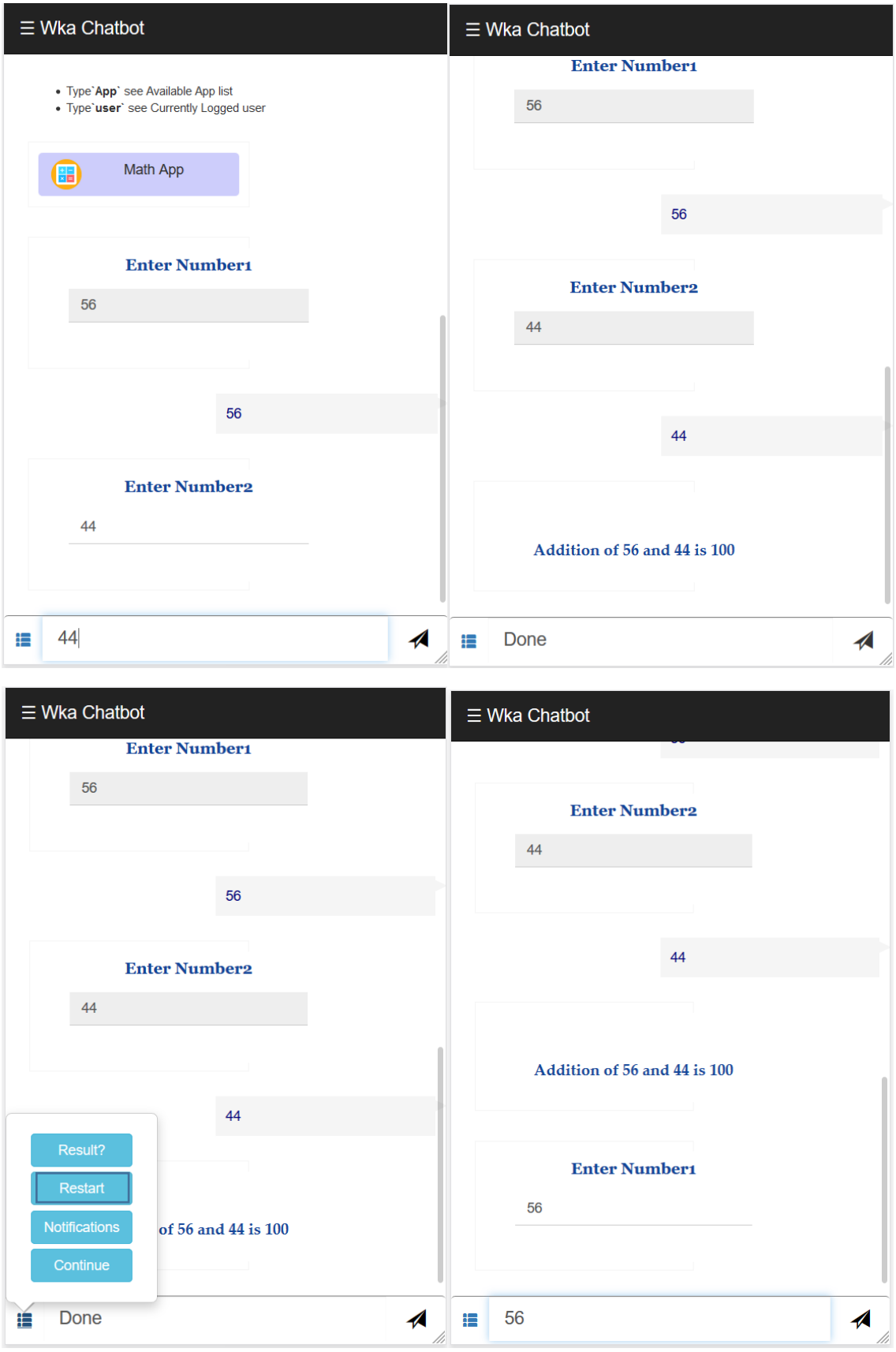

Math App
Value Not In Range the range is: 0 - 100
mber1
124
Next
Back
Math App
Enter Number1
56
Next
Back


Math App
Enter Number2
44
Next
Back
Math App
Addition of 56 and 44 is 100
Next



Step 6: Chatbot App Execution




☰ Wka Chatbot
Type`App` see Available App list
Type`user` see Currently Logged user
Math App
Enter Number1
56
56
Enter Number2
44
44
Result?
Restart
Notifications
Continue
Addition of 56 and 44 is 100
Done

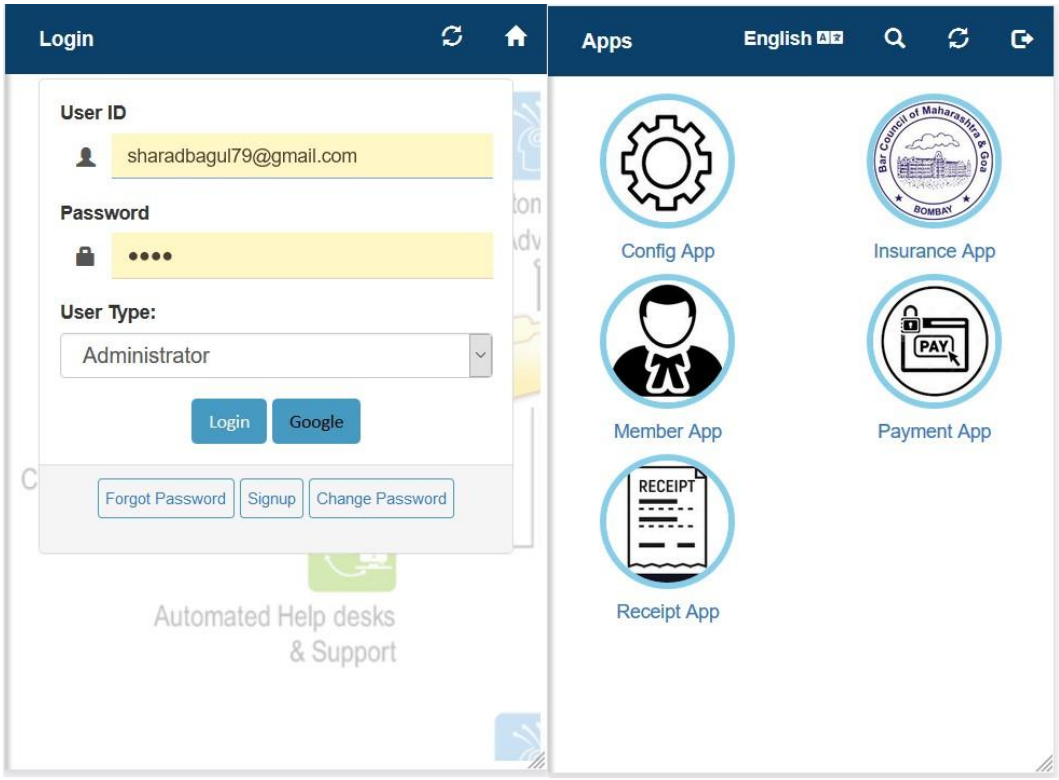
3.5 More about WKAS-based apps

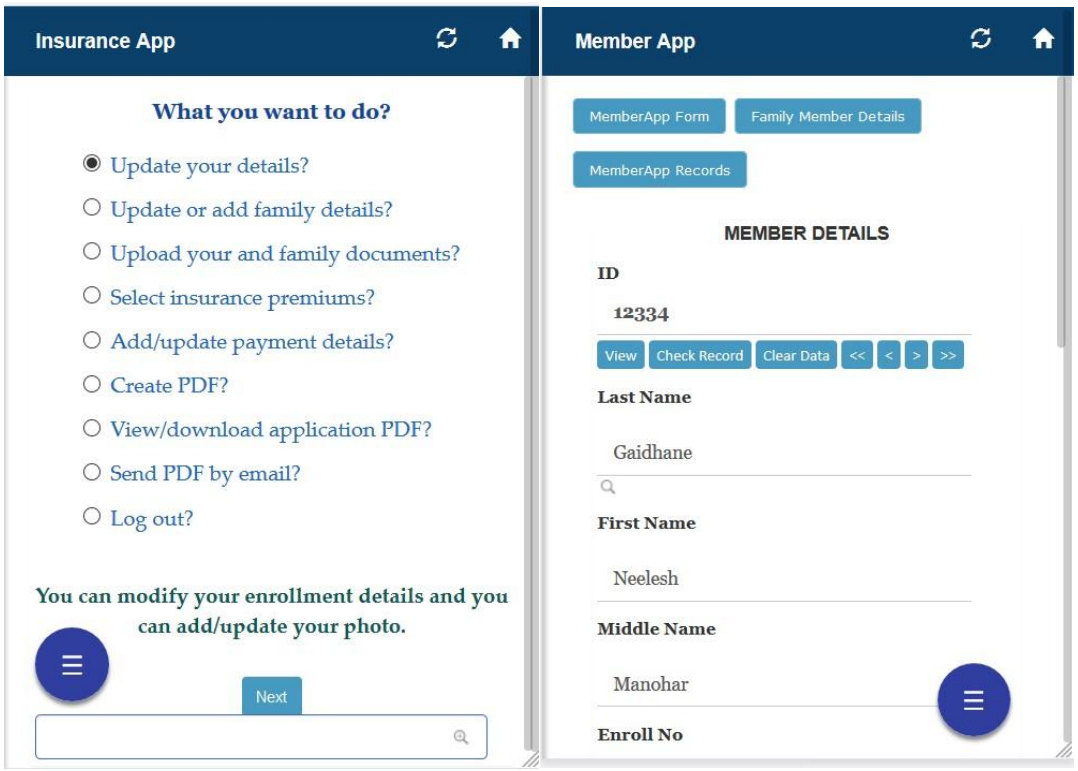
All these apps are backed by expert system approach, so they work in Q&A mode (so no main menu, sub-menu etc., like conventional apps) next form/input invoked depends upon current selection. However, if all inputs are given or loaded before app execution, app does not ask inputs from the user. All apps are created in DIY mode so can be reconfigured/modified whenever required even if they are live.

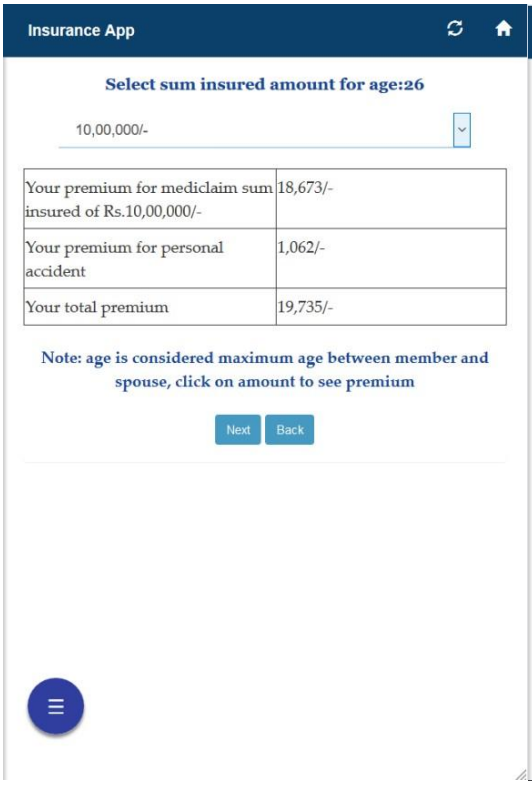
All apps use common icons and options, once user knows use of buttons/icons, the use can use any application based on WKAS. Following are screenshots of live applications.

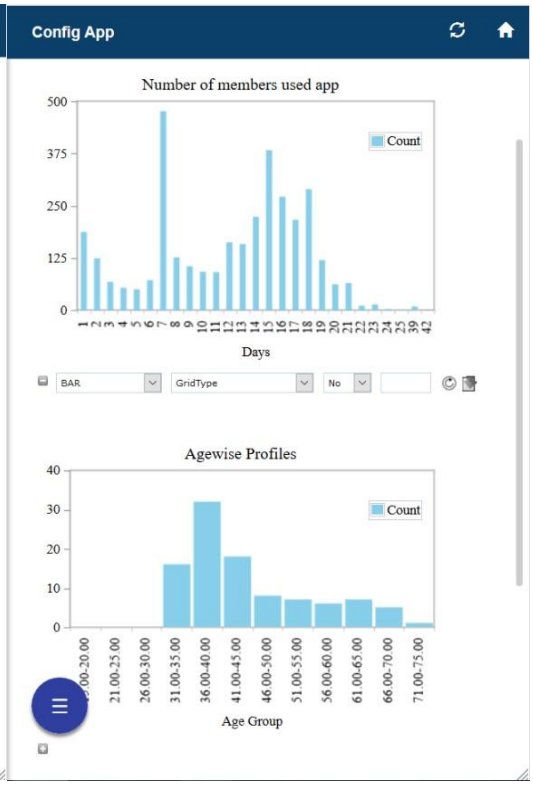
4 APPENDIX

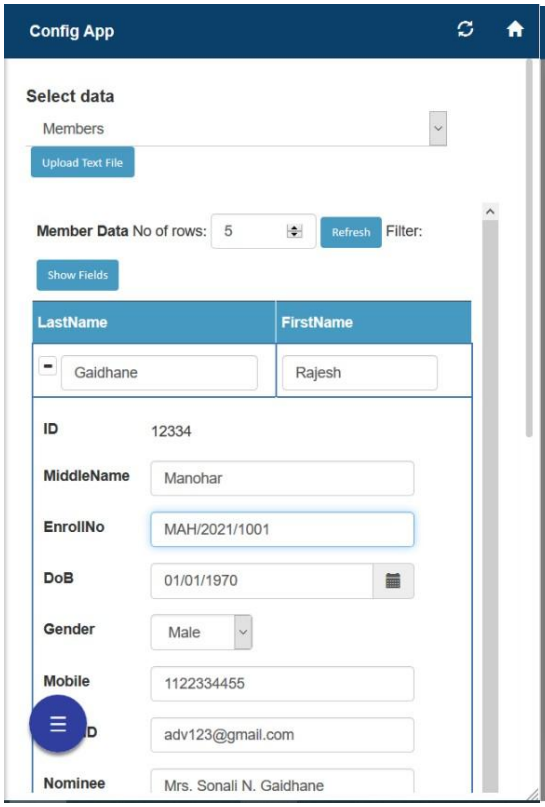
Sample screen-shots of currently live applications






Insurance App
Select sum insured amount for age:26
10,00,000/-
Your premium for mediclaim sum insured of Rs.10,00,000/- 18,673/-
Your premium for personal accident 1,062/-
Your total premium 19,735/-
Note: age is considered maximum age between member and spouse, click on amount to see premium
Next
Back


Config App
Number of members used app
Count
Days
BAR
GridType
No
Agewise Profiles
Count
Age Group


Config App
Select data
Members
Upload Text File
Member Data No of rows: 5
Refresh
Filter:
Show Fields
LastName
FirstName
Gaidhane
Rajesh
ID 12334
MiddleName Manohar
EnrollNo MAH/2021/1001
DoB 01/01/1970
Gender Male
Mobile 1122334455
adv123@gmail.com
Nominee Mrs. Sonali N. Gaidhane


Config App
Close
Members registered last year
No:1
id 12402
fullname Aage Pornima Prakash
enrollno MAH/1761/2013
Created Date 6/16/2020 10:26:49 PM
No:2
id 14736
fullname Abdul Raheed Vaseem Ahmad
enrollno MAH/2564/2006
Created Date 7/2/2020 12:06:12 PM
No:3
id 15915
fullname Abhale Adv.Uttam Ramesh
enrollno MAH/150/2006
Created Date 7/7/2020 3:09:16 PM
No:4
id 16409
fullname Abuj Yogesh Abuj
enrollno MAH/2985/2003
Created Date 7/9/2020 5:18:08 PM
No:5
id 16267
fullname ACHALIYA SHIRISH SHANTILAL
enrollno MAH/1315/1980
Created Date 7/9/2020 11:55:40 AM

Figure 4-1 Backward chaining algorithm: detailed