FSI (FM Solutions) Limited
Exchange House
180-182 St. Mary's Lane
Upminster, Essex, RM14 3BT

GO Plus Functional Requirements

Version 1.0

t: +44(0)1708 251900
e: info@fsifm.com
w: www.fsifm.com

Contents

1 Overview

GO Plus is a product development project building on top of the FSI GO platform to deliver a brand new software delivery platform.

The new platform will enable the rapid creation of new applications that can be seamlessly delivered through Web and Mobile technologies. Applications will be built using a set of graphical designers, much like the current Mobile App designer in FSI GO.

High level components include:

- Web Application Builder
- Mobile Application Builder
- Online Database Builder
- Workflow Builder.

Whilst this platform will be used to create a new suite of Facilities Management applications, the delivery will not just focus in this sector, and will support the building of a wide variety of applications for different business sectors, both enterprise and consumer facing.

The purpose of this document is to detail the requirements of this platform as well as to outline the functionality that the platform will be capable of delivering.

2 Business Drivers

There are a number of key business drivers that are required to be addressed by the delivery of the new product, these are defined below.

2.1 Platform

GO Plus will be delivered as a platform that comprises of features and functionality that enable the rapid delivery of business applications for both Web and Mobile

The platform is to be built using industry standard technologies, and intellectual property is to be fully owned by FSI. Third party controls and tools may be used; however there will be no on-going runtime licence costs associated with any future sale of the product, unless explicitly authorised by FSI.

2.2 Dynamic rich web applications

The new product is to provide features and functionality that enable new applications to be delivered quickly, and with rich dynamic functionality.

Applications are to be accessible through any web browser, using industry standard technologies.

The user interface is to have a modern look and feel, with a dynamic layout that adapts to different form factors such as phones, tablets and more traditional devices.

2.3 Advanced and configurable business logic

The new product is to provide features and functionality that enable business logic to be created and edited via graphical designers and workflow technologies.

Business logic can be consumed by the platform designer's e.g. new web and mobile applications, as well as providing an integration layer with in-process and scheduled capabilities.

2.4 Cloud Integration

By exposing the configurable business logic capabilities as Web Services, the option of Cloud Integration is also realised. This will enable systems to consume the platform features and integrate systems together, even if the applications are not delivered through the GO platform. For example SAP to Oracle integration.

2.5 Dynamic and rich mobile applications

The new product is to provide features and functionality that enable Mobile applications to be created and deployed to multiple mobile platforms, including Apple IOS, Android, Windows Phone, Windows 8 tablets.

2.6 Flexible and Scalable Database Schema

Building on top of the FSI GO Data Objects functionality, a fully flexible data schema is required, and is to be easily designed through a graphical designer. The database schema is to be fully multi tenanted and support SaaS deployments.

The data schema is to be built using Microsoft SQL Server and support scalable deployments and offer enterprise level performance.

2.7 Cross module features

Each of the four key modules – Web Applications, Mobile Applications, Workflow and Integration, are to share data and interact seamlessly with each other.

For example, it must be possible to create a Web application that has all business logic built using Workflow, extended out to one or more Mobile applications and have integration to third party systems through the Integration API.

2.8 Platform Commercialisation

The completed platform is to be designed as a commercial software product, with the option of delivering a Platform as a Service (PaaS), technology licencing or white labelling. A fully functional licencing model is required to lock features and functionality, limit data usage, data storage and have support for time controlled trials.

To fully enable the commercialisation of the platform, there will be no specific line of business features included in the platform. The platform itself will provide all of the building blocks to deliver line of business applications.

2.9 Remove Technical Debt

The new platform is to have no dependencies on legacy FSI products (excluding FSI GO); this includes code base, database schema and any other installation tools that could compromise design and thinking.

Note: There will be no upgrade path from Concept applications such as Evolution, C500 or Workflow.

2.10 Partnership Value Proposition

The new product, including both the platform and applications built on the platform, are to provide a suite of products that are flexible and configurable.

This is essential to create a value proposition for the FSI partnership channel that enables partners to build a successful business around the FSI product suite, creating IP and delivering customer value through application delivery and customisation.

2.11 Build Services Business

The new product suite is to provide a set of designer tools that will enable the customisation of applications and business logic. The tools will mask the complexities of building web and mobile applications, enabling technical consultants to deliver change requests and new line of business applications.

It is a requirement that a technical power user can build rich, engaging web and mobile applications using the platform tools and designers, and that a developer is not required unless advanced highly custom features are required.

The platform is to be extensible and enable partners to build custom modules and functionality to further embed the platform within their business.

3 Key Features

3.1 Enterprise Feature Support

The platform is to support enterprise and globalisation features out of the box. This includes multiple currency, multiple time zone features, as well as multi language, Right to Left support and translatable user interfaces.

These features apply to both the Platform and any designed application built on top of the platform.

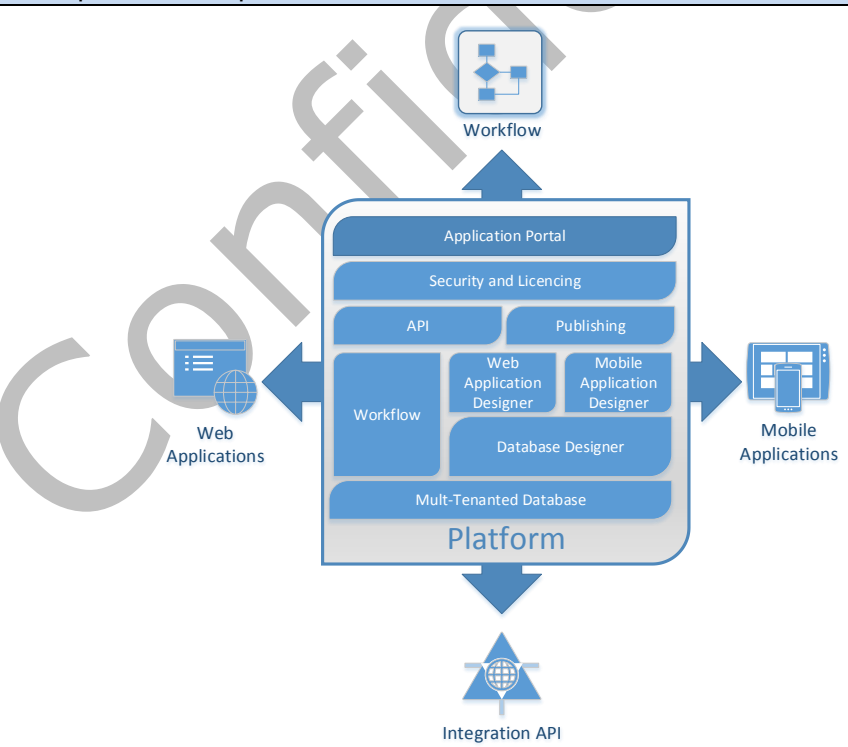
3.2 Scalable

The platform and applications built on the platform are to be fully scalable from small to enterprise deployments. The platform is to be deployable both to the public cloud and private cloud for local client deployment.

3.3 High Level Architecture

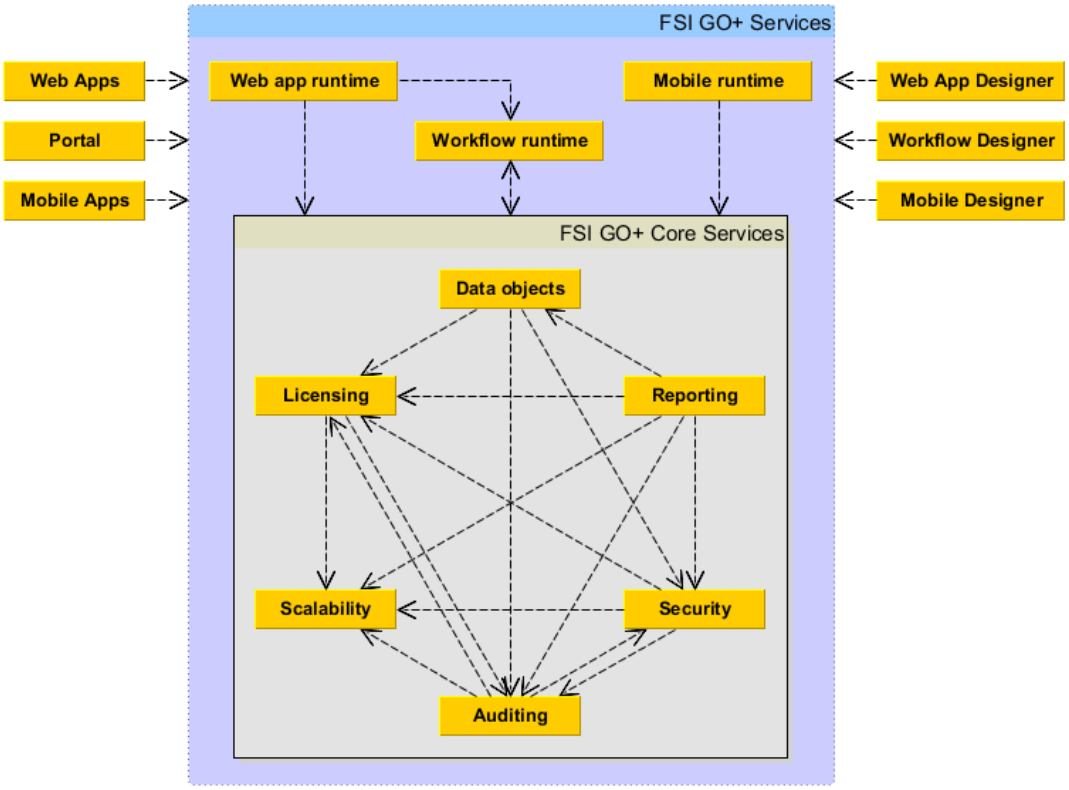
The below diagram depicts a possible high level architecture showing the key components of the platform.

Note: The final configuration of modules and components will be defined in the detailed specification phase.



3.4 High Level Dependency Diagram

The below diagram depicts key subsystems of the platform and their dependencies. All of the subsystems have internal or web service API. This diagram is referred to later to show which parts of the platform are affected by a specific feature.



3.5 General User Experience

The platform designers and importantly any applications built using the platform need to conform to the following user experience principles:

- Touch First Design – Web pages for both Management portal and published applications is to have a "Touch First" design - this is essential to supporting multiple devices and form factors especially tablet devices which utilise swiping and other screen gestures
- Stylus and Mouse Support – Although Touch first design is key, many users will interact using Stylus or Mouse
- Multi Browser – Desktop, Laptop, Tablet & Mobile
- Applications are to be responsive with web pages typically rendering in sub second, and up to a maximum of 2 seconds.

3.6 Ecosystem

It is important to note that there are no design limitations that makes either the platform or resulting applications built on top of the platform more suitable for one platform over another.
The ecosystem is to work seamlessly across desktop smartphone, tablets etc.

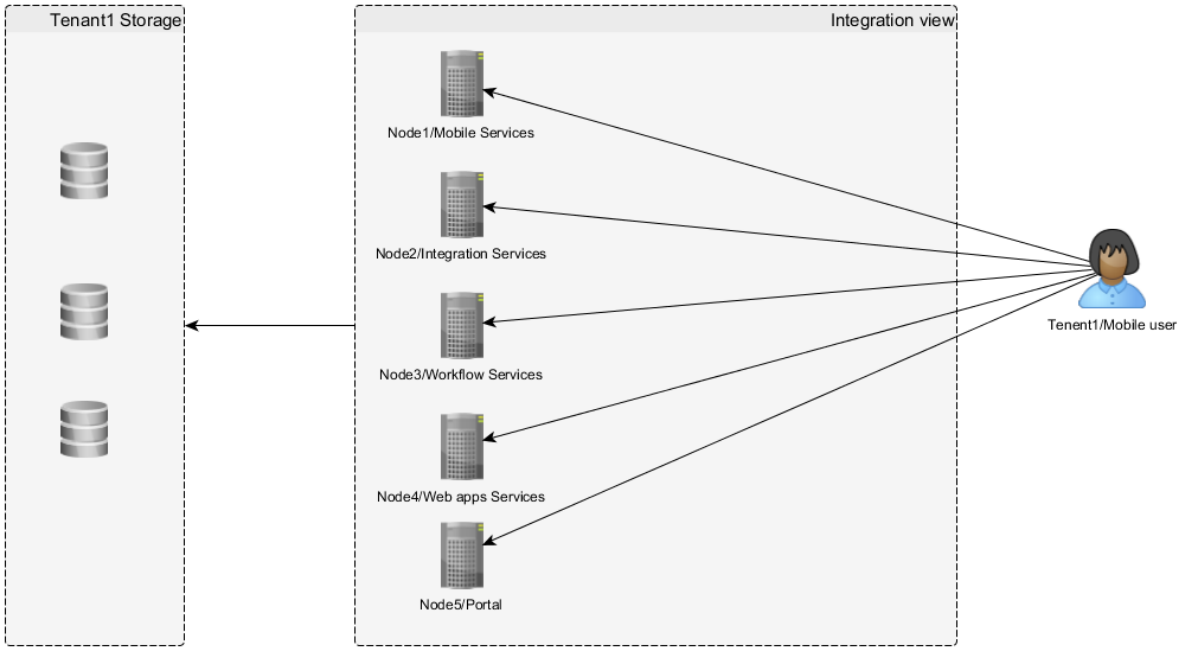
3.7 Performance and Scalability

The finished product is to support full scalability options and be responsive in all areas of execution. This includes:

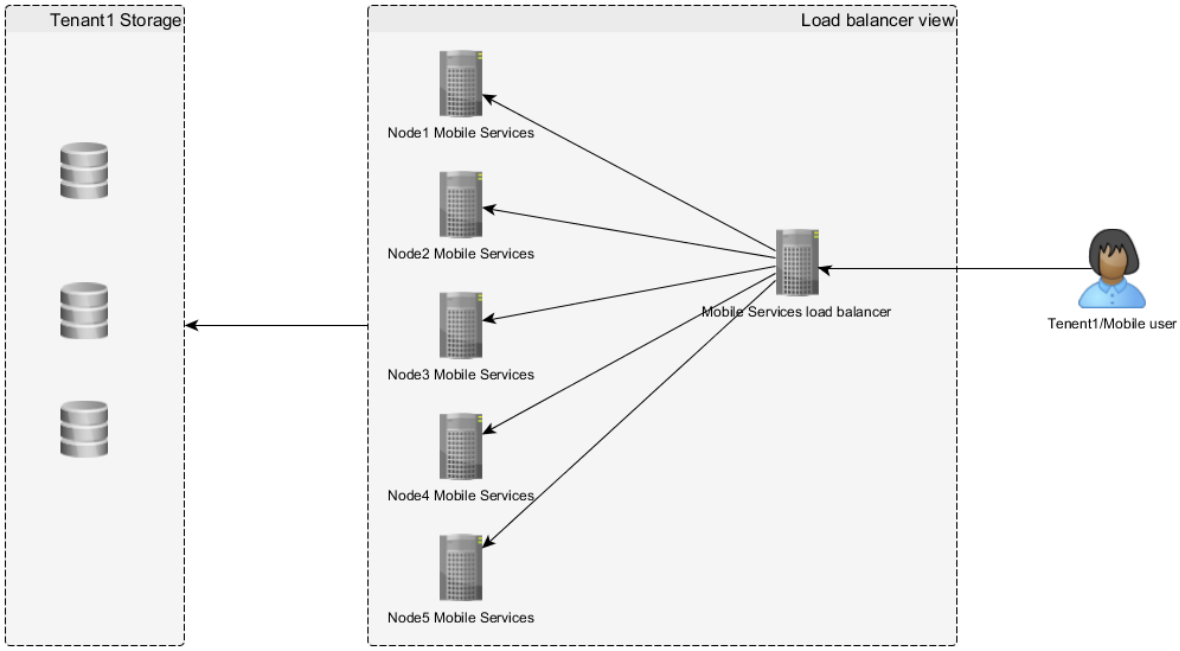
- Load Balancing for all IIS hosted components
- Web Farm support
- Load balancing for non IIS hosted components e.g. windows services
- Support for Database mirroring and clusters

Scalability of *REE (Runtime Execution Environment)* is the ability to create such a configuration that each of the roles available for tenant is arranged on any node. This means that tenant could have five different nodes and each of node could have separate role:

- Mobile services
- Workflow services
- Web apps
- Integration services
- Portal



Definition of REE could also contain load balancers which are associated with any number of nodes in the context of their role. Load balancers act as nodes responsibility. This means that the definition of the REE can be theoretically spans a large number of physical nodes.



3.8 Runtime Execution Environments

The runtime execution environments provide the infrastructure and logical separation to execute Web Apps, Workflow and Mobile Services.

For each tenant it is to be possible to configure and assign which Runtime Execution Environments are allocated. This includes:

- Web App Production
- Workflow
- Mobile Services

Execution environments are defined in detail later in this document.

All portal activities including the designers, notifications and publishing will be hosted on a single default server or farm environment.

Test environment for web, workflow, and mobile will be realized as a separate tenant. App definitions will be transferred between environments using "Import and Export" feature. Export and import will have tenant's global configuration excluded to not overwrite test configuration with production configuration or vice versa.

Test tenant licensing model is the same as for production tenant.

3.9 Security

All access is to be secured by a minimum of a user name and password. Other required features include:

- Minimum and Maximum Password length (per tenant)
- Password complexity – configurable per tenant
- All passwords are to be encrypted
- Portal Single sign on support (such as Active Directory, federation services & oAuth)
- Active directory user sync
- Full permission sub system – every action is to be securable for both design and run time
- Web Services are to be fully secured
- Workflow execution is to be secured
- Extendible security model – custom permissions for apps
- User and User Role support
    - With permission management

3.9.1 Platform Security

It is a requirement that all components of the platform are secured against known attacks such as:

- Cross Site Scripting (XSS)
- SQL Injection
- Denial of Service
- Cookie Stealing

3.9.2 Cookies

It is a requirement that no sensitive information is stored and persisted in a cookie, and that cookie stealing is fully prevented.

3.9.3 SSL

Network traffic is to be protected using SSL certificates.

3.10 Auditing

A full audit is required to record all actions that relate to

- Changes in security such as setting permissions or changing passwords
- Changes in data – this is to be optionally setup through the Data Object designer. For example setting which fields are required to be audited.
- User activities (e.g. log in, log out etc.)

Audits will apply to both the master tenant and client tenants.

Access to the audits will be via permission, and audits will record before and after data values.

3.10.1 Audit Viewer

It is a requirement that a user with the correct permission can view the recorded audits for each data object. The description of audits is to be recorded in clear English by default. For example:

"DateOfBirth changed from 1/1/1970 to 1/2/1970 by Jon Smith"

3.11 Licencing

Platform features will be licenced per tenant, and each licence will be verified by the FSI licensing server, as it is currently done so in FSI GO.

On top of the current licencing options the following additional licences are required:

- Web User
  - Named user
  - Concurrent user
  - Guest Concurrent user
- Web Designer
  - Named user
- Number of Web Applications
  - Maximum number of web applications that can be created for the tenant. The software will restrict this from being exceeded
- Pages per Web Application
  - Maximum number of pages that can be created within a single Web application
- Data Objects
  - Maximum number of data objects that can be created within the tenant
- Data Object Fields
  - Maximum number of user definable fields per data object
- Number of Mobile Activities
  - Maximum number of Mobile activities that can be created for the tenant. The software will restrict this from being exceeded
- Forms per Mobile Activity
  - Maximum number of forms that can be created within a single Mobile activity
- Number of Integration API Calls (per 24 hours)
  - Licenced number of API calls allowed within a 24 hour period (midnight to midnight UTC). If exceeded the system will notify the tenant administrator but **not** block access.
- Data Usage Threshold
  - Total storage consumed by the tenant, including application data, documents, audit logs, workflow trace logs. If exceeded the system will notify the tenant administrator but **not** block access.

Workflow Specific

- Workflow Node
    - Ability to licence workflow nodes individually. Each node is licensed separately for each tenant. Only licensed nodes are available in the Graphical Workflow Designer. Attempt to save workflow definition using unlicensed node will result in an error. Revoking workflow node license stops any running instances which are using the node and prevent any such workflow definitions to spawn new instances.
- Workflow Licencing
    - Ability to licence specific workflows to stop workflows from being copied from system to system.

3.12 Metering and Monitoring

As the platform is multi-tenanted and primarily hosted as a cloud solution, the product usage needs to be metered and monitored against the available licences for each tenant.

Some of the metering will be fairly static such as number of users and data objects, however it is important to record usage for other licensable options such as API calls.

The master tenant is to have the ability to view all metered data against each tenant, such as:

- Number of static licences consumed (users, features etc.)
- Workflow execution
    - Current and historic workflow execution to show peak usage times within a day/week/month
- Number of integration API calls made per day
    - It is to be possible to trend this data to show peak usage times within a day/week/month
- Dynamic+ data storage by Data/Documents/Images
    - It is to be possible to trend this data to show peak usage times within a day/week/month

The usage information is also to be available to the tenant administrators

3.13 Workflow Health Monitoring

It is important that all workflow executions are monitored and can be reported on through each tenant. Information is to include:

- Currently Executing Workflows
- Number of Items in each execution queue
- List any failed workflows
- Retrieve information regarding workflow errors
- Information is to be recorded so that trend analysis is possible. For example:
  - Peak usage
  - Most executed workflows
  - Longest running Workflows

3.14 Use case scenarios description

3.14.1 Display list of running workflows instances

1. **Tenant administrator** selects *Workflow Health Monitoring* option from portal's menu.
2. **System** shows a grid of currently running workflow instances in tenant's Runtime Execution Environment. Columns: *Workflow definition name, Workflow definition version*, *Workflow user*, *Start date*, *Last activity date*, *Number of errors*, *Queue*, *View execution history*, *View state*.
3. **System** shows a grid of failed workflow instances from the last X hours.
4. **System** shows a chart for trend analysis of number of running instances (average running time, maximum running time) in last X days with a resolution of Y minutes.
5. **System** shows summary of last X days for top Y workflows definitions. Columns: *Peak executions per day*, *Total number of executions*, *Longest running instance*, *Peak errors per day*, *Total number of errors*.
6. **Tenant administrator** orders the table results by any column of the set: *Workflow definition*, *Workflow user*, *Start date*, *Last activity date*.
7. **Tenant administrator** adjusts chart settings: *From Date*, *To Date.*
8. **System** updates

3.14.1.1 Pre-conditions

1. **Tenant** has workflow subsystem licensed.
2. **Tenant administrator** has successfully authenticated against the portal.

3.14.1.2 Post-conditions

None.

3.14.2 Display execution history of a running workflow instance

Tenant administrator selects an instance and clicks *Execution history* button.

System displays *Started On*, *Last Executed On*, *Next Execution On*. If tracing is enabled for workflow definition, then *View Trace* link is available, otherwise.

3.15 Use case scenarios description

3.15.1 Display Deployment Configuration Summary

1. **Master tenant administrator** selects if the results should be grouped by load balancing machines or not.
2. **System** presents a grid grouped by machine names showing rows with: service name (*Mobile, Workflow*, etc.), tenant name and licensing information. If the grouping by load balancers was not selected then every machine name is post fixed with "load balanced by ..." information.

Pre-conditions

1. **Master tenant administrator** successfully authenticated against the portal.
2. **Master tenant administrator** navigated to *Deployment Configuration Summary* view.

3.16 SaaS and Cloud

The solution is to be multi-tenanted to support a single-instance SaaS solution. Data is to be fully segregated to ensure data integrity between tenants.

3.17 Codeless Design

The platform is to provide as much functionality through graphical designers enabling "Codeless Design" capabilities to deliver fully custom Web, Mobile and Business logic systems.

It is accepted that some advanced functionality will need to be delivered through code, however this will only be considered if the graphical designer does not lend itself to achieving the function through property configuration.

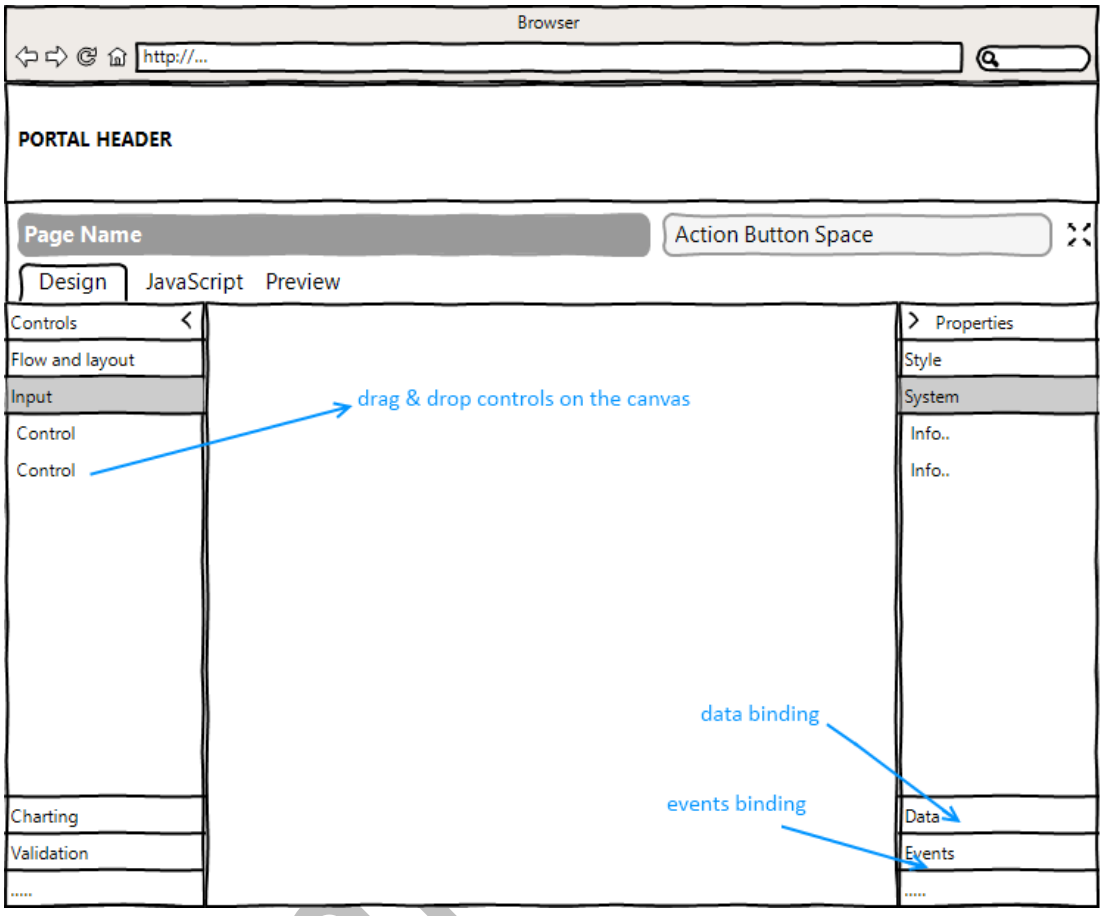
Features that are built into the designer will also be accessible through code. For example, hiding a control from the user must be a settings in the designer but also available through JavaScript (JavaScript tab).

A Web App is created using a graphical designer that has the following functionalities to support "codeless design":

- Support drag and drop of controls on the design canvas
- Support control data binding to Data Objects
- Support control to control data (e.g. cascading combo-boxes)
- Support control binding to Web Services

- Support events – JavaScript and Workflow binding

The image below shows a possible layout for the designer with mentioned capabilities.

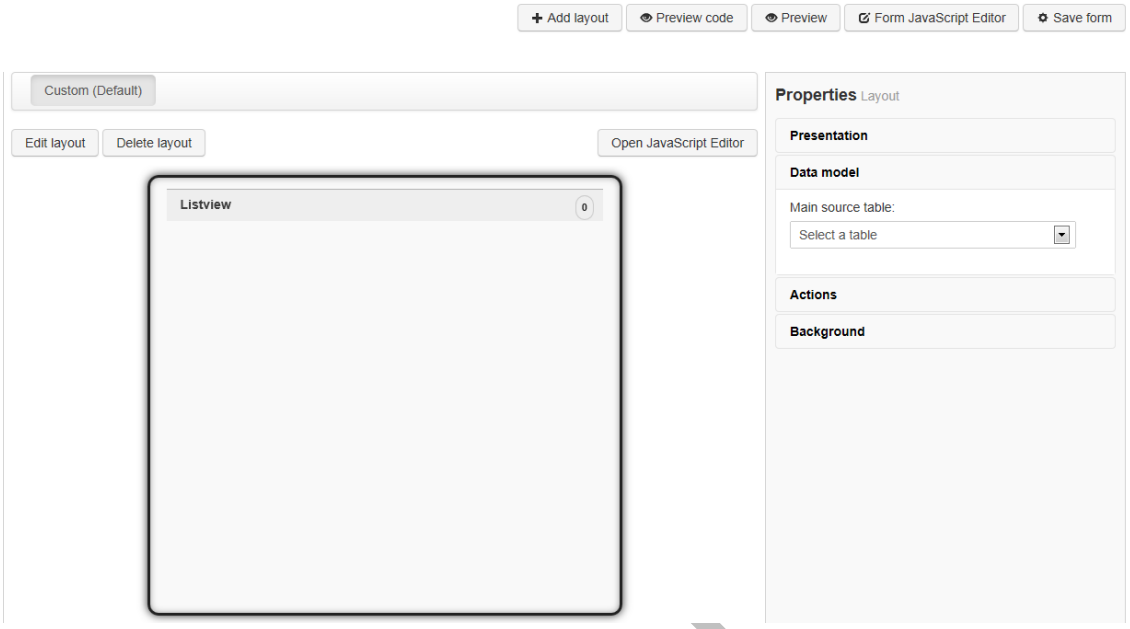


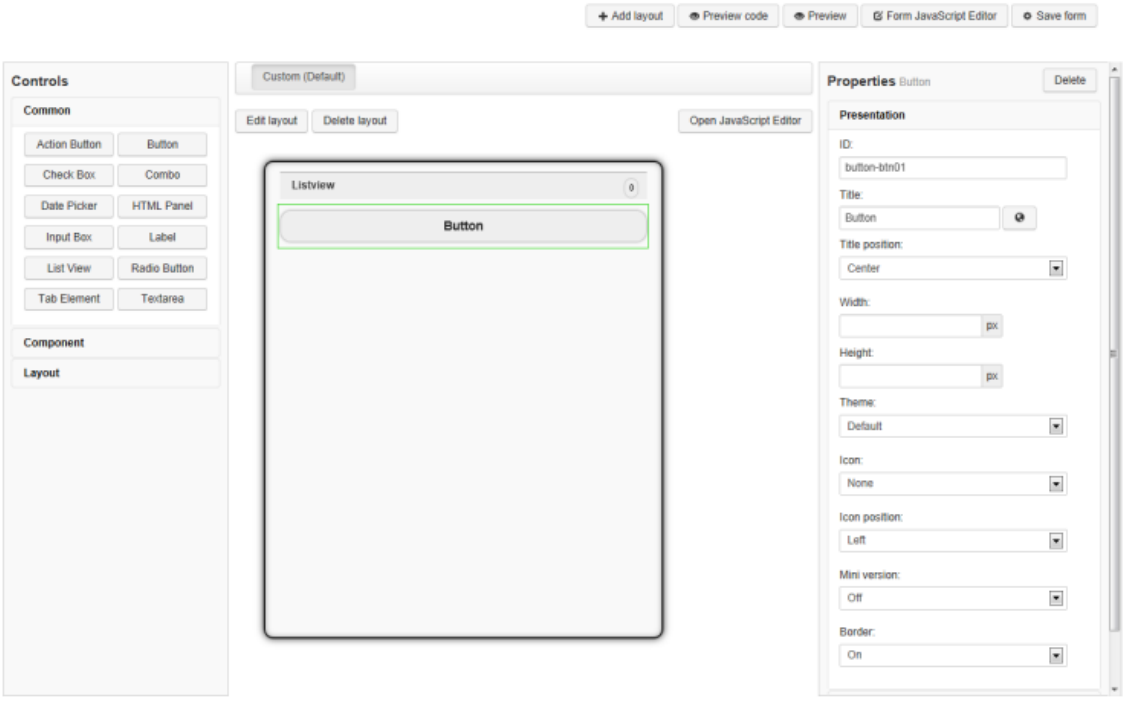
When **Web App designer** selects an object on the canvas, the *Properties View* shows different control-specific properties.

Properties can be broken down into a few groups, for example:

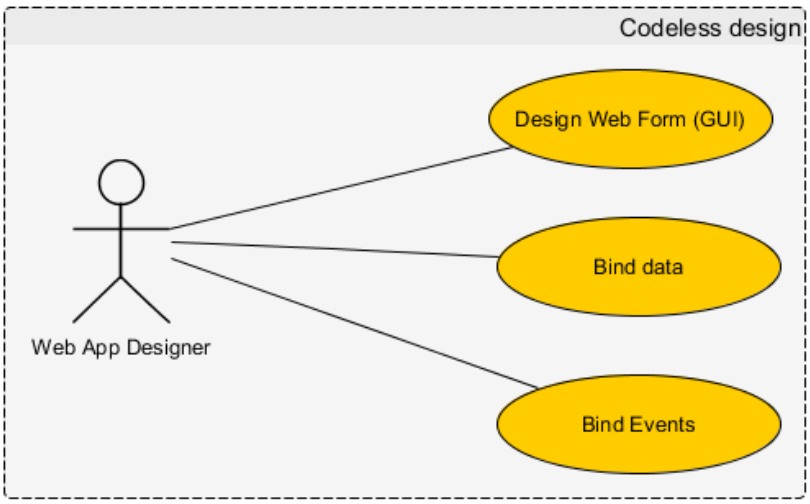
- Presentation
- Data model
- Actions
- Background

Figures below show screens from FSI GO Mobile App designer, shown only for illustration purposes.





3.17.1 Use case diagram



3.17.2 Use case scenarios description

3.17.2.1 Design Web Form (GUI).

Basic flow of events: design new application

1. **Web App designer** creates new application in the Web App designer tool
2. **Web App designer** creates new page
3. **Web App designer** designs web page by simply dragging and dropping various elements (controls)
4. **Web App designer** adjusts page layout using mouse and setting properties of controls (e.g. width, height, margin etc.)
5. Use case ends

Alternate flow: existing application, new page

A.1. **Web App designer** opens an existing application

Alternate flow: change existing page

B.1. **Web App designer** opens existing application
B.2. **Web App designer** opens existing page

Pre-conditions

- **Tenant** has license to use the Web App designer tool
- **Tenant** is logged to the Portal

Post-condition

- Web page has been created.

3.17.2.2 Bind data.

3.17.2.3 Bind control to data source

Basic flow of events:

1. **Web App designer** selects control placed on the web page
2. If data source exists **Web App designer** selects data source
3. If data source doesn't exist **Web App designer** creates new data source
4. **Web App designer** binds control to data source by setting *data properties* of control
5. Use case ends

Pre-conditions

- **Tenant** has license to use the Web App designer tool
- **Tenant** is logged to the Portal

Post-condition

- Control is bound to data sources.

3.17.2.4 Control to control binding

Use case describes control filtering. E.g. value of one combo can be used as a filter parameter on another combo

3.17.2.5 Basic flow of events:

1. **Web App designer** selects control placed on the web page
2. **Web App designer** selects control that represents *data source* from the list of controls
3. **Web App designer** binds control to control by setting *data properties* of control
4. Use case ends

Pre-conditions

- **Tenant** has license to use the Web App designer tool
- **Tenant** is logged to the Portal

Post-condition

- Control is bound to other control.

3.17.2.6 Bind control to Web Service

Basic flow of events:

1. **Web App designer** selects control placed on the web page
2. **Web App designer** clicks *Add New Data Source* and types the URL address or selects web service method from the list of available web services and sets parameters if needed
3. Use case ends

Pre-conditions

- **Tenant** has license to use the Web App designer tool
- **Tenant** is logged to the Portal
- Web services list is available

Post-condition

- Control is bound to web service

3.17.2.7 Bind Events

Basic flow of events:

1. **Web App designer** selects control placed on the web page
2. **Web App designer** selects control's event (e.g. *On Click* for button; *On Change* for Combo-box etc.)
3. **Web App designer** binds control's event to action from the list of available actions
4. Use case ends

Pre-conditions

- **Tenant** has license to use the Web App Designer tool
- **Tenant** is logged to the Portal

Post-condition

- Control is bound to events

3.18 FSI GO Migration

It is a requirement that mobile activities and data objects that are built using the FSI GO platform can be migrated to the new GO Platform. It is a requirement that functionality is built to support this migration, although this feature does not necessarily need to reside inside the platform and can be a standalone feature.

3.19 3rd Party Controls/Components

It is acceptable to use 3rd party controls or components to deliver the platform, however there are some key criteria that must be met:

- No runtime licencing costs
- Open source is acceptable, however final acceptance on usage is to be agreed by FSI
- Web controls need to be cross platform HTML5

3.20 Browsers

For the web applications created, all modern browsers are to be supported across multiple form factors. The following is a list of current browsers, however this list may be subject to change on commencement of the development project:

Windows 7

- IE 8, 9, Chrome Latest, Firefox Latest

Windows 8.1

- IE 10, 11, Chrome Latest, Firefox Latest

MAC

- Safari Latest, Chrome Latest

IOS

- Native Browser, Chrome

Android

- Native Browser, Chrome

Windows Phone

- Native Browser

Windows RT

- Native Browser

Platform Features

The GO platform is to be extended to incorporate the new components defined bellow.

4 Web App

Building web applications is a core feature and requires advanced designer capabilities to allow users to create rich and immersive experience. Each web project is to be known as a Web App and a tenant can have any number of web apps up to their licence restrictions.

A Web App is created using a graphical designer and has functionality to bind user controls to data objects, as well as consume web services and other services such as workflow.

4.1 Web App Designer

Web app designer's goal is to be able to build complete applications *(Web Activities)* using tools provided by the management portal. Web App Designer is an application that provides a rich design experience with full drag drop capabilities. The designer is to be intuitive to use, and the user should be able to create a simple app with zero "coding" through a drag and drop interface.

The Web App Designer is integrated (like present Mobile Designer from FSI Go) within the management portal and has its own address like http://portal/[tenant]/webappdesigner. Designer is build on JavaScript using popular frameworks like Backbone, jQuery, Require, Lodash. Basic interface is provided by Twitter Bootstrap.

Designer saves Web Apps in a database and at the same time deploys the Web App to the local Portal server so that the user can perform a live preview. All pages created by the Web App Designer are stored as templates, which will be compatible with JavaScript frameworks like Backbone or Ember.

The designer is to have the following capabilities:

- Support "Codeless Design"
- Support drag and drop of controls on the design canvas
- Support control data binding to Data Objects
- Support control binding to Web Services
- Support control binding to Workflow
- Support JavaScript and other libraries such as JQuery
- Support events – JavaScript and Workflow binding
- Design canvas will be as large as possible with control panels and property panels docked with the ability to hide/show/pin.

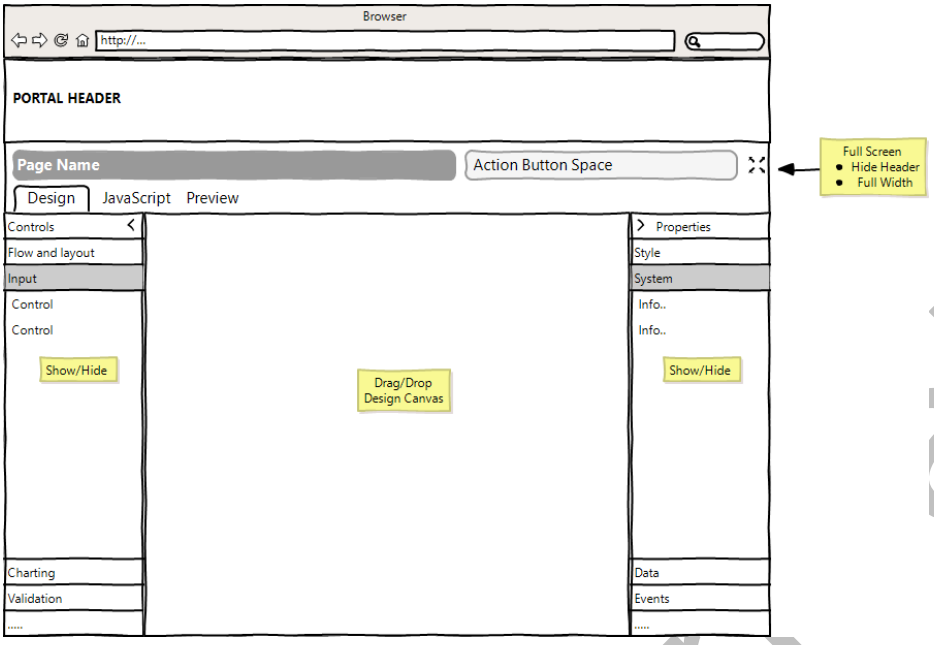
- Support full screen mode for page designing.
- Support App creation, renaming and Page renaming
- Support the importing of an App logo
- Support custom and extendable permissions
- Support Responsive Design

The designer is divided into separate elements:

- Web App Manager - place where **Web App Designer** can see list of *Web Apps* for current tenant displayed as a grid (Web App ID, Web App Name, Description and Latest Version/Number of Versions). Depending on user permissions **Web App Designer** can add, rename, delete and manage versions. Also gives ability to import logo to selected *Web App*.
- Version Manager – place where **Web App Designer** can see all available versions of current Web App displayed as grid (Version Number, Version Name, Create Date, Archival and Publish Status). Depending on user permissions **Web App Designer** can design, rename, delete, duplicate, import, export, publish, and see the publish history for selected version.
- Page Manager – place where **Web App Designer** can see all pages saved for selected version of *Web App*. Depending on user permissions **Web App Designer** can create, duplicate, design, rename and delete selected page.
- Menu Designer – place where **Web App Designer** can build the navigation that will be used at runtime to open and work with the web pages.
- Page Designer – is the most interactive part of the designer components, and requires full drag & drop functionality. Gives ability to build complete page, bind data object to controls, bind custom JavaScript actions and bind workflow runtimes to any element with proper behavior.

Live Preview – place where **Web App Designer** can run selected *Web App* and test its functionality.

The following image shows a possible layout for the designer page.



4.1.1 Use case scenarios description

4.1.1.1 Ability to add new Web App

Steps:

1. **Web App Designer** selects *Web Apps* option from portal's menu.
2. **System** shows page loader and in the meantime loads all modules to run designer.
3. **System** calls web-service to get previously saved *Web* Apps and renders them as grid.
4. **System** checks permissions and shows proper actions above/under the grid.
5. **Web App Designer** clicks on button "Add new Web App".
6. **System** adds new Web App to database and re-renders/prepends the grid.

Pre-conditions:

1. **Portal User** is logged to the portal.
2. **Portal User** has **Web App Designer** license**.**
3. **Web App Designer** has permission to add new *Web App.*

4.1.1.2 Ability to remove selected Web App

Steps:

1. **Web App Designer** selects *Web Apps* option from portal's menu.
2. **System** shows page loader and in the meantime loads all modules to run designer.

3. **System** calls web-service to get previously saved *Web Apps* and renders them as grid.
4. **System** checks permissions and shows proper actions above/under the grid.
5. **Web App Designer** selects one or more *Web Apps*.
6. **Web App Designer** clicks on button "Remove selected Web Apps".
7. **System** shows small popup with confirmation text and buttons "Yes" and "No".
8. **Web App Designer** clicks on button "Yes".
9. **System** updates selected Web Apps in database, sets archival flag to true.
10. **System** re-renders the grid/removes selected elements from the grid.

Pre-conditions:

1. **Portal User** is logged to the portal.
2. **Portal User** has **Web App Designer** license**.**
3. **Web App Designer** has permission to add new *Web App.*
4. **Web App Designer** has previously created few *Web Apps.*

4.1.1.3 Ability to rename selected Web App

Steps:

1. **Web App Designer** selects *Web Apps* option from portal's menu.
2. **System** shows page loader and in the meantime loads all modules to run designer.
3. **System** calls web-service to get previously saved *Web Apps* and renders them as grid.
4. **System** checks permissions and shows proper actions above/under the grid.
5. **Web App Designer** selects *Web App*.
6. **Web App Designer** clicks on button "Edit selected Web App".
7. **System** shows a popup with two inputs (Name and Description).
8. **Web App Designer** sets new values and clicks on "Save" button.
9. **System** updates selected Web Apps in database.
10. **System** re-renders the grid/updates selected element on the grid.

Pre-conditions:

1. **Portal User** is logged to the portal.
2. **Portal User** has **Web App Designer** license**.**
3. **Web App Designer** has permission to add new *Web App.*
4. **Web App Designer** has previously created few *Web Apps.*

4.1.1.4 Ability to manage versions for selected Web App

Steps:

1. **Web App Designer** selects *Web Activities* option from portal's menu.
2. **System** shows page loader and in the meantime loads all modules to run designer.
3. **System** calls web-service to get previously saved *Web Apps* and renders them as grid.
4. **System** checks permissions and shows proper actions above/under the grid.
5. **Web App Designer** selects *Web App*.
6. **Web App Designer** clicks on button "Design selected Web App".
7. **System** shows grid with currently saved versions.

Pre-conditions:

1. **Portal User** is logged to the portal.
2. **Portal User** has **Web App Designer** license**.**
3. **Web App Designer** has permission to add new *Web App.*
4. **Web App Designer** has previously created few *Web Apps.*

4.1.1.5 Ability to duplicate current version for selected Web App

Steps:

1. **Web App Designer** selects *Web Apps* option from portal's menu.
2. **System** shows page loader and in the meantime loads all modules to run designer.
3. **System** calls web-service to get previously saved *Web Apps* and renders them as grid.
4. **System** checks permissions and shows proper actions above/under the grid.
5. **Web App Designer** selects *Web App*.
6. **Web App Designer** clicks on button "Version Manager".
7. **System** shows grid with currently saved versions.
8. **Web App Designer** selects version.
9. **Web App Designer** clicks on button "Duplicate".
10. **System** duplicates selected version incrementing number of versions
11. **System** shows new version at the top of the list.

Pre-conditions:

1. **Portal User** is logged to the portal.
2. **Portal User** has **Web App Designer** license**.**
3. **Web App Designer** has permission to add new *Web App.*
4. **Web App Designer** has previously created few *Web Apps.*
5. **Web App Designer** has previously created some versions.

4.1.1.6 Ability to add new page inside selected Web App

Steps:

1. **Web App Designer** selects *Web Apps* option from portal's menu.
2. **System** shows page loader and in the meantime loads all modules to run designer.
3. **System** calls web-service to get previously saved *Web Apps* and renders them as grid.
4. **System** checks permissions and shows proper actions above/under the grid.
5. **Web App Designer** selects *Web App*.
6. **Web App Designer** clicks on button "Version Manager".
7. **System** shows grid with currently saved versions.
8. **Web App Designer** selects version.
9. **Web App Designer** clicks on button "Page Manager".
10. **System** loads from server list of all saved pages, java scripts and branding.
11. **System** shows new page with 3 tabs (Pages, JavaScript, and Branding).
12. **System** selects first tab by default.
13. **System** shows grid with currently saved pages (or empty grid for new *Web Apps*).
14. **Web App Designer** clicks on button "Add new page".
15. **System** shows popup with input.
16. **Web App Designer** fills input with new name for page.
17. **System** appends/re-renders grid.

Pre-conditions:

1. **Portal User** is logged to the portal.
2. **Portal User** has **Web App Designer** license**.**
3. **Web App Designer** has permission to add new *Web App.*
4. **Web App Designer** has previously created few *Web Apps.*

4.1.1.7 Ability to design selected page inside Web App

Steps:

1. **Web App Designer** selects *Web Apps* option from portal's menu.
2. **System** shows page loader and in the meantime loads all modules to run designer.
3. **System** calls web-service to get previously saved *Web Apps* and renders them as grid.
4. **System** checks permissions and shows proper actions above/under the grid.
5. **Web App Designer** selects *Web App*.
6. **Web App Designer** clicks on button "Design selected Web App".
7. **System** hides current page.
8. **System** loads from server list of all saved pages, java scripts and branding.

9. **System** shows new page with 3 tabs (Pages, JavaScript, and Branding). First tab selected by default.
10. **System** shows grid with currently saved pages (or empty grid for new *Web Apps*).
11. **Web App Designer** selects Page on the grid.
12. **Web App Designer** clicks on button "Design selected page".
13. **System** hides current page.
14. **System** loads from server complete page
15. **System** shows controls area with grouped components.
16. **System** shows canvas area (depending on 3rd part, loads all saved components).
17. **System** shows properties area.

Pre-conditions:

1. **Portal User** is logged to the portal.
2. **Portal User** has **Web App Designer** license**.**
3. **Web App Designer** has permission to add new *Web App.*
4. **Web App Designer** has previously created few *Web Apps.*

4.1.1.8 Ability to drag & drop components

Steps:

1. **Web App Designer** drag & drops selected component to a canvas area.
2. **System** gets drop coordinates and creates new component on canvas.

Pre-conditions:

1. **Portal User** is logged to the portal.
2. **Portal User** has **Web App Designer** license**.**
3. **Web App Designer** has opened Page Designer component.

4.1.1.9 Ability to set properties

Steps:

1. **Web App Designer** selects component on canvas area.
2. **System** catches selection and gets current components model.
3. **System** refreshes properties area with all needed properties for selected component.
4. **Web App Designer** adjusts properties.
5. **System** listens to changes and when needed, refreshes selected component on canvas area.

Pre-conditions:

1. **Portal User** is logged to the portal.
2. **Portal User** has **Web App Designer** license**.**
3. **Web App Designer** has opened Page Designer component.

4.1.2 Full Screen Mode

To assist with a rich and immersive design experience, the Web App designer is to support a full screen mode. When in full screen the Header is removed to create more design real estate. The image below shows a possible layout for full screen mode.

4.1.3 Controls

To create a great user experience, a rich set of user controls for input, display and layout is required.

Both open source and commercial (runtime free) components are acceptable. However a common style/theme must run through the controls.

Controls have a set of properties. Properties can be divided into groups. Not every control has to have all kinds of properties. There are six major properties groups:

- System - contains main elements that are bound to components, like identifier. Mostly they are used as selectors inside custom JavaScript.
- Presentation – contains properties that are involved in how current component looks like. Even small changes will refresh components view.
- Validation – contains properties that helps validate input components.
- Data Model – contains properties that can bind current component to Data Objects. It's possible to have extra elements that can filter and sort selected Data Objects.
- Initial Data – contains properties that will affect running activities. Elements set here will be visible while executing selected activity and will act as default values.
- Action - Every active component (like button) has set of events and action which can be handled. Every action can be scripted using defined

JavaScript or can be handled with predefined actions, like go to next page, save and synchronize, clear form etc. Second approach does not require any "coding".

Required controls include:

- Charting
  - Full range of HTML5 charts.
    - Tenant data objects, web services methods and JavaScript will provide data for controls.
- Input Controls
  - Text, Multi-Line Text & Labels
  - Number box (spinner)
  - Date and Time, Date only & Time only
  - Calendar Control with Switch between Gregorian and Hijri Calendars - the control is to take the date entered and be able to calculate the change into the other calendar format.
  - Check Box, Option Box, Toggle Button
  - Buttons, Button with stock images
  - Image button (custom image)
  - Link
  - HTML Panel
  - Slider, Progress Bar
  - RSS Feed reader
  - Data List
  - Combo – there is to be a setting through the designer to show no selection if the data has not already been selected.
  - Image panel, Icon
  - Colour Picker
  - Tree Views
  - View drawings/images and annotate the image
  - Calendar View
  - Annotation of graphics/overlay
  - Data Tag (see social features)
  - Document Linker (see document management features)

- Grids
  - Data Grids
    - Editable data grids
    - Control over style and row/cell colour
    - Row actions, Update/Delete
    - Multi column Search feature
    - Multi column sort feature
    - Column personalisation
    - User definable views
    - Persistency – remember last settings
- Pivot control
  - To support ad-hoc real-time data analysis. Please see section Pivot Control
  -
- Flow Control
  - Table flow (rows and columns)
  - Dock areas
  - Collapse Sections
  - Accordion
  - Show/Hide Sections
  - Group By Box
  - Tabs
- Validation
  - User Interface input validation with translatable error text
- Mapping
  - Map control (see Mapping section for details)
- Social
  - Share Button
    - Facebook
    - Twitter
    - LinkedIn
    - Google+

4.1.3.1 Presentation

Components are standard html elements. Both canvas area (place where you can drop components) and web app runtime are using the same frameworks and libraries to draw components (for example Button component will be drawn by Twitter Bootstrap or Foundation framework and Chart component by jqPlot).

All components by default have flexible width and height (automatically they fit its parent sizes). To be able to create few components inline, Web App Designer has to drag Container component with specified number of columns.

Other common element for almost all components is theme (see: Branding and Style).

4.1.3.2 Data model

Many components can be bound to Data Objects. Depending on activity flow those components can read/write Data Objects. Basic flow allows binding one table per Activity page, so each component can be bound to specific columns from this table.

Joining tables are part of codeless design too, so when the component is bound to FK column, a system renders a helper dropdown with columns from related table. More complex joins are only available through custom JavaScript.

4.1.3.3 List of available components:

NOTE: Where Width and Height properties are available, the control should default

All visual controls have default properties

- Width (in percent or pixels)
- Height (in percent or pixels)

All visual control with actions could call java script functions and workflows through the web service.

Page

Page is a component too, but it's not possible to drag & drop it anywhere and it's not listed inside Controls Area. It's a static element. Page has a set of properties that globally interacts with all components on it.

- Presentation
  - Themes
- Data model
  - Data Object (each page can be bound to one main Data Object)
- Action
  - On load (dropdown: functions)

Container

Container defines layout areas where **Web App Designer** can put other controls. Container is built from cells. Each cell has its own properties.

- System
  - Identifier (unique)
- Presentation
  - Number of rows (number 1 – 20)
  - Number of columns (number 1 – 6)
  - Column widths (slider - sum 100%) or
    - Container and columns to Fixed width in pixels
  - Container and columns width in pixels
  - Themes
- Cell (Grids have editable cells, selecting them will populate extra properties for each cell)
  - Horizontal alignment (top, middle, bottom)
  - Vertical alignment (left, center, right)

Collapsible section

Collapsible section defines area where **Web App Designer** can put other controls. **Web App User** can show/hide its area by simply clicking on sections header.

- System
  - Identifier (unique)
- Presentation
  - Status (opened/closed)
  - Width
  - Height
  - Theme
- Action
  - On click (function)

Divider

Dividers can be defined as visual lines that allows to gap components from each other. Line is an optional element, can be defined as empty space.

- Presentation
  - Margin top
  - Margin bottom
  - Line visibility (on/off)
  - Line color (color picker)
  - Line width

Accordion

Accordion defines areas where **Web App Designer** can put other components. **Web App User** can show/hide its area by simply clicking on sections header.

- System
  - Identifier (unique)
- Presentation
  - Width
  - Height
  - Theme
- Sections
  - Elements (ability to add a set of items)
    - Title
    - Status (which accordion section will be visible by default)
- Action
  - On click (function)

Tabs

Like Accordions, Tabs defines areas where **Web App Designer** can put other components.

- System
  - Identifier (unique)
- Presentation
  - Width
  - Height
  - Theme
- Sections
  - Elements (ability to add a set of items)
    - Title
    - Status (which tab section will be visible by default)
- Action
  - On click (function)

Text Box

Basic input element with ability to read specific data objects. Allows the user to view, enter and edit text data.

- System
  - Identifier
  - Type (text, password)
  - Read-only (true, false)
- Presentation
  - Width
  - Theme
  - Placeholder
  - Hidden
  - Text alignment (left, center, right)
- Validation
  - Required
  - Pattern (regex)

    - Minimum (min number of typed letters)
    - Maximum (max number of typed letter)
    - Error message (when element is not valid, show this message)
- Data model
    - Column (dropdown)
    - Display as (when selected column is a FK, show related columns instead)
- Initial Data
    - Default value
- Action
    - On change (function)

Number Box

Number box is an input where **Web App User** can add only numbers. Allows the user to view, enter and edit numbers. Other chars are not allowed.

- System
    - Identifier
    - Step
    - Read-only (true, false)
- Presentation
    - Width
    - Theme
    - Hidden
    - Text alignment (left, center, right)
- Validation
    - Required
    - Pattern (regex)
    - Minimum (value check)
    - Maximum (value check)
    - Error message (when element is not valid, show this message)
- Data model
    - Column (dropdown)
    - Display as (when selected column is a FK, show related columns instead)
- Initial Data
    - Default value
- Action
    - On change (function)

Text Area

Multiline input component. Allows the user to view, enter and edit text data.

- System
    - Identifier
    - Read-only (true, false)
- Presentation
    - Width
    - Height
    - Theme

    - Hidden
    - Text alignment (left, center, right)
- Validation
    - Required
    - Pattern (regex)
    - Minimum (value check)
    - Maximum (value check)
    - Error message (when element is not valid, show this message)
- Data model
    - Column (dropdown)
    - Display as (when selected column is a FK, show related columns instead)
- Initial Data
    - Default value
- Action
    - On change (function)

Label

Label is a read-only element that has ability to show selected data object value.

- System
    - Identifier
- Presentation
    - Width
    - Height
    - Theme
    - Text alignment (left, center, right)
    - Font size (in pixels)
    - Font color (color picker)
    - Hidden (on, off)
- Initial Data
    - Default value
- Data model
    - Column (dropdown)
    - Display as (when selected column is a FK, show related columns instead)

Button

A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it. Clickable button that is able to execute an action such as JavaScript, Form Navigation or any other predefined action.

- System
    - Identifier
    - Disabled (true, false)
- Presentation
    - Text
    - Text alignment (left, center, right)
    - Width

    - Height
    - Theme
    - Border (on, off)
    - Hidden (on, off)
- Background
    - Up state
        - Image
        - Color
        - Size (Original, Tile, Stretch, Fill, Fit)
        - Position X
        - Position Y
    - Down state
        - Image
        - Color
        - Size (Original, Tile, Stretch, Fill, Fit)
        - Position X
        - Position Y
    - Hover state
        - Image
        - Color
        - Size (Original, Tile, Stretch, Fill, Fit)
        - Position X
        - Position Y
- Action
    - Type (JavaScript, Save, Link, Clear, Refresh, Delete)
    - Redirect (shows if selected type is able to handle redirection)
    - On click (depending on type: function name)

Toggle Button

Clickable button that is able to execute an action such as JavaScript, Form Navigation or any other predefined action. A toggle button allows the user to change a setting between two states (normal state and pressed state).

- System
    - Identifier
    - Disabled (true, false)
- Presentation
    - Text
    - Text alignment (left, center, right)
    - Width
    - Height
    - Theme
    - Border (on, off)
    - Hidden (on, off)
- Background
    - Normal state
        - Image
        - Color
        - Size (Original, Tile, Stretch, Fill, Fit)

        - Position X
        - Position Y
    - Pressed state
        - Image
        - Color
        - Size (Original, Tile, Stretch, Fill, Fit)
        - Position X
        - Position Y
- Action
    - On change (dropdown: function name)

Check Box

Check Box permits the user to make a binary choice (yes / no)

- System
    - Identifier
    - Disabled (true, false)
- Presentation
    - Text
    - Text alignment (left, center, right)
    - Width
    - Theme
    - Hidden (on, off)
- Background
    - Normal state
        - Image
        - Color
        - Size (Original, Tile, Stretch, Fill, Fit)
        - Position X
        - Position Y
    - Checked state
        - Image
        - Color
        - Size (Original, Tile, Stretch, Fill, Fit)
        - Position X
        - Position Y
- Validation
    - Required (means its status have to be checked, otherwise element is not valid)
    - Error message (when element is not valid, show this message)
- Initial Data
    - Status (checked / not checked)
- Data model
    - Column (dropdown)
- Action
    - On change (dropdown: function name)

Combo Box

The Combo Box component contains a drop-down list from which the user can select a single value.

- System
  - Identifier
  - Disabled (true, false)
- Presentation
  - Width
  - Theme
  - Hidden (on, off)
- Validation
  - Required
  - Error message (when element is not valid, show this message)
- Data model
  - Source table
  - Source column (dropdown: column list, runtime: display them as options label)
  - Target column (dropdown: column list)
  - Sort by (dropdown: column list)
  - Sort order (asc / desc)
  - Filter By (dropdown: column list)
    - Controls: data list, other combo box, radio button
    - Columns: list of columns that previously selected component is bound to
  - Filter values
  - Limit
- Dependency
  - Parent component (codeless – parent takes control over data model)
  - Column (dropdown: column list, codeless – FK representation)
- Initial Data
  - Options (ability to add a set of initial elements to combo as pair <value, label>)
- Action
  - On change (dropdown: function name)

Radio Box

Radio buttons allow the user to select one option from a set.

- System
  - Identifier
  - Disabled (true, false)
- Presentation
  - Width
  - Theme
  - Hidden (on, off)
- Validation
  - Required
  - Error message (when element is not valid, show this message)

- Initial Data
  - Options (ability to add a set of initial elements to combo as pair <value, label>)
    - Label
    - Value
  - Status (default selected option)
- Action
  - On change (dropdown: function name)

Date Picker

The Date Picker component is tied to a standard form input field. Focus on the input (click, or use the tab key) will open an interactive calendar in a small overlay.

- System
  - Identifier
  - Disabled (true, false)
- Presentation
  - Width
  - Theme
  - Hidden (on, off)
- Validation
  - Required
  - Minimum
  - Maximum
  - Error message (when element is not valid, show this message)
- Initial Data
  - Default date
- Action
  - On change (dropdown: function name)
- Hijri support

Color Picker

Color Picker is a simple component to select color. (See: http://jscolor.com/)

- System
  - Identifier
  - Disabled (true, false)
- Presentation
  - Width
  - Theme
  - Hidden (on, off)
- Validation
  - Required
  - Error message (when element is not valid, show this message)
- Initial Data
  - Default color
- Action
  - On change (dropdown: function name)

HTML Panel

HTML Panel is a Rich-text editor.

- System
  - Identifier
- Presentation
  - Width
  - Hidden (on, off)
- Data model
  - Column (dropdown)
  - Display as (when selected column is a FK, show related columns instead)
- Initial Data
  - Text

Image Panel

Image Panel is a component with ability to upload images and icons. During runtime this component has ability to allow annotations.

- System
  - Identifier
- Presentation
  - Width
  - Height
  - Hidden (on, off)
  - Size (Original, Tile, Stretch, Fill, Fit)
  - Position X
  - Position Y
- Data model
  - Column (dropdown)
  - Display as (when selected column is a FK, show related columns instead)
- Initial Data (Both upload at design time to set an image and at runtime to upload an image to a Web App and store in data Object)
  - Upload a file

Link

Component similar to label but has ability to add an address. During runtime: clicking on this element will redirect **Web App User** to specified place.

- System
  - Identifier
- Presentation
  - Width
  - Height
  - Theme
  - Text alignment (left, center, right)
  - Font size (in pixels)
  - Font color (color picker)

    - Hidden (on, off)
- Initial Data
    - Text
    - Link
- Data model
    - Text Column (dropdown)
    - Link Column (dropdown)
- Behavior
    - Target (The target attribute specifies where to open the linked document)
        - _blank – load in a new window
        - _self – load the same frame as it was clicked
        - _parent – load in the parent frameset
        - _top – load in the full body of a window
        - *Frame name* – load in named frame

Slider

The basic slider is horizontal and has a single handle that can be moved with the mouse or by using the arrow keys.

- System
    - Identifier
- Presentation
    - Width
    - Theme
    - Min
    - Max
    - Step
    - Hidden (on, off)
- Initial Data
    - Default value
- Data model
    - Column (dropdown)
    - Display as (when selected column is a FK, show related columns instead)
- Action
    - On change (dropdown: function)

Progress Bar

Provide up-to-date feedback on the progress of a workflow or action with simple yet flexible progress bars.

- System
    - Identifier
- Presentation
    - Width
    - Theme
    - Min (default: 0)
    - Max (default: 100)

    - Hidden (on, off)
- Initial Data
    - Default value
- Data model
    - Column (dropdown)
    - Display as (when selected column is a FK, show related columns instead)

RSS Feed reader

List of aggregated content such as news headers

- System
    - Identifier
- Presentation
    - Width
    - Height
    - Theme
    - Hidden (on, off)
- Initial Data
    - Default RSS address
- Data model
    - Column (dropdown)
    - Display as (when selected column is a FK, show related columns instead)

Calendar

Calendar is a graphical date picker that has ability to switch between Gregorian and Hijri calendar (http://www.islamicfinder.org/Hcal/index.php).

- System
    - Identifier
    - Type (Gregorian, Hijri)
- Presentation
    - Width
    - Height
    - Theme
    - Hidden (on, off)
    - Switcher (on/off, when "on", new button appears on control with ability to change calendar type on runtime)
- Initial Data
    - Default date
- Data model
    - Column (dropdown)
    - Display as (when selected column is a FK, show related columns instead)
- Action
    - On change (dropdown: function)

Data List

Data List is very similar to List View in current implementation on mobile. Data List is a set of Data Objects presented as horizontal list with extra elements that helps to describe it. Has ability to show thumbnails.

- System
  - Identifier
- Presentation
  - Width
  - Height
  - Theme
  - Hidden (on, off)
  - Label
  - Label alignment (left, center, right)
- Data model
  - Source table
  - Options (ability to add a set of columns, first column will be described as header)
    - Column
    - Display as (when selected column is a FK, show related columns instead)
  - Thumbnail column (dropdown: column list)
  - Sort by (dropdown: column list)
  - Sort order (asc / desc)
  - Filter By (dropdown: column list)
  - Filter values
  - Limit
  - Lazy loading (on/off)
  - Lazy loading step
- Action
  - Redirect url
  - On click (dropdown: function)

Tree View

Tree View gives ability to see grouped data. Web App user has the ability to show/hide each node.

- System
  - Identifier
- Presentation
  - Width
  - Height
  - Theme
  - Hidden (on, off)
- Initial Data
  - Default nodes
- Data model
  - Column
  - Display as (when selected column is a FK, show related columns instead)

- Action
  - On click (dropdown: function)

Grid

Grid is an editable Data Object table.

- System
  - Identifier
- Presentation
  - Width
  - Height
  - Theme
  - Hidden (on, off)
  - Pagination (on, off)
- Data model
  - Source table
  - Options (ability to add a set of columns, first column will be described as header)
    - Column
    - Display as (when selected column is a FK, show related columns instead)
  - Sort by (dropdown: column list)
  - Sort order (asc / desc)
  - Filter By (dropdown: column list)
  - Filter values
  - Limit
- Action
  - On click (dropdown: function)

Web App runtime behavior:

- Editable data grids
- Control over style and row/cell colour
- Row actions, Update/Delete
- Multi column Search feature
- Multi column sort feature
- Column personalisation
- User definable views
- Persistency – remember last settings

Chart

Ability to draw charts (bars, lines, pies and areas) based on Data Object columns.

- System
  - Identifier
- Presentation
  - Width
  - Height
  - Theme

- Hidden (on, off)
- Data model
  - Source table
  - Options (ability to add a set of columns, first column will be described as header)
    - Column
    - Display as (when selected column is a FK, show related columns instead)
  - Sort by (dropdown: column list)
  - Sort order (asc / desc)
  - Filter By (dropdown: column list)
  - Filter values
  - Limit

Map

The maps have to be extensible to enable points to be plotted on them based upon GPS data within the page, via JavaScript or from data sources such as data objects or web services. Preferable library: http://www.openstreetmap.org/.

- System
  - Identifier
- Presentation
  - Width
  - Height
  - Theme
  - Hidden (on, off)
- Initial data
  - Pinpoints (multi-values)
    - Longitude
    - Latitude
- Data model
  - Source table
  - Source column (dropdown: column list)
  - Sort by (dropdown: column list)
  - Sort order (asc / desc)
  - Filter By (dropdown: column list)
  - Filter values
  - Limit

Document Linker

Document Linker gives ability to upload selected file to a Data Object,

- System
  - Identifier
  - Disabled (true, false)
- Presentation
  - Width
  - Height
  - Theme
  - Hidden (on, off)

- Data model
  - Target Column
- Action
  - On click (dropdown: function)

Pivot

A pivot style control is required so that end users will be able to complete real-time data analysis. The control is to have the following functionality:

- Ability to set Row, Colum and Value fields for the pivot
- Ability to bind to pre-defined user queries
- Ability to build user defined queries
- Ability to export to PDF and Print

Note: A suitable runtime royalty free is required to provide the core Pivot Functionality.

Social

Social component is a button with social share ability. It links current activity with social platforms such as Facebook, Google+, LinkedIn and Twitter

- System
  - Identifier
  - Disabled (true, false)
- Presentation
  - Width
  - Height
  - Theme
  - Hidden (on, off)
  - Social Platform (Facebook, Google+, LinkedIn or Twitter)
- Action
  - On click (dropdown: function)

4.1.4 Control Filtering

The designer is to support filtering of all data based controls such as combo's and list views. The requirement is that through the designer and through JavaScript, it is possible to set the current value of one control to be the filter value for another control. For example:

Control 1: Date Picker

Control 2: List of tasks

On change of the Date, the list of tasks will be filtered to show only tasks for the selected date.

The filtering capabilities must also be advanced to support complex filters such as:

- =, <, >, <>, In, Between

An example of this is:

Control 1: Date From - Picker

Control 2: Date To - Picker

Control 3: Priority - Combo

Control 4: List of tasks

On change of control 1, 2 or 3 the list of tasks will filter to show only tasks between the from and to date where the priority equals the priority combo value.

4.1.5 Control Data Sorting

For all data controls that show more than one record, the data will be sortable. This setting of sort order is to be settable through the designer as well as through JavaScript.

4.1.6 Page Parameters

Page parameters are used to pass information between pages when navigating. It is a requirements that page parameters are managed visually within the page designer so that it is clear what the input parameters and output parameters are.

It is also a requirement that page parameters can be set to the output of any control on the page through the designer, and that control values can be set to be a page parameter on page load. This will enable simple passing of data without the need for complex JavaScript.

It is also a requirement that page parameters can be read and written through JavaScript

4.1.7 Mapping

The designer is to support maps on the web pages. The maps have to be extensible to enable points to be plotted on them based upon GPS data within the page, via JavaScript or from data sources such as data objects or web services.

Note: Open source options such as http://www.openstreetmap.org/ are preferable

4.1.8 Internationalisation

Both the platform and applications built on the platform are to support internationalisation. This includes the data model which is to be UNICODE compliant and including support for:

- Multi-language
- Multi-time zone
- Multi-calendar
- Multi-currency
- Multi-directional (right-to-left)

Note: To support certain features the database should be automatically adjusted, for example new fields for multi-currency – base, actual & exchange rate.

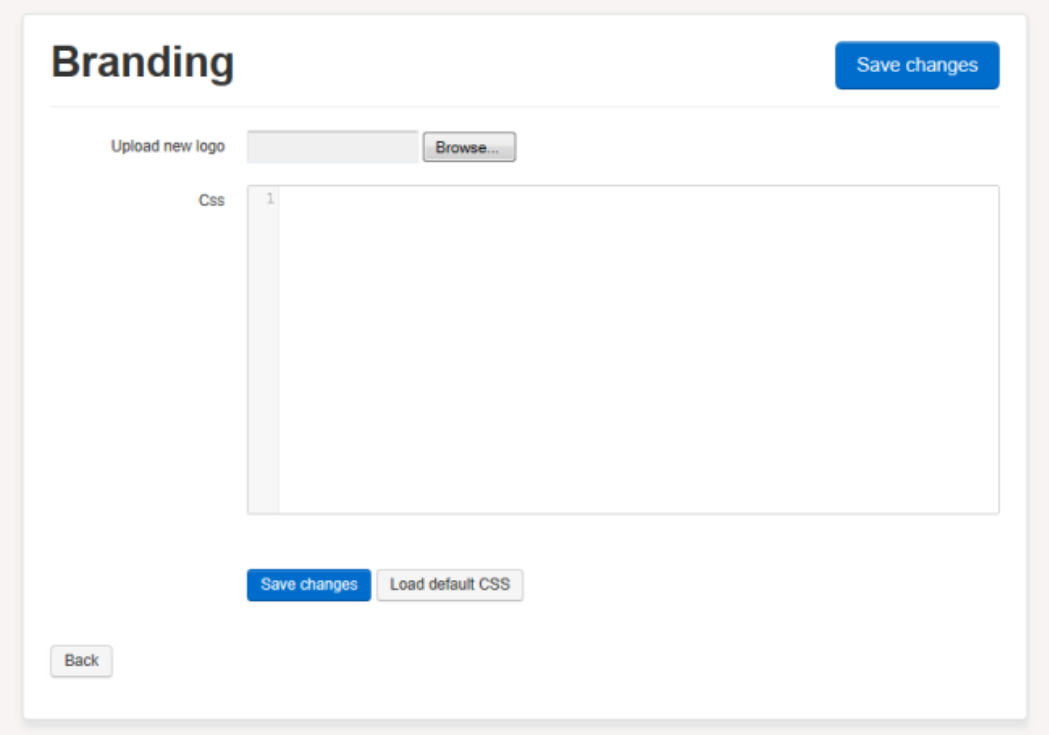
4.1.9 Branding and Style

It is a requirement that Web Apps support CSS for style, and enable pages to be branded according to the client/product.

It is also a requirement that all user interface controls support full styling including but not limited to the following:

- Height & Width
- Control Alignment
  - Vertical & Horizontal
- Text Alignment
  - Vertical & Horizontal
- Background Colour
  - Block and Gradient
- Background Image
  - Original, Tile, Fit, Stretch etc.
  - For buttons, check boxes, radio toggle etc – all press states must have colour and image options.
- Font
  - Type, Bold, Italic, Size & Colour
- Containers – Alignment – Vertical & Horizontal
  - Percentage & Pixel widths

The image bellow shows possible mock-up of the *Branding and Style* window:



Themes

It is a requirement that the Web Pages support style "Themes" to simplify and unify style across web applications. A default theme is to be included, with the option of creating new themes.

A theme will style controls and web pages such as colours, fonts, borders, transparency etc.

4.1.10 Live Preview

Through the designer it is a requirement to be able to preview the page and review all of the page functionality, with full features, data and styles enabled.

The live preview will be part of application portal along with web app designer.

4.1.11 Responsive Design

It is a requirement to support different form factors and devices that the web application layout supports "Responsive" design. The definition of how the design behaves with the different factors is to be managed through the designer in a codeless manner.

4.2 Custom Pages

As we are building a designer to create web applications, there will be some examples of bespoke pages that will not be possible or just too difficult to create using the web app designer.

In this instance, it is required that custom web pages can be imported into the web app and used within the application flow.

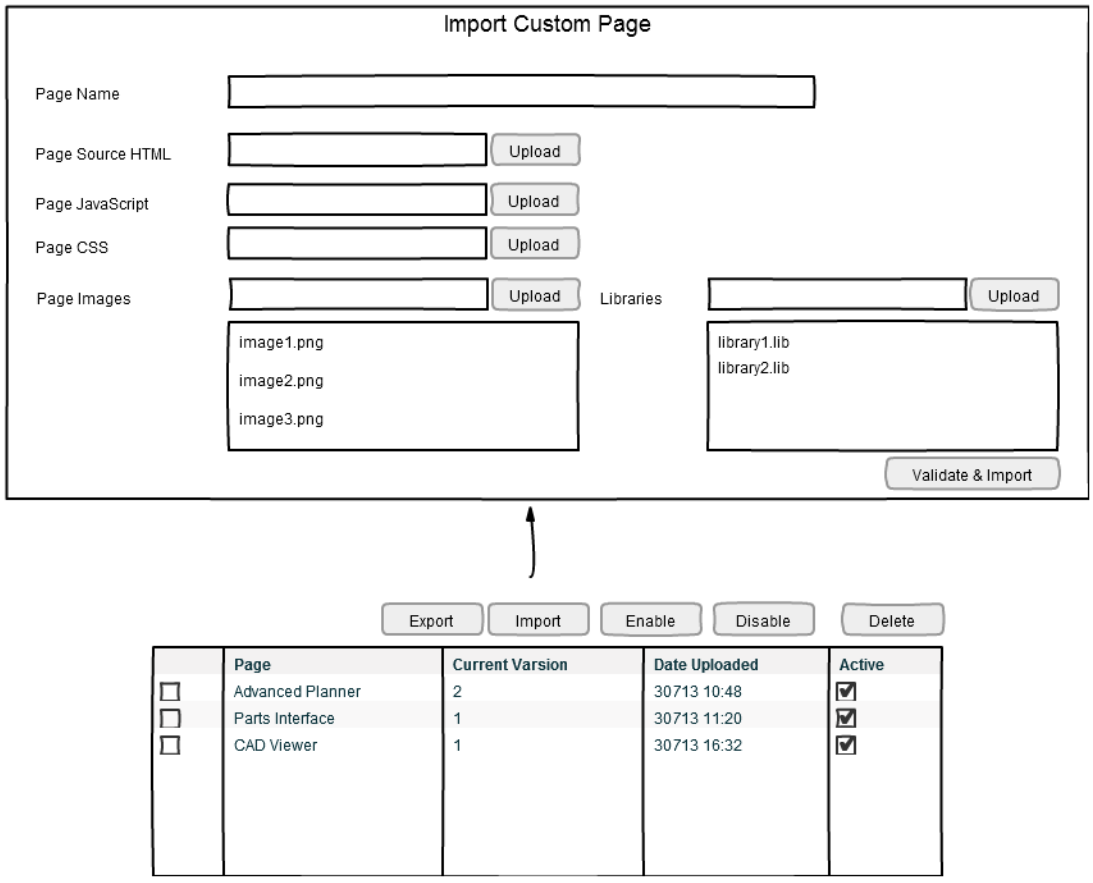
Custom pages will communicate with the platform through a defined API which will support data binding, events and page navigation features.

Custom pages will comprise of the html, JavaScript and any additional libraries or images.

The Web App designer component is to provide a user interface to manage the importing and exporting of custom pages.

Custom pages will be included in the Web App export function.

Below is an option on how this feature may be designed.



4.2.1 Custom Page API

To support the custom pages and enable the pages to be created in 3rd party tools a Custom Page API is required. The API will support the following features.

- Platform Authentication
- Access to Authenticated User information
- Data Object Binding
  - CRUD
- Platform JavaScript functions
- Page Navigation
- Event binding
  - Workflow, Web Services

Note: This list will increase as the design phase progresses

The Platform will have a Blank Custom Page template available to download for Web App designers and developers. The template will include a header with all system JavaScript libraries included. Depending on the framework used jQuery, Bootstrap, etc. will be linked there, but primarily a JavaScript library encapsulating all available GO Platform web service methods will be included.

This "fsigoplus.js" library has to be code-generated to not hinder a further development of the system.

4.3 JavaScript

To enable advanced capabilities, full JavaScript capabilities including support for 3rd party libraries such as JQuery is to be supported. Full access to the page document object model is also required.

The designer is to provide a JavaScript editor similar to the FSI GO editor.

4.4 Events

To support commercial applications it is a requirement that web designer through UI is to support manage web service and java script events binding. Web designer will allow to create, modify and delete existing binding for controls defined bellow. Web designer is to provide two options for web service and java script binding e.g.:

- Events for user control
- Page events

User controls which support triggering web services:

- Button
- Button with stock image

- Image button
- Link
- Data list
- Combo

Supported page events:

- On load
- On unload
- On resize
- On scroll
- On click
- On error
- On before unload – works correct only in chrome browser
- Before Page Changed

Web designer is to provide functionality to bind input and output parameters of web services to user control. Outcome of web services could be bind to selected components e.g.:

- Number Box
- Text
- Multi-line
- Labels

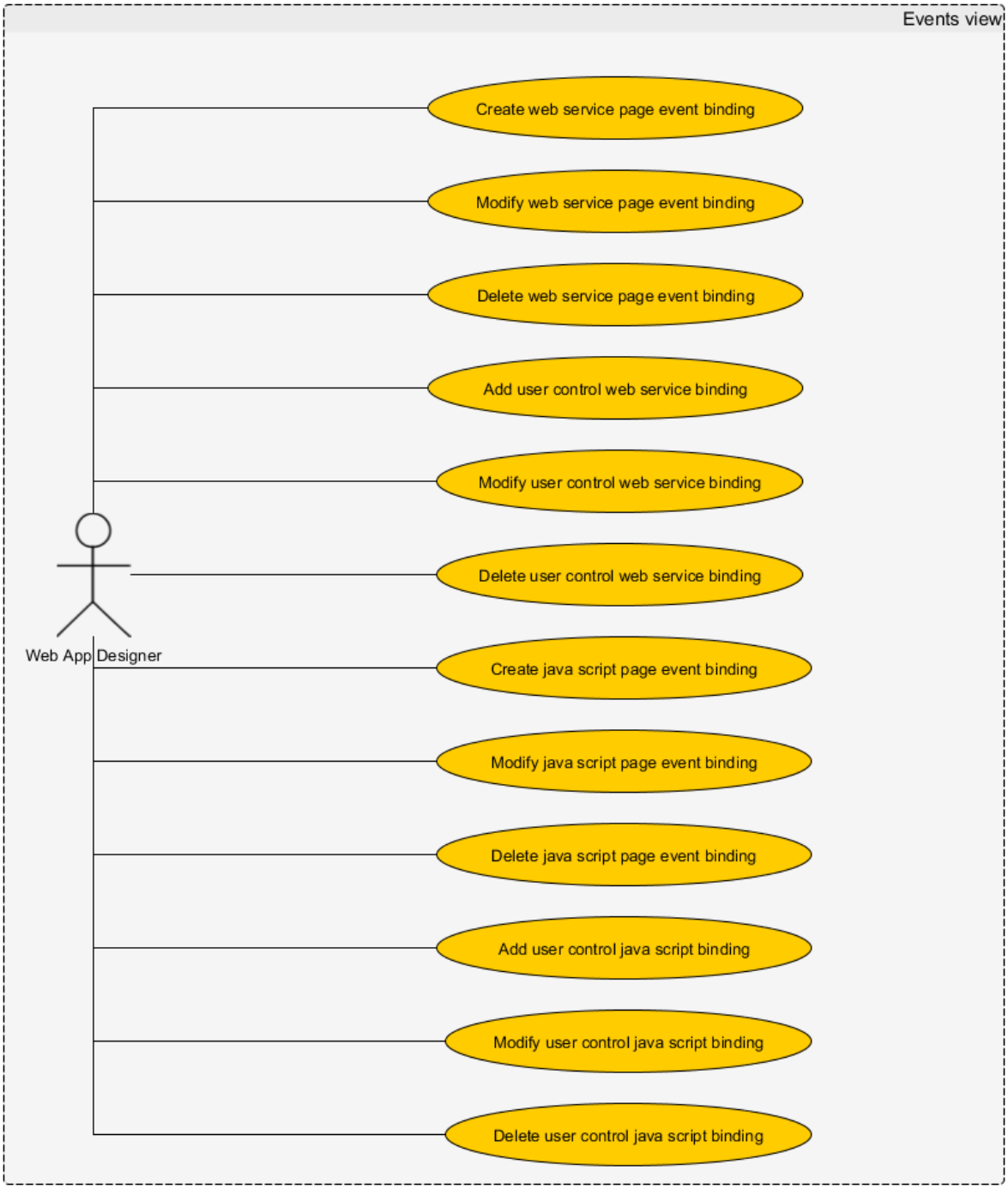
Web designer supports passing parameters only for web service binding. For java script binding there is requirement to selecting only no-arguments functions.

Features available in the designer will be accessible through java script. It means that there will be possibility create binding via java script function.

4.4.1 Assumptions

- Workflows will be exposed via web service.
- Possibility to bind java script function and web service together to each event e.g.: page event or control event
- Tenant will see only his web services and workflows.

4.5 Use case diagram



4.6 Use case scenarios description

4.6.1 Create web service page event binding.

1. **Web Designer** selects in web app designer button to bind web service to page events
2. **System** presents GUI to page events binding
3. **System** presents page events which could trigger web service. Default *on load* event is selected.
4. **System** presents registry of web services which could be triggered by event. Default nothing is selected.
5. **Web Designer** selects page event.
6. **Web Designer** selects web service.
7. **System** presents input and output parameters of web service. All required input parameters are highlighted and need to be fill.
8. **Web Designer** bind input parameters of web service with existing data objects.
9. **Web Designer** bind output parameters of web service with existing data objects. This step could be skipped.
10. **Web Designer** confirm operation by click submit button
11. **System** saves page event binding configuration in database.
12. Flow ends

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Page event binding has been stored in database.

4.6.2 Modify web service page event binding.

1. **Web Designer** selects in web app designer button to bind web service to page events
2. **System** presents GUI to page events binding
3. **System** presents page events which could trigger web service. Default *on load* event is selected.
4. **System** presents registry of web services which could be triggered by event. Default nothing is selected.
5. **Web Designer** selects page event.
6. **System** presents web service which is bound to event.
7. **System** presents input and output parameters of web service.
8. **Web Designer** changes input parameters of web service binding.
9. **Web Designer** changes output parameters of web service binding.
10. **Web Designer** confirm operation by click submit button
11. **System** saves page event binding configuration in database.
12. Flow ends

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Page event binding has been modified in database.

4.6.3 Delete web service page event binding.

1. **Web Designer** selects in web app designer button to bind web service to page events
2. **System** presents GUI to page events binding
3. **System** presents page events which could trigger web service. Default *on load* event is selected.
4. **System** presents registry of web services which could be triggered by event. Default nothing is selected
5. **Web Designer** selects page event.
6. **System** presents web service which is bound to event.
7. **Web Designer** clear selection of web service
8. **System** hide web service parameters.
9. **Web Designer** confirm operation by click submit button
10. **System** saves page event binding configuration in database.
11. Flow ends

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Page event configuration has been deleted from database.

4.6.4 Create web service control binding.

1. **Web Designer** selects user control on design canvas
2. **System** presents selected control properties
3. **Web Designer** selects button to bind web service with events
4. **System** presents GUI to page events binding
5. **System** presents user control events which could trigger web service. Default *on Click* event is selected.
6. **System** presents registry of web services which could be triggered by event. Default nothing is selected
7. **Web Designer** selects control event.
8. **System** presents web service which is bound to event.
9. **System** presents input and output parameters of web service. All required input parameters are highlighted and need to be fill.
10. **Web Designer** bind input parameters of web service with existing data objects.
11. **Web Designer** bind output parameters of web service with existing data objects. This step could be skipped.

12. **Web Designer** confirm operation by click submit button
13. **System** checks if required parameters are set. If parameters are not set **system** presents warning and flow starts from point 9.
14. **System** saves control event binding configuration in database.
15. Flow ends

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Control event binding has been stored in database.

4.6.5 Modify web service control binding.

1. **Web Designer** selects user control on design canvas.
2. **System** presents selected control properties.
3. **Web Designer** selects button to bind web service with events.
4. **System** presents GUI to page events binding.
5. **System** presents user control events which could trigger web service. Default *on Click* event is selected.
6. **Web Designer** selects control event.
7. **System** presents web service which is bound to event.
8. **System** presents input and output parameters of web service.
9. **Web Designer** changes input parameters of web service binding.
10. **Web Designer** changes output parameters of web service binding.
11. **Web Designer** confirm operation by click submit button.
12. Flow ends

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Control event binding has been modified in database.

4.6.6 Delete web service control binding.

1. **Web Designer** selects user control on design canvas.
2. **System** presents selected control properties.
3. **Web Designer** selects button to bind web service with events.
4. **System** presents GUI to page events binding.
5. **Web Designer** selects page event.
6. **System** presents web service which is bound to event.
7. **Web Designer** clear selection of web service.
8. **System** hide web service parameters.
9. **Web Designer** confirm operation by click submit button.
10. **System** saves control event binding configuration in database.
11. Flow ends.

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Control event binding configuration has been deleted from database.

4.6.7 Create java script page event binding.

1. **Web Designer** selects in web app designer button to bind java script function to page events
2. **System** presents GUI to page events binding
3. **System** presents page events which could trigger java script function. Default *on load* event is selected.
4. **System** presents registry of java script functions which could be triggered by event. Default nothing is selected.
5. **Web Designer** selects page event.
6. **Web Designer** selects function.
7. **Web Designer** confirm operation by click submit button
8. **System** saves page event binding configuration in database.
9. Flow ends

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Page event binding has been stored in database.

4.6.8 Modify java script page event binding.

1. **Web Designer** selects in web app designer button to bind java script function to page events
2. **System** presents GUI to page events binding
3. **System** presents page events which could trigger java script function. Default *on load* event is selected.
4. **System** presents registry of java script function which could be triggered by event. Default nothing is selected.
5. **Web Designer** selects page event.
6. **System** presents java script function which is bound to event
7. **Web Designer** changes java script function.
8. **Web Designer** confirm operation by click submit button
9. **System** saves page event binding configuration in database.
13. Flow ends

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Page event binding has been modified in database.

4.6.9 Delete java script page event binding.

1. **Web Designer** selects in web app designer button to bind java script function to page events
2. **System** presents GUI to page events binding
3. **System** presents page events which could trigger java script function. Default *on load* event is selected.
4. **System** presents registry of java script function which could be triggered by event. Default nothing is selected
5. **Web Designer** selects page event.
6. **System** presents java script function which is bound to event.
7. **Web Designer** clear selection of java script function.
8. **Web Designer** confirm operation by click submit button
9. **System** saves page event binding configuration in database.
10. Flow ends

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Page event configuration has been deleted from database.

4.6.10 Create java script control binding.

1. **Web Designer** selects user control on design canvas
2. **System** presents selected control properties
3. **Web Designer** selects button to bind java script function with events
4. **System** presents GUI to page events binding
5. **System** presents user control events which could trigger java script function. Default *on Click* event is selected.
6. **System** presents registry of java script function which could be triggered by event. Default nothing is selected.
7. **Web Designer** selects control event.
8. **System** presents java script function which could be bind to event.
9. **Web Designer** selects java script function from list
10. **Web Designer** confirm operation by click submit button
11. **System** checks if required parameters are set. If parameters are not set **system** presents warning and flow starts from point 9.
12. **System** saves control event binding configuration in database.
13. Flow ends

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Control event binding has been stored in database.

4.6.11 Modify java script control binding.

1. **Web Designer** selects user control on design canvas.
2. **System** presents selected control properties.
3. **Web Designer** selects button to java script service with events.
4. **System** presents GUI to page events binding.
5. **System** presents user control events which could trigger java script. Default *on Click* event is selected.
6. **Web Designer** selects control event
7. **System** presents java script which is bound to event.
8. **Web Designer** changes java script function.
9. **Web Designer** confirm operation by click submit button.
10. Flow ends

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Control event binding has been modified in database.

4.6.12 Delete java script control binding.

1. **Web Designer** selects user control on design canvas.
2. **System** presents selected control properties.
3. **Web Designer** selects button to bind java script function with events.
4. **System** presents GUI to page events binding.
5. **Web Designer** selects page event.
6. **System** presents java script function which is bound to event.
7. **Web Designer** clear selection of java script function.
8. **Web Designer** confirm operation by click submit button.
9. **System** saves control event binding configuration in database.
10. Flow ends.

Pre-conditions

- **Web Designer** is logged to the portal.

Post-condition

- Control event binding configuration has been deleted from database.

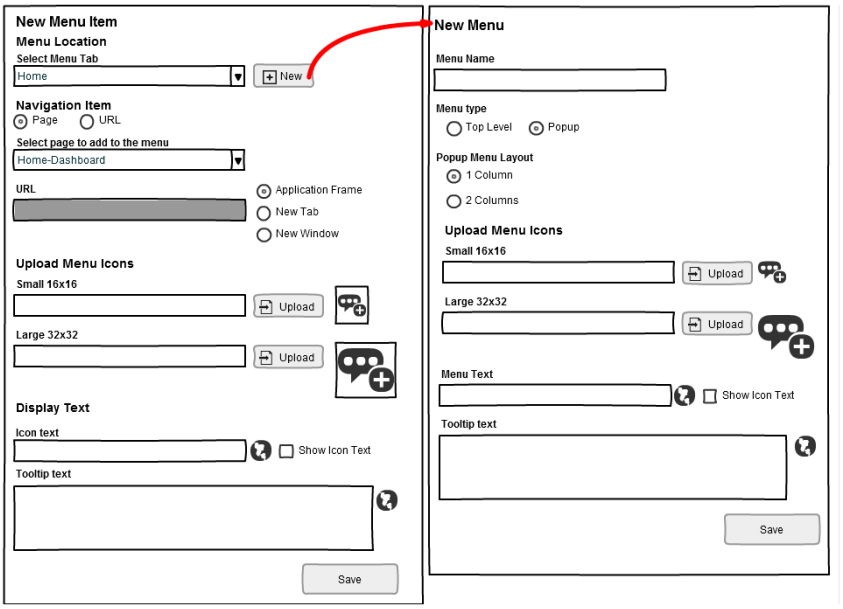
4.7 Multi Language

It is a requirement that the Web Application supports multiple languages. All aspects of the web application designer is to support the setting of different language translations. This relates to all controls that display fixed text such as labels, buttons and navigation.

4.8 Menu Designer

The menu designer forms an intrinsic part of the App design process. The designer enables the app designer to build the navigation that will be used at runtime to open and work with the web pages. The menu is to also support navigation to outside URL's.

The following image shows a possible layout for the menu designer page.



4.9 Custom permission functionality

4.9.1 Design Time

It is a requirement that portal would have implemented permission subsystem which could be used by graphical designers:

- Web designer
- Mobile designer

Both of designers are to have capability to create custom permission and assign them to visual control properties. For example, a designer may wish to have a button on a web page that authorizes a payment.

To support this, the designer must have functionality to:

- Create a permission
- Assign a permission to an action event such as Button Click
- Assign a permission to a control property such as Control Hide
- Check for a permission at runtime through JavaScript

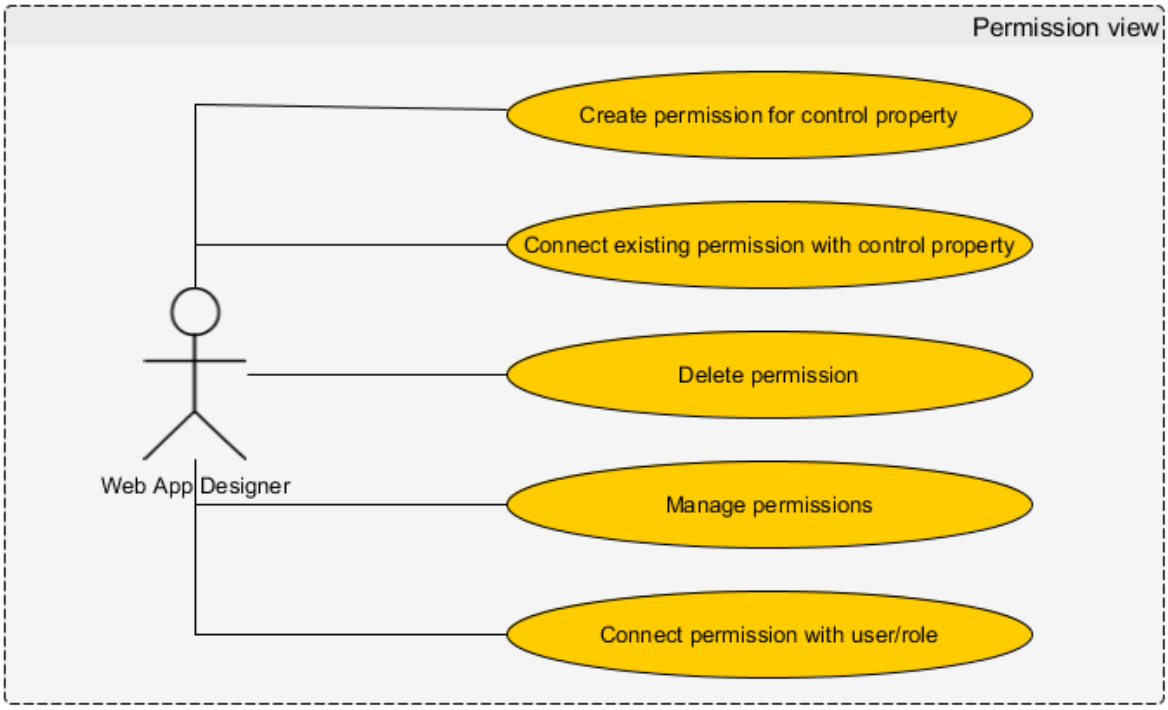
Permission should be shared between key platform features.

Custom permissions could be used in the designers and it will influence on other subsystem such as integration services/data objects services.

4.9.2 Setting Custom Permissions

It is a requirement that there is a special user interface in the platform which enables the setting of custom permissions to users/roles. This requirement is so that a system administrator can be given access to assign the permissions without having access to set platform permissions.

4.10 Use case diagram



4.11 Use case scenarios description

4.11.1 Create permission for control property.

The use case describes process which allow to create permission for control property.

1. **Web app designer** selects visual control at design canvas
2. **Web app designer** selects property to set permission e.g.: visibility
3. **Web app designer** submits button to create permission
4. **System** presents GUI to create permission
5. **System** presents inputs for permission name and short description
6. **Web designer** enters permission unique name and short description
    a. When permission name is not unique **system** rises exception
7. **Web app designer** submits confirmation button
8. **System** registers permission associated with control
9. When control is associated with data object **system** propagates permission to it
10. Flow ends

Pre-conditions

- **Web app designer** is logged to system

Post-condition

- Permission for control property has been registered

4.11.2 Connect existing permission with control property.

The use case describes process which allow to connect existing permission with control property.

1. **Web app designer** selects visual control at design canvas
2. **Web app designer** selects property to set permission e.g.: visibility
3. **Web app designer** submits button to bind existing permission
4. **System** shows lists of custom permission for tenant
5. **Web app designer** selects right permission
6. **Web app designer** confirm operation by submitting button
7. **System** registers association between permission and control property
8. Flow ends

Pre-conditions

- **Web app designer** is logged to system

Post-condition

- Association between permission and control property has been registered

4.11.3 Delete permission.

The use case describes process which allow to delete permission for control property.

1. **Web app designer** selects visual control at design canvas
2. **Web app designer** selects property to delete permission e.g.: visibility
3. **Web app designer** submits button to delete permission
4. **System** deletes permission associated with control
5. When control is associated with data object **system** deletes permission to it
6. When there are other controls bound to the permission system shows warning information and flow ends. It must not leave the system in an inconsistent state.
7. Flow ends

Pre-conditions

- **Web app designer** is logged to system

Post-condition

- Permission for control property has been deleted

4.11.4 Manage permissions.

The use case describes process which allow to manage permissions

1. **Web app designer** opens permission registry
2. **System** presents permission lists for particular tenant
3. **System** presents panel with followed buttons
    a. Connect permission
    b. Show permission details
4. Flow ends

Pre-conditions

- **Web app designer** is logged to system

Post-condition

4.11.5 Connect permission with user/role.

The use case describes process which allow to connect permission to user/role.

1. **Web app designer** opens permission registry
2. **System** presents permission lists for particular tenant
3. **Web app designer** selects permission for particular tenant
4. **System** presents lists of users and roles which could be associated with selected permission
5. **Web app designer** selects users, roles
6. **Web app designer** submits confirmation button
7. **System** binds permission with users, roles

8. Flow ends

Pre-conditions

- **Web app designer** is logged to system

Post-condition

- Permission has been connected with users roles

4.12 Import & Export

Support for Web App import and export is required. Where possible importing will support the merging of objects (such as data tables). The following files are required to be included in the Export/Import file.

- App Pages
- JavaScript
- CSS
- Custom Pages
- Navigation
- Data Objects

4.13 Social Features

The GO platform is to provide social features.

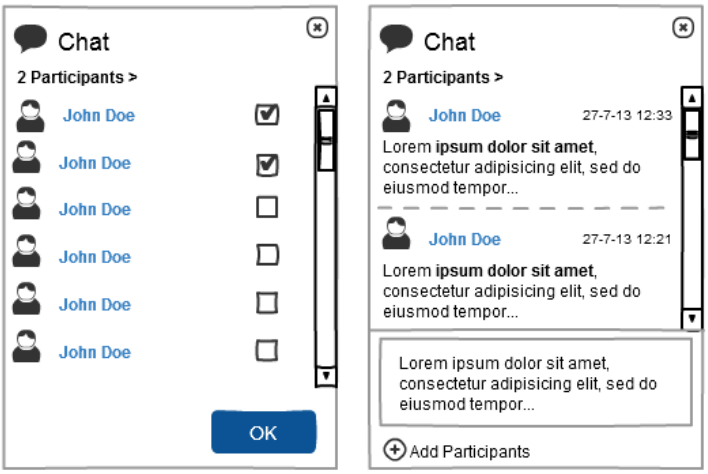
Both the Chat and Share features are accessed via the master page at runtime, with the Data Tag optional on each page through the page designer.

Note: As there may be high volume of social data created, progressive loading of data is to be implemented to ensure the UI does not become unresponsive.

4.13.1 Chat

Chat functionality is to be available (if turned on) to start a chat session with one or more tenant users.

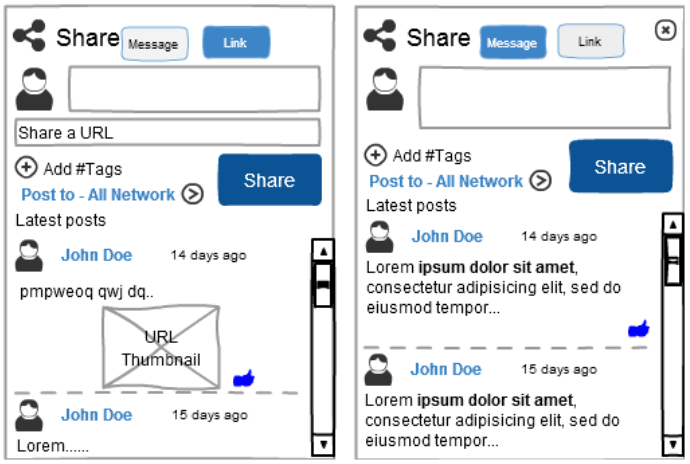
Below is an example of how the chat feature may look.



4.13.2 Share

The share feature is to be available (if turned on) to all tenant users. The feature allows for the user to share any kind of comment or web links with other users.

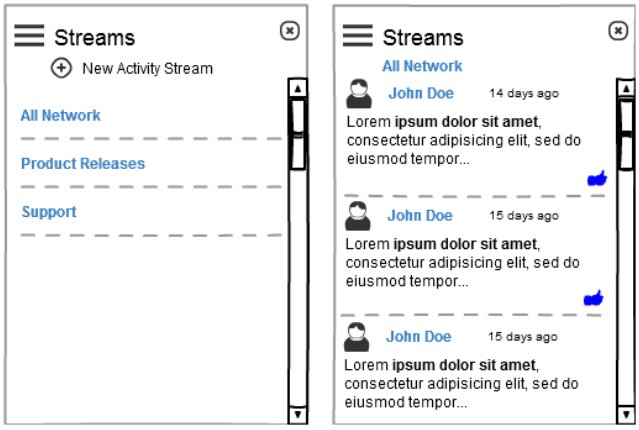
Below is an example of how the share feature may look.



4.13.3 Activity Streams

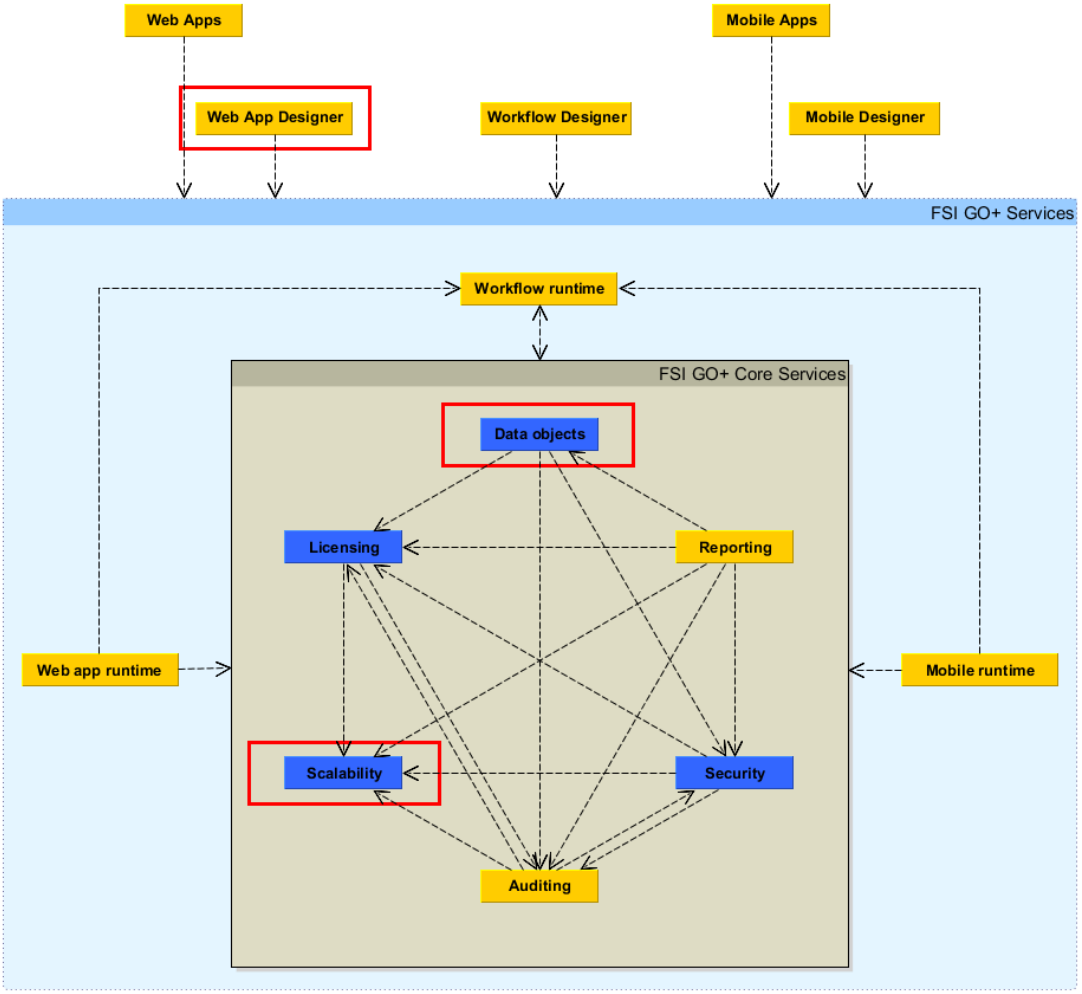
Activity Streams are simply used to group posts into logical streams. There is to be support for the creation of new Activity Streams and viewing all posts relating to the stream.

Note: The creation of new Activity Streams is to be protected with a permission.



4.13.4 Data Tags

To enable searching across application data, a new system field will be added to every Data Object. This field will support the addition of Tags for each data row.



Tags will be entered via the Tag control on the page designer, so it will be the responsibility of the page designer to add data tags where required.

The Tag control could be designed like the following, where the user types a tag and adds it to a list below the control.

The user is to be able to remove tags at any time.

4.14 Data Tag Search

It is a requirement to search across data that has been tagged in the system and return a list of data. The user will then be able to drill into the data by opening the required data view page.

Each Data Object row in the system can have a set of Data Tags associated with it (like #Asset, #Client, etc.). It is to enable searching by data tags in future versions of the GO+ Platform. However, it has to be implemented in Data Object layer and Web App Designer now, to avoid reimplementation of core subsystem and subsequent verification of whole system later.

4.15 Use case scenarios description

4.15.1 Edit Data Tags of a Data Object row

1. **Tenant administrator** selects Data Object rows using checkboxes in *selected* column.
2. **Tenant administrator** clicks *Data Tags* button.
3. **System** shows a window displaying number of rows being edited and a input box containing a union of all Data Tags of selected rows.
4. **Tenant administrator** modifies Data Tags using comma separated list format.
5. **Tenant administrator** click *save* button.

4.15.1.1 Pre-conditions

1. **Tenant administrator** successfully authenticated against the portal.
2. **Tenant administrator** navigated to Data Object Rows view.

4.15.1.2 Post-conditions

1. All selected Data Object rows have their Data Tags replaced by the tags provided by **Tenant administrator**.

4.16 Document Management

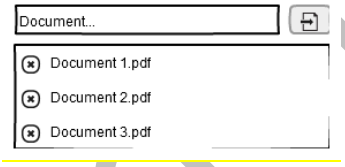
Documents are currently supported in FSI GO. Data object documents are stored in the data objects structure and are not the same as published documents.

With GO plus document management is to be an integral part of the Web App run time. It must be possible to:

- Add one or more documents to any record in a data object. A new system field/linker will be added to each Data Object
- Have standard user interface to manage documents
- Support version history
- Support folders
- Support file compression
- Full audit of who created, read and deleted documents.

Documents stored within the data model have to have maximum compression applied.

Uploading documents control for use in the page designer may be designed like the following.



4.17 Publishing

There is to be a publishing mechanism for Web Apps. This will support versioning and enable web sites to be published to the assigned Web App Runtime environment (see Runtime Execution for more information).

A Web App can also be published in either Test or Production mode, this will enable changes to be verified prior to general release. There can only be one version published at a time for either Test or Production.

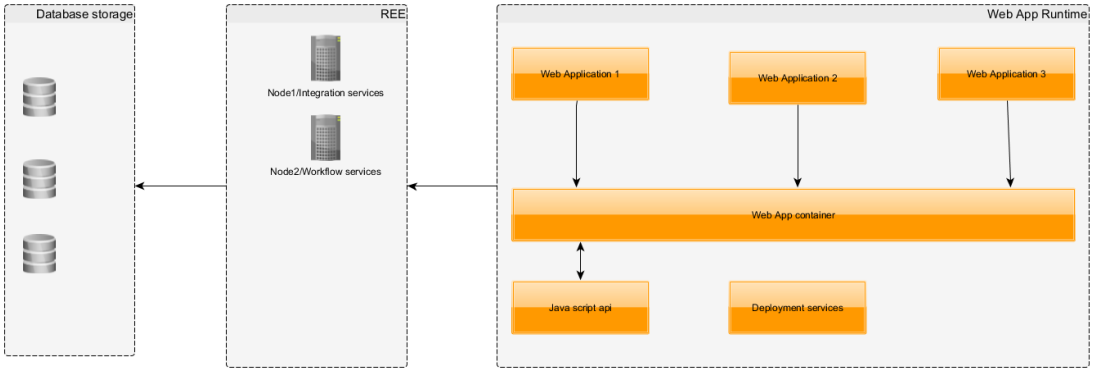
When a web application is published, the user is to be notified of the URL to access the application.

It is a requirement that a two stage prompt is made to ensure accidental publishing is prevented.

Note: The data model is shared between all mobile, and web applications within the tenant. If a client requires complete isolation, multiple tenants will be required.

5 Web App Runtime (WAR)

WAR is the container which hosts any application created/designed in Portal Web App Designer. *WAR* is to supports only one version of web application. *WAR* is to be self-hosted application working independently from portal application and services. *WAR* isn't to provide any business logic itself. Whole business logic is to be served by external services and *WAR* is to only consume them.



WAR is to have hot-deployment capability. It means that from portal there is possible to deploy web application (html template) remotely without restarting container.

Public deploy is a process which will be used to propagate the web application outside the portal ecosystem. In this mode the web application will be visible and accessible for tenant web users.

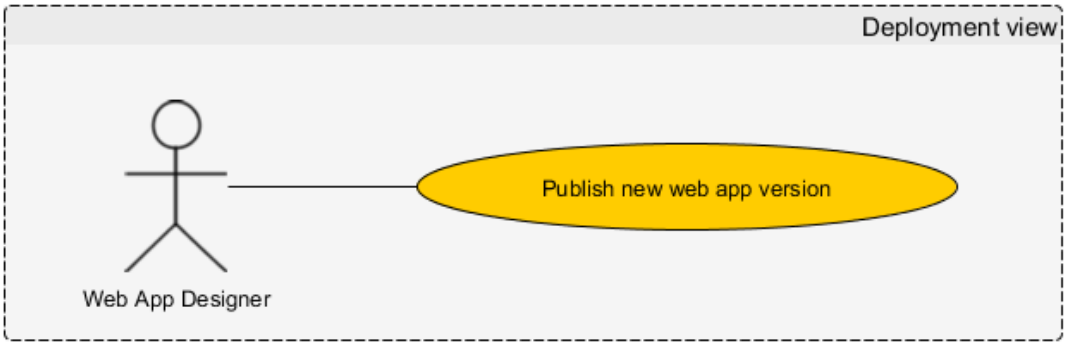
Assumptions

- All API calls from Web Apps will go through WAR service layer. WAR acts as a proxy.
- Html template transformation for security purpose will be realized in runtime by java script API.
- Publication of html template will be done directly into WAR.

WAR itself is to have a variety of built-in security models/options such as:

- Guest access
- Login page
- Self-registration

5.1.1 Use case diagram



5.1.2 Use case scenarios description

5.1.2.1 Publish new web app version.

The use case describes process which allow to deploy new version of web application for production purpose

1. **Web app designer** selects in the portal a button to deploy selected web application version into the **web app runtime**.
2. **System** connects to **web app runtime**
3. **System** deploys selected web application version with *public deployment* flag
4. **Web app runtime** deletes existing version of application
5. **Web app runtime** adds new application to the container
6. **System** presents **Web app designer** message that application has been published successfully.
7. Flow ends

Pre-conditions

- **Web app designer** is logged to the portal.

Post-condition

- New version of web app application has been installed. Application is ready to use.

5.2 Authentication

DThe runtime must present the user with the required authentication page and ensure access is not granted unless the user has been authenticated and has the required permissions.

Users will consume either a Named user or Concurrent user licence while they are using the web application.

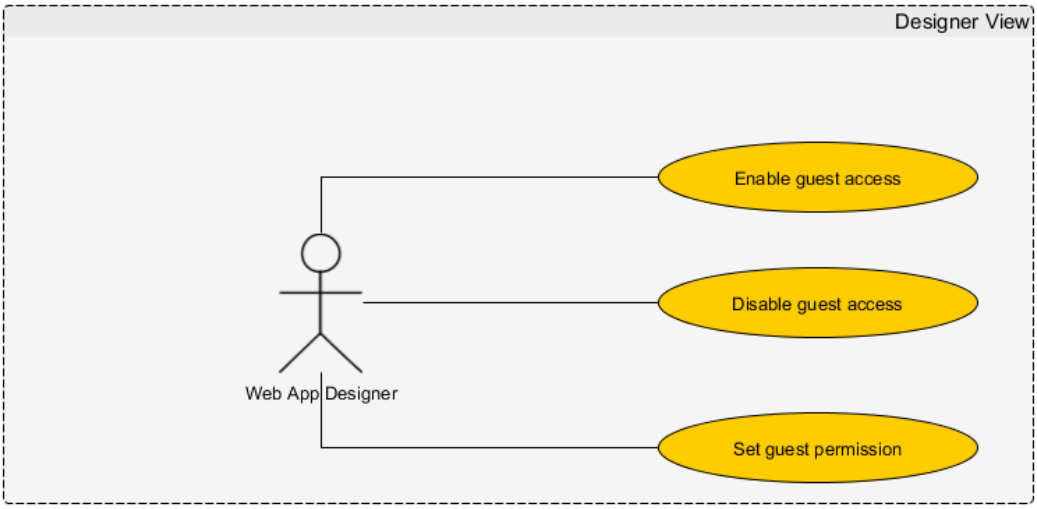
5.2.1 Guest Access

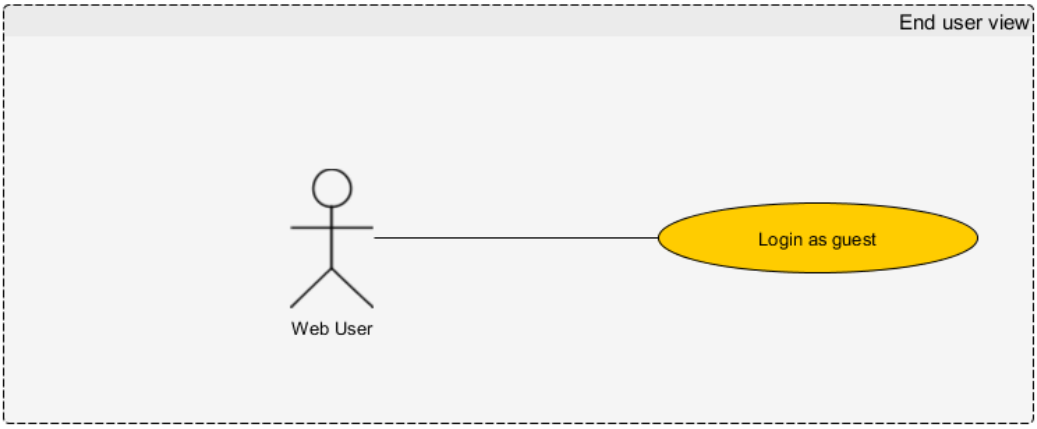
It is a requirement that web apps can be configured to allow Guest user access. This is particularly important for self-service or order processing types of applications.

The Application designer and or tenant administrator is to have the ability to assign which permissions and application features are available to the Guest users.

Each Guest user will consume one Guest Concurrent licence while using the web application.

5.2.2 Use case diagram





5.2.3 Use case scenarios description

5.2.3.1 Enable/Disable guest access

The use case describes process which enables or disables guest access for web application. Defaults guest access is disabled.

1. **Web app designer** in web application designer area submits button to enable/disable guest access for current developing application.
2. **System** saves configuration option for guest access.
3. Flow ends

Pre-conditions

- **Web app designer** is logged to the portal

Post-condition

- Configuration for guest access has been saved

5.2.3.2 Login as guest

The use case describes process which allow to login as guest.

1. **Web user** opens web application
2. **System** presents login screen
3. **System** presents button *login as guest* when application has this option enabled
4. **Web user** submits button *login as guest*
5. **System** logins user to the web app with guest privileges
6. Flow ends

Pre-conditions

- **Web app designer** is logged to the portal

Post-condition

- User has been logged to the system with guest privileges

5.2.4 Self-Registration

To support commercial applications it is a requirement that users are able to self-register an account with the system. Web apps will provide variety of options for self-registration process e.g.:

- With email confirmation option
- Without email confirmation option

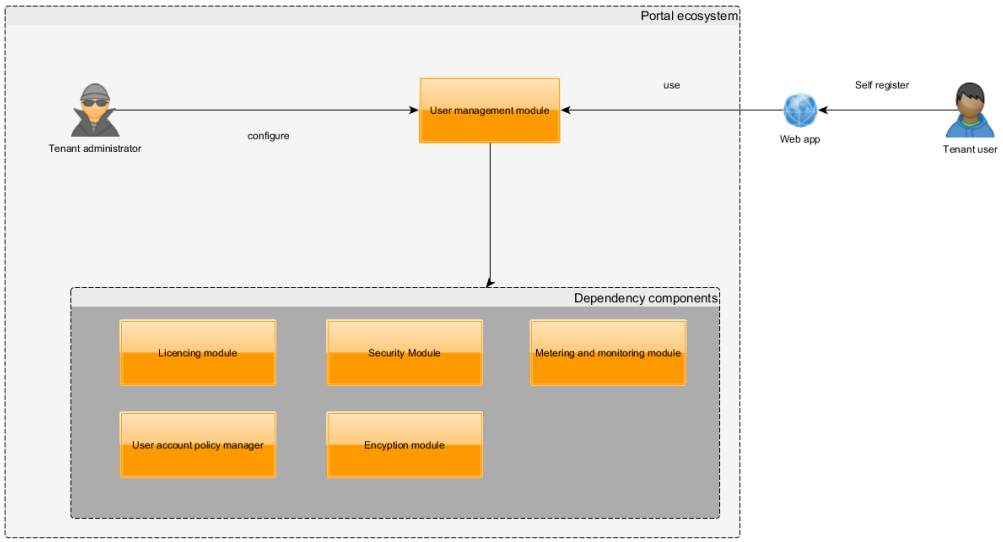
Self-registration will depends on licencing capabilities per tenant (*please refer to licencing chapter*). Self-registration-process will require the new **tenant-user** to fill required data e.g.:

- First name
- Last name
- Email address
- Password
- Captcha

All newly registered **tenant-users** will be assigned to default role (*please refer to roles and privileges chapter*). There is a requirement for an account request authorisation process that will ensure a **Web App Designer** or a nominated user can confirm the user and authorise. When **tenant-user** has been confirmed and authorised access will be enabled to log into the web apps with permissions assigned through the default granted role.

5.2.5 Portal self-registration ecosystem description

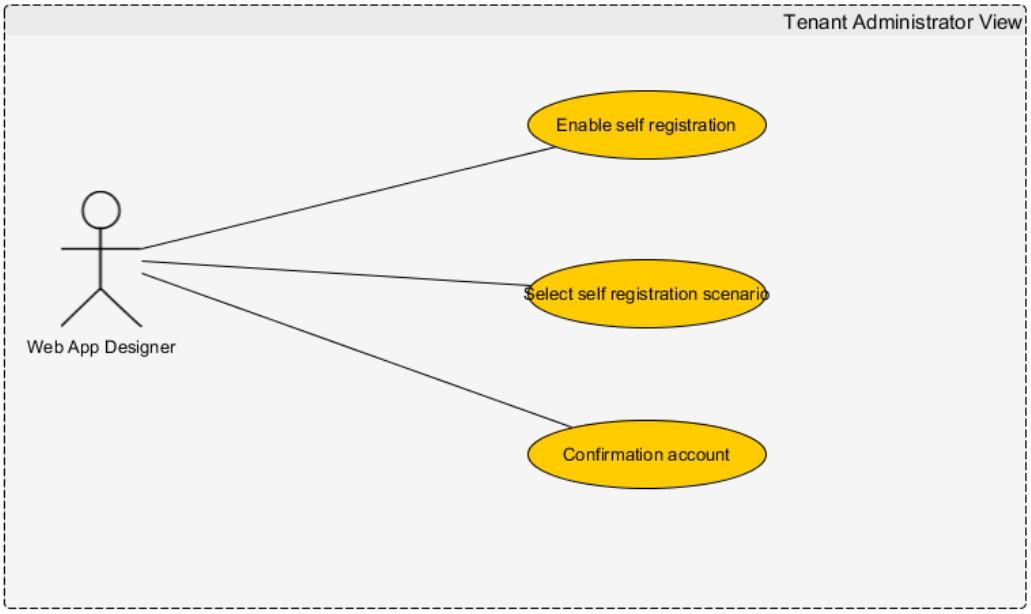
The following diagram presents the bird eye view on portal in context of self-registration



Self-registration process will use the following dependency modules

- Licencing module which for example controls access to register another user e.g.: check limits
- Metering and monitoring module which will check number of licences consumed
- User account policy manager which will set password expiration date
- Encryption module which will be responsible for password encryption (*please refer to encryption chapter*)

5.2.6 Use case diagram



5.2.7 Use case scenarios description

5.2.7.1 Confirmation account use case scenario.

1. **Portal Administrator** selects in portal menu page for confirmation and authorization user accounts.
2. **System** presents table with user accounts.
3. **Portal Administrator** selects user from table and activates account.
4. **Portal Administrator** confirms operation by submits button.
5. **System** changes user state from *inactive* to *active*.
6. Flow ends

Pre-conditions

- **Portal Administrator** is logged to the portal

Post-condition

- **Tenant user** account has been activated

5.2.7.2 Select self-registration use case scenario.

1. **Web App Designer** selects Web App setting option to manage self-registration scenario.
2. **System** presents list with self-registration scenarios e.g.:
    a. With email confirmation
    b. Without email confirmation
3. **Web App Designer** selects scenario from the list.
4. **Web App Designer** confirms operation by submits button.
5. **System** changes self-registration scenario.
6. Flow ends

Pre-conditions

- **Web App Designer** is logged to the portal

Post-condition

- Self-registration scenario has been changed

5.2.7.3 Tenant user registration use case.

1. **Tenant user** visits the registration page
2. **System** present form with registration data
    a. First name
    b. Last name
    c. Email address
    d. Password
    e. Password confirmation
    f. Captcha
3. **Tenant user** fills form data
4. **Tenant user** submits data by submit button
5. **System** registers new user with status *registered*
6. If chosen *without email confirmation* **system** change **tenant user** account status to *inactive* and flow ends
7. If chosen *with email confirmation scenario* **system** sends confirmation email to **tenant user**
    a. **Tenant user** clicks the confirmation link
    b. **System** changes **tenant user** account status to *inactive*
    c. Flow ends
8. Flow ends

Pre-conditions

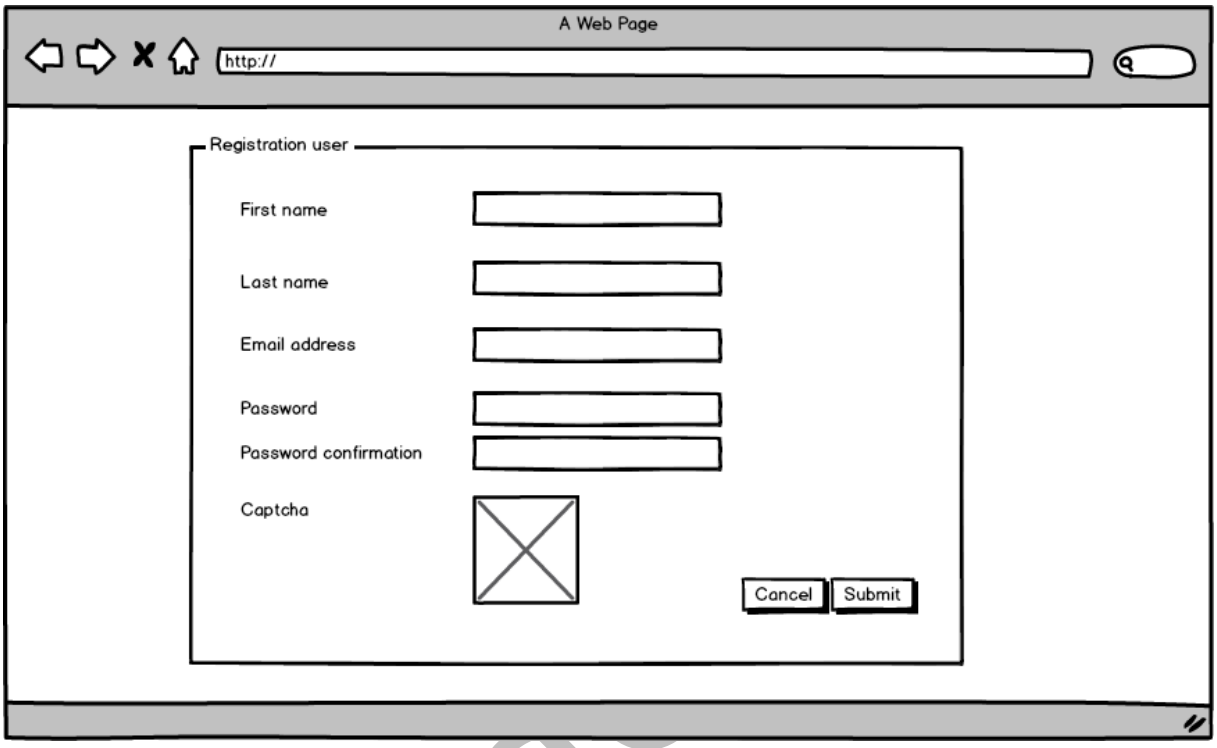
- License granted allows to self-registration

Post-condition

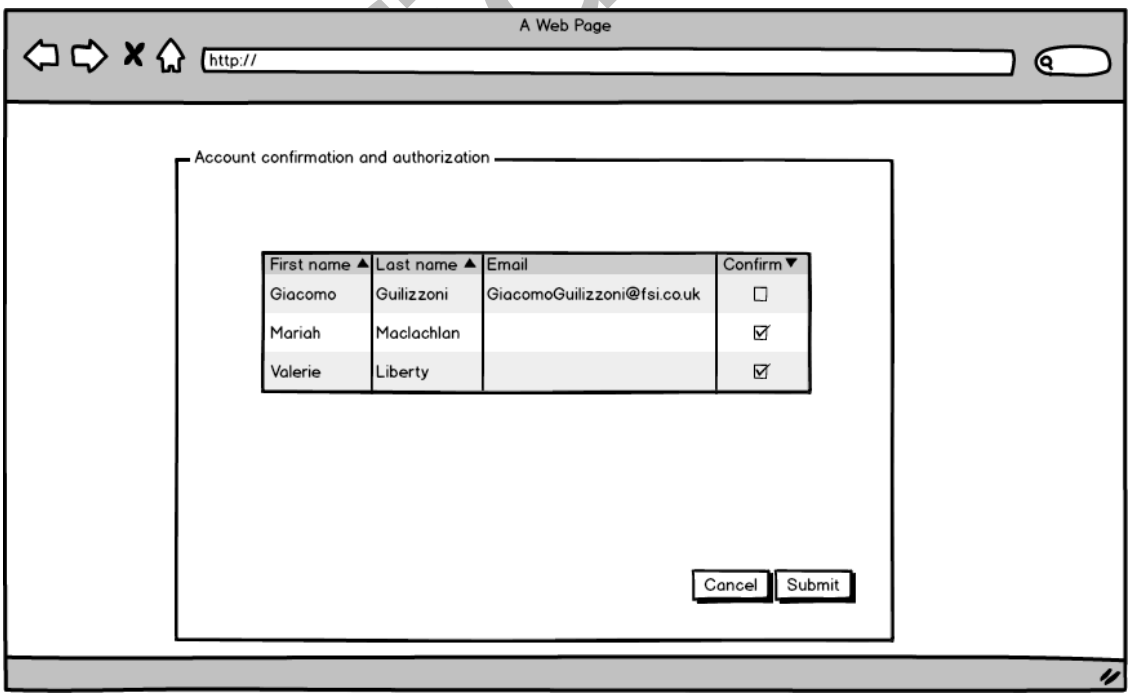
- The user tenant is *inactive*

5.2.8 User interface mockups

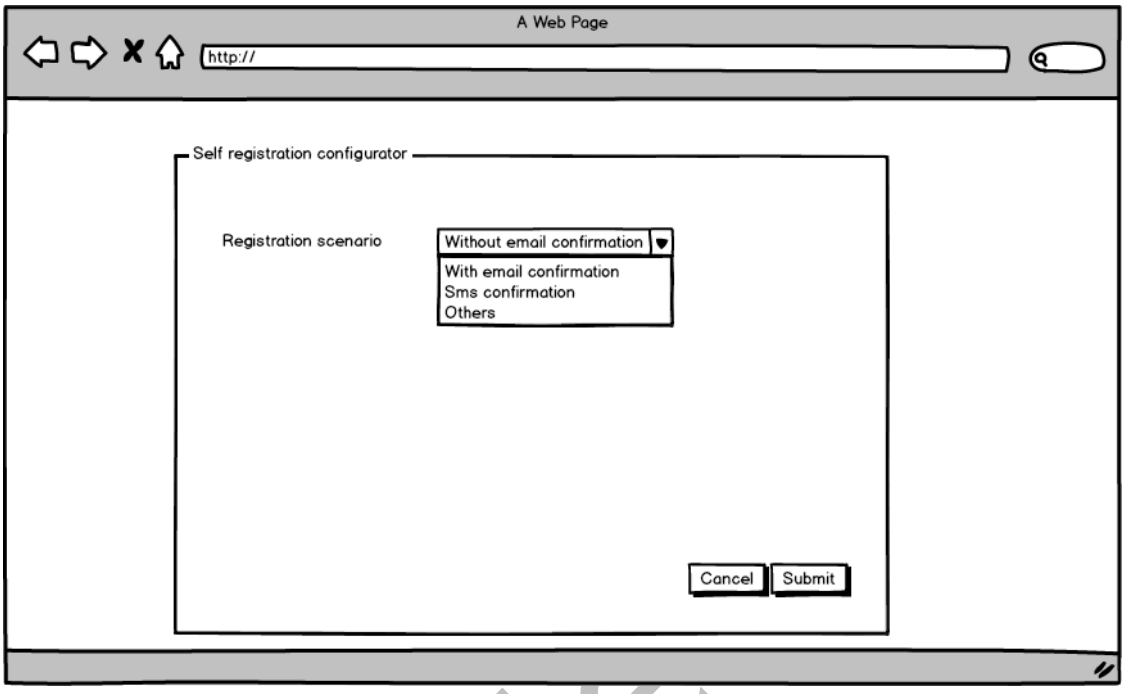
Tenant user self-registration user interface could looks like below.



Account confirmation and authorization user interface could looks like below.

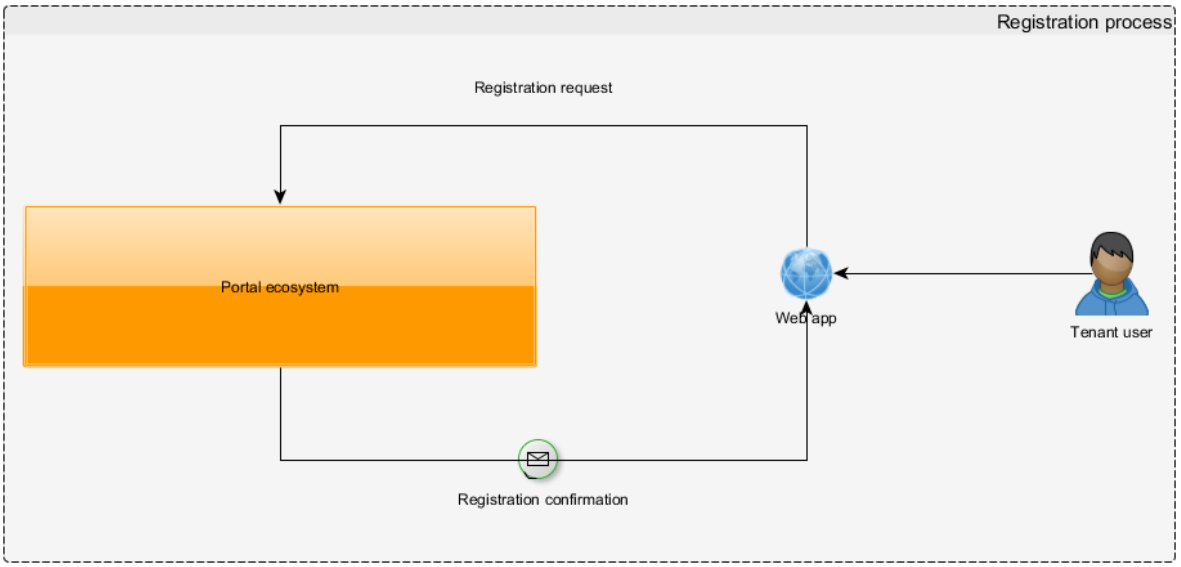


Self-registration configurator user interface could looks like below.



5.2.9 Registration process

The following diagram presents self-registration process with email confirmation option.



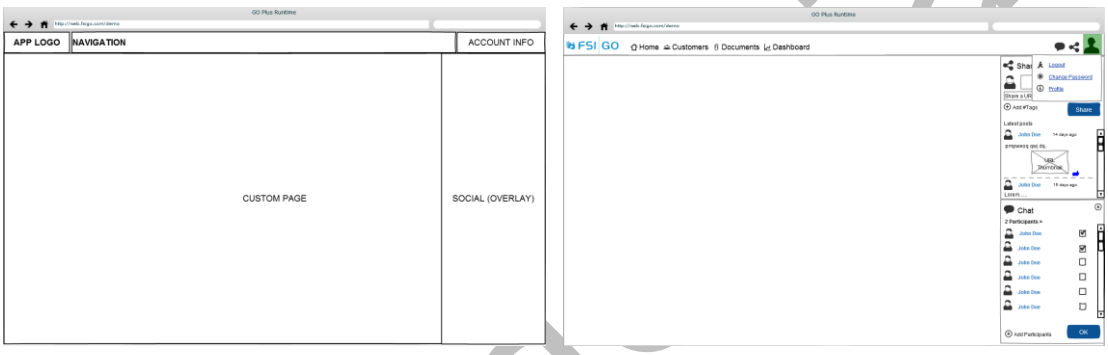
5.3 Master Page

The primary master page for displaying designed pages will be fixed as far as layout of app components.

The master page is to be simple in design, leaving as much real estate for the custom designed pages. There are five layout components for the Web App:

- App logo
- Navigation
- Account Info
- Custom Page
- Social Overlay

The components will be hosted in a master page with the following layout.

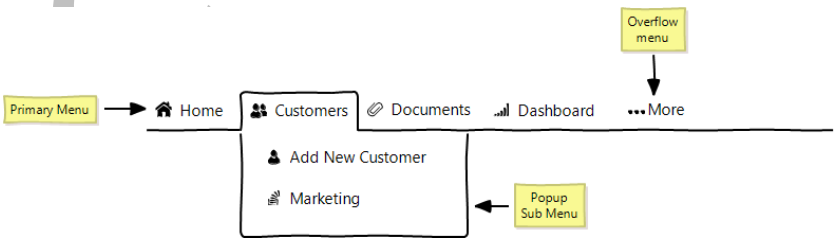


5.3.1 App Logo

The App Logo will be settable within the App designer. The App Logo Area will support an App logo of size of 40 by 160 pixels, and be importable through the App designer.

5.3.2 Navigation

Users access pages directly through the Menu Navigation components. Menus are designed using the menu designer and are included when the Web App is published. The following shows how the menu navigation may look.



5.3.3 Touch Design

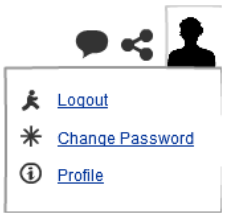
All areas of the Web Application Runtime need to be touch friendly. This includes:

- Large buttons
- Large hit areas on the navigation
- Large hit areas on links

5.3.4 Account Info

The Account info are hosts the following features

- Picture of logged on user with drop down menu
  - Link to Logout
  - Account details
  - Change password
- Link to open "Chat" feature
- Link to open "Share" feature



5.3.5 Social Overlay

The Social overlay area on the master page is where the Chat and Share functionality is overlaid on top of the custom page area and should include the ability for the area to be collapsible and pin/dock to the master page.

6 Workflow

The workflow feature of the platform provides a fully configurable component for the following high level functionality.

- Configurable Business logic
- Native support for Data Objects
- Connect 3rd party systems – cloud integration
- Database, web services connectivity

This is achieved through a graphical workflow designer.

6.1 Workflow Design & Management Area

The following functionality is accessed by each tenant through the portal. Only data relating to their tenant is to be available.

6.1.1 Workflow administration

The following functionalities for managing workflows need to be supported:

- **Tracing/Auditing and Monitoring**

  Provide details of executions, errors and warnings of each workflow visible to the authenticated user
- **Enable/Disable workflows**

  Provide the ability to enable and disable the workflow and set Valid from and to periods.
- **Versioning**

  Each workflow will be versioned with the ability for the user to check out, check in and roll back to a specific version. Each workflow node library will be versioned in the same way. There will be only one valid workflow definition version and one valid workflow node library version. When publishing new workflow node library version all workflow definitions and instances have to start using the library (e.g. Assembly Version Redirection has to be applied in the whole scope of the library).
- **Permissions**

  Control which users and roles can access, edit, delete and execute a workflow
- **Workflow triggers and scheduling management**

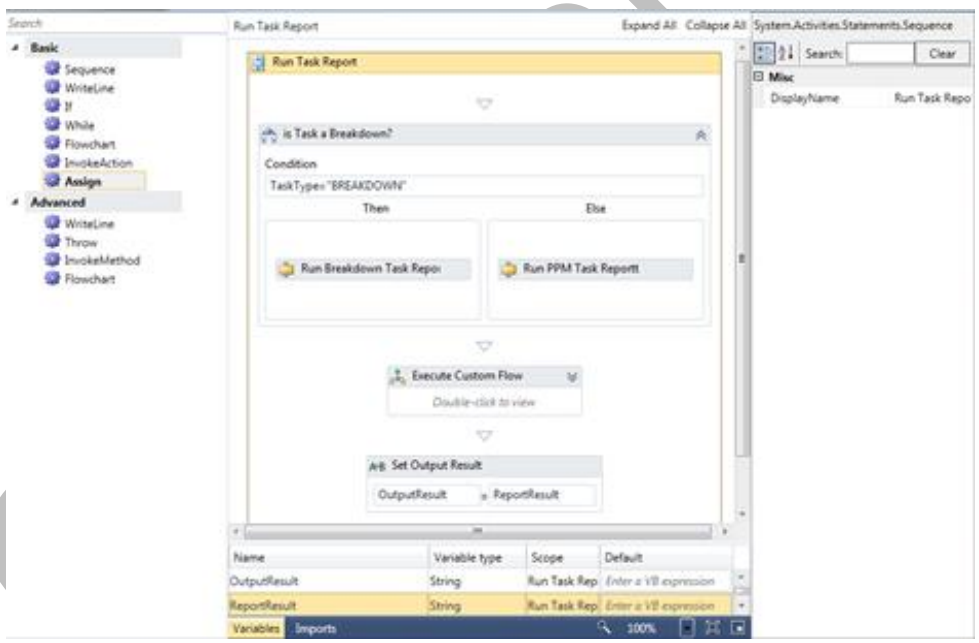
  Configure workflow triggers and define workflow schedules. A workflow trigger is a method to invoke the execution of a defined workflow.
- **Export and Import**

  Full import and export of workflows, workflow triggers and schedules.

6.2 Graphical Workflow Designer

A graphical workflow designer will be required to create workflows. The designer is to support the following functionalities:

- **List of available Workflow and Integration Nodes**
  Limited by licensing
- **Drag & Drop**
  Drag workflow nodes to the workflow designer space and link them together to describe execution path
- **Properties and variables**
  Used to store information/results to use within the workflow
- **Input & Output Parameters**
  Used to pass information to and from a workflow; must support simple and complex data types that can be bound to custom user interfaces defined in the Configurable UI area
- **Debugging**
  View a trace of the workflow as it executes put breakpoints and inspects parameters and variables.
- **Flow chart and State machine support**
  Designer will support two modes of creating workflows flowchart and state machine



The .Net 4.0 workflow designer control

Note: The workflow designer component supplied within the Microsoft .Net 4.0 Framework provides the building blocks of designing workflows. The component also ships with some default Nodes for flow control etc.

6.3 Use case scenarios description

This section contains use cases as in standard Workflow Foundation. If a solution that uses custom designer is proposed, those use cases have to be supported.

6.3.1 Create new workflow definition (flowchart)

Scenario describe creation of new workflow definition with loaded flowchart component

1. **Workflow designer** selects workflow activities in portal
2. **System** displays workflow administration panel
3. **Workflow designer** clicks *'create new flowchart workflow'* button
4. **System** displays graphical workflow designer with flowchart component loaded on designer workspace

Pre-conditions:

1. **Tenant** is logged to the portal
1. **Tenant** has workflow subsystem licensed
2. **Tenant** has flowchart component licensed

Post-conditions:

1. Workflow definition draft is created

6.3.2 Create new workflow definition (state machine)

Scenario describe creation of new workflow definition with loaded state machine component

1. **Workflow designer** selects workflow activities in Portal
2. **System** displays Workflow administration panel
3. **Workflow designer** clicks *'create new state machine workflow'* button
4. **System** displays graphical workflow designer with state machine component loaded on designer workspace

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
2. **Tenant** has sate machine component licensed

Post-conditions:

1. Workflow definition draft is created

6.3.3 Modify workflow definition

Scenario describe edit functionality for workflow definition

1. **Workflow designer** selects workflow activities in portal
2. **System** displays workflow administration panel
3. **System** displays list of available workflow definitions for modify (not used)
4. **Workflow designer** selects workflow definition for modify
5. **System** sets lock on workflow definition

6. **System** displays graphical workflow designer with workflow definition loaded to designer workspace
7. **Workflow designer** modifies workflow definition
8. **Workflow designer** clicks *'update workflow definition'* button
9. **System** updates workflow definition
10. **System** releases lock from workflow definition

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. Workflow definition is created and available for modify

Post-conditions:

1. Workflow definition is modified

6.3.4 Create workflow Input & Output arguments

Scenario describe creation of arguments for workflow definition

1. **Workflow designer** selects *arguments* tab on the bottom of the graphical workflow designer
2. **System** displays panel with list of arguments
3. **Workflow designer** clicks *'create new argument'* button
4. **System** creates new row in arguments list grid with default values
    a. **Name:** *distinct argument name*
    b. **Direction:** *In*
    c. **Property Type:** *string*

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition

Post-conditions:

1. New argument for workflow definition is created
2. Argument can be used in workflow definition context

6.3.5 Modify workflow Input & Output arguments

Scenario describes modification of workflow input and output arguments

1. **Workflow designer** selects *arguments* tab on the bottom of the graphical workflow designer
2. **System** displays panel with loaded list of input and output arguments for current workflow definition
3. **Workflow designer** selects argument (row in grid) for edit
4. **System** displays row in edit mode
5. **Workflow designer** provides distinct argument name (textbox)

6. **Workflow designer** selects direction property from *dropdown list (in, out, property)*
7. **Workflow designer** selects argument type from *dropdown list (basic types)*
8. **Workflow designer** provides default value for argument *(VB expression)*
9. **System** validates argument properties

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
4. Argument is added to workflow definition

Post-conditions:

1. Argument for workflow definition is modified

6.3.6 Create the workflow variable

Scenario describe creation of variable for workflow definition

1. **Workflow designer** selects *variable* tab on the bottom of the graphical workflow designer
2. **System** displays panel with loaded list of variables for current workflow definition
3. **Workflow designer** clicks *'create new variable'* button
4. **System** creates empty row in variable list grid and default values
   Name: *distinct argument name*
   Scope:
   Property Type: *string*

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition

Post-conditions:

1. New variable for workflow definition is created
2. Variable can be used in workflow definition context

6.3.7 Modify the workflow variable

Scenario describe edition of variables for workflow definition

1. **Workflow designer** selects *variable* tab on the bottom of the graphical workflow designer
2. **System** displays panel with loaded list of variables for current workflow definition
3. **Workflow designer** selects variable (row in grid)for edit

4. **System** displays row in edit mode
5. **Workflow designer** provides distinct variable name in textbox
6. **Workflow designer** selects scope property from *dropdown list (available scope)*
7. **Workflow designer** selects variable type from *dropdown list (basic types)*
8. **Workflow designer** provides default value for variable *(VB expression)*
9. **System** validates variable properties

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition

Post-conditions:

Variable for workflow definition is modified

6.4 Workflow Nodes

Workflow nodes provide the functional building blocks to define a workflow. Each node is predefined to complete a specific task and can interact with inputs/outputs from other nodes within the workflow. If not specified otherwise, the following use cases describe standard WF components. The main point is that custom codeless property binder has to be available when providing an expression for input parameters.

6.4.1 General

6.4.1.1 Add workflow node to designer workspace

User scenario describe functionality to add workflow node to workflow designer canvas

1. **Workflow designer** drag & drops selected workflow node from *nodes toolbox* to designer workspace.
2. **System** renders new node on designer workspace and update workflow definition draft

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition

Post-conditions:

1. New node is added to designer workspace and workflow definition draft

6.4.1.2 Select node on designer workspace

User scenario describe functionality to select workflow node from graphical workflow designer workspace

1. **Workflow designer** clicks workflow node on designer workspace
2. **System** marks selected node with different border on designer workspace
3. **System** displays node attributes in property window

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition

Post-condition:

1. Workflow node is selected

6.4.1.3 Link nodes on designer workspace

User scenario describe linking functionality nodes

1. **Workflow designer** mouse hovers on node on workflow workspace
2. **System** renders handles for node links
3. **Workflow designer** clicks on handle and drags to other node
4. **System** renders link between nodes with proper label

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
4. Al least two nodes are added to designer workspace

Post-conditions:

1. Link between nodes is rendered
2. Workflow definition draft is updated

6.4.1.4 Delete workflow node from designer workspace

User scenario describe functionality to delete workflow node from designer's workspace

1. **Workflow designer** selects workflow node from WF designer canvas
2. **Workflow designer** press 'delete' key
3. **System** removes selected workflow node from designer workspace
4. **System** removes all incoming and outgoing links related to node from designer workspace
5. **System** removes selected workflow node from workflow definition
6. **System** removes all incoming and outgoing links related to node from workflow definition

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
4. Workflow node is added to designer workspace

Post-conditions:

1. Workflow node is deleted from designer workspace and from workflow definition draft

6.4.1.5 Property binding add-on

Property building add-on is a feature to provide *user-friendly* binding values to node properties

1. **Workflow designer** selects workflow node on workflow canvas
2. **System** displays properties in property window
3. **Workflow designer** focus on property in property window
4. **System** displays box with list of available arguments, variables, properties, nodes outputs, predefined values within current workflow definition (filtered according to node property context)
5. **Workflow designer** selects item from list
6. **System** binds selected item to node's property

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
4. Workflow node is added to designer workspace

Post-conditions:

1. Value is bound to property

6.4.2 Flow Decision Node (WF's Flow Decision Activity)

6.4.2.1 Add Flow Decision Node

As in "Add workflow node to designer workspace" use case

6.4.2.2 Configure Flow Decision Node

User scenario describe use of Flow Decision node

1. **Workflow designer** double-clicks Flow Decision node placed on designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** provides first expression manually or by *property binding add-on*

4. **Workflow designer** provides operation manually or selects from predefined set: <,>,<=,>=,= …
5. **Workflow designer** provides second expression manually or by *property binding add-on*
6. **Workflow designer** clicks button 'apply'
7. **System** validates node properties

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
4. Flow Decision node is added to designer workspace

Post-conditions

1. Workflow definition draft is updated

6.4.2.3 Link Flow Decision node with other nodes

User scenario describe linking functionality of Flow Decision node

1. **Workflow designer** mouse hovers on Flow Decision node on workflow canvas
2. **System** renders handles for *true link* and *false link*
3. **Workflow designer** clicks on handle and drags to other node
4. **System** renders link between nodes with proper label

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has Flow Decision node licensed.
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Flow Decision node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.3 Case node (WF's Switch<T> Activity)

6.4.3.1 Add Case Node

As in "Add workflow node to designer workspace" use case

6.4.3.2 Configure Case Node

User scenario describe use of case node

1. **Workflow designer** double-clicks case node placed on designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in

3. **Workflow designer** selects type property form predefined set or browse for type
4. **Workflow designer** provides a *VB expression* into expression property (manual or by property binding add-on)
5. **System** validates provided expression

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has case node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Case node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.4 Loop Node (WF's ForEach<T> Activity)

6.4.4.1 Add Loop Node

As in "Add workflow node to designer workspace" use case

6.4.4.2 Configure Loop Node

User scenario describe use of Loop Node

1. **Workflow designer** double-clicks case loop placed od designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** selects type property form predefined set or browse for type
4. **Workflow designer** provides a VB expression into values property (manual or by property binding add-on)
5. **System** validates expressions

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has loop node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Loop node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated
2. In loop node scope is available variable *'item'* that corresponds to the iterated element in the collection (*values* property of loop node)

6.4.4.3 Add node into Loop Node body

User scenario describe functionality of adding activities (nodes) into Loop Node

1. **Workflow designer** clicks *'detail mode'* button on loop node
2. **System** displays loop node in detail mode
3. **Workflow designer** drag & drops node from nodes toolbox to body of loop node

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has loop node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Loop node is added to designer workspace

Post-conditions:

1. Node is added to body of loop node
2. Workflow definition draft is updated

6.4.4.4 Link Loop Node

As in "Link nodes on designer workspace" use case

6.4.5 Terminate Node (WF's TerminateWorkflow Activity)

6.4.5.1 Add Terminate Node

As in "Add workflow node to designer workspace" use case

6.4.5.2 Configure Terminate Node

User scenario describe use of terminate node

1. **Workflow designer** double-clicks terminate node placed on designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow Designer** provides a *reason* property manually or selects *reason* form predefined set
4. **System** validates expressions

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has terminate node licensed
4. **Workflow Designer** successfully opened existing workflow definition or created new workflow definition
5. Terminate node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.5.3 Link Terminate Node

As in "Link nodes on designer workspace" use case

6.4.6 Parallel Node (WF's Parallel Activity)

6.4.6.1 Add parallel Node

As in "Add workflow node to designer workspace" use case

6.4.6.2 Configure parallel Node

User scenario describe use of parallel node

1. **Workflow designer** double-clicks parallel node placed on designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow Designer** provides a *completion condition* property manually (VB expression)
4. **System** validates expressions

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has parallel node licensed
4. **Workflow Designer** successfully opened existing workflow definition or created new workflow definition
5. Parallel node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.6.3 Add node into parallel node body

User scenario describe functionality of adding (nodes) into parallel Node

1. **Workflow designer** clicks *'detail mode'* button on parallel node (on workflow designer canvas)
2. **System** display parallel node in detail mode
3. **Workflow designer** drag & drops node from nodes toolbox to body of loop node

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has parallel node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Parallel node is added to designer workspace

Post-conditions:

3. Node is added to body of parallel node
4. Workflow definition draft is updated

6.4.6.4 Link parallel Node

As in "Link nodes on designer workspace" use case

6.4.7 Invoke Workflow Node

6.4.7.1 Add Invoke Workflow Node

As in "Add workflow node to designer workspace" use case

6.4.7.2 Configure Invoke Workflow Node

User scenario describe use of invoke workflow node

1. **Workflow designer** double-clicks invoke workflow node placed on designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** selects available workflow instance to be invoked
4. **Workflow designer** selects return property from workflow
5. **System** validates expressions

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has invoke workflow node licensed
4. **Workflow Designer** successfully opened existing workflow definition or created new workflow definition
5. Invoke workflow node is added to designer workspace
6. **Tenant** has licensed different workflow instances

Post-conditions:

1. Workflow definition draft is updated

6.4.7.3 Link Invoke Workflow Node

As in "Link nodes on designer workspace" use case

6.4.8 SMS Node

6.4.8.1 Add SMS Node

As in "Add workflow node to designer workspace" use case

6.4.8.2 Configure SMS Node

User scenario describe use of SMS node

1. **Workflow designer** double-clicks SMS node placed od designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** provide *recipient* property
4. **Workflow designer** provide *originator* property
5. **Workflow designer** provide *replay e-mail* property
6. **Workflow designer** provide *text* property
7. **System** validates expressions

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has SMS node licensed
4. **Workflow Designer** successfully opened existing workflow definition or created new workflow definition
5. SMS node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.8.3 Link Invoke Workflow Node

As in "Link nodes on designer workspace" use case

6.4.9 SQL Service Reporting Service Node

Execute an SSRS report and retrieve PDF to be used as an attachment or write to a path/ftp

6.4.9.1 Add SSRS Node

As in "Add workflow node to designer workspace" use case

6.4.9.2 Configure SSRS Node

User scenario describe use of file SSRS node

1. **Workflow designer** double-clicks SSRS node placed od designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** select SSRS from *Tenant Context* to be executed
4. **Workflow designer** provides parameters for SSRS manually or by *property binding add-on*
5. **Workflow designer** enters *path* of file, or selects a path from *Tenant Context,* or build a path with path builder where SSRS result as pdf file will be stored
6. **System** validates expressions

Pre-conditions:

1. **Tenant** is logged to the portal

2. **Tenant** has workflow subsystem licensed
3. **Tenant** has SSRS node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. SSRS node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.9.3 Link SSRS Node

As in "Link nodes on designer workspace" use case

6.4.10 File Writer Node

Write flat files to a specified file path relative tenant specific file storage. No paths outside this storage are allowed. No other file system resources as disk drives, network paths, nor external drives are available.

6.4.10.1 Add File Writer Node

As in "Add workflow node to designer workspace" use case

6.4.10.2 Configure File Writer Node

User scenario describe use of file writer node

1. **Workflow designer** double-clicks file writer placed od designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** enters *path* of file, or selects a path from *Tenant Context,* or build a path with path builder
4. **Workflow designer** selects encoding type form predefined set of encodings.
5. **Workflow designer** choose policy if file exist
    a. Overwrite file
    b. Append content
6. **System** validates expressions

Pre-conditions:

6. **Tenant** is logged to the portal
1. **Tenant** has workflow subsystem licensed
2. **Tenant** has file writer node licensed
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
4. File writer node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.10.3 Link File Writer Node

As in "Link nodes on designer workspace" use case

6.4.11 String Function Node

Length, compare, equals, sub string, trim etc.

6.4.11.1 Configure String Function Node

User scenario describe use of code node

1. **Workflow designer** double-clicks code node placed od designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** provides string variable manually or by *property binding add-on*
4. **Workflow designer** selects function witch will be executed on *string variable*
5. **Workflow designer** provides parameters for selected function
6. **System** validates parameters

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has string function node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. String function node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.12 Date Function Node

Now, today, date add, UTC, time zone, conversion etc.

6.4.12.1 Configure Date Function Node

Similar to 'Configure String Function Node;

6.4.13 Number Function Node

Add, compare, divide, floor etc. Formula style

6.4.13.1 Configure Number Function Node

Similar to 'Configure String Function Node;

6.4.14 Executing User Info Node

Retrieve at runtime the user executing the workflow

6.4.15 Code node

6.4.15.1 Add Code Node

As in "Add workflow node to designer workspace" use case

6.4.15.2 Configure code node

User scenario describe use of code node

1. **Workflow designer** double-clicks code node placed on designer workspace
2. **System** displays new window showing a required text field to enter code block.
3. **Workflow designer** provide VB expression to be executed
4. **System** validate provided code

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has code node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Code node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

Special Requirements:

1. At runtime workflow instance will be persisted after *code* node execution.

6.4.15.3 Link code node

As in "Link nodes on designer workspace" use case

6.4.16 HTML Node

Designer to build HTML output (use in emails etc.)
The designer will support variables (like a mail merge) to build content at runtime

6.4.16.1 Add HTML Node

As in "Add workflow node to designer workspace" use case

6.4.16.2 Configure HTML Node

User scenario describe use of set variable node

1. **Workflow designer** double-clicks HTML node placed od designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** provides variables names for merge with html output

4. **Workflow designer** provides content in *HTML "WYSIWYG" Designer* including placeholders for variables
5. **System** validates expressions

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has HTML node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. HTML node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.16.3 Link HTML Node

As in "Link nodes on designer workspace" use case

6.4.17 Set Variable Node (WF's Assign Activity)

Node can be used to set workflow variables

6.4.17.1 Add Set Variable Node

As in "Add workflow node to designer workspace" use case

6.4.17.2 Configure Set Variable Node

User scenario describe use of set variable node

1. **Workflow designer** double-clicks set variable node placed od designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** provides variable name manually or by *property binding add-on*
4. **Workflow designer** provides VB expression result which will be assigned to variable
5. **System** validates expressions

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has set variable node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Set variable node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.17.3 Link Set Variable Node

As in "Link nodes on designer workspace" use case

6.4.18 Try Catch Node (WF's Try/Catch Activity)

6.4.18.1 Add Try Catch Node

As in "Add workflow node to designer workspace" use case

6.4.18.2 Add node into Try section of Try Catch node

User scenario describe functionality of adding (nodes) into *Try section* of *Try Catch node*

1. **Workflow designer** clicks *'detail mode'* button on try catch node (on workflow designer canvas)
2. **System** displays try catch node in detail mode
3. **Workflow designer** drag & drops node from nodes toolbox to try section of try catch node

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has try catch node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Try catch node is added to designer workspace

Post-conditions:

1. Node is added to try section of try catch node
2. Workflow definition draft is updated

6.4.18.3 Add node into Catch section of Try Catch node

User scenario describe functionality of adding (nodes) into *catch section* of *Try Catch node*

1. **Workflow designer** clicks *'detail mode'* button on try catch node (on workflow designer canvas)
2. **System** displays try catch node in detail mode
3. **Workflow designer** select exception which trigger execution of *catch* block
4. **Workflow designer** drag & drops node from nodes toolbox to catch section of try catch node

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has try catch node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Try catch node is added to designer workspace

Post-conditions:

1. Node is added to catch section of try catch node
2. Workflow definition draft is updated

6.4.18.4 Add node into Finally section of Try Catch node

User scenario describe functionality of adding (nodes) into *catch section* of *Try Catch node*

1. **Workflow designer** clicks *'detail mode'* button on try catch node (on workflow designer canvas)
2. **System** displays try catch node in detail mode
3. **Workflow designer** select exception which trigger execution of *finally* block
4. **Workflow designer** drag & drops node from nodes toolbox to finally section of try catch node

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has try catch node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Try catch node is added to designer workspace

Post-conditions:

1. Node is added to finally section of try catch node
2. Workflow definition draft is updated

6.4.18.5 Link Try Catch Node

As in "Link nodes on designer workspace" use case

6.4.19 Trace Node

6.4.19.1 Add Trace Node

As in "Add workflow node to designer workspace" use case

6.4.19.2 Configure Trace Node

User scenario describe use of delay node

1. **Workflow designer** double-clicks Trace node placed od designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** provides a *log message* manually or by *property binding add-on*
4. **System** validates expressions

Pre-conditions:

1. **Tenant** is logged to the portal
2. **Tenant** has workflow subsystem licensed
3. **Tenant** has trace node licensed
4. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
5. Trace node is added to designer workspace

Post-conditions:

1. Workflow definition draft is updated

6.4.19.3 Link Trace Node

As in "Link nodes on designer workspace" use case

6.4.20 Delay Node (WF's Delay Activity)

6.4.20.1 Add Delay Node

As in "Add workflow node to designer workspace" use case

6.4.20.2 Configure delay Node

User scenario describe use of delay node

1. **Workflow designer** double-clicks code node placed od designer workspace
2. **System** displays new window showing required and optional parameters. Required parameters are highlighted and have to be filled in
3. **Workflow designer** provides a delay *value(timespan)* manually or by *property binding add-on*
4. **System** validates expressions

Pre-conditions:

6. **Tenant** is logged to the portal
7. **Tenant** has workflow subsystem licensed
8. **Tenant** has delay node licensed
9. **Workflow designer** successfully opened existing workflow definition or created new workflow definition
10. Delay node is added to designer workspace

Post-conditions:

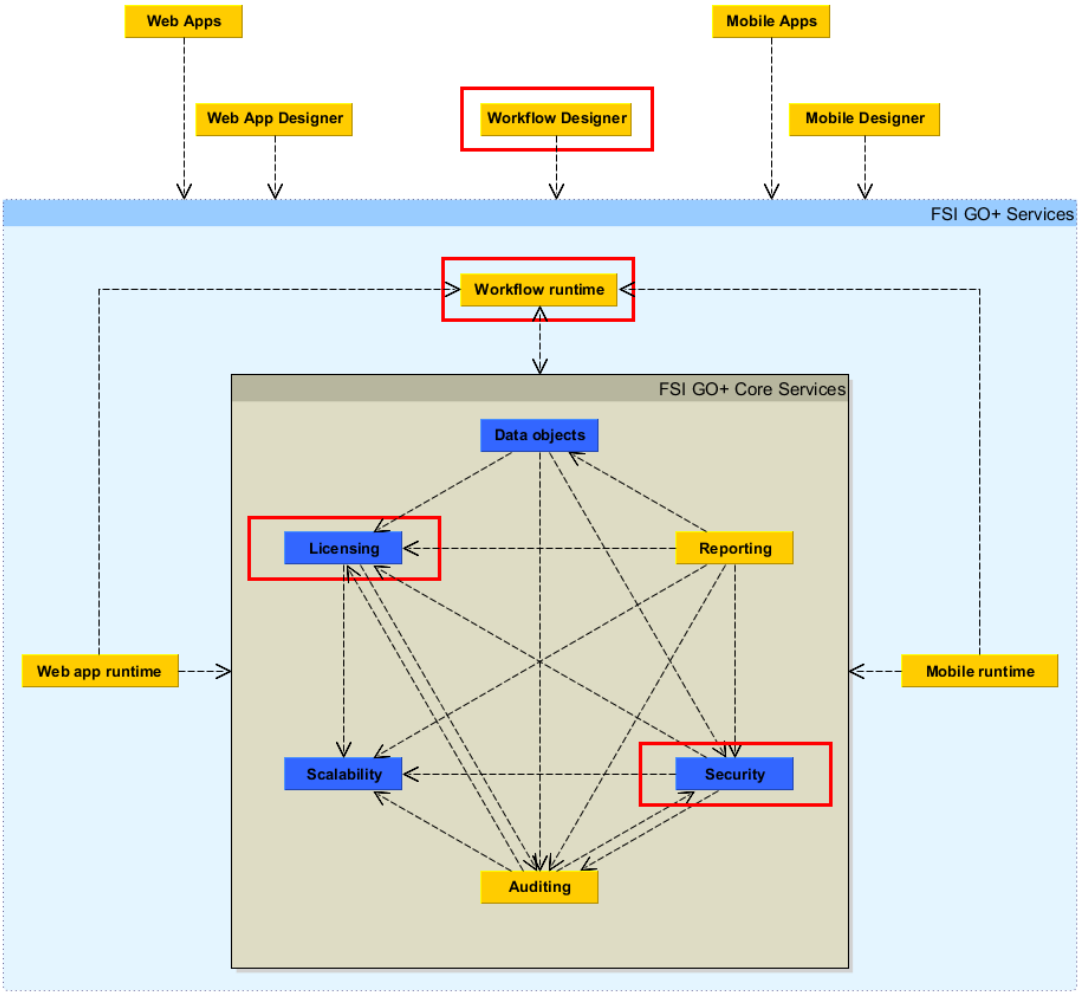
1. Workflow definition draft is updated

6.4.20.3 Link Delay Node

As in "Link nodes on designer workspace" use case

6.5 Integration Nodes

Besides *Code Node* which allow to implement arbitrary communication between systems, there has to be a set of user-friendly nodes that provide code free methods to implement common integration scenarios.



Integration nodes are used within Workflow and are special nodes that connect to other systems. The following integration nodes are required:

- Web Services
  - REST
  - SOAP
- OData Read
- CSV Read/Write
- Xml Read/Write
- FTP / SFTP Read/Write
- RSS Reader

- Email
  - POP / IMAP / TLS
  - Google
  - Exchange Web Services, Office 365
- Calendar
  - Exchange / iCAL
  - Google, Office 365
- Database
  - ODBC
  - SQL/Oracle Native
  - HADOOP (Big Data)
- Active Directory
- Social Media
  - Facebook, Twitter
  - LinkedIn, Google+

As an example, workflow may need to connect to Building Management Systems, Energy Systems, BIM and CAD data.

Integration Nodes require common facilities. The facilities are provisioned outside FSI GO+ but configured through Workflow Administration Panel:

- temporary files storage shared between load-balanced machines,
- certificates store accessible to all load balanced machines (for consuming web services requiring client certificate authentication),
- credentials store accessible to all load balanced machines,
- email server.

Temporary files are workflow instance scoped and are not shared between workflow instances. Instance specific files have to be cleaned-up when removing (active or terminated) workflow instance from the system. Temporary files of all workflow instances of given workflow definition have to be cleaned-up when the definition is removed from the system. Total size of temporary files is recorded for licensing purposes.

Security Context containing originating user information is accessible from any node in the instance and is automatically passed to workflows invoked downstream. *Security Context* contains *Original Caller*, *Direct Caller*, and *Caller Chain*. Examples:

- If a web app invokes workflow through JavaScript, then Original Caller and Direct Caller are the same identity that is of the user "X" who's using the app.
  Security Context = { Original Caller: X, Direct Caller: X, Caller Chain: [] }
- If a Data Object trigger invokes a workflow, then Original Caller is the identity of a user "X" who modified the DataObject, and Direct Caller is the

identity of triggering service "T".
Security Context = { Original Caller: X, Direct Caller: T, Caller Chain: [X] }

- If the above workflow invokes another workflow, then Original Caller is taken from the security context of invoking workflow, and Direct Caller is the identity of workflow service "W".
  Security Context = { Original Caller: X, Direct Caller: W, Caller Chain: [X, T] }

The purpose of *Security Context* is to allow workflow designer to decide against which identity to authorize operations. Original user identity is useful for reports – different users see different results. Service identity is useful for custom auditing – users don't need access to audit tables.

Every node that tries to access resources which require authorization has an option to select *Original Caller* or *Direct Caller* as an identity on which behalf to access a resource: temporary file, invocation of another workflow, etc.

Tenant Context containing tenant-wide configuration settings (connection strings, web service endpoints, shared storage URIs) is accessible from any node to support portability between production and test environments.

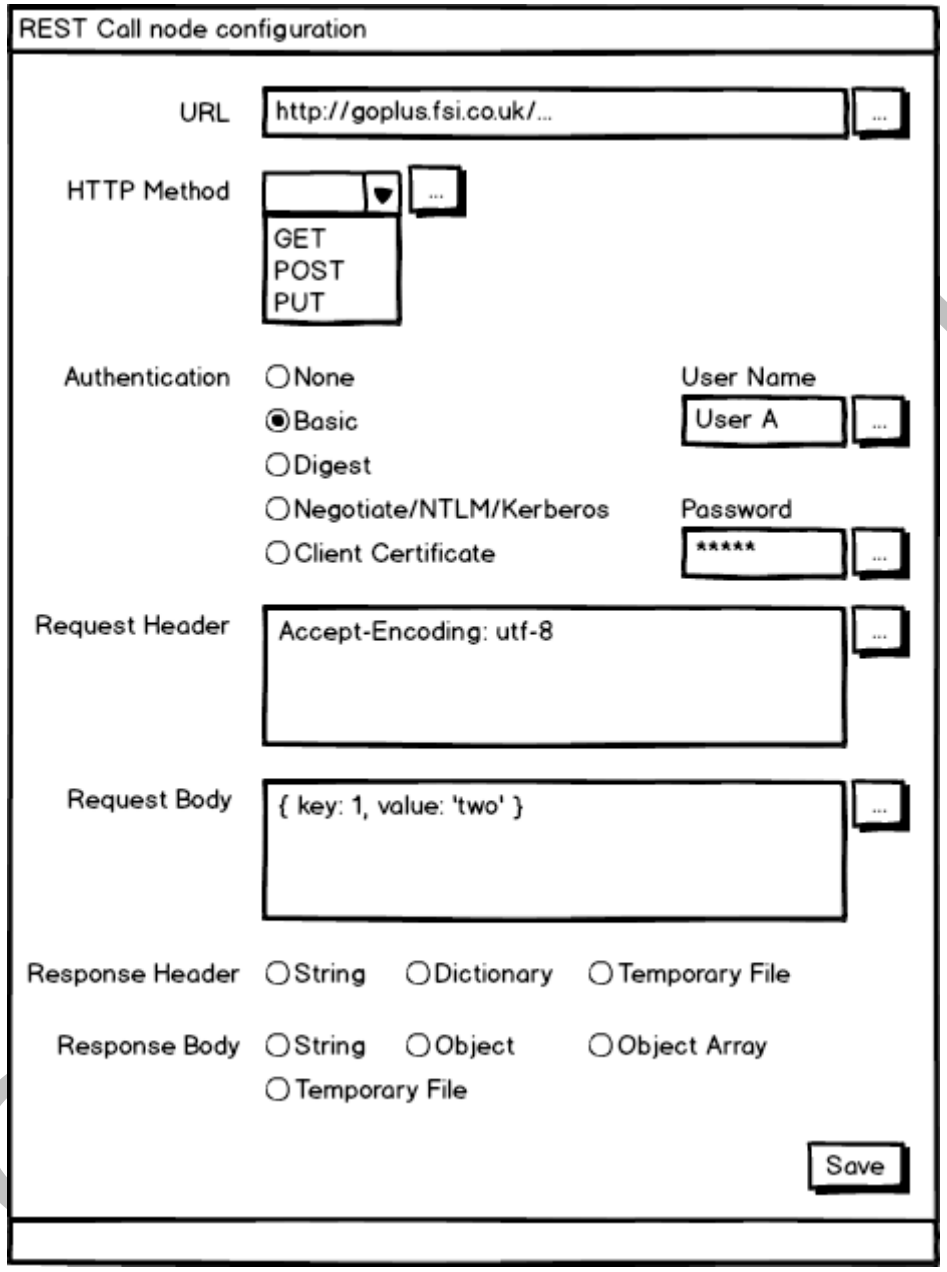
It is up to the workflow designer to decide, if trace records for a workflow definition should be encrypted. If he switches the setting *Parameter values in trace* to *Encrypted*, then every input and output parameter of every node comprising the definition will be encrypted in the trace output. Otherwise, every parameter value will be traced in clear text. The only exception is *Tenant Context* node which is exempted from input and output parameter tracing irrespective of encryption settings.

6.6 Use case scenarios description

Only *Configure Node* use cases are described in this section. Every node has *Add Node to Designer Workspace* and *Remove Node from Designer Workspace* use cases which are described in *Workflow Nodes* section.

6.6.1 Configure REST Call node

This node can be used to download files from any external and internal URL and make their contents available in current workflow instance scope.



1. **Workflow designer** double-clicks *REST Call* node placed on designer workspace.
2. **System** presents a new window showing required and optional parameters. Required parameters are highlighted and have to be filled in.
3. **Workflow designer** enters *URL* of REST endpoint, or selects a URL from *Tenant Context*, or uses *URL Builder* (see *Configure URL Builder* use case).
4. **Workflow designer** enters *HTTP Method*, or selects one from predefined set: *GET*, *POST*, *PUT*, *DELETE*, *HEAD*, *OPTIONS*, *TRACE*, *CONNECT*.

5. **Workflow designer** selects *Authentication* from predefined set: *None*, *Basic*, *Digest*, *Negotiate/NTLM/Kerberos*, *Client Certificate*.
    a. If *Basic* is selected then *User Name* and *Password* has to be provided.
    b. If *Digest* is selected then *User Name* and *Password* has to be provided.
    c. If *Negotiate/NTLM/Kerberos* is selected then *User Name* and *Password* has to be provided, or *Workflow Service Identity* has to be selected.
    d. If *Client Certificate* is selected then *String*, or *Byte Array* with certificate contents has to be provided, or certificate from *Tenant Certificate Store* has to be selected
6. **Workflow designer** enters *HTTP Request Header*, or selects its source from predefined set: *String Variable*, *Dictionary Variable*, *Temporary File*.
7. **Workflow designer** enters *HTTP Request Body*, or selects its source from predefined set: *String Variable*, *Object Variable*, *Temporary File*. If *Object Variable* is selected then **Workflow designer** additionally selects *Serializer* from predefined set: *XML*, *JSON*.
8. **Workflow designer** enters *HTTP Response Body Output Type* from predefined set: *String Variable*, *Object Variable, Object Array Variable*, *Temporary File*.
    a. If *Object Variable* is selected then **Workflow designer** additionally selects *Deserializer* from predefined set: *XML*, *JSON*.
    b. If *Object Array Variable* is selected then a top level array
9. **Workflow designer** enters *HTTP Response Header Output Type* from predefined set: *String Variable*, *Dictionary Variable, Temporary File*.
10. **Workflow designer** clicks *Preview* button.
11. **System** presents response of sample data with data schema applied.
12. **Workflow designer** clicks *Save* button.

6.6.1.1 Pre-conditions

3. **Tenant** has workflow subsystem licensed.
4. **Tenant** has *REST Call* node licensed.
5. **Workflow designer** successfully opened existing workflow definition or created new workflow definition.
6. **Workflow designer** had *REST Call* node added to designer workspace.

6.6.1.2 Post-conditions

1. **Workflow designer**'s draft of workflow definition is updated to include *REST Call* node.

6.6.1.3 Special Requirements

1. *REST Call* node will be the base node for custom web service nodes. It has to be possible to use the node as a base class for custom nodes. Moreover, it has to provide a way to customize authentication, header, and body processing for derived nodes.

2. At runtime *REST Call* node execution fails if the HTTP response code is interpreted by the .NET Framework as error code.
3. At runtime workflow instance will be persisted after *REST Call* node execution.
4. At runtime original caller's access rights to temporary storage, credential store, or certificate store will be verified.

6.6.2 Configure SOAP Call node

1. **Workflow designer** double-clicks *SOAP Call* node placed on designer workspace.
2. **System** presents a new window showing required and optional parameters. Required parameters are highlighted and have to be filled in.
3. **Workflow designer** enters *WSDL URL* of SOAP endpoint, or selects a URL from *Tenant Context*, or uses *URL Builder* (see *Configure URL Builder* use case).
4. **Workflow designer** select *SOAP Method* from the set available from selected WSDL.
5. **Workflow designer** selects *Authentication* from predefined set: *Basic*, *Digest*, *Negotiate/NTLM/Kerberos*, *Client Certificate*.
    a. If *Basic* is selected then *User Name* and *Password* has to be provided.
    b. If *Digest* is selected then *User Name* and *Password* has to be provided.
    c. If *Negotiate/NTLM/Kerberos* is selected then *User Name* and *Password* has to be provided, or *Workflow Service Identity* has to be selected.
    d. If *Client Certificate* is selected then *String*, or *Byte Array* with certificate contents has to be provided, or certificate from *Tenant Certificate Store* has to be selected
6. **Workflow designer** enters *SOAP Header*, or selects its source from predefined set: *String Variable*, *Dictionary Variable*, *Temporary File*.
7. **Workflow designer** enters *SOAP Body*, or selects its source from predefined set: *String Variable*, *Object Variable*, *Temporary File*.
8. **Workflow designer** enters *SOAP Response Body Output Type* from predefined set: *String Variable*, *Object Variable, Object Array Variable*, *Temporary File*.
    a. If *Object Array Variable* is selected then it is assumed there's top level array of objects in data received.
9. **Workflow designer** enters *SOAP Response Header Output Type* from predefined set: *String Variable*, *Object Variable, Temporary File*.
10. **Workflow designer** clicks *Preview* button.
11. **System** presents response of sample data with data schema applied.
12. **Workflow designer** clicks *Save* button.

6.6.2.1 Pre-conditions

1. **Tenant** has workflow subsystem licensed.
2. **Tenant** has *SOAP Call* node licensed.

3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition.
4. **Workflow designer** had *SOAP Call* node added to designer workspace.

6.6.2.2 Post-conditions

1. **Workflow designer**'s draft of workflow definition is updated to include *SOAP Call* node.

6.6.2.3 Special Requirements

1. *SOAP Call* node will be the base node for custom web service nodes. It has to be possible to use the node as a base class for custom nodes. Moreover, it has to provide a way to customize authentication, header, and body processing for derived nodes.
2. At runtime *SOAP Call* node execution fails if the HTTP response code is interpreted by the .NET Framework as error code.
3. At runtime workflow instance will be persisted after *SOAP Call* node execution.
4. At runtime original caller's access rights to temporary storage, credential store, or certificate store will be verified.

6.6.3 Configure CSV Import node

1. **Workflow designer** double-clicks *CSV Import* node placed on designer workspace.
2. **System** presents a new window showing required and optional parameters. Required parameters are highlighted and have to be filled in.
3. **Workflow designer** selects *Input Type* from predefined set: *String Variable, Temporary File*.
4. **Workflow designer** defines data schema either:
    a. by adding each column and specifying it's: *Name*, *Position*, *Separator*, *Destination Type*,
    b. or by: selecting sample data file and specifying: *Column Separator*, *Row Separator*, *Text Qualifier*.
5. **Workflow designer** previews the output of applying data schema to sample data by clicking *Preview* button.
6. **System** presents first 100 rows of sample data with data schema applied.
7. **Workflow designer** clicks *Save* button.

6.6.3.1 Pre-conditions

5. **Tenant** has workflow subsystem licensed.
6. **Tenant** has *CSV Import* node licensed.
7. **Workflow designer** successfully opened existing workflow definition or created new workflow definition.
8. **Workflow designer** had *CSV Import* node added to designer workspace.

6.6.3.2 Post-conditions

1. **Workflow designer**'s draft of workflow definition is updated to include *CSV Import* node.

6.6.3.3 Special Requirements

1. At runtime workflow instance will be persisted after *CSV Import* node execution.
2. At runtime original caller's access rights to temporary storage will be verified.

6.6.4 Configure Data Object Select node

1. **Workflow designer** double-clicks *Data Object Select* node placed on designer workspace.
2. **System** presents a new window showing required and optional parameters. Required parameters are highlighted and have to be filled in.
3. **Workflow designer** selects *Authorize against* from predefined set: *Original Caller*, *Direct Caller.*
4. **Workflow designer** selects *Data Object* from the tenant's available data objects.
5. **Workflow designer** selects *Source* from predefined set: *Before Trigger, After Trigger, Query*.
    a. If *Query* is selected, then **Workflow designer** inputs the *Query Text*.
6. **Workflow designer** previews the data schema by clicking *Preview schema* button.
7. **System** presents first 100 rows of the selected Data Object.
8. **Workflow designer** clicks *Save* button.

6.6.4.1 Pre-conditions

1. **Tenant** has workflow subsystem licensed.
2. **Tenant** has *Data Object Select* node licensed.
3. **Workflow designer** successfully opened existing workflow definition or created new workflow definition.
4. **Workflow designer** had *Data Object Select* node added to designer workspace.

6.6.4.2 Post-conditions

1. **Workflow designer**'s draft of workflow definition is updated to include *Data Object Select* node.

6.6.4.3 Special Requirements

1. At runtime selected caller's access rights to dynamic database will be verified.
2. At runtime encrypted data object's attributes will be decrypted.

6.7 Importing Workflow Nodes

The node architecture will be defined so as to easily add additional nodes to a system via an Import feature.

Nodes will be imported using the Master Tenant, and will be available for use in any of the defined tenants via the tenant licencing feature.

Nodes will be imported as DLL files that have to be Strong Named and have to adopt versioning. It won't be possible to replace node library DLL with a DLL with already existing version. Libraries can't be stored in GAC to avoid cross-tenant visibility. Database is the preferred storage.

Every newly created and re-hydrated workflow instance will use the newest DLL version containing required nodes. Simultaneous use of multiple versions of the node library DLL isn't supported.

Any dependencies of node libraries have to be uploaded first. Dynamic assembly loading isn't supported, because Workflow Designer will have to download and load all node libraries and their dependencies.

Note: Upgrades must not affect any custom imported nodes.

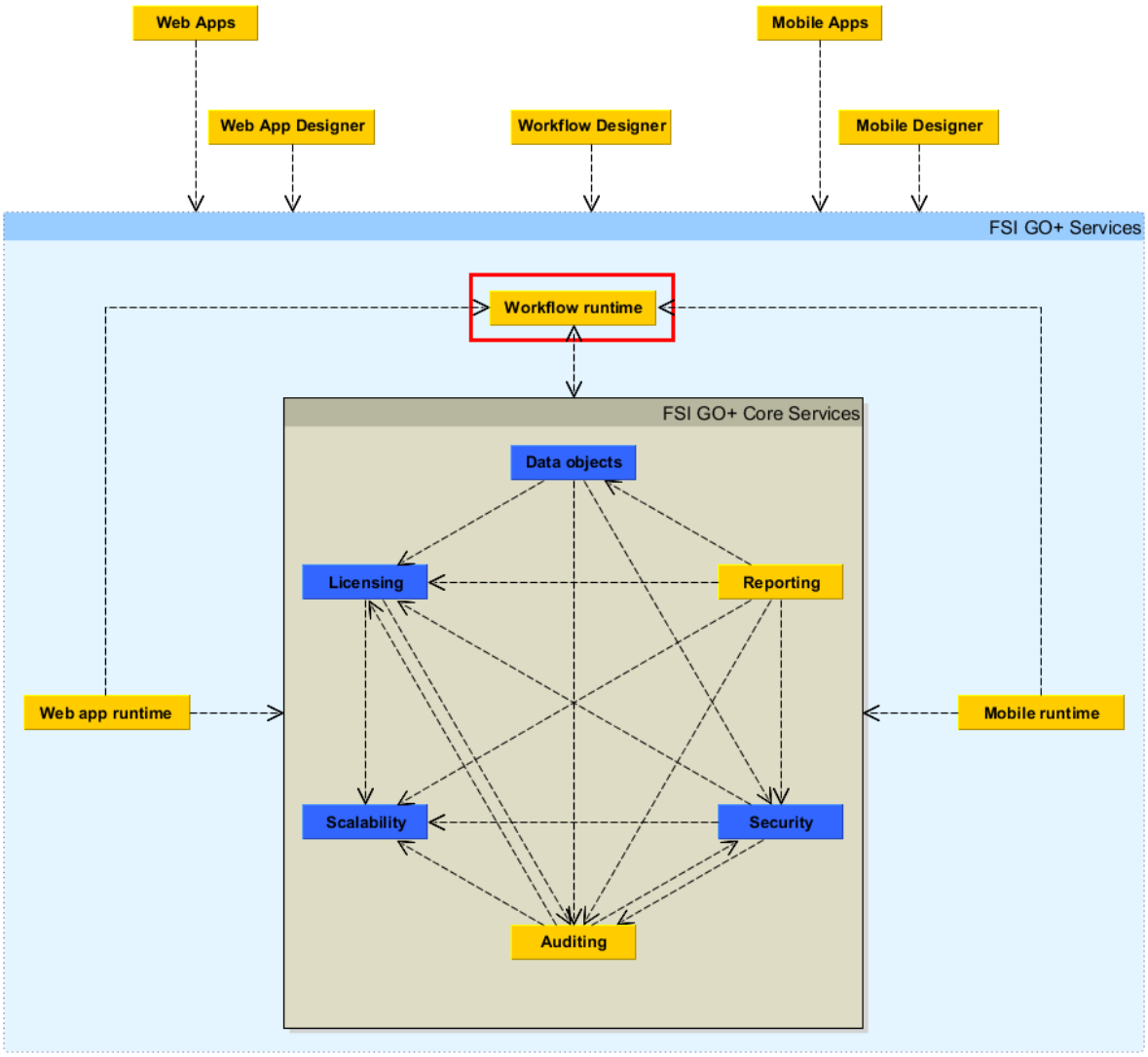
6.8 Workflow Maintenance

As the recording of execution data for tracing and logging, the current GO cleaner service settings will need to be extended to set automated data clean-up for the following:

- Workflow Trace History
- Workflow Execution History

6.9 Workflow prioritisation

A queuing mechanism is to be provided with the workflow engine to let users specify priorities of the different workflows.



Different workflow queues allow for executing critical tasks in separate workflow runtimes: *In-Process*, *Queue 1*, *Queue 2*, *and Queue 3*. The workflow queue is selected as part of workflow definition and all instances of the definition are executed accordingly.

Queues are licensed separately. Each license specifies maximum number of simultaneously running workflow instances in a queue. Each tenant can have only a subset of queues licensed.

Each queue requires separate clean-up process for terminated workflow instances.

6.10 Use case scenarios description

6.10.1 Modify workflow queue of workflow definitions

1. **Workflow designer** optionally orders or group available workflow definitions by *Queue* column.
2. **Workflow designer** selects workflow definitions using checkboxes in *selected* column.
3. **Workflow designer** clicks *Change Queue* button.
4. **System** prompts for of a queue to run selected definitions in.
5. **Workflow designer** selects Queue from predefined set: *In-Process*, *Queue 1*, *Queue 2*, *and Queue 3*.
6. **Workflow designer** clicks *Apply* button.
7. **System** displays information that the changes will apply to the next instance execution of each one of the selected definitions, but not to the currently running ones.

Pre-conditions

1. **Tenant** has workflow subsystem licensed.
2. **Workflow designer** successfully authenticated against the portal.
3. **Workflow designer** navigated to *Workflow Definitions* view.

Post-conditions

1. Newly spawned instances or rehydrated instances of modified workflow definitions will execute in set queues.

6.10.2 Display historic workflow instances in workflow queues

1. **Workflow designer** clicks *Historic executions* button.
2. **Workflow designer** selects *From Date* and *To date* for trend analysis.
3. **System** presents a chart with one day resolution of the selected time interval showing number of workflow definitions and number of workflow instances executed each day.

Pre-conditions

1. **Tenant** has workflow subsystem licensed.
2. **Workflow designer** successfully authenticated against the portal.
3. **Workflow designer** navigated to *Workflow Definitions* view.

6.10.3 Display current workflow instances in workflow queues

1. **Workflow designer** clicks *Currently executing* button.
2. **Workflow designer** selects *From Date* and *To Date* (with one-minute resolution) for limiting the number of displayed instances.
3. **System** presents a grid grouped by *Queue* and showing current number of instances of each workflow definition executing.

Pre-conditions

1. **Tenant** has workflow subsystem licensed.
2. **Workflow designer** successfully authenticated against the portal.
3. **Workflow designer** navigated to *Workflow Definitions* view.

6.10.4 View licensing for workflow queues

1. **Master tenant administrator** selects workflow queues using checkboxes in *Selected* column.
2. **Master tenant administrator** clicks *Licensing* button.
3. **System** contacts FSI licensing server and then displays number of instances which are licensed for selected queues.
4. If FSI licensing server can't be reached, **System** displays the last known licensing information with a warning that the numbers are not up-to-date.

Pre-conditions

1. **Master tenant administrator** successfully authenticated against the portal.
2. **Master tenant navigated** to *Workflow Queues* view.

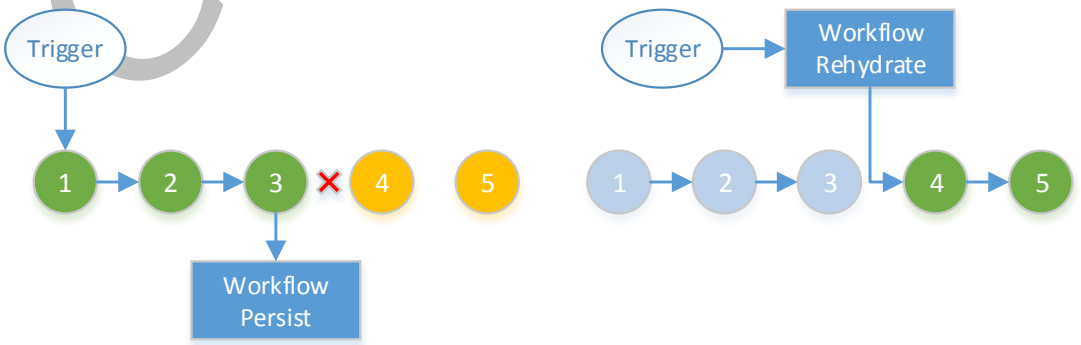
Post-conditions

1. Newly spawned instances or rehydrated instances of workflow definitions assigned to reconfigured queues will respect set licensing limit.

6.11 Persistency and re-hydration of workflows

Workflows might require to be paused and wait for additional business logic to be executed on the caller (or another external system) before continuing with their flow of actions.

The workflow engine must be able to save the state of a workflow at any point in time, and then restore it as and when required to resume its operations (this functionality is supported by Microsoft Workflow Foundation). Relevant low level facility provided by WF is bookmark. However, web service receive activities and delay activities encapsulate bookmark functionality. Using them as base classes for custom nodes or composing custom nodes from them is the most productive way of extending the GO platform



There are also cases when persistence should be enforced without dehydration. Nodes which change (GO+ or external) system state in non-idempotent way have to enforce persistence of workflow instance at the end of their execution. That's because after recovering from Workflow Runtime failure, every workflow instance is executed from the last persistence point. Not having workflow persisted right after state modification will result in replaying the modification after recovery. If the modification is not idempotent (e.g. incrementing a counter) and in the absence of distributed transactions consistency will become a problem. FSI GO+ nodes have to be designed taking this fact into consideration, likewise any node created in the future. Example of such nodes are REST/SOAP Calls, DB insert node, (S)FTP send, etc.

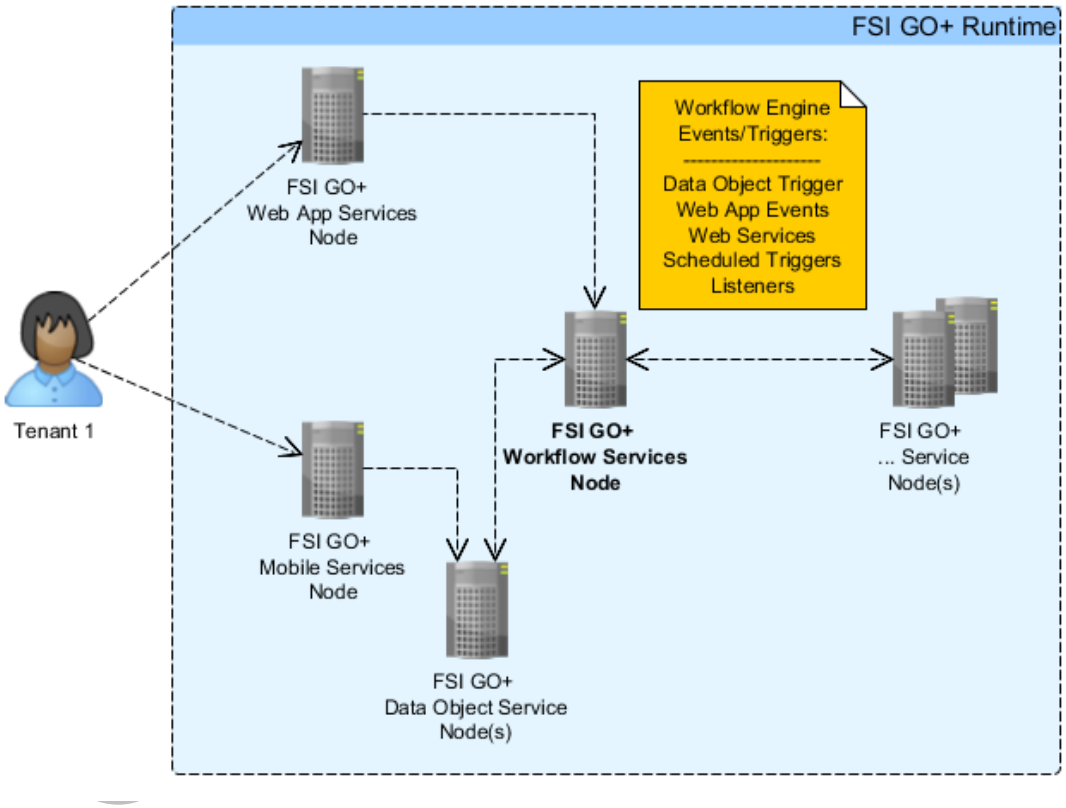
6.12 Triggers

Triggers define when a workflow executes. The platform is to support several trigger types that will ease the integration of workflows into the wider platform as well as allowing 3rd party systems to utilise the workflow feature set.

The platform is to support the following types of triggers for launching the workflows and setting an execution schedule:

- Data Object Trigger
- Web App Events
- Web Services
- Scheduled Triggers
- Listeners

The image below shows the architecture of the conceptual model, defining workflow execution functionality.



Triggers will have the option of being enabled or disabled by the tenant user (with appropriate permissions assigned).

6.13 Data Object Trigger

Data objects need to be extended to allow the configuration of workflows to be executed based upon defined actions. In its simplest form one or more workflows could be triggered based upon

In its simplest form one or more workflows could be triggered based upon:

- ON INSERT
- ON UPDAT.
- ON DELETE

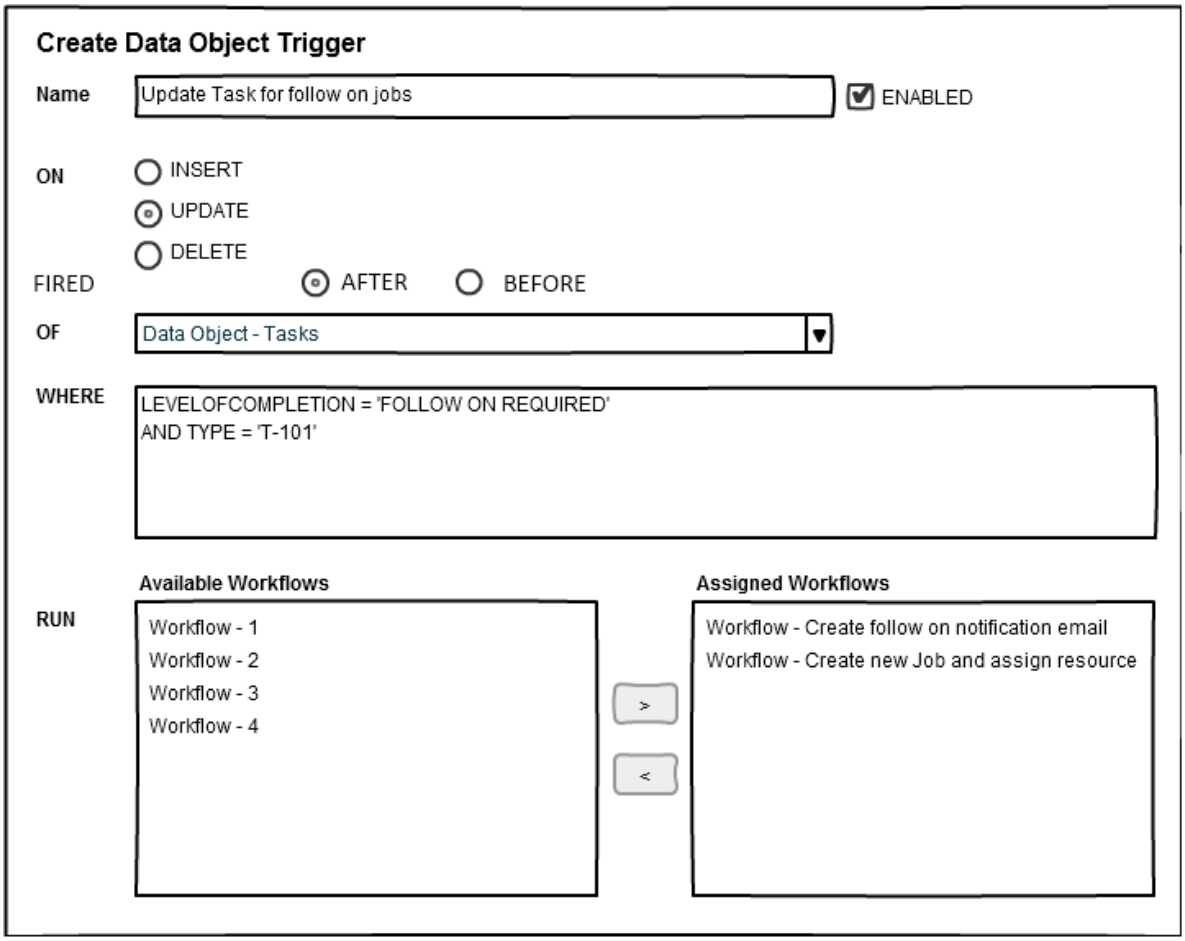
The system must support retrieving information about the data object state. State is the data snapshot at the time of the trigger. This is required to be able to use in a workflow logic for data comparison. E.g. Old v New data in an Update. Data object is always in any one of the following states:

- **Added** (*inserted*)
- **Deleted** (*deleted*)
- **Modified** (*updated*)
- Unchanged

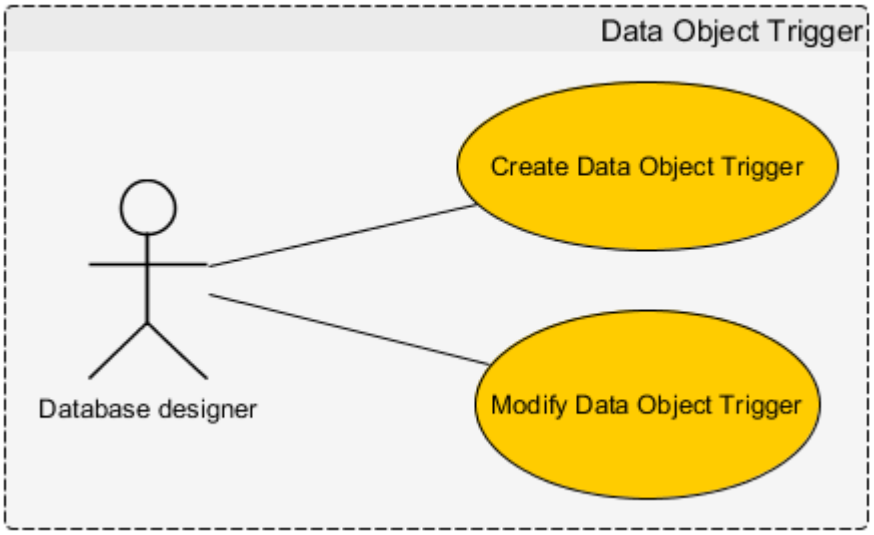
Taking this further, additional WHERE clause capabilities are required such as:

- ON UPDATE WHERE "ContractType = Maintenance"
- ON DELETE WHERE "CompleteDate > PlanedDate"

A user interface is required to create Data Object triggers. This could be designed as below.



6.13.1 Use case diagram



6.13.2 Use case scenarios description

6.13.2.1 Create Data Object Trigger.

Basic flow of events:

1. **Database designer** enters trigger name into Name box
2. **Database designer** selects the option to enable trigger (checkbox)
3. **Database designer** defines type of event, an option when the trigger should be fired
4. **Database designer** selects Data Object (entity) from the list of objects
5. **Database designer** defines WHERE clause
6. **Database designer** assigns workflow(s) from the list of available workflows
7. If **Database designer** clicks OK, the list of Data Object Triggers is updated
8. If **Database designer** clicks CANCEL, the user is returned to the previous menu without saving changes.
9. Use case ends

Pre-conditions

- **Tenant** has license to use the Database Designer tool and create Data Object Triggers
- **Tenant** is logged to the Portal

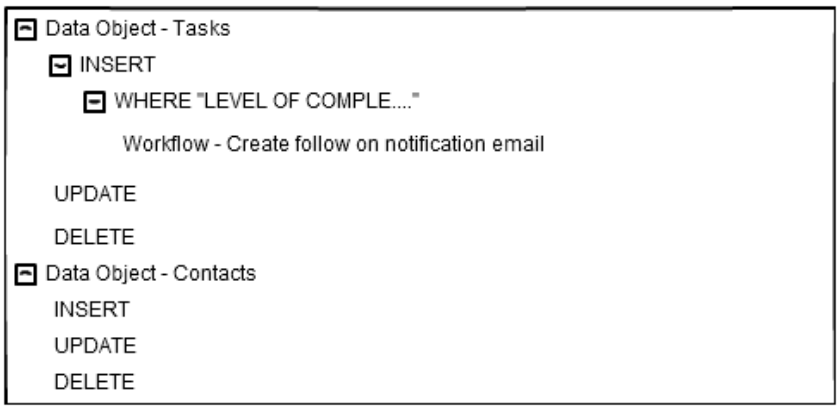
Post-condition

- Data Object Trigger has been created

There also needs to be a user interface to view and open for edit all defined Data Object Triggers. This could be represented as a tree view with the following structure.

- Data Object
  - ON (Insert Update Delete)
    - Trigger Name

Clicking or selecting the Data Object Trigger Name node, the trigger designer will be opened.



6.13.2.2 Modify Data Object Trigger.

Basic flow of events:

1. **Database designer** selects the trigger by clicking or selecting node on the tree view control; trigger design window will be opened in the workspace
2. **Database designer** changes the data trigger definition
3. If **Database designer** clicks OK, the list of Data Object Triggers is updated
4. If **Database designer** clicks CANCEL, the user is returned to the previous menu without saving changes.
5. Use case ends

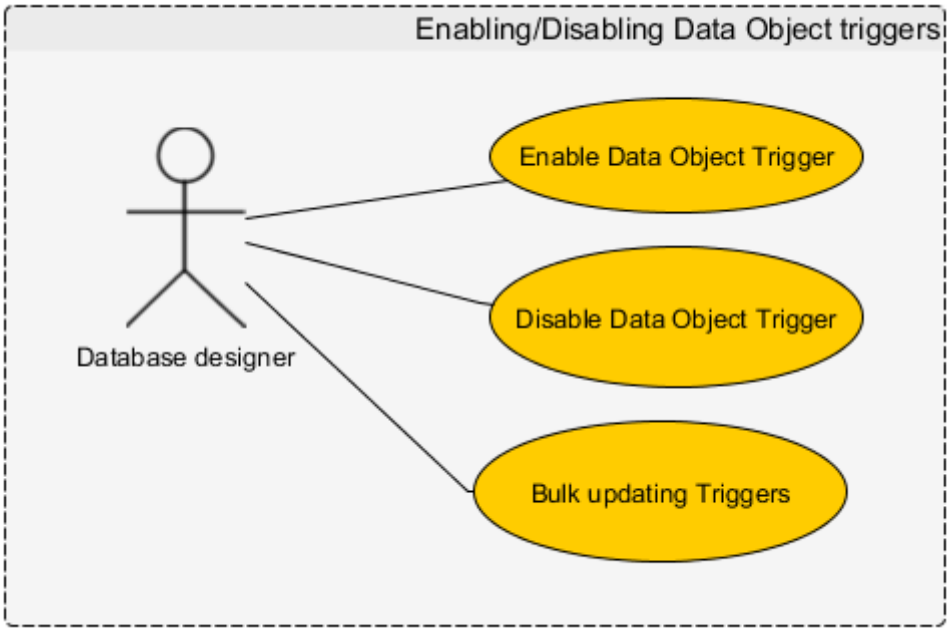
Pre-conditions

- **Tenant** has license to use the Database Designer tool and create Data Object Triggers
- **Tenant** is logged to the Portal

Post-condition

- Data Object Trigger has been modified.

6.13.3 Use case diagram



6.13.4 Use case scenarios description

6.13.4.1 Enable/Disable Data Object Trigger.

Basic flow of events:

1. **Database designer** opens window with *Data Object Triggers list*
2. **Database designer** searches for the trigger, scrolling list of triggers
3. **Database designer** enables/disables *trigger execution* using checkbox control (tick)
4. If **Database designer** clicks OK, the list of triggers is updated
5. If **Database designer** clicks CANCEL, the user is returned to the previous menu without saving changes.
6. Use case ends

Pre-conditions

- **Tenant** has license to use Database Designer tool and create Data Object Triggers
- **Tenant** is logged to the Portal

Post-condition

- Data Object trigger has been enabled/disabled *or*
- Data Object triggers have been enabled/disabled

6.13.4.2 Bulk updating Data Object Triggers.

Basic flow of events:

1. **Database designer** opens window with *Data Object Triggers list*
2. **Database designer** selects Data Object from the list of objects
3. **Database designer** "checks a checkbox" with the option *Disable Triggers*
4. If **Database designer** clicks OK, the list of triggers is updated and triggers are disabled/enabled
5. If **Database designer** clicks CANCEL, the user is returned to the previous menu without saving changes.
6. Use case ends

Pre-conditions

- **Tenant** has license to use Database Designer tool and create Data Object Triggers
- **Tenant** is logged to the Portal

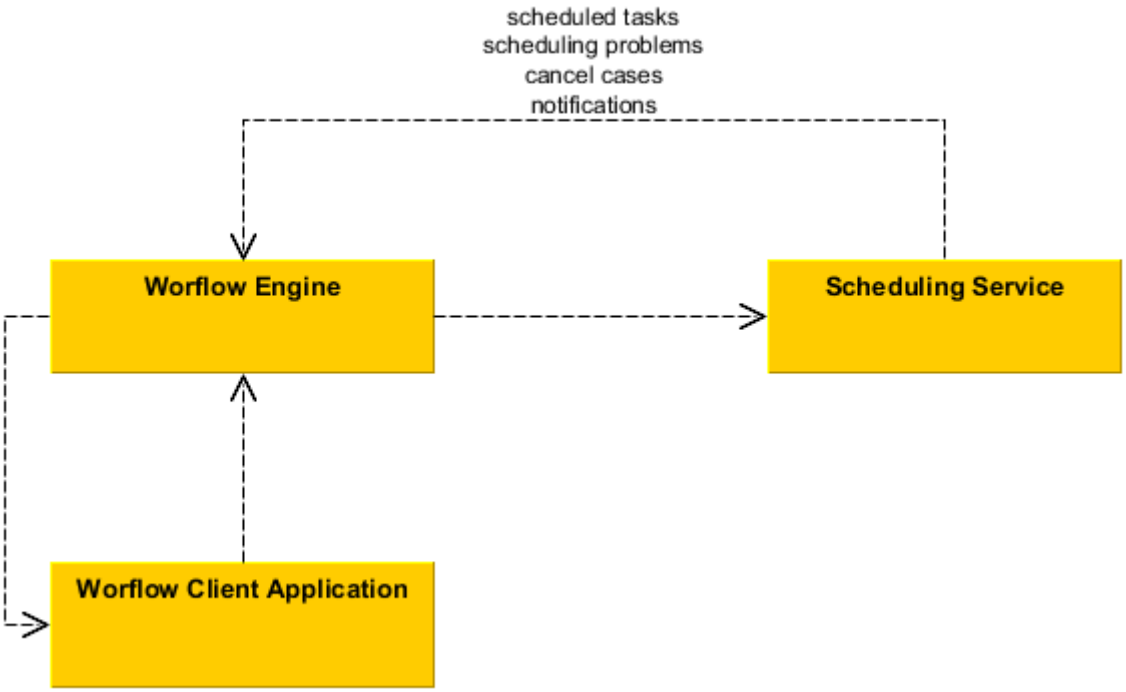
Post-condition

- Data Object triggers have been disabled/enabled

6.14 Scheduled

A scheduling engine is required to support executing workflows at specific times or using a recurring pattern.

Figure below shows the architecture of the conceptual model, defining workflow scheduling functionality.



The architecture consists of three components:

- Workflow Engine
- Scheduling Service
- Workflow Client Application

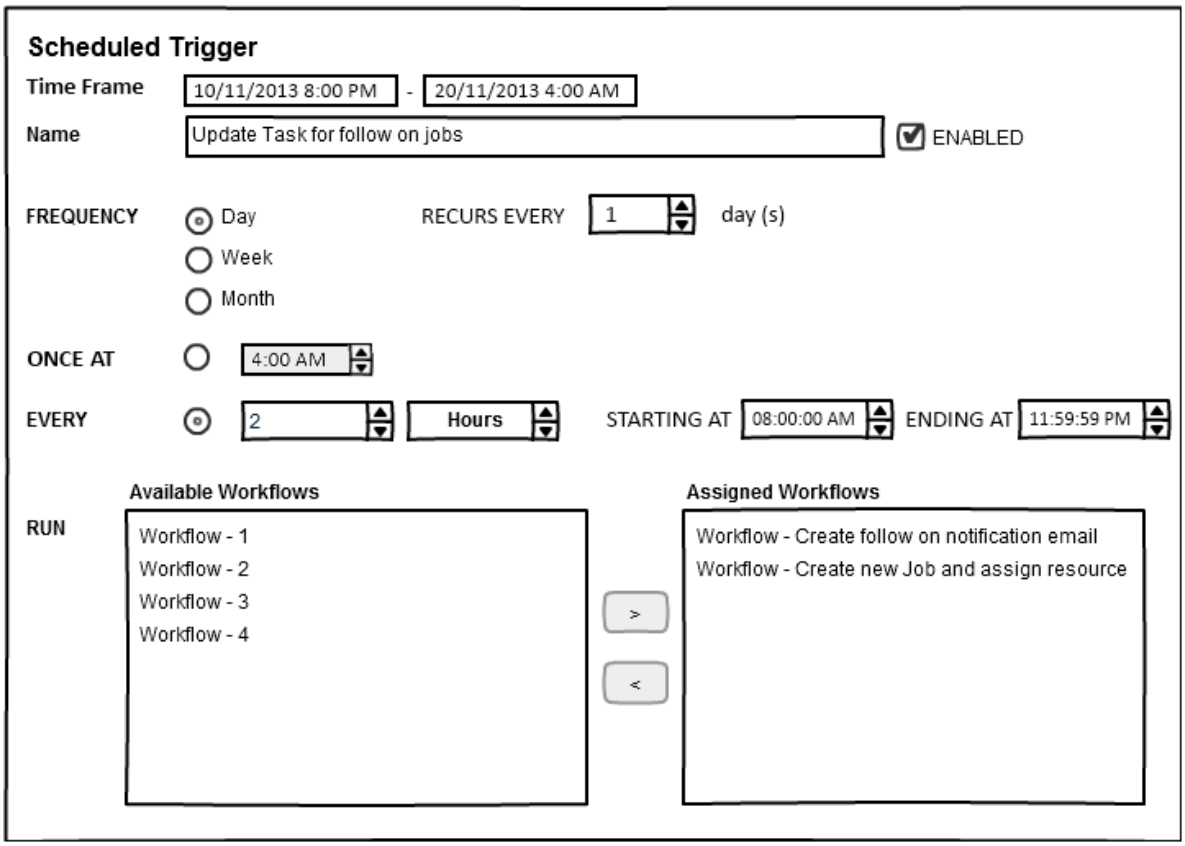
Scheduling Service is to support creation of the scheduled tasks with workflows execution.

6.14.1 Scheduled trigger definition

Schedule should have the following time options:

- Every x Second/Minute/Hour/Days/Months
- Specific date/time (one off)
- Every x day of the week (e.g. Tuesday and Thursday at 10am)
- First or Last Day of the month (1st or 28th/29th/30th/31st depending on month)
- First or Last x of the month (e.g. first Thursday of month)

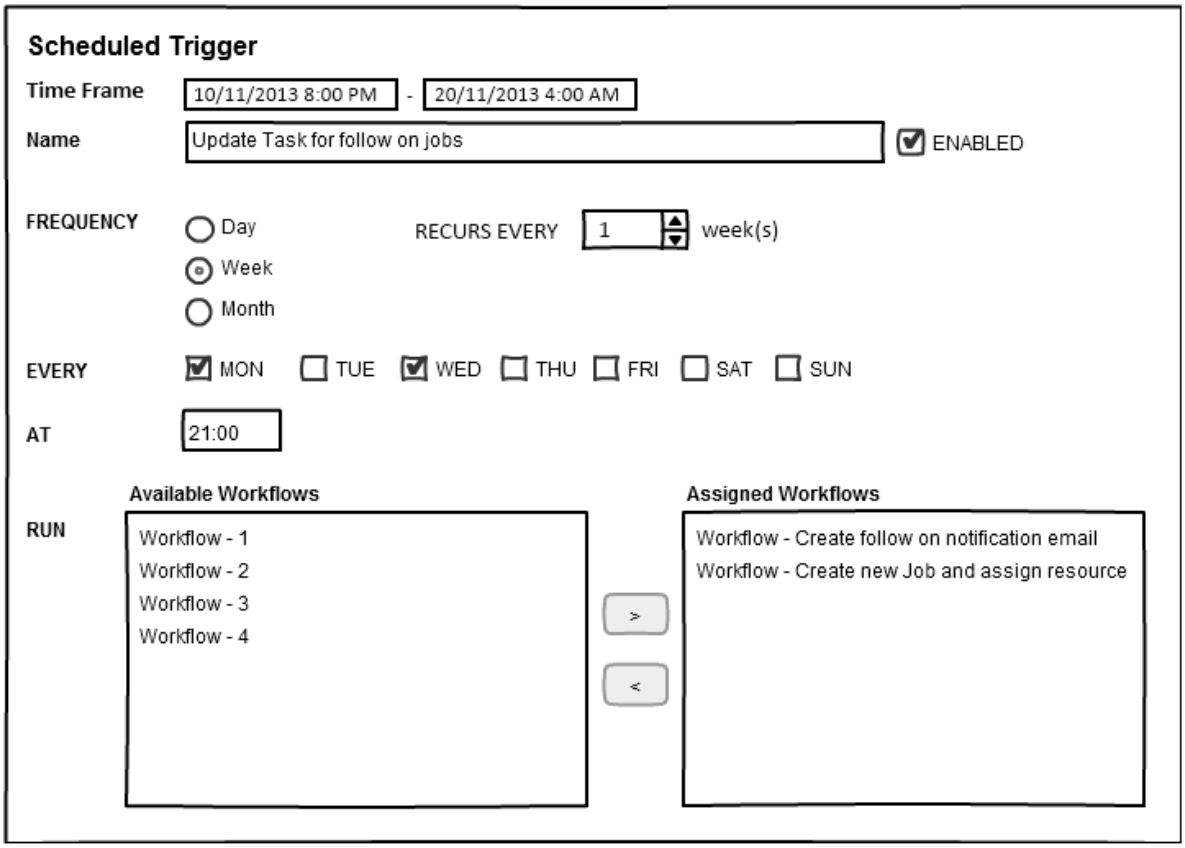
Figure below shows the example of *Scheduled trigger* definition window with *day frequency*.



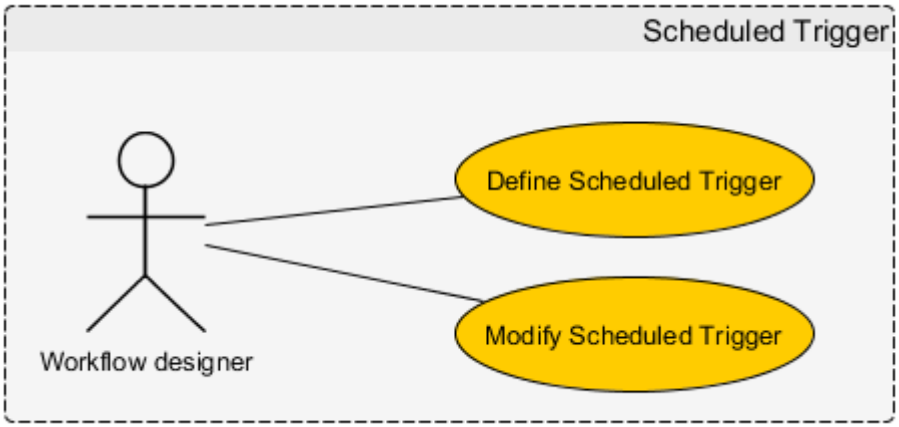
The *scheduled trigger* should have the following inputs:

- Time Frame – continuous period of time, in which the scheduled trigger works. If end date is blank, it means that there is no end date.
- Name – name of the scheduled trigger
- Enable - option switches the trigger on and off
- Frequency – recurring pattern
- Once at/Every, starting and endig time, etc. – occurrences for recurring pattern
- Run
    - Available Workflows – list of available worklfows that can be selected
    - Assigned Workflows – list of assigned workflows for the trigger

Figure below shows the example of *Scheduled trigger* definition window with *week frequency*.



6.14.2 Use case diagram



6.14.3 Use case scenarios description

6.14.3.1 Define Scheduled trigger.

Basic flow of events: define scheduled trigger, *daily recurring pattern*

1. **Workflow designer** defines Time Frame period using Date Time picker controls
2. **Workflow designer** enters trigger name into Name box
3. **Workflow designer** selects the option to enable trigger (checkbox)
4. **Workflow designer** selects *Day recurring pattern*
    a. **Workflow designer** enters *Recurs every* option (e.g. recurs every 1 day)
    b. **Workflow designer** selects *Daily frequency* option
        i. If **Workflow designer** selects *Occurs once at* option; *Occurs every* option shall be disabled; **Workflow designer** sets hour in the time control box
        ii. If **Workflow designer** selects occurs *Every* option; *Occurs once at* option shall be disabled; **Workflow designer** sets frequency and time unit
5. **Workflow designer** assigns workflow(s) from the list of available workflows
6. **Workflow designer** submits form
7. Use case ends

Alternate flow: *weekly recurring pattern*

A.4. **Workflow designer** selects *Week recurring pattern*
a. Workflow designer enters *Recurs every* option (e.g. recurs every 2 weeks)
b. Workflow designer selects days from occurs *Every* options
c. Workflow designer sets starting time in the 'start *at'* box

Alternate flow: *monthly recurring pattern*

B.4. **Workflow designer** selects Month recurring pattern

a. Workflow designer enters *Recurs* every option (e.g. recurs every 1 month)

b. Workflow designer selects day (e.g. first or last day of the month, 1st or 28th/29th/30th/31st depending on month)

c. Workflow designer sets starting time in the 'start *at'* box

Pre-conditions

- **Tenant** has license to use the Workflow Designer tool
- **Tenant** is logged to the Portal

Post-condition

- Scheduled trigger has been created

6.14.3.2 Modify Scheduled trigger.

Basic flow of events:

1. **Workflow designer** opens window with *Scheduled triggers list*
2. **Workflow designer** searches for the workflow, scrolling list of triggers
3. **Workflow designer** selects the trigger by clicking or selecting node on the tree view control; trigger design window will be opened in the workspace
4. **Workflow designer** changes the data trigger definition
5. If **Database designer** clicks OK, the scheduled trigger is updated
6. If **Database designer** clicks CANCEL, the user is returned to the previous menu without saving changes.
7. Use case ends

Pre-conditions

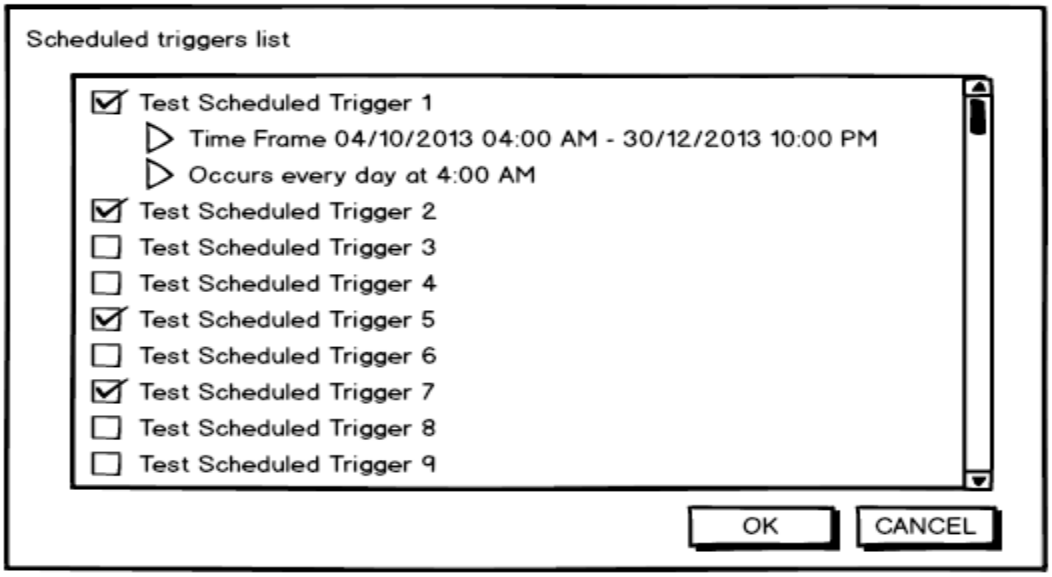
- **Tenant** has license to use the Workflow Designer tool
- **Tenant** is logged to the Portal

Post-condition

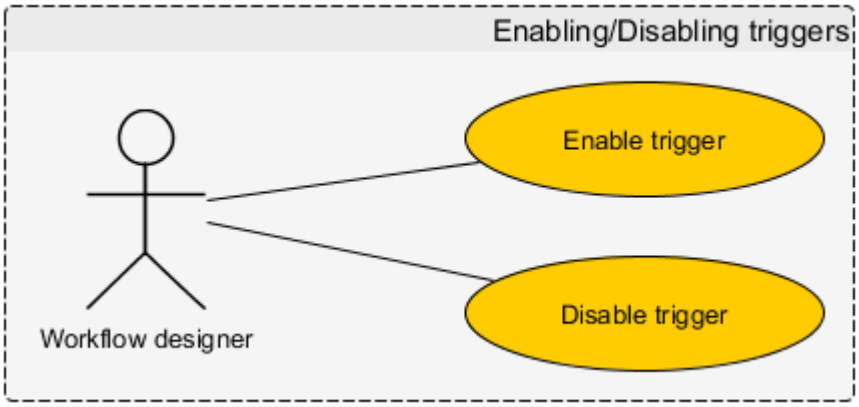
- Scheduled trigger has been modified

6.15 Enabling/Disabling triggers

This option allows the enabling/disabling of a trigger.



6.15.1 Use case diagram



6.15.2 Use case scenarios description

6.15.2.1 Enable/Disable trigger.

Basic flow of events:

1. **Workflow designer** opens window with *Scheduled triggers list*
2. **Workflow designer** searches for the workflow, scrolling list of triggers
3. **Workflow designer** enables/disables *workflow execution* using checkbox control (tick)
4. If **Workflow designer** clicks OK, the list of scheduled triggers is updated

5. If **Workflow designer** clicks CANCEL, the user is returned to the previous menu without saving changes.
6. Use case ends

Pre-conditions

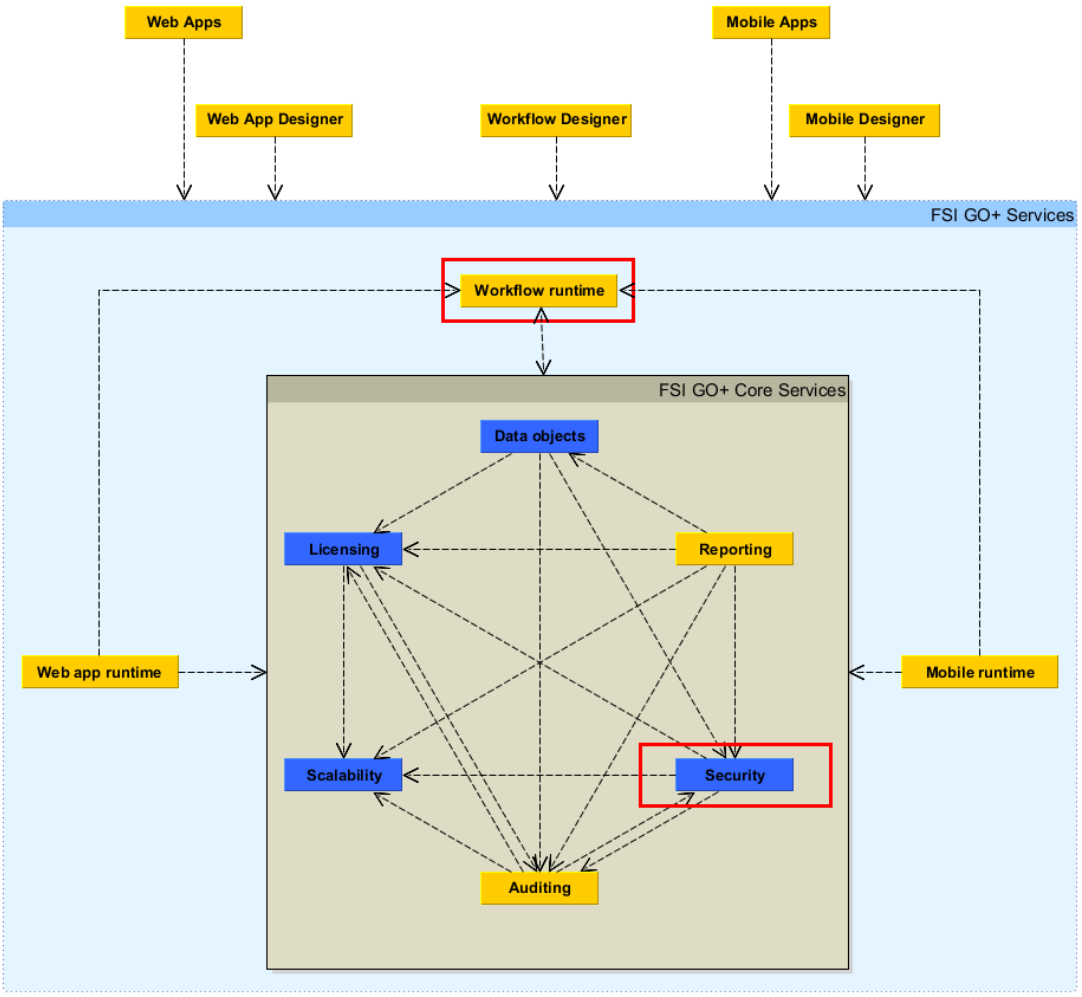
- **Tenant** has license to use the Workflow Designer tool
- **Tenant** is logged to the Portal

Post-condition

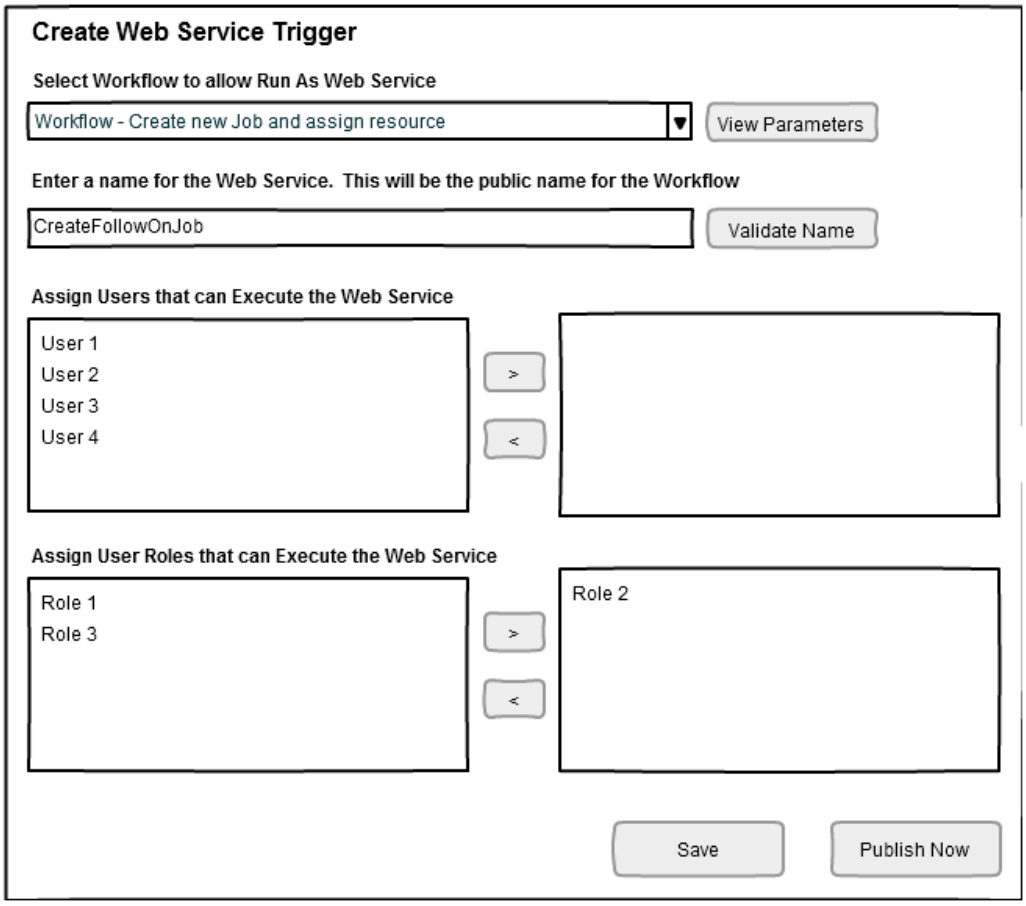
- Scheduled trigger has been enabled/**disabled** *or*
- Scheduled triggers have been enabled/disabled

6.16 Web Services

The workflow process can be exposed as a Web Service. To trigger a workflow via a web service, a 3rd party system must initiate the trigger by calling a valid web service address that has been configured for the workflow. The 3rd party system must also be authenticated and have the rights to execute the workflow.



The platform is to provide a user interface to configure workflows to run as web services. The platform will then make these services available when enabled by publishing the Web Service End Point.



There are a number of default characteristics that are required for a Web Service enabled workflow:

- Optional input parameters
- Result of Success or Failure is required
    - Optional detailed error information
- Web Service workflows will need to adhere to the prioritization settings of the workflow they are linked to (see Workflow Prioritization)
    - In Process (synchronous web services)
    - FIFO queue (asynchronous web services, a queue monitoring service is to be available to verify the state of a request)

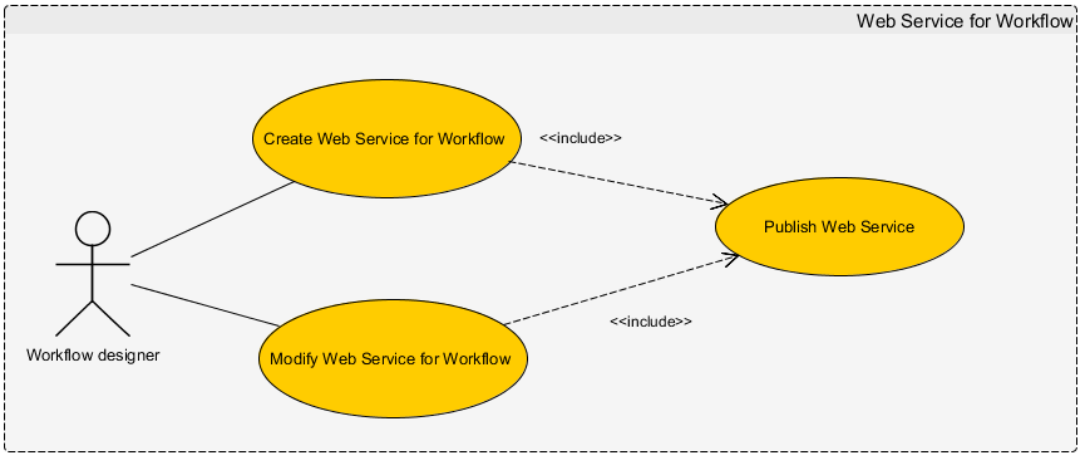
Workflows will be exposed internally and externally as web services. They are always invoked through web service call irrespective of a workflow runtime chosen as part of the workflow definition.

Internal workflow web services use FSI GO+ authentication scheme but can be exposed externally for clients supporting this kind of authentication.

External workflow web services are exposed to systems bound to use other standard authentication schemes: basic, digest authentication (e.g.: md5), certificate mapping.

Each workflow web service call is recorded for licensing purposes.

6.16.1 Use case diagram



6.16.2 Use case scenarios description

6.16.2.1 Create Web Service for Workflow.

Basic flow of events:

1. **Workflow designer** selects Workflow to run as a Web service from the list of workflows (***include*** *Define Web Service parameters*)
2. **Workflow designer** enters Web Service name into Name box
3. **Workflow designer** validates the Web Service name. Web Service name is valid
4. **System** generates web service parameters based on workflow parameters.
5. **Workflow designer** assigns *Users* that can execute the Web Service
6. **Workflow designer** assigns *Roles* that can execute the Web Service
7. If **Workflow designer** clicks Save, the definition of the Web Service is saved
8. If **Workflow designer** clicks *Published Now*, the definition of the Web Service is saved. System opens window with publishing features.
9. If **Workflow designer** clicks CANCEL, the user is returned to the previous menu without saving changes.
10. Use case ends

Pre-conditions

- **Tenant** has license to use the Workflow Designer tool, create and publish Web Services for Workflows
- **Tenant** is logged to the Portal

Post-condition

- Web Service has been created

6.16.2.2 Modify Web Service for Workflow.

Basic flow of events:

1. **Workflow designer** opens window with Web services *list*
2. **Workflow designer** searches for the web service, scrolling list of services
3. **Workflow designer** selects the Web Service by clicking or selecting node on the tree view control; Web Service design window will be opened in the workspace
4. **Workflow designer** changes the web service definition
5. If **Workflow designer** clicks Save, the definition of the Web Service is saved
6. If **Workflow designer** clicks *Published Now*, the definition of the Web Service is updated. System opens window with publishing features.
7. If **Workflow designer** clicks CANCEL, the user is returned to the previous menu without saving changes.
8. Use case ends

Pre-conditions

- **Tenant** has license to use the Workflow Designer tool and create Web Services for Workflows
- **Tenant** is logged to the Portal

Post-condition

- Web Service has been updated

6.16.2.3 Publish Web Service.

Basic flow of events:

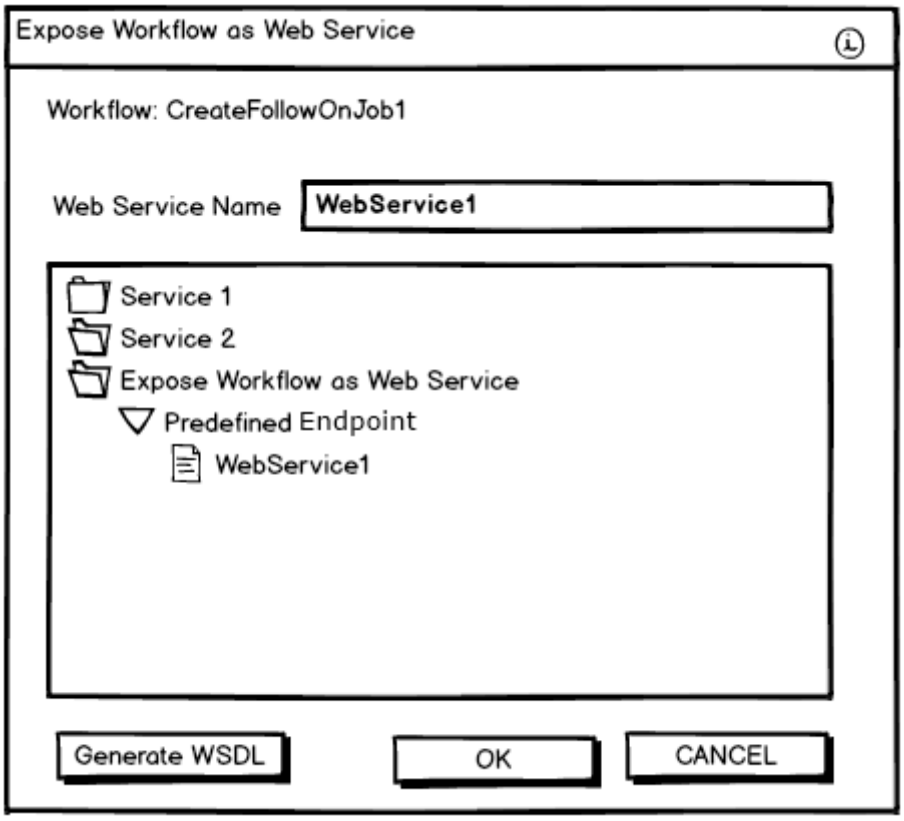
1. **Workflow designer** examines definition of workflow parameters
2. **Workflow designer** clicks Publish Now button; *Expose Workflow as Web Service* window will be opened in the workspace
3. **Workflow designer** defines the operation name for the new web service
4. If **Workflow designer** selects *External Endpoint* then he also provides parameters: protocol (https), *URL, authentication method*.
5. **Workflow designer** clicks button *Generate WSDL.* System generates WSDL file. (Note: even in the case of REST endpoint WSDL is useful for .NET developer. Contracts defined by WSDL can be recreated as native classes in the client-side code and then serialized using JSON at runtime. There's no need to build JSON objects manually then.)
6. **Workflow designer** clicks finish. System creates Web Service.
7. If **Workflow designer** clicks CANCEL, the user is returned to the previous menu without saving changes.
8. Use case ends

Pre-conditions

- **Tenant** has license to use the Workflow Designer tool, create and publish Web Services for Workflows
- **Tenant** is logged to the Portal

Post-condition

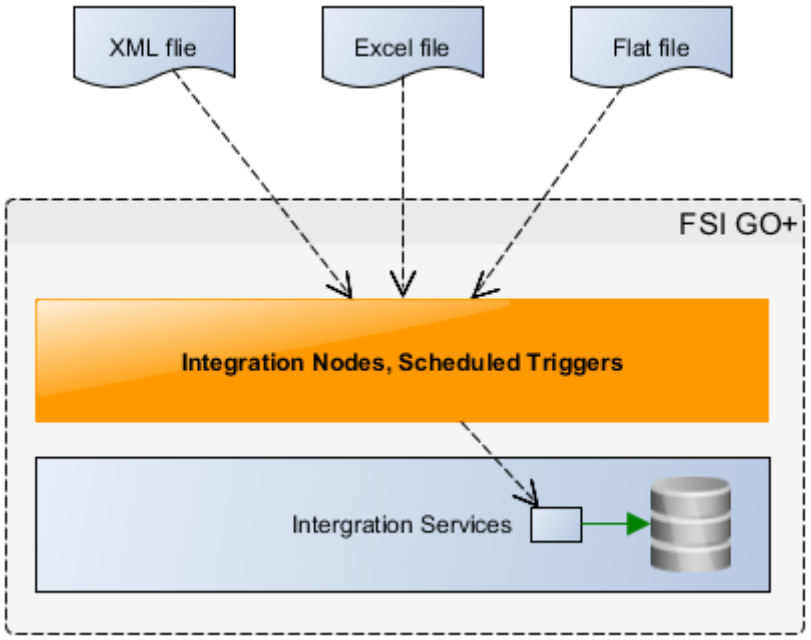
- Web Service has been published



6.17 Listeners

To integrate with other types of system such as Flat Files and FTP, the user can configure a Schedule Trigger and utilise the Integration nodes to interact with these systems.

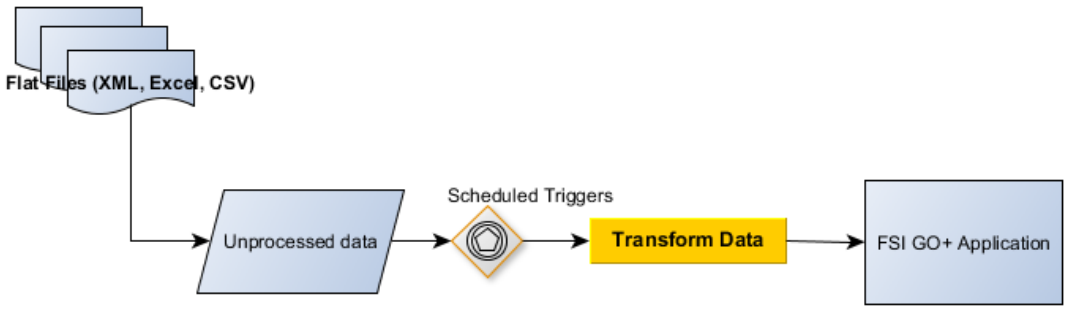
The image below shows the architecture of the conceptual model, defining flat files integration functionality.



Flat files integration process includes design-time and runtime support, including:

- Support for mapping between flat files, XML documents and spreadsheets
- Support for data inspection and transformation
  - creating *transformation rules* between source and target data
- Codeless (graphical) process and data flows modelling (Integration nodes)
- Support for Events and Schedules configuration

The image below shows example schema that consolidates flat file processing from different sources:



The process consists of the following steps:

- Scheduled Trigger checks the arrival of new flat file and rises an event, which triggers data transformation process
- The flat file is transformed by the Integration Services
- The target data is transported to its destination

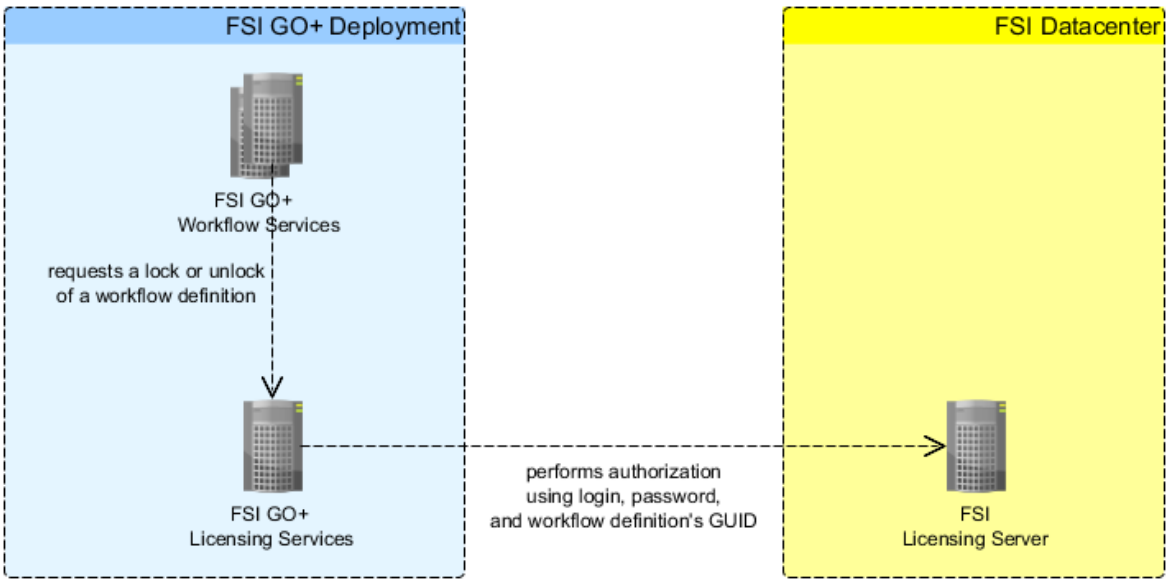
6.18 Locking of system workflows

To aid the delivery of workflows through our services business, it is essential that we support a mechanism for the "locking of workflows", this is to prevent users from viewing or editing FSI's "system" workflows.

Additional level of access control to workflow definitions will be provided to protect IP of their authors. This applies both to system workflows created by FSI and to workflows created by FSI's partners. A workflow author can request to lock its definition in order to not allow to view, copy, or modify the definition without explicit unlock request. Locked definition can still spawn new workflow instances and behave as not locked definition in any other regard. The only difference lies in necessity to request to unlock the workflow definition to be able to work with it in Graphical Workflow Designer.

Lock and unlock requests are authorized against a server external to FSI GO+ Deployment. The server is property of FSI. Every lock and unlock request consists of a user name and password verification against the server. The user name and password are credentials managed by the FSI's server not by FSI GO+ Deployment.

Workflow definitions stored in the database have to be encrypted using a key accessible only to account on which Workflow Services are executed.



6.19 Use case scenarios description

6.19.1 Lock a workflow definition

1. **Workflow designer** selects workflow definition rows using checkboxes in *selected* column.
2. **Workflow designer** clicks *Lock Definition* button.
3. **System** prompts for username and password.
4. **Workflow designer** provides user name and password valid for FSI's server.
5. **System** shows status of the selected definitions as *Locked*.

6.19.1.1 Pre-conditions

1. **Tenant** has workflow subsystem licensed.
2. **Workflow designer** successfully authenticated against the portal.
3. **Workflow designer** navigated to *Workflow Definition List* view.

6.19.1.2 Post-conditions

1. Selected workflow definitions become encrypted using dedicated encryption key for the rest of their life-time.

6.19.2 Unlock a workflow definition

1. **Workflow designer** selects a row with workflow definition that is locked.
2. **Workflow designer** clicks *Edit* button.
3. **System** opens *Graphical Workflow Designer*.
4. **System** prompts for username and password.
5. **Workflow designer** provides user name and password valid for FSI's server
6. **System** makes the definition available to edit until the end of the session with *Graphical Workflow Designer*.

6.19.2.1 Pre-conditions

1. **Tenant** has workflow subsystem licensed.
2. **Workflow designer** successfully authenticated against the portal.
3. **Workflow designer** navigated to *Workflow Definition List* view.

6.19.2.2 Post-conditions

1. Selected workflow definitions become encrypted using dedicated encryption key for the rest of their life-time.

6.20 Workflow Engine

A workflow engine is required to execute workflows. The workflow engine is to be provided using the Microsoft Workflow Foundation (WF) included within the .Net 4.5 Framework.

6.20.1 Workflow engine hosting

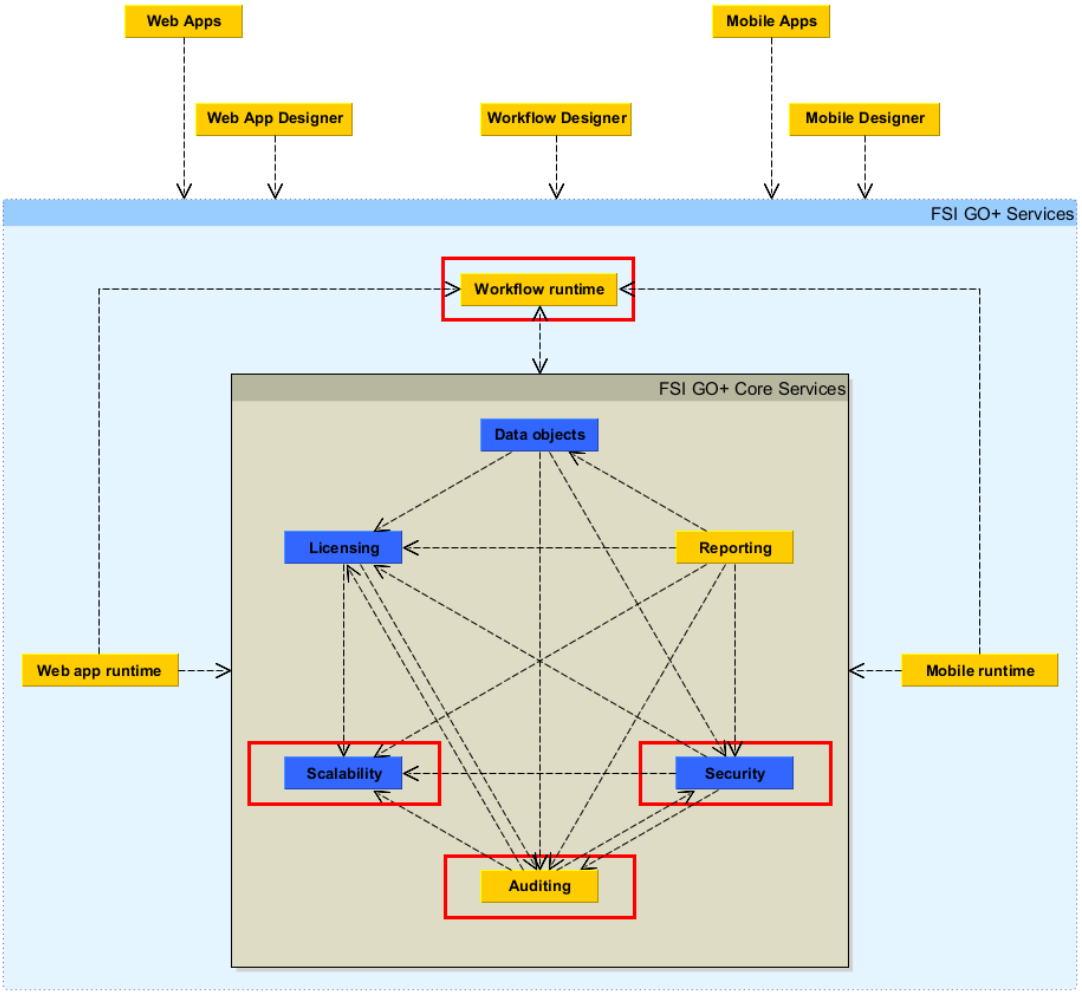
Workflows will be hosted on a scalable Runtime Execution environment (please see section Runtime Execution for full details) that is configured for each tenant.

Note: Microsoft IIS provides out-of-the-box scalability features and is capable of executing WF 4 workflows using the additional AppFabric module. AppFabric, however, does not provide an API that can be used to easily deploy and control workflows. Depending on the final architecture, a custom hosting application is therefore also acceptable, if its licensing model satisfies GO+ requirements stated earlier.

For more information on runtime environments see the section defining the Runtime Execution

6.20.2 Workflow Tracing

The engine must support tracing of workflow execution down to individual node level. The trace will be configurable per workflow, and keep a history of traces per workflow execution.



Workflow designers need a tool to see workflow instance execution in progress and workflow instance execution results after its termination. They have to be able to confirm that a workflow instance was executed and that expected execution path was followed. In general they need a means of debugging and analyzing workflows.

For those purposes it has to be possible to trace every input and output parameter of every executed node. In the special case of Trace Node a trace record is created directly by the node. The exact level of detail is specified in *trace definition*. Trace definition consists of:

- trace level,
- trace definition's validity period,
- trace records expiration period.

Tracing configuration is done by:

- creating *a trace definition*,
- applying *the trace definition* to a specific version of *workflow definition* or to the newest possible version of *workflow definition*.

Trace records are stored in tracing database. It should be possible to enable caching for trace (e.g. using AppFabric Caching Services) because of scalability requirements. Tracing directly to a database is a potential bottleneck in the system caused by its write-intensive nature, and amount of locking introduced. Caching trace records and queuing their persistence is a recommended solution.

Tracing requires a scheduled clean-up process to enforce trace records expiration, i.e. to delete expired records from the system. The clean-up process is configured through Workflow Administration Panel.

Tracing has to respect encryption settings of workflow definition. Values of all fields or parameters are to be traced in encrypted form using tenant's global encryption key if the "Parameter values in trace" is set to "Encrypted".

Tracing UI follows master-detail design where Workflow Designer can start from workflow definitions or trace definitions, then select trace instances, and finally inspect trace records. Because of high volume of data multi-select feature has to be provided to manage tracing configurations, trace definitions, and trace instances.

The trace log is to include as a minimum the following data:

For each workflow

- Start and finish date and time (UTC)
- Execution Runtime Environment
- Workflow name and version
- Node libraries names and versions
- Trigger action
- Requestor address (if web service trigger)
- Executing user

For each action within the workflow

- Start and finish date and time (UTC)
- Action name/identifier
- Node executing
- Input/output parameters/variables
- Status – Success/Fail etc.

Full Trace

Full tracing is designed to be used during workflow development and trouble shooting. This is due to the volume of data that will be collected. A full trace will record the following:

- Records all activity during workflow execution
- Records variables, input and output values

To ensure data does not grow out of control and offer some storage protection, it must be possible to set a time period such as number of hours to automatically turn off full tracing.

Information Trace

Information tracing is designed to log only key information that the workflow designer has explicitly requested.

Data for information tracing is created through the workflow "Trace Node"

Error Trace

Error tracing if enabled will record all stages in the workflow Activity that error or fail. Errors can be managed using the try catch workflow node, or could be due to an unhandled error.

Error Trace Full

Error trace full is simply an extension of the Error trace functionality, however when a workflow is set to this trace level, and an error occurs, the "Full Trace" information is also recorded.

To achieve this all workflow executions will record a full trace while the workflow executes.

This information is only persisted to the log file if the trace level is set to Full Trace or an Error has occurred.

Note: When logging data that is retrieved from a workflow instance that is based on workflow definition where the *Parameters values in trace* is marked as *Encrypted*, the log must not display unencrypted data. See the Data Encryption section for more information.

6.20.3 Tracing UI

It is a requirement that there is a user interface to manage the assignment of trace settings for workflow. The user interface is to support bulk updating.

6.21 Use case scenarios description

6.21.1 Create trace definition

1. **Workflow designer** clicks *New Trace Definition* button.
2. **System** presents a page to provide the details of trace definition. All input parameters are required and highlighted.
3. **Workflow designer** selects one of the predefined *Trace Levels*: *Full Trace*, *Information Trace*, *Error Trace*, *Error Trace Full*.
4. **Workflow designer** enters *Valid From Date* and *Valid To Date* with one minute resolution.
5. **Workflow designer** enters *Trace Records Expiration* in minutes.
6. **Workflow designer** selects whether the trace definition is *Enabled*.
7. **Workflow designer** enters name of the trace definition.
8. **Workflow designer** clicks *Save* button.
9. **System** presents a page with workflow definitions to which the trace definition can be applied.

6.21.1.1 Pre-conditions

1. **Tenant** has workflow subsystem licensed.
2. **Workflow designer** successfully authenticated against the portal.
3. **Workflow designer** navigated to *Workflow Tracing* page in the portal.

6.21.1.2 Post-conditions

1. Newly created trace definition becomes available for **Tenant**'s **Workflow designers**.

6.21.2 Delete trace definition

1. **System** presents a list of Tenant's trace definitions in a form of grid with columns: *Selected*, *Enabled*, *Trace Level*, *Valid From*, *Valid To*, *Expiration [min.]*.
2. **Workflow designer** selects trace definition using checkboxes from the column *Selected*.
3. **Workflow designer** clicks *Delete* button.
4. **System** warns about all trace records coming from the selected definitions being also deleted.
5. **Workflow designer** confirms the choice.

6.21.2.1 Pre-conditions

1. **Tenant** has workflow subsystem licensed.
2. **Workflow designer** successfully authenticated against the portal.
3. **Workflow designer** navigated to *Workflow Trace Definitions* page in the portal.

6.21.2.2 Post-conditions

1. Selected trace definitions become unavailable for **Tenant**'s **Workflow designers**.
2. Selected trace definitions will not start tracing for new workflow instances.
3. Selected trace definitions stop tracing for running workflow instances.
4. Records created by selected trace definitions are removed.

7 Database and Schema

7.1 New System Fields

To support some of the new platform features the following new fields are required.

- Tags
  - Field to store data tags for each record
  - There could be zero to many tags per row
  - Tags will be searchable for reporting purposes
- Document Links
  - Link zero to many documents to any record

7.2 Multi Tenancy

Both the platform and tenant data is to be fully multi tenanted. The current design of separation by Database Schema currently in place for FSI GO is the preferred design.

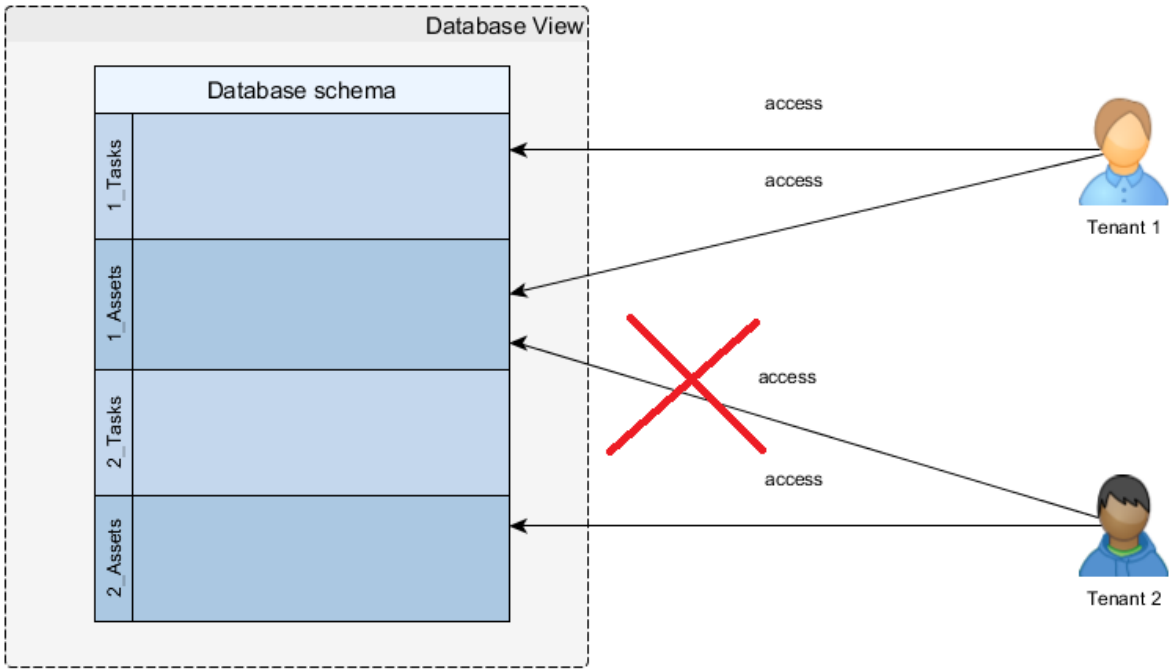
Multi tenancy means that platform in context of *Runtime Execution Environment* is to provide opportunity to work tenants in single environment and their data is to be separated from each other. All tenant assets dynamic and static is to be separated from others tenants.

Separation of static assets is to be provided for:

- Certificates for https connection
- Temporary files
- DLL library files

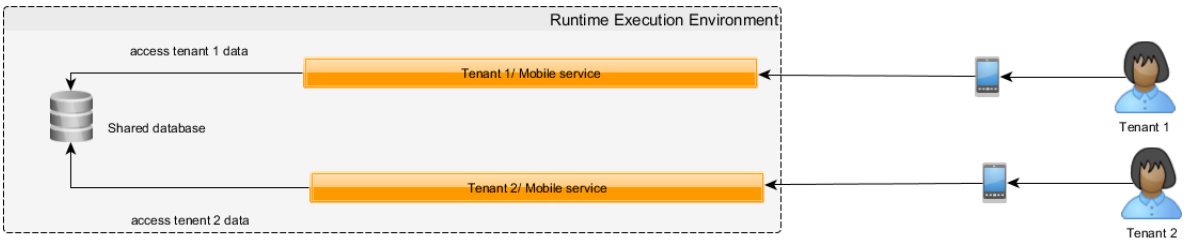
Tenant's dynamic data could be stored in a single database schema. Each tenant is to have access only for its data and he shouldn't see others tenant's assets via GUI. At this level separation of data is provided by the platform. Each data table created by Database Designer is to be identified by tenant prefix and it could be used only by tenant who created it. Access to the tenant dynamic data is to be restricted from designers:

- Web App Designer
- Mobile Designer
- Workflow Designer
- Database Designer



The picture above describes scenario which should be prevented by Runtime Execution Environment. *Tenant 2* shouldn't get access to *Tenant 1* dynamic data tables.

All services exposed via *Runtime Execution Environment* should support multi tenancy.



Dynamic Database

As currently implemented in the FSI GO platform, all tenant data relating to Data Objects is stored in the "Dynamic" database.

For GO Plus, it is required for scalability to be able to host Dynamic data across multiple databases/servers. To achieve this the following fundamental rule applies:

All data Objects for a specific tenant resides in the same Dynamic database,

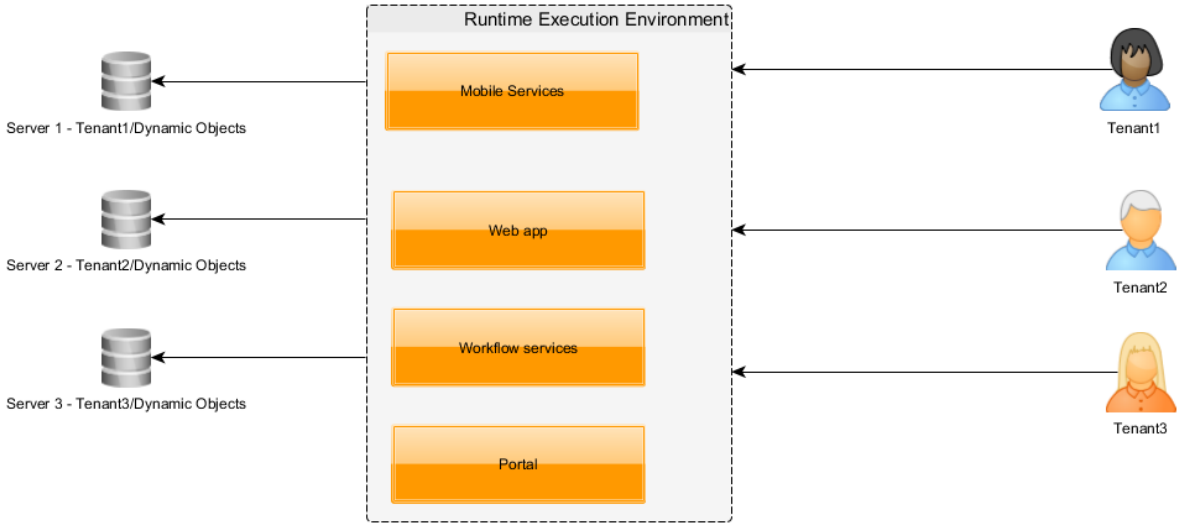
Data Objects for the tenant cannot be distributed across databases or servers.

A user interface is required in the master tenant to manage Dynamic Database connections. For each tenant it is a requirement to set the Dynamic Database connection to use.

It must be possible for migration purposes to change the Dynamic Database connection for a tenant, however the following must be in place:

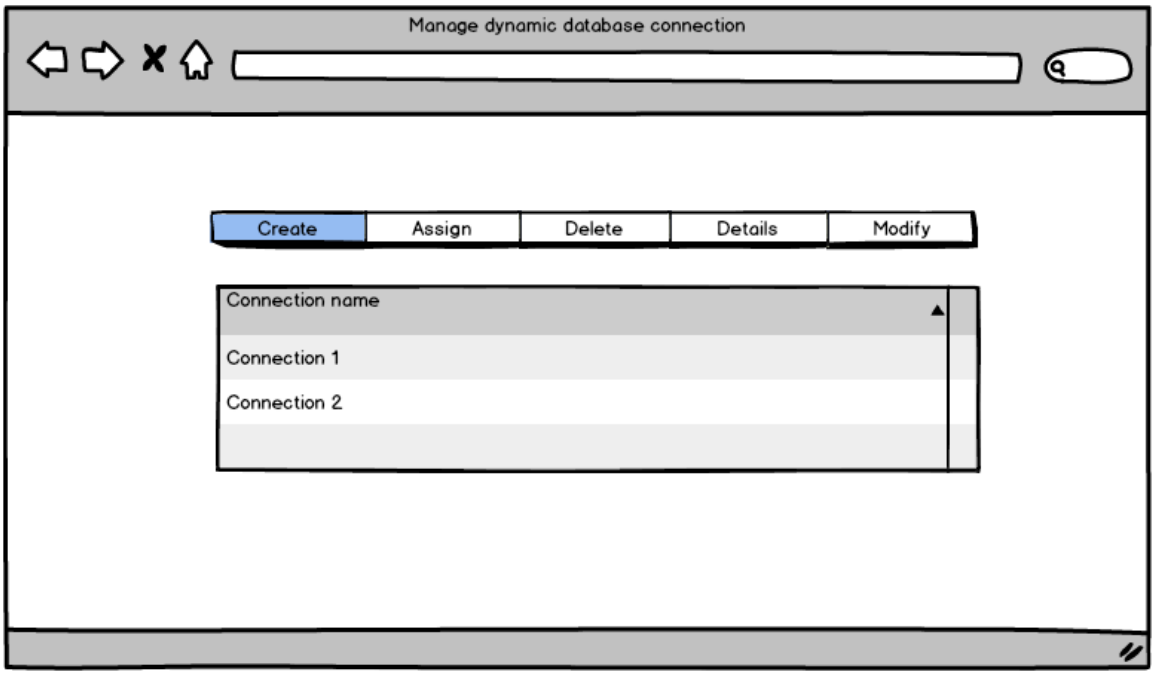
User will be warned of the dangers in changing Dynamic Database connection.

Runtime execution environment allow to each tenent to have their own database server. It means thata REE have to handle multiple connection to different database servers.

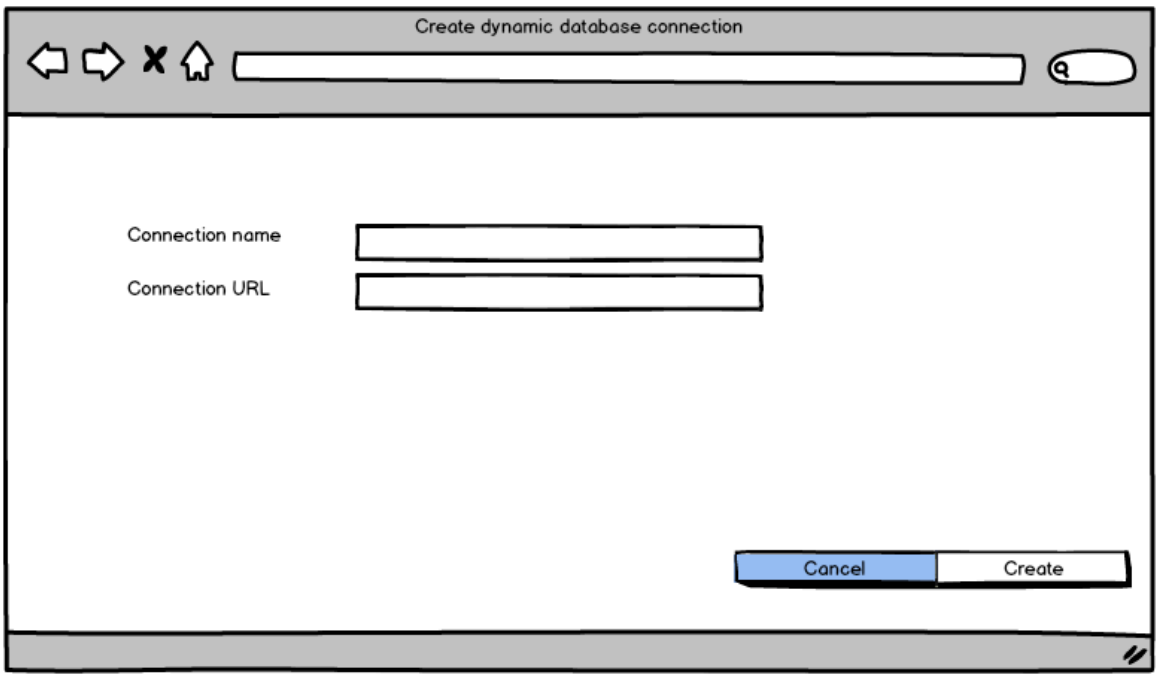


7.3 Mock-ups

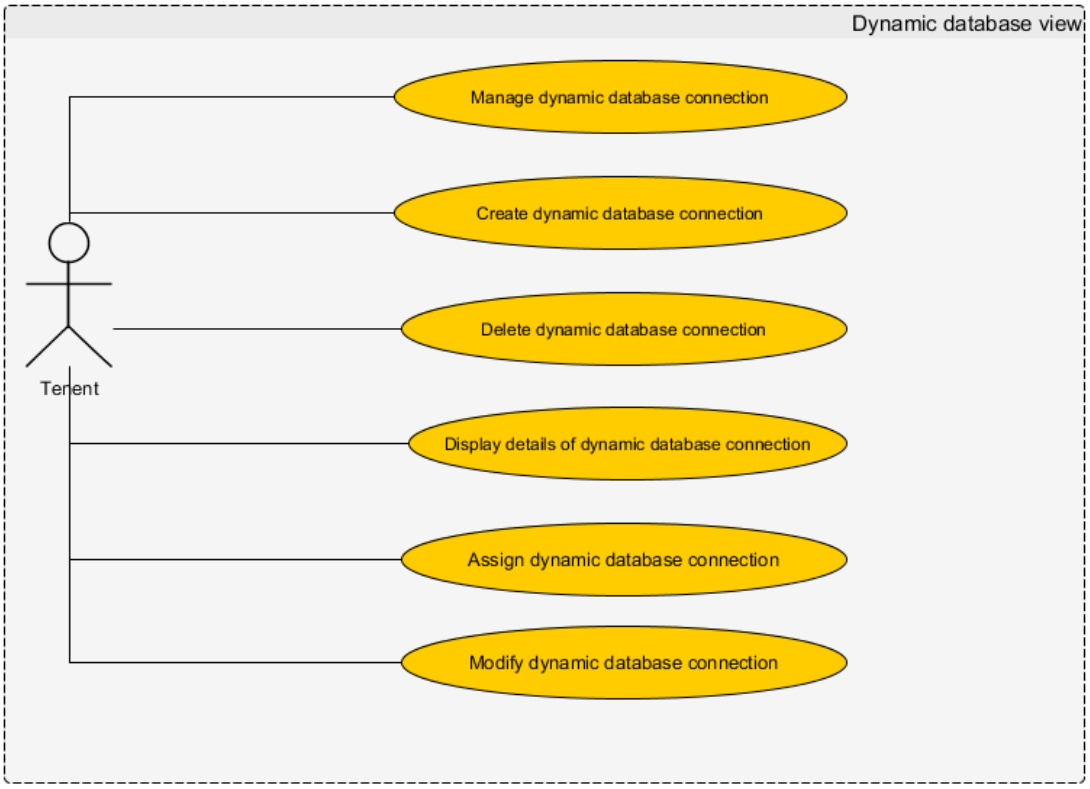
The user interface to manage *dynamic database connection* could be designed as below:



The user interface to create *dynamic database connection* could be designed as below:



7.4 Use case diagram



7.5 Use case scenarios description

7.5.1 Manage dynamic database connection.

Use case describes process which allow to manage dynamic database connection.

1. **Tenant administrator** selects button to manage dynamic database connections.
2. **System** presents GUI to manage dynamic database connections
3. **System** presents table with dynamic database connections**.** Table contains followed columns
   - **a.** Connection name
4. **System** presents panel with button
   - **a.** Create
   - **b.** Assign
   - **c.** Delete
   - **d.** Details
   - **e.** Modify
5. Flow ends

Pre-conditions

- **Tenant administrator** is logged to the system

7.5.2 Create dynamic database connection.

Use case describes process which allow to create dynamic database connection.

1. **Tenant administrator** selects button to manage *dynamic database connections*.
2. **System** presents GUI to manage *dynamic database connections*
3. **Tenant administrator** selects button *create*
4. **System** navigates to the next page
5. **System** presents GUI to create *dynamic database connection*
6. **System** presents input for connection name
7. **System** presents input for connection URL
8. **System** presents input for user name
9. **System** presents input for password
10. **Tenant administrator** enters connection name
11. **Tenant administrator** enters connection URL
12. **Tenant administrator** enters user name
13. **Tenant administrator** enters password
14. **Tenant administrator** submits confirmation button
15. **System** validates entered data
    a. When connection name is not unique **system** raises exception
    b. When connection URL not unique **system** raises exception
    c. When connection URL is not proper **system** raises exception
16. **System** encrypts connection URL
17. **System** registers *dynamic database connection* in *system registry*
18. Flow ends

Pre-conditions

- **Tenant administrator** is logged to the system

Post-condition

7.5.3 Delete dynamic database connection.

Use case describes process which allow to delete dynamic database connection.

1. **Tenant administrator** selects button to manage *dynamic database connections*
2. **System** presents GUI to manage *dynamic database connections*
3. **Tenant administrator** selects button *delete*
4. **System** asks **Tenant administrator** for confirmation
5. **Tenant administrator** confirms the operation
6. **System** validates operation
    a. When the dynamic database connection is used **system** raises exception
7. **System** deletes *dynamic database connection* from *system registry*
8. Flow ends

Pre-conditions

- **Tenant administrator** is logged to the system

Post-condition

7.5.4 Display details of dynamic database connection.

Use case describes process which allow to display details of dynamic database connection.

1. **Tenant administrator** selects button to manage *dynamic database connections*
2. **System** presents GUI to manage *dynamic database connections*
3. **Tenant administrator** selects button *details*
4. **System** presents tenants connected with *dynamic database connection*
5. **System** presents connection URL
6. **System** presents connection name
7. Flow ends

Pre-conditions

- **Tenant administrator** is logged to the system

Post-condition

7.5.5 Assign dynamic database connection.

Use case describes process which allow to connect dynamic database connection with tenants.

1. **Tenant administrator** selects button to manage *dynamic database connections*.
2. **System** presents GUI to manage *dynamic database connections*
3. **Tenant administrator** selects button *assign*
4. **System** navigates to the next page
5. **System** presents GUI to connect *dynamic database connection* with tenants
6. **System** presents tenants list
7. **Tenant administrator** selects tenants
8. **Tenant administrator** submits confirmation button
9. **System** registers association between tenants and dynamic database connection
10. Flow ends

Pre-conditions

- **Tenant administrator** is logged to the system

Post-condition

7.5.6 Modify dynamic database connection.

Use case describes process which allow to modify dynamic database connection.

1. **Tenant administrator** selects button to manage *dynamic database connections*.
2. **System** presents GUI to manage *dynamic database connections*
3. **Tenant administrator** selects button *modify*
4. **System** navigates to the next page
5. **System** presents GUI to modify tenants associated with *dynamic database connection*
6. **System** presents tenants list
7. **System** highlights tenants associated with current *dynamic database connection*
8. **Tenant administrator** selects new tenants
9. **Tenant administrator** submits confirmation button
10. **System** registers association between new tenants and dynamic database connection
11. Flow ends

Pre-conditions

- **Tenant administrator** is logged to the system

Post-condition

7.6 Data Object Security

It is a requirement that access to data objects is controllable through user permissions.

To support this, a user interface is required to enable the setting of the following permission for each data object in the Tenant. It is also a requirement that these permissions can be set at user role level.

There will be two types of permissions for data objects, Designer Access and Integration Access.

Designer Users

- Web App
  - If set then the user can work with the data object in the Web App designer
- Mobile Activity
  - If set then the user can work with the data object in the Mobile Activity designer
- Workflow
  - If set then the user can work with the data object in the Workflow designer
- Schema Update
  - If set the user can update the data object Schema
- Schema Delete
  - If set the user can delete the schema

Integration Users

- Create
  - Create new rows
- Read
  - Read rows
- Update
  - Update rows
- Delete
  - Delete rows
- Get Schema
  - Retrieve the schema for the data object
- Read Encrypted
  - Retrieve un-encrypted data, see the Data Encryption section for more information

7.7 Data Encryption

It is a requirement that the data schema designer can mark individual fields as Encrypted. When a field is marked as encrypted, the data will be fully encrypted to ensure that data is not readable if accessed through the database back end.

Encrypted data will be decrypted for displaying within the Web App, Mobile Apps and Workflow components.

When encrypted data is retrieved through the integration services, data will only be readable if the "Read Encrypted" permission is set.

8 Business Intelligence & Analytics

Business Intelligence (BI) and Analytics are becoming a staple for any new system. While GO plus does not intend to create a fully featured BI tool. The platform is to include a number of features to ease the delivery of data to be consumed by any 3rd party BI tool.

8.1 Pivot Control

A pivot style control is required so that end users will be able to complete real-time data analysis. The control is to have the following functionality:

- Ability to set Row, Colum and Value fields for the pivot
- Ability to bind to selected sets of Data Objects – the query will be generated automatically based on relationships
- Ability to do server-side filtering to make it work client-side on subsets of large datasets
- Ability to export to PDF and Print

Pivot Control binds only to Data Objects. Any data not in Data Objects has to be imported into them.

Note: There are 3rd party controls that provide this type of functionality. Many of them provide this using Silverlight. It is our preference to have HTML5 controls not Silverlight, however this may be reviewed for this control only if the functionality outweighs the technology choice.

8.2 SQL Server Reporting Services

It is a requirement to support the execution of SQL Server Reporting Services (SSRS) reports from within the Web App. To support this, the solution must be able to provide tenant data in a secure manner to the SSRS reporting engine, for both report design and runtime execution.

There should be no reliance on third party applications such as Citrix to design/execute reports.

To execute reports using GO+ credentials, custom SSRS authentication provider should be used. Queries providing data for the report will have access only to a set of automatically generated views. No base table will be accessible to report service account. Views will use credentials passed by the custom authentication provider and thus restrict results according to the permissions granted to the user.

To design reports without direct access to SQL Server a Report Builder ClickOnce app with a prepared Report Model can be used. Report model should be recreated on each Data Object structure change using Report Server Web Services.

The report definitions stored in Reporting Services have to be also saved in GO+ database. The latter is considered the primary source of definitions. It is necessary to have the RDL files synchronized back, should the Reporting Services instance get replaced.

8.2.1 Dashboards

The Web App functionality of the platform, is to fully support a combination of charts and grids arranged on a web page to form a dashboard. It must be easy to create these dashboards through the graphical designer.

Provision in the Charting controls to bind to data using the Web App designer is required. This is to include group by, aggregation and filtering of data to present to the charting components

9 Consuming Data

Data objects are currently defined within the FSI GO platform. The GO Plus project requires easier access to consume data objects.

9.1 oData

Open Data Protocol (oData) is to be available as a data source to access tenant Data Objects.

Note: Full security must be available

9.2 Simple REST

Use standard REST notation for retrieving data. For example to retrieve

https://server/tenant/dataobjects/mydataobject

https://server/tenant/dataobjects/mydataobject/21?firstName=John

Note: REST must still be secure and tenant specific through the Integration user account.

10 Mobile Features

Building upon the FSI GO platform, there are a number of new features required on the mobile. These features are to be cross platform enabled.

There are two modifications necessary in functionality of already existing FSI GO platform. Because of rapid expansion on Arabic markets, there is a need to deliver a date picker with Hijiri calendar support. Also, because of large number of co-existing activities are now have been developed, there has to be a possibility to run (start) another activity from another activity.

10.1 New Mobile Controls

The following new user controls are required for the Mobile:

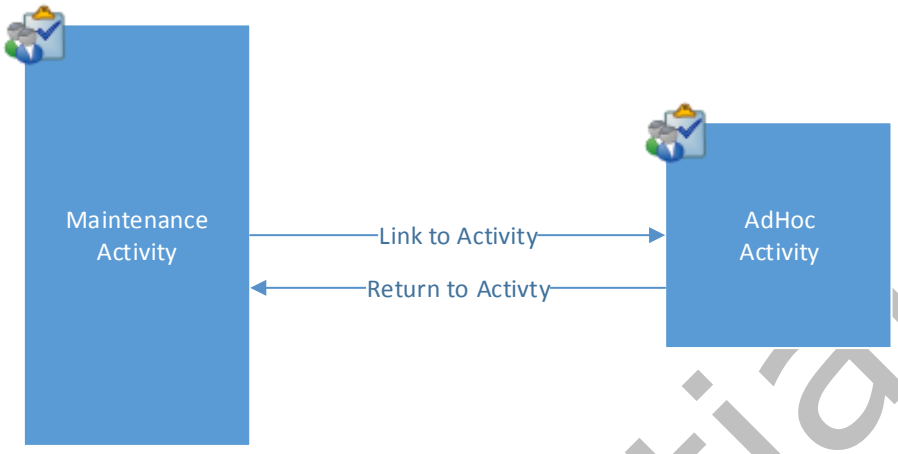
- **Calendar Control** with Switch between Gregorian and Hijri Calendars
  The control will take the date entered and be able to calculate the change into the other calendar format. Existing date picker has to have a functionality of switching between standard (gregorian) calendar and Arabic Hijiri calendar. Additionally, updated calendar control should default to proper calendar based on the system calendar on the mobile device.

10.2 Linked Activities

Currently Activities are stand alone, allowing the user to open one Activity at a time. It is a requirement to be able to link Activities together to perform a wider business function.

For example, an engineer is working on some planned maintenance, and he spots a broken window. Within the Maintenance Activity, it will be possible to have a button/link that will launch the Ad Hoc Task Activity so that the engineer can report the broken window, and then return back to the Maintenance Activity.

Linking Activities will follow the normal Activity permissions, and if the Activity is not available to the user, the Activity will notify the user that it is not possible to open the Activity.



10.3 Run Activity Action

New action has to be introduced in the activities. A designer has to be able to set an action of starting another activity from designed activity. It will be a mobile device that will start modally a second activity allowing users to interact with them and return to previous activity when the executed one will be dismissed.

Please note, that permissions will still be handled the same it was done. In other words if the user will not have a permission to run given **child activity** from given mobile device a warning alert view should be presented informing about the situation. No further actions should be undertaken.

10.4 Runtime Environments for mobile

Mobile services are to become more scalable with support for distributed services. This is further defined in the section Runtime Execution

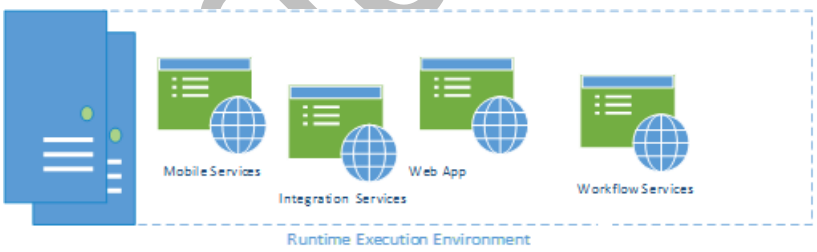
11 Runtime Execution

11.1 Terms dictionary

- Role - responsibility which can serve any node
  - Mobile services
  - Workflow services
  - Integration services
  - Web apps
- Runtime Execution Environment(REE) - set of roles grouped into logical whole
- Load balancer – system that could distribute requests between nodes. Distribution can be handled on hardware or software layer.
- Node - a physical object (computer) which have one or more roles. The node can be part of multiple REE. Each node should have IIS installed.
- Tenant - a user group operating under the REE.

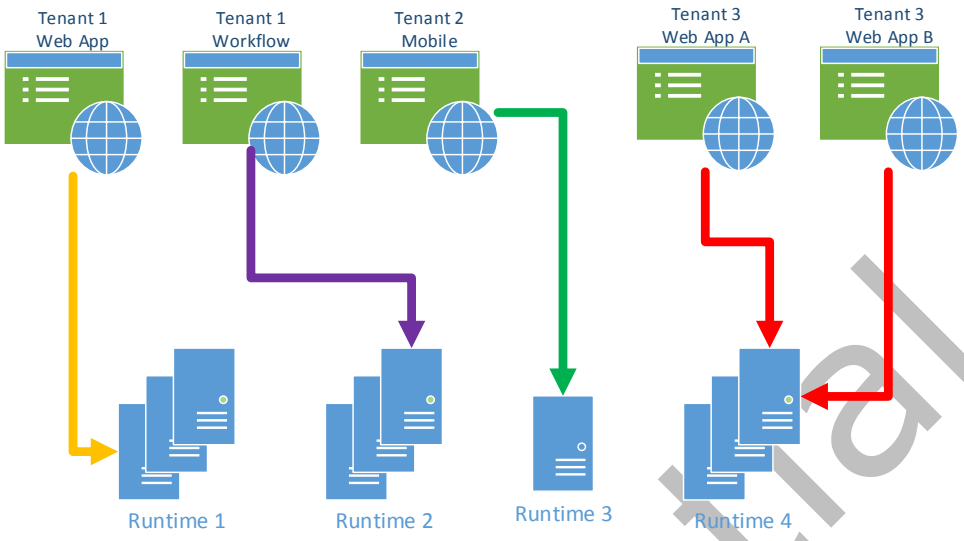
REE provide infrastructure and logical separation to execute Web Apps, Workflow, Integration Services and Mobile Services. For each tenant it is possible to configure and assign which REE are allocated.

Each server or server farm configured within the GO platform is to have all REE components installed and available.



Each deployment of the GO platform as a minimum will require one REE, however will support multiple environments to enable full scalability. Each REE can reside

on a single web server or a web server farm, providing resilience and to support load balancing.

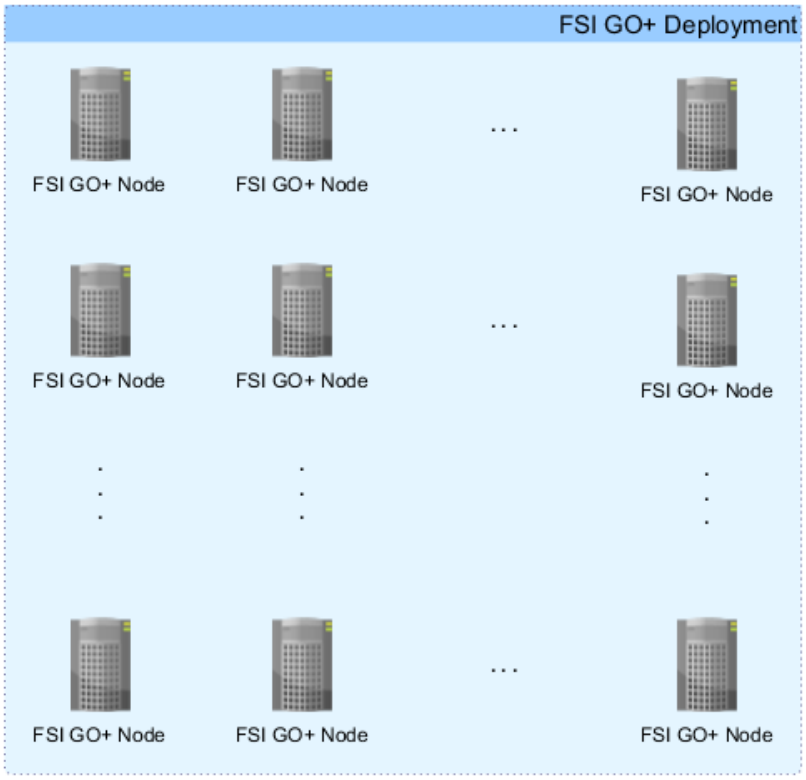


11.2 REE features

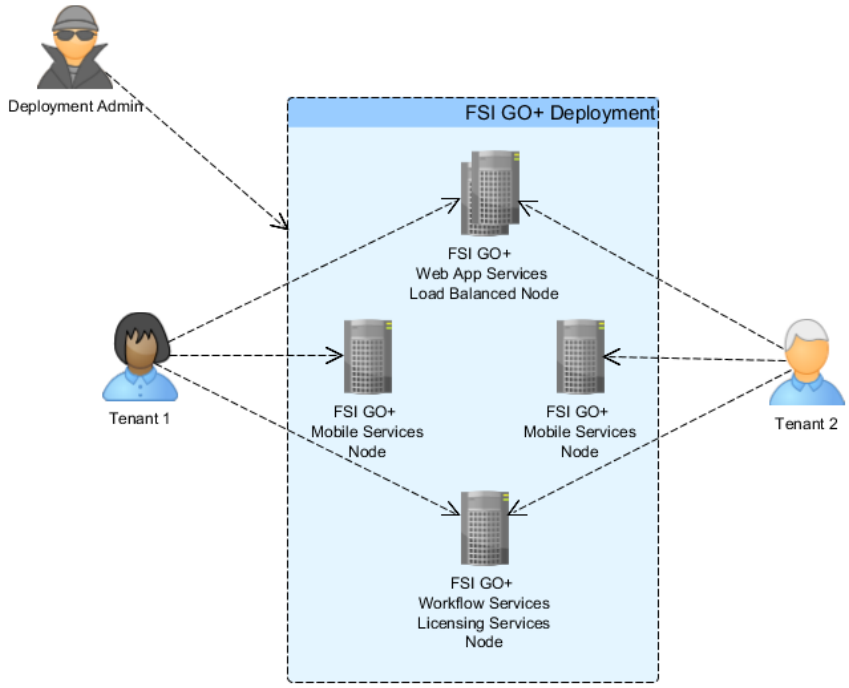
- Each *REE* can be associated with one or more tenants
- Many *REE* can share the *roles* between them
- The *REE* definition may also include load balancers instead of *nodes*
- Definition of *REE* is a set consisting of at least one node or load balancer
- Each node can perform at least one role. Each newly formed tenant will be assigned to the *REE* dependency on the license status.
- A *node* can perform any *role*, or a lot.
- *Role* communicate with each other using the soap protocol.
- *Roles* are independent applications/services under IIS
- *Node* can take part in multiple *REE*.

11.3 Deployment

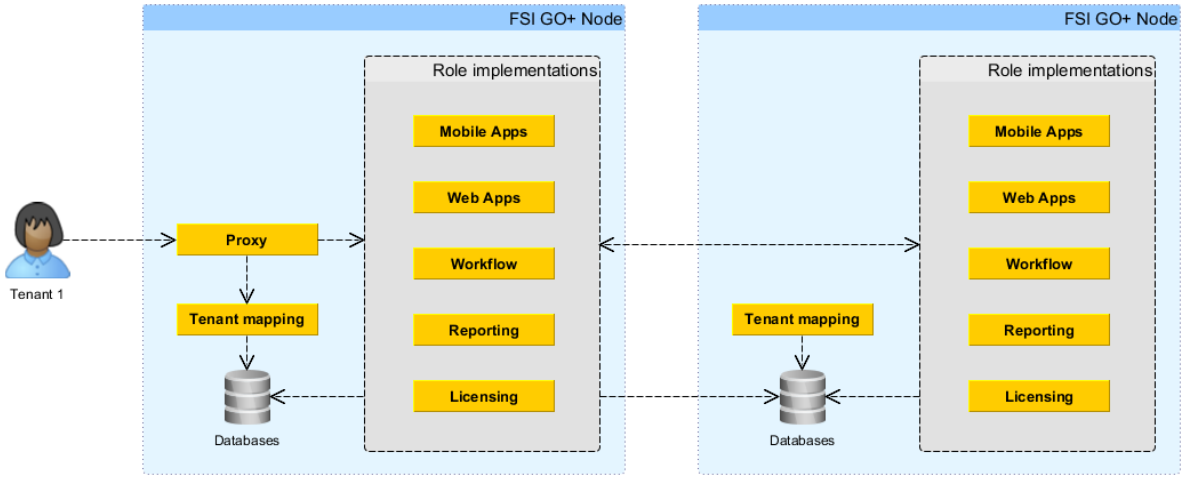
Deployment is a set of nodes. Each of node has installed GO+ services.



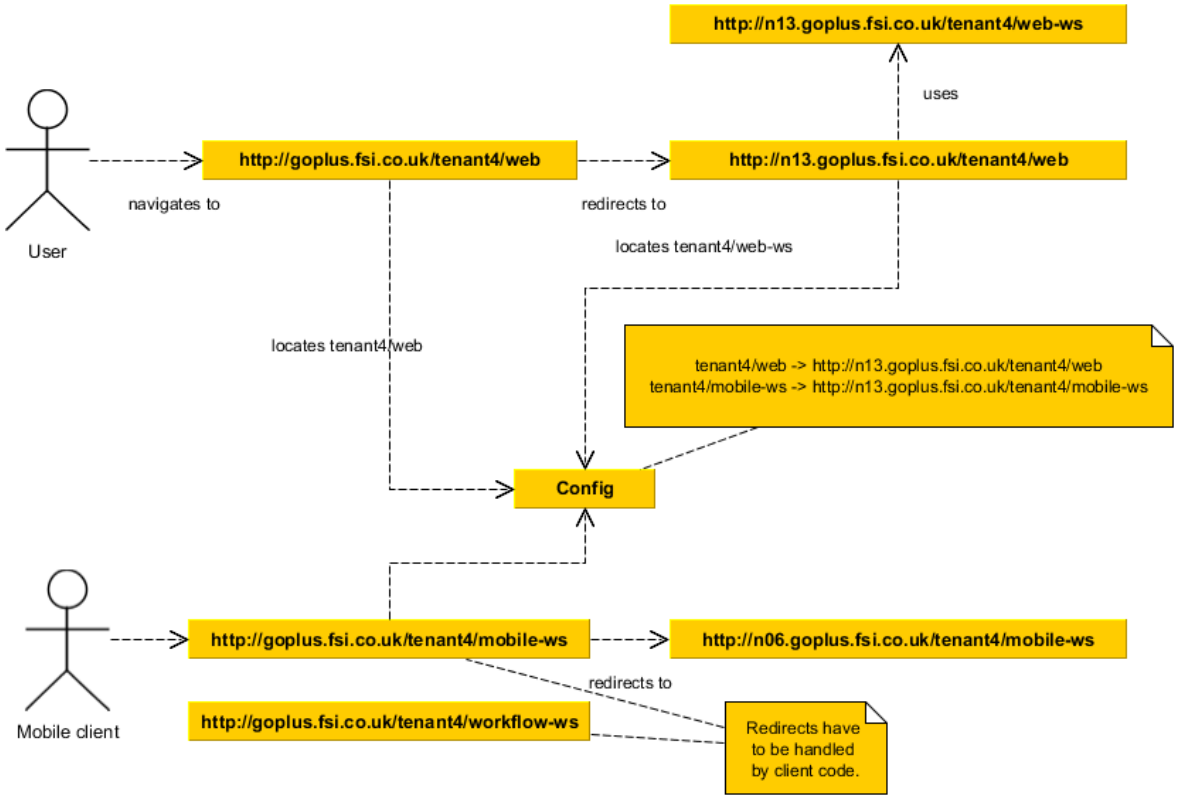
Each role can have more than one node assigned. Each node can serve in more than one role. Each node can be load balanced.



Tenant could interacts with REE through proxy node. Nodes interact directly. This could be designed as below.

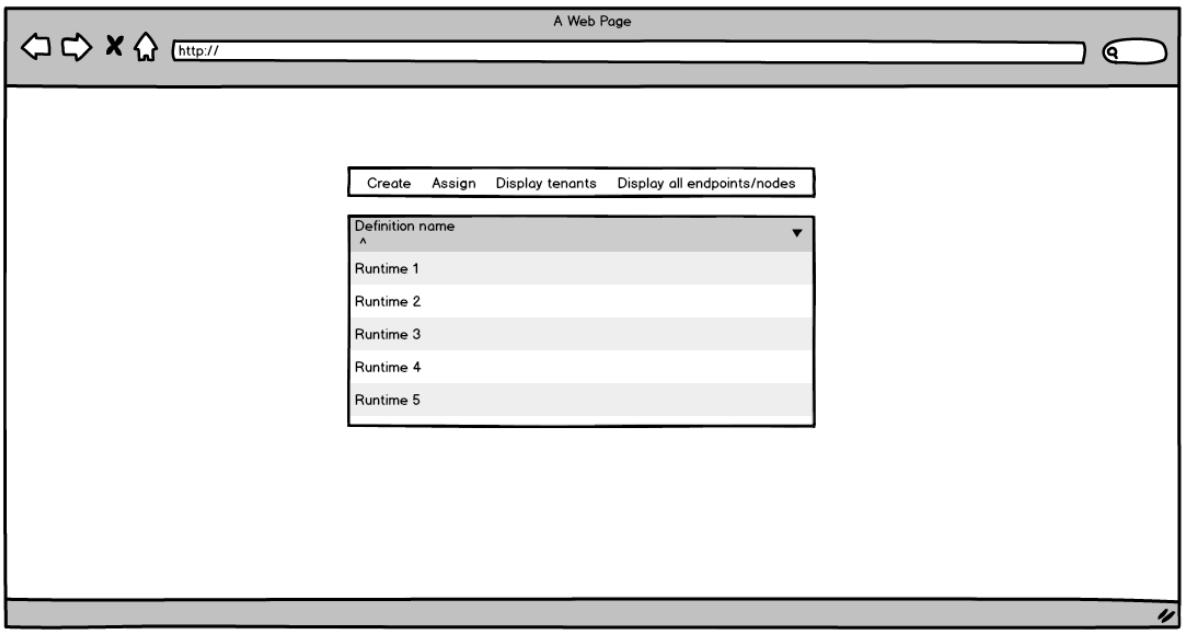


Scenario of communication with shared configuration could be designed as below.

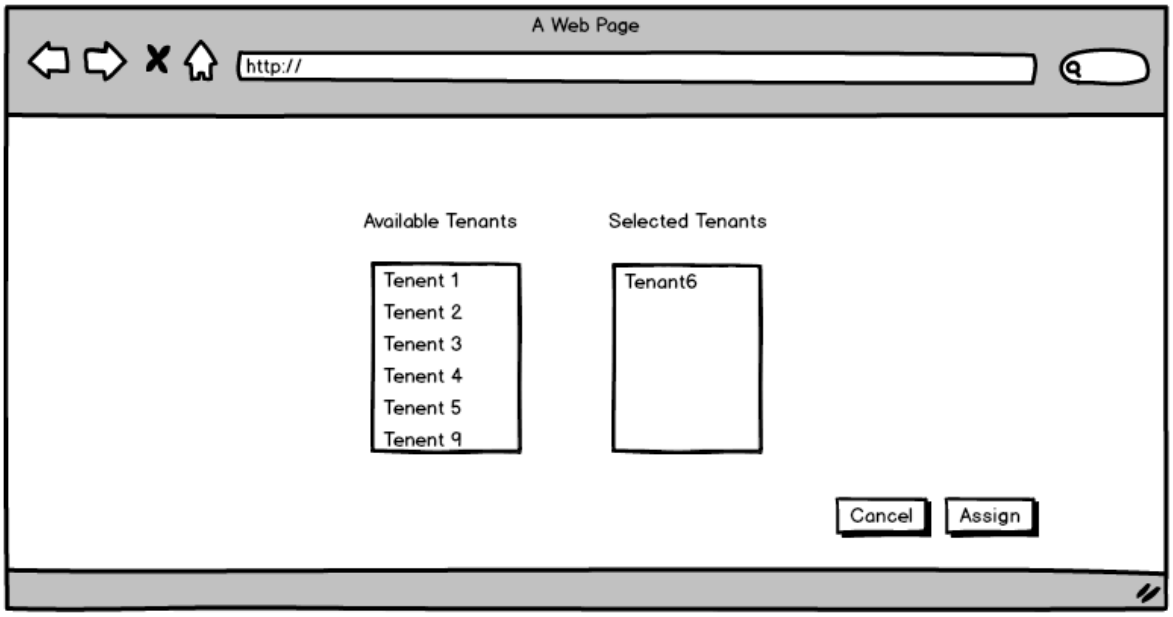


11.4 Mock-ups

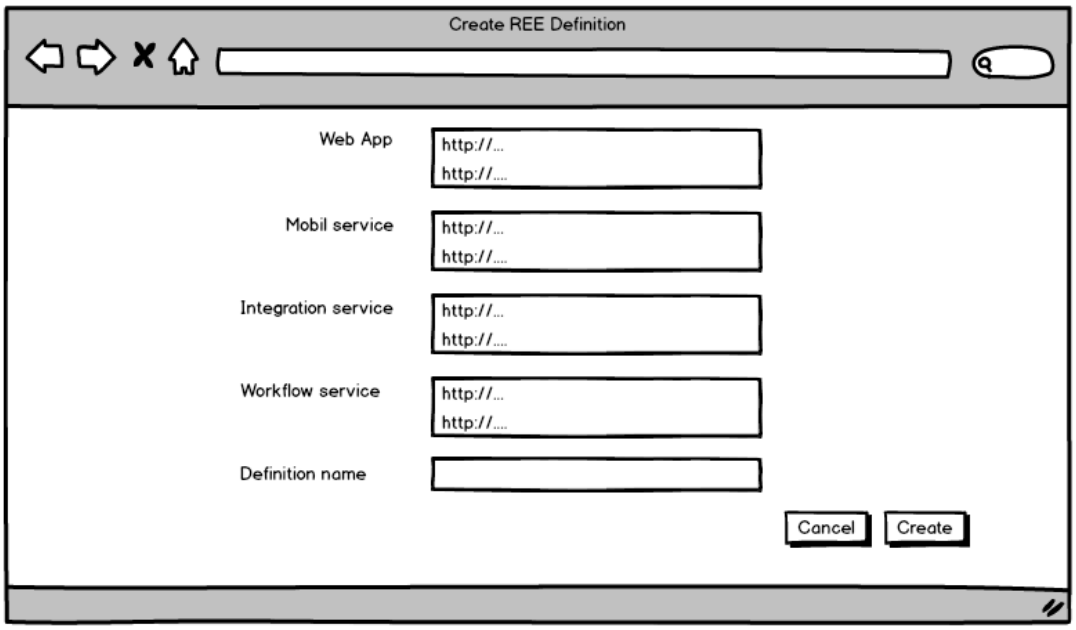
The user interface to manage definition of REE could be designed as below:



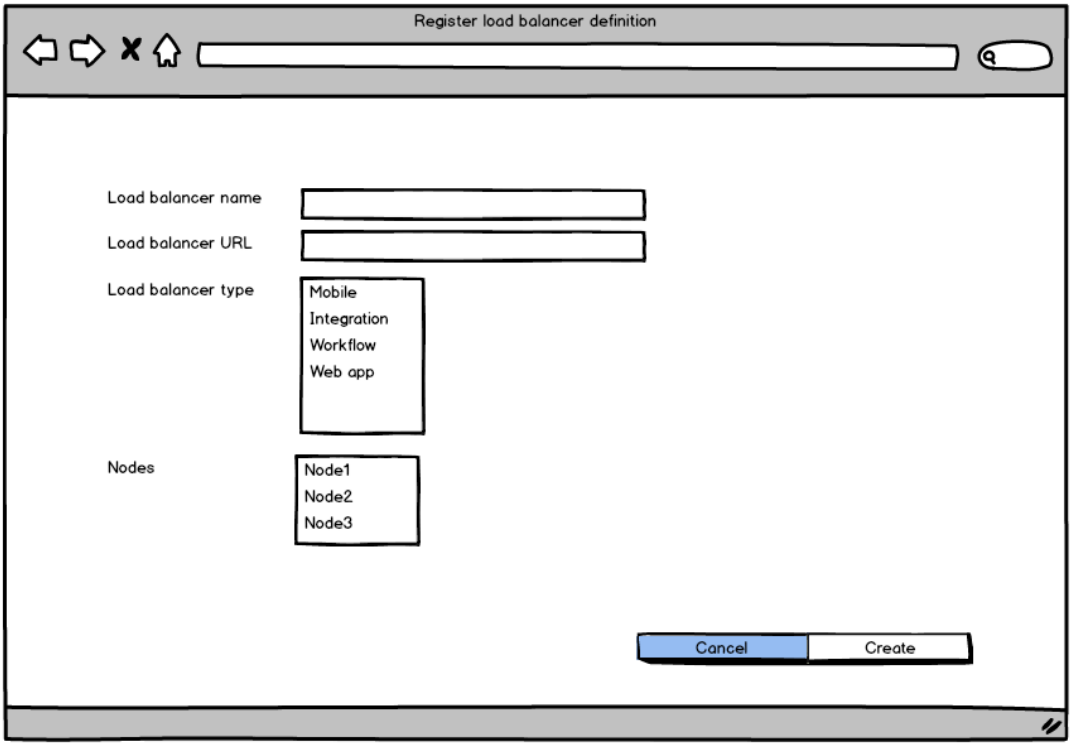
The user interface to assign tenant to REE could be designed as below:



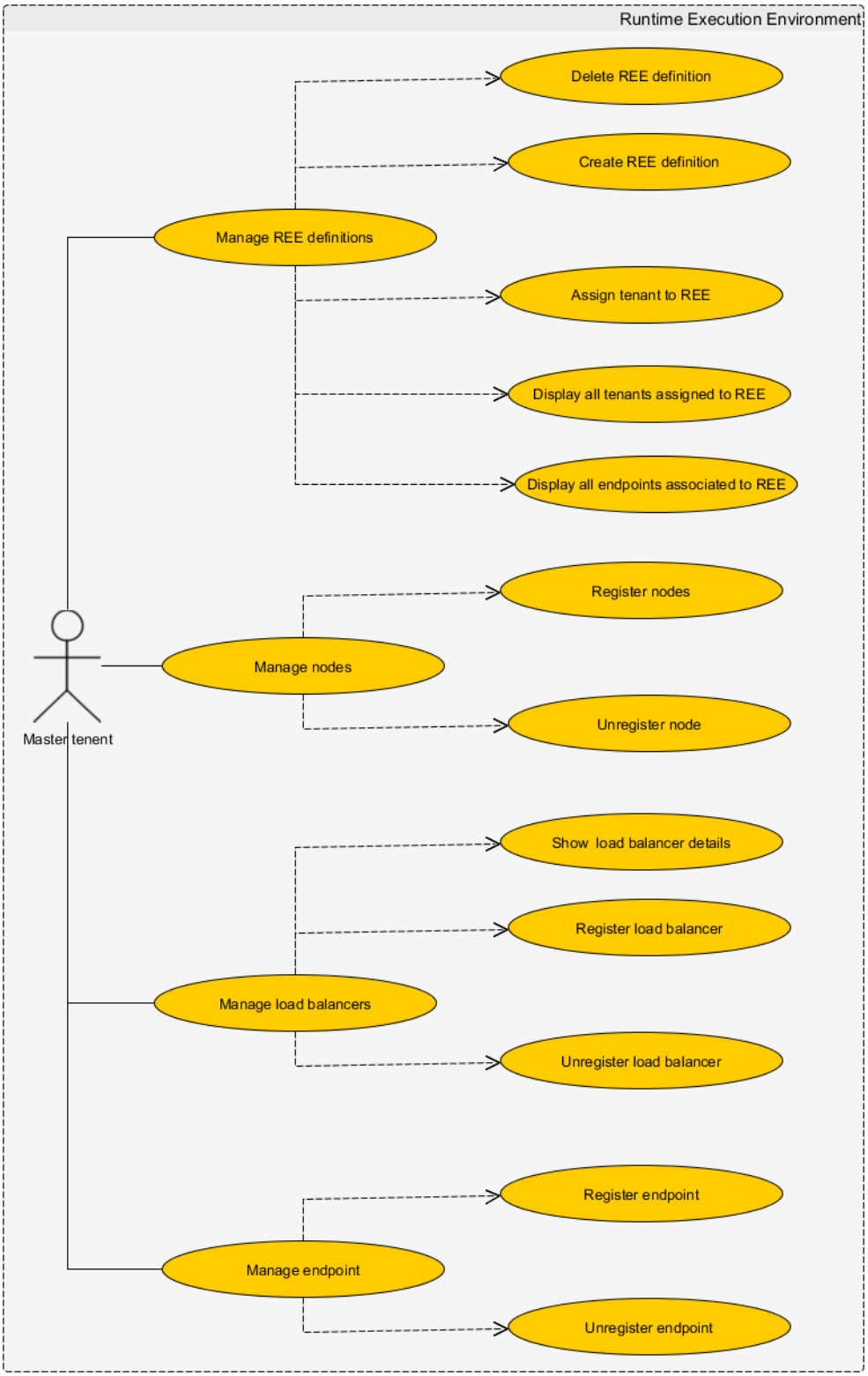
The user interface to create REE definition could be designed as below:



The user interface to create load balancer could be designed as below:



11.5 Use case diagram



11.5.1 Use case scenarios description

11.5.1.1 Manage endpoints.

1. The use case describes process which allow managing endpoints.
2. **Master tenant** selects in portal button to manage endpoints
3. **System** presents table with registered endpoints categorized by endpoint type
4. **System** presents panel with buttons which allows to invoke followed actions
5. Register new endpoint
6. Unregister existing endpoint
7. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

11.5.1.2 Register endpoint.

The use case describes process which allow register particular service in registry.

1. **Master tenant** selects in portal button to register new endpoint
2. **System** presents GUI to register endpoint
3. **Master tenant** selects endpoint type
4. Web app
5. Mobile service
6. Integration service
7. Workflow
8. **Master tenant** fill endpoint URL input
9. **Master tenant** submits button add
10. **System stores** endpoint in registry
11. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

- Endpoint for tenant has been persisted in registry.

11.5.1.3 Unregister endpoint.

The use case describes process which allows to unregister service from registry.

1. **Master tenant** selects in portal button to unregister existing endpoint
2. **System** presents GUI to unregister endpoint
3. **Master tenant** selects endpoint to unregister
4. **When** there are **tenants** associated with endpoint the warning will be presented and operation canceled.
5. **Master tenant** submits button unregister
6. **System unregisters** endpoint from registry
7. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

- Endpoint for tenant has been unregistered from registry.

11.5.1.4 Manage nodes.

The use case describes process which allow managing nodes. Particular node should have all set of services such as:

- Mobile service
- Web app
- Workflow service
- Integration service

1. **Master tenant** selects in portal button to manage nodes
2. **System** presents table with registered nodes categorized by endpoint type
3. **System** presents panel with buttons which allows to invoke followed actions
4. Register new node
5. Unregister existing node
6. Flow ends

Pre-conditions

- **Master tenant** has been logged to the portal.

Post-condition

11.5.1.5 Register node.

The use case describes process which allows to register physical machine which have all services on board, such as:

- Web app
- Mobile service
- Integration service
- Workflow

1. **Master tenant** selects in portal button to register new node
2. **System** presents GUI to register node
3. **Master tenant** fill node URL input
4. **Master tenant** submits button add
5. **System stores** node in registry
6. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

- Node for tenant has been persisted in registry.

11.5.1.6 Unregister node.

The use case describes process which allows to unregister physical machine from registry. Node has all services on board such as:

- Web app
- Mobile service
- Integration service
- Workflow

1. **Master tenant** selects in portal button to unregister existing node
2. **System** presents GUI to unregister node
3. **Master tenant** selects node to unregister
4. **When** there are **tenants** associated with node system presents warning and cancels operation.
5. **Master tenant** submits button unregister
6. **System unregisters** node from registry
7. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

- Node for tenant has been unregistered from registry.

11.5.1.7 Manage REE definition.

The use case describes process which allow managing REE definitions.

1. **Master tenant** selects in portal button to manage REE definitions
2. **System** presents table with registered REE definitions. Table contains followed columns
3. Definition name
4. **System** presents panel with buttons which allows to invoke followed actions
5. Create new REE definition
6. Delete REE definition
7. Assign tenant to REE
8. Display all tenants assigned to REE
9. Display all endpoints associated with REE
10. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

11.5.1.8 Create REE definition.

The use case describes process which allows to create REE definition which could be bind with tenants.

1. **Master tenant** selects in portal button to create REE definition.
2. **System** presents GUI to create REE definition
    a. **System** presents list of endpoint for each category
    b. Web app
    c. Mobile service
    d. Integration service
    e. Workflow
3. **System** presents input to fill definition name
4. **Master tenant** selects combination of endpoints which will be part of REE definition.
5. **Master tenant** fill definition name
6. **Master tenant** submits button create
7. When combination of endpoints duplicate existing definition system raises warning. Flow stops.
8. When name of REE duplicate existing definition name system raises warning. Flow stops.
9. **System** registers definition in registry
10. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.
- It is required to selects on endpoint.

Post-condition

- REE Definition has been persisted in registry.

11.5.1.9 Assign tenant to REE.

The use case describes process which allows assign tenant to REE. Tenant can have only one REE associated with it.

1. **Master tenant** selects in portal button for assigning tenant to REE definition.
2. **System** presents GUI which displays in grid all REE environments
3. **Master tenant** selects REE.
4. **Master tenant** submits assign button
5. **System** presents list of tenants to assign with REE
6. **Master tenant** selects tenants
7. **Master tenant** confirm assignation by clicking button
8. **System** writes changes to database
9. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

- REE definition has been changed in database.

11.5.1.10 Delete REE definition.

The use case describes process which allows to delete REE definition.

1. **Master tenant** selects in portal button for deleting REE definition.
2. **System** lists all available REE definition
3. **Master tenant** selects REE definition to delete
4. **Master tenant** confirm operation by submitting button
5. When there is tenant associated with REE definition, **system** raises warning to the end user and cancels **operation**.
6. **System** removes definition from database
7. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

- REE definition has been deleted from registry.

11.5.1.11 Display all tenants assigned to REE.

Use case describes process which allows to display all tenants assigned to particular REE definition.

1. **Master tenant** selects in portal button for displaying tenants assigned to REE definition.
2. **System** presents grid which displays all tenants assigned to REE definition.
3. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

11.5.1.12 Display all endpoints associated with REE.

The use case describes process which allows to display all endpoints associated with particular REE definition.

1. **Master tenant** selects in portal button for displaying all endpoints associated with REE definition.
2. **System** presents grid which displays all endpoints associated with REE definition.
3. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

11.5.1.13 Manage load balancers.

Use case describes process which allow managing load balancers.

1. **Master tenant** selects in portal button to manage load balancers
2. **System** presents table with registered load balancers
3. **System** presents panel with buttons which allows to invoke followed actions
4. Register new load balancer
5. Unregister load balancer
6. Modify load balancer
7. Show load balancer nodes
8. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

11.5.1.14 Register load balancer.

The use case describes process which allows to register load balancer in system.

1. **Master tenant** selects in portal button to register new load balancer
2. **System** presents GUI to register load balancer
3. **System** presents input control for load balancer name
4. **System** presents types of load balancer
    a. Mobile service
    b. Workflow services
    c. Integration services
    d. Web apps
5. **System** presents lists of end points and nodes which could be part of load balancer
6. **System** presents load balancer URL
7. **Master tenant** fills endpoint URL input
8. **Master tenant** fills load balancer name
9. **Master tenant** selects type of load balancer
10. **Master tenant** selects nodes or end points
11. **Master tenant** submits button add
12. **System** validates user data
13. When required fields are not set **system** raises exception
14. **System stores** load balancer in registry
15. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

- Load balancer has been persisted in registry.

11.5.1.15 Unregister load balancer.

The use case describes process which allows to load balancer from system.

1. **Master tenant** selects in portal button to unregister existing load balancer
2. **System** presents GUI to unregister load balancer
3. **Master tenant** selects load balancer to unregister
4. **When** there are **tenants** associated with load balancer system presents warning and cancels operation.
5. **Master tenant** submits button unregister
6. **System unregisters** load balancer from registry
7. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

- Load balancer has been unregistered from registry.

11.5.1.16 Show load balancer details.

The use case describes process which allows to present load balancer details.

1. **Master tenant** selects in portal button to present load balancer details
2. **System** presents details of load balancer
3. All endpoints
4. All nodes
5. Load balancer URL
6. Flow ends

Pre-conditions

- **Master tenant** is logged to the portal.

Post-condition

12 Installation

It is required that all installations are provided via MSI, or other suitable installation technologies. The installation process must at a minimum conform to the following:

- Each component e.g.: Mobile, Workflow, Runtime must be simple and support upgrades/patching with minimal effort and disruption to production systems.
- All upgrades must take the current settings and not require manual copy/paste of configuration settings.
- Installation must be quick and not require sizable downtime.
- Subsequent upgrades to production releases have to be backward compatible.
- It must be possible to install and upgrade the core components (Web, Mobile & Workflow) independently. For example supporting a Mobile Only deployment of the platform.

13 Testing

It is imperative that the final solution is scalable and we have confidence with load and performance as well as feature stability. To ensure this, the solution has to be well tested.

Testing must be built into the development cycle to ensure we are not coding software that does not function or perform as expected.

13.1 Feature Testing

All of the features are required to be fully tested based upon repeatable test scripts. The following functional areas will need to be tested:

- Design Time Tests
  - Web Designer Tests
  - Mobile Designer Tests
  - Workflow Designer Tests
- Runtime Tests
  - Browser testing
  - Mobile Testing
  - Workflow Testing

13.2 Load Testing

The final solution must perform well as a multi tenanted solution or a stand-alone solution. The system must therefore respond well under stress and load. The following functional areas will require load tests:

- Design Time
  - Parallel Web Designer Tests
  - Parallel Mobile Designer Tests
  - Parallel Workflow Designer Tests
- Runtime Tests
  - Multi Tenants
  - Browser testing
  - Mobile Testing
    - Parallel Data Synchronisation
  - Workflow Testing
    - High Volume
    - Parallel Execution

13.3 Test Strategy

A detailed test strategy document is required, and should detail the key phases of the testing activities

14 Acceptance

The following defines the assets that are to be delivered at the end of the project.

- Fully working functional platform
- Installation files to install all software into a production environment
- Documentation
    - Technical
        - Architecture, Platform Implementation etc.
        - 3rd Party Component details
        - Open Source Licences
        - Server configuration / setup guides
    - Functional
        - API guides
            - JavaScript
            - Web Application - Custom Page API
        - Workflow guides
    - Test Cases
        - Load results
        - Functional test results
- All source code
    - Source code to be documented

15 Scenarios

This section focuses on different application scenarios that FSI expect the final platform to be able to deliver.

15.1 Web Application Scenarios

15.1.1 Contact Management

Ability to record and manage contact information

15.1.2 Help Desk

- Record new tasks/tickets
- Have a configurable State model e.g. New/Active/Complete
- Build and assign Service Level Agreements (SLA) to tasks/tickets
- Review list of outstanding tasks
    - Colour code tasks in a grid/list view based on data values e.g. SLA
- Costing
    - Labour, expenses other costs

15.1.3 Planned Maintenance & Scheduled Work

- Build schedules for planned maintenance
- Build schedules for other work types e.g. cleaning job
- Define and assign SLA
- Dynamic maintenance based on system feeds e.g. runtime of a water pump
- Manage physical Assets e.g. air conditioner units
- Assign and Issue work to engineers
- Assign work to mobile workers
- Print job sheets
- Review outstanding jobs
- Visualise maintenance plan e.g. multi resource calendar
- Costing
    - Parts, Labour, other costs

15.1.4 Property and Estates

- Buildings, Locations

- Flexible location hierarchy e.g. global region, country, country region, town, district site.
- Manage Utilities, Rent, Rate & Leases

15.1.5 Asset Management

- Asset Register
- Warranty
- Whole Life Cycle Costs (reporting)
- Asset Condition

15.1.6 Resource Management

- Managing resource availability
- Scheduling of resources to complete work
- Recording time
- Absence
- Holiday

15.2 Workflow Scenarios

15.2.1 Process Building Data

- Accept Building Alarm feed from Building system via REST trigger
- Read Data
- Create Help Desk ticket for Alarm data

15.2.2 Integrate with Finance System

- Web Page event triggers workflow
- Workflow reads data from GO data Objects
- Workflow connects to finance system by Web Service or writes Flat File (CSV)
- Workflow updates GO data objects with integration information

15.2.3 Web App requires complex logic

- GO Web App triggers In Process workflow, passing in data via the workflow parameters, and waits for response.
- Workflow uses complex process flow to determine answer
- Workflow returns answer to the GO Web App

15.2.4 Configurable State Model

- Workflow is created with all required States and State Transitions
- GO Web App requests a state change using parameters and triggers the In Process workflow. The Web app waits for a response
- Workflow executes the State Transitions and returns the next State

15.2.5 Scheduled BI Data Dump

- Workflow is created to extract data, consolidate and refactor into a defined BI data object.
- Workflow is scheduled to run every 1 hour
- Schedule engine executes workflow

15.2.6 3rd party to 3rd party integration

- Web Service Triggered workflow is created
- When executed, workflow reads data from 1st system and writes data to second system
- Workflow returns completion status

15.2.7 Database Trigger

- An application creates an Update to an SQL table
- On Update trigger, then triggers a Workflow passing in the required data
- Workflow processes data and writes new data back to the database.

15.2.8 System Aggregating

- Workflow runs on a schedule every evening
- Workflow retrieves data from
  - Database
  - Web Service
  - Flat File
- Workflow aggregates data and processes output