FinCyan Web



|  |  |  |  |
| --- | --- | --- | --- |
| S No | Date | By | Remarks |
| 0.1 | 11 November | Srikant |  |
|  |  |  |  |

Contents

[1. Introduction 4](#_Toc25256669)

[2. Suggested Architecture 4](#_Toc25256670)

[3. Development Methodology 5](#_Toc25256671)

[4. Cards - Explanation 5](#_Toc25256672)

[5. User Data Model 6](#_Toc25256673)

[6. Page flow 8](#_Toc25256674)

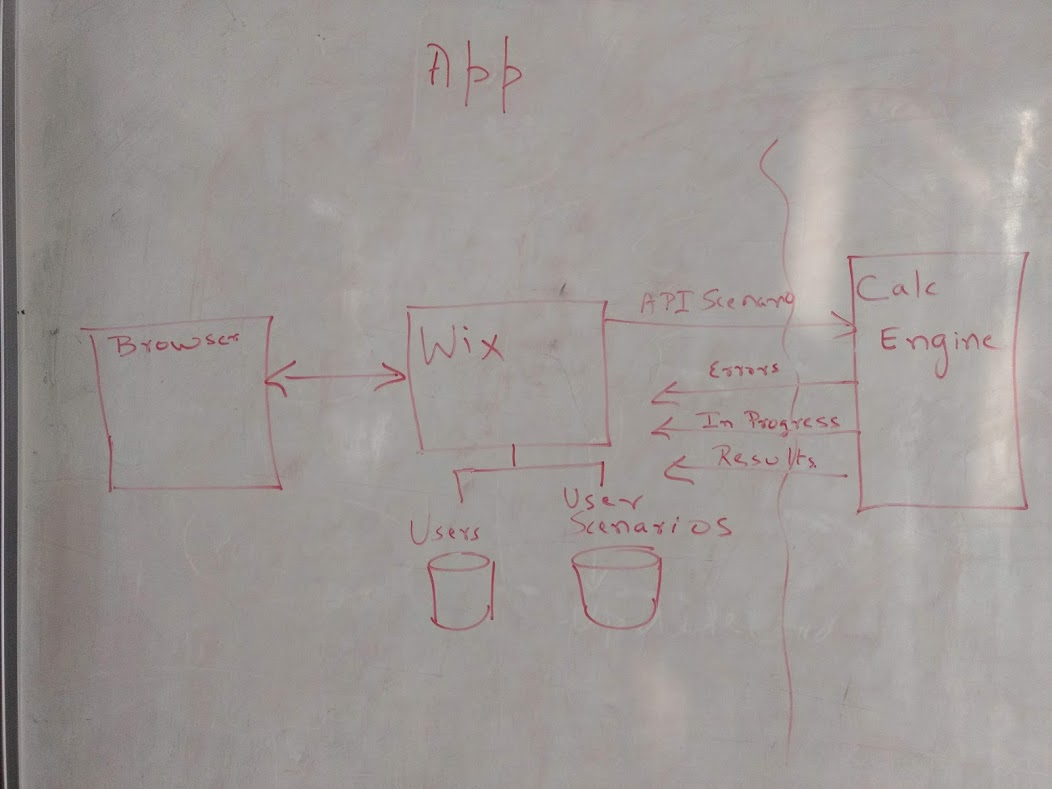
# Introduction

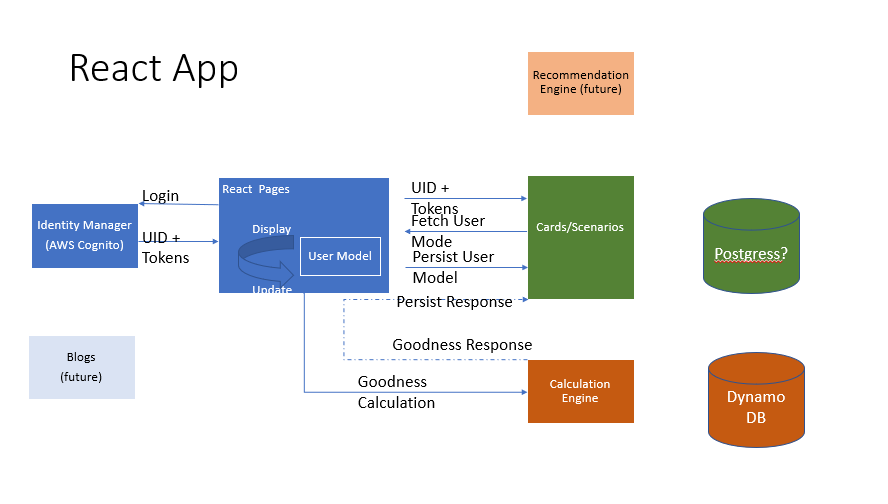
FinCyan engine produces a rather dry list of probabilities based on the user entered data. The purpose of this mail document is to details the requirements which are expected from the FrontEnd.

We have already done one round of development with Wix as a platform and the next round of development will be done using React as the framework for building the web pages. The FinCyan engine is built using Flask/AWS Lambda and Python . However, from the view point of this project (FinCyan Web) , it will only be taken as Rest API which we need to interact with

# Suggested Architecture of overall architecture

The following diagram shows the current app as it is developed. This app is hosted on the Wix platform which manages the User data. This user data consists of the User login/signup and also a picture of the users financial profile under different scenarios as a series of cards. The structure of the cards is describe in a section below. The basic flow of the app is given below



During the current phase, we will migrate the code from Wix to React. All functionalit will now be served using a React based App called WebFinC. 

The scope of this task is only this portions marked in blue

The React Application will interact with

1. AWS Cognito user pools for User login/sign up data –
2. AWS User – Scenarios service for storing the scenario data
3. Calc Engine which consumes the scenarios and gives back the results after validation

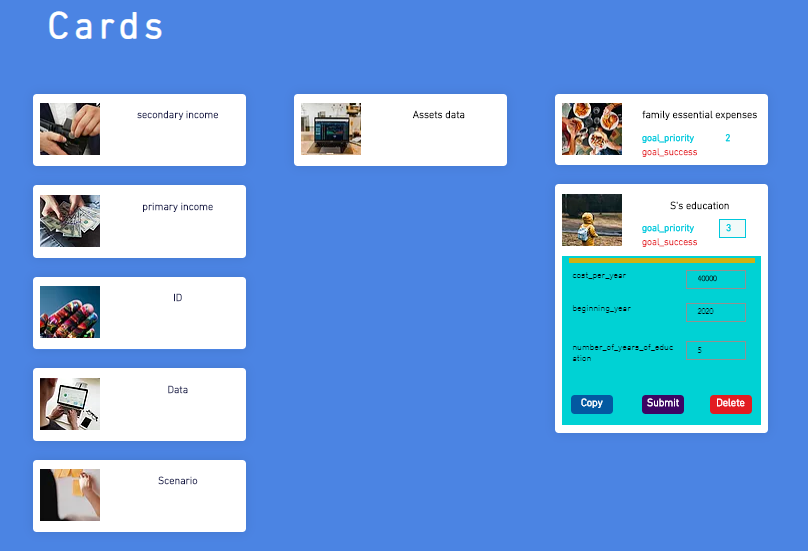
# Development Methodology

This document details the full set of functionalities which we want to develop into the future. We will use daily incremental builds to ensure that different developers can interact with each other.

1. The base version of the code is stored at <https://gitlab.com/fincyan/webfinc>
2. Version given out to freelancers is stored at <https://gitlab.com/msrik/webfinc.git>
   * Please share your git id , so that I can give access to you
3. The functionality required in the first phase is given below in the section Scope - Page flow
4. I expect the developer to come with a schedule of delivery and deliver at frequent intervals.
5. Once initial phase is done, we will integrate with the rest of the codebase and give a fresh set of documents
6. A working model is kept at <https://msrikant.wixsite.com/website-11/plan-with-fincyan>
7. The scope of this phase includes the functionality of this page
8. Sign-up and login do not form part of this phase

# Cards - Explanation

Each user has a series of cards associated with him and each card has modifiable data about one facet of a persons life. The web front end should be able to manage the state of the cards.



Each card has some common data fields, but the expanded portion of the last card gives a set of data fields which are relevant for only that card. We determine which data fields are necessary for a card by looking at its Card Type

# User Data Model

1. A user can have multiple scenarios associated with himself and each scenario can have multiple cards associated with him .
2. Each Card belongs to specific card type.
3. Multiple cards can have the same card type. Card types can also be not present
4. The engine provides the following information
   1. ISO goodness values **or**
   2. Errors

A sample of the user data model is given in the data directory of GIT. I will provide a full test version later

|  |
| --- |
| {      "version": "1.1",      "Data": {          "UserData": {              "name": "srikant",              "login": "xxx@gmail.com",              "userId": "xxx-yyy-zzzzzzz"          },          "Scenario\_data": [              {                  "Scenario": {                      "scenario\_id": 7,                      "title": "Default",                      "description": "This is the default scenation "                },                  "cards": [                      {                          "card\_type": "\_id",                          "\_id": "USR1",                          "card\_description": "User ID",                          "title":"Yourname",                          "card\_category":"profile",                          "icon":"icon.png",                          "priority":100,                          "scale":1000,                          "form":{                              "customer\_id": "USR1",                              "group\_id": "Indi"                          },                          "goal\_success":0,                          "errors":{                              "group\_id": [                                  "Value has to be India"                              ]                          }                      }                  ],                  "iso\_goodness\_value": 99.09,                  "Engine\_hash": {                      "scenarioHash":"abc-def-hij",                      "errors":  "aaa-pas-err",                      "CalcHash":"aaa-calc-resp"                    }              }          ]      }  } |

# Scope - Page flow

1. Load the user data
   1. provided the JSON is equal to variable named user Id
   2. you can hard code the user id
2. Filter the JSON to the scenario
3. Split the cards as per category
   1. Categories can be Profile
   2. Assets
   3. Goals
4. Display the following fields for each card
   1. . icon
   2. Title
5. Each card should have a mechanism to input the form details
   1. Key
   2. Value (input )
6. If scenario hash is the same as errors hash display the list of errors
   1. Pseudo code for calculating scenario hash will be provided by Srikant .
7. If scenario hash is the same as Calchash display the goal success as progress bar across the card
8. Provide an duplicate/delete function for the cards
9. If any of the form data changes (including addition/deletion of cards) , set the scenario hash to null and refresh the page
10. Provide a button for update which calls an API and displays an alert (“API updated ) when the API returns with a value. You can use any sample API for this as all we are doing is waiting for a response

# Next Steps

1. Review the workpacket
2. Provide cost estimate
3. Provide Project steps and timelines
4. Establish milestones