1. Purpose

This document outlines the background, role, user needs, and proposed features for Reporting within the FICO Platform. The purpose of this document is to provide high-level requirements; however, the detailed Epics, Stories and their associated acceptance criteria or standards required by FICO PDLC/SDLC processes are outside the scope of this document. Finally, this document does not outline any technical approaches, except in cases where there are specific standards or protocols that must be met because they are a growing or prevailing standard. Other reasons include specific asks on standards or protocols that apply to our customers and may be requested in RFPs.

2. Introduction

FICO Platform aims to serve organizations as an Applied Intelligence and Decisioning platform. Reporting in the platform is envisioned to serve the visual self-service needs based on the applied intelligence and insights generated in the platform. Specifically, it is meant to provide the user with the ability to model, explore, view, share and analyze data from different data sources in the form of quick overview, visualizations, KPIs, and metrics that can be shared with authorized users of the platform. It is a vital feature in any decision platform to have the ability to turn data into insights that inform a variety of business actions. Reporting must allow the user to create, edit, collaborate, and share information as dashboards consisting of visualizations and their underlying data, e.g., in the form of tables. These dashboards are in turn used by stakeholder personas to analyze, monitor, identify, and potentially mitigate risks by making key decisions based on the information presented.

3. Market Requirements

Reporting has its roots in providing accountability (usually written) that one has observed, heard, done, or investigated. It's use in business and decision support systems dates back several decades and has been instrumental in describing "interrelationships of presented facts in such a way as to guide action towards a desired goal". Traditionally, business reporting was built by individuals with specialized technical skills who were involved in assembling the data required for reporting, storing that data in systems, and then coding visualizations, dashboards and integrating bespoke mathematical or statistical analyses with the dashboards These steps both manual and automated were time consuming resulting in delays that hampered business and decision support. Further in businesses where there is a need for auditability, the manual activity, email trails make operations and governance a challenge. The need for streamlined "fact-based systems" to improve business decisions became the foundation of the offerings in this software category.

Reporting is a mature software category that has existed in the software market since the late 80s and 90s. What began as a category that needed to be installed, managed by technology professionals has seen marked improvements to consume data from different sources, accommodate essential mathematical and statistical measures all while enabling a greater level of self-service for its users. This move for self-service has been driven in large part by the shift in the primary and often only user(s) being analyst(s) to the inclusion of decision maker personas. For these decision maker personas , reporting is a key part of how they view, interact, and inform strategic and operational actions. Today, Reporting is embedded in the productivity suites where these users operate and derive the necessary information required for decisions across the enterprise. Especially in industries where this information powers key strategic and operational decisions, there has always been the need to audit and explain decisions. It is this kind of transparency and accountability that is driving the growing yet somewhat unaddressed need for governance to track provenance and usage of these reports across the organizations.

As such, the needs of this software category have advanced well beyond collecting data, storing that data in a store, and producing or sharing dashboards. Given the number of decision maker personas that this category counts in its install base, several providers are adding features that

1. Enable collaboration and sharing of insights
2. Improve governance, augment human insights through AI
3. Support insight driven workflows

These features provide non-technical users with self-service capabilities to enable them to achieve actions that may have previously required code. The evolution of these Reporting features is aligned with the goal for Reporting in the FICO platform – to allow users to have a high degree of self-service to deliver contextualized, outcome-aware insights - while ensuring that the checks and balances of disseminating business information are met. Today FICO uses Tableau for Reporting as part of many of its offerings. FICO and Tableau (now Salesforce) have been in negotiations and the platform needs an alternative to Tableau that can be used by users for reporting.

Data Visualisation & Integration with Charting Library

Purpose

This document outlines the component design of the Data Visualisation Layer of the Reporting Capability with the intention of developing its detailed technical design document.

Overall Design Proposal

The purpose of creating the Data Visualisation Layer is to develop a strong, reusable and organised component layer that will provide a dependable solution for displaying various charts. The data visualisation component layer, which should not be confused with the authoring layer, will receive an API response containing all the layout IDs that will contain the individual visualisations and the form data needed to construct each visualisation. This form data will include visualisation configuration information such as the visualisation title text, legend position, and type of visualisation, among others. Please note that the model is still in development.

The chart-agnostic angular component will receive the form data and call the adapter (bridge) layer with the *chart type value*. This adapter(bridge) layer will then dynamically load the chart configuration object. The chart configuration object will be filled with the form data values received from the API response and passed to the chart-agnostic angular component. This component will then request the data for this chart by making an API call.

Components

- Container Component

  The container component will be placed within each visualization layout section of the page. It will occupy the entire width and height of that visualisation section. The container component will receive the form data and layout ID as props from the API response. It will then encapsulate the chart-agnostic Angular component, which will be loaded dynamically within the container component.

- Chart Agnostic Component

  The chart-agnostic angular component serves as the UI for displaying various charts. When the container component passes the props to this component, it will call the adapter layer which returns the chart object necessary for building the chart.Creating a chart-agnostic component eliminates the need for chart-specific wrapper components. The component will be dynamically loaded by the container component.This component will make an API call to get the chart data.

- Adapter Layer

  The adapter layer is responsible for loading the visualisation configuration file and mapping the FICO-specific visualisation form data with the library chart object. Once this is done, the layer populates the library chart object with the form data and returns it to the chart agnostic component.

- Chart Model

  The chart model is the FICO centric chart properties. This properties will be mapped onto the charting library.

Design Diagram

DRAFT C3 design diagram of the data visualisation layer

HighChart Integration

The reporting feature will utilize the HighCharts library to create visual representations of data. The configuration files will call the HighChart's chart constructor using a basic object to create the chart object. This chart object will be passed to the chart agnostic component, which will render the chart. This document in this link HighChart Configuration Options has list down the API and configuration object which is used to build the chart.

Module Import

For browser side(target ES6) projects the following bundle should be used. It also provides support for typescript.

import Highcharts from 'highcharts/es-modules/masters/highcharts.src';

Other bundles for gantt,geospatial and stock are

- highcharts/highcharts-gantt
- highcharts/highmaps
- highcharts/highstock

- Created by BhaskarRay Karmakar, last modified by ShrikarGangarekalve on Sept 11, 2024

Requirement

The data visualization library will provide developers with a wide variety of options for visualizing data using a graphical interface. The user can configure visualization properties without having to know much about how the internal implementations work.

Goals

- Should adhere to the North Star architecture.
- Ensure no vendor is locking while selecting the charting library.
- The design should stick to the IRIS design guide.
- Consumer-defined visualization properties should be independent of the charting library.

Detail Design

The library consists of three integral parts.

1. **Data Visualization Chart Components (Angular)**

   Consumers can use these angular components to display the chart. The consumer needs to provide the mandatory value required for chart visualization. These components internally implement their types (typescript types, a contract that needs to be implemented), and the IDE will auto-suggest the properties during implementation. Additionally, these components will internally use the adapter component. Please refer to the table below to find the configurable properties.

2. **Adapter**

   The design is based on the adapter design pattern. Using this adapter, data visualization-defined chart properties can interact with APIs of third-party chart libraries. As a result of this design, data visualization-defined chart properties remain independent of the charting library and won't be affected by changes to the third-party/open-source library.

Additionally, this design clearly defines the scope of unit testing and allows us to exclude the charting library from unit testing.

In the current implementation, the Echart adapter component exposes a required angular signal input of type FicoVisualizations. FicoVisualizations is a union of various visualizations such as FicoAreaModel, FicoColumnModel, FicoBarModel, etc. The adapter component doesn't need to know which visualization component is calling it; it only accepts a signal input containing the visualization type and the necessary attributes to draw the visualization. Based on the visualization type, the adapter will delegate responsibility to helper methods that will dynamically load a typeScript file containing the logic to create a specific visualization. This approach also ensures a small bundle size and avoids loading the entire visualization library initially.

3. Open Source Charting Library

The data visualization charting library is internally using the ECharts library.