CENG 491

DETAILED DESIGN REPORT

HTML5 Canvas Workflow Diagram Editor

iFlowEdit

Sponsored by

INNOVA IT Solutions Inc.

TriUlti

KARAOĞUZ, Mehmet Ozan

KAYRAK, Alaattin

KORKMAZ, Ozan

January 10, 2012

Contents

1. Introduction

This document contains the software requirements for iFlowEdit which is an editor to create and edit workflow diagrams. The approach used in this specification is adapted from IEEE Std 830-1998. The project group, namely TriUlti, guarantees that everything about the project is described clearly and well-organized.

1.1. Problem Definition

The workflow diagrams are used to organize steps of plans, describe processes and their relations among them with input-output information. To organize and manage their projects faster and easier, many companies and organizations use workflow editors. Because of this huge demand, software designers try to produce the most suitable product for costumers. There are diagram editors working on Internet which fulfill the needs of some customers. However, there are a lot of problem that customers encounter. First one is the compatibility issues of the programs, which are caused by the plugins that these programs use to work properly. The second problem is once again compatibility issues which are brought about using different Internet browsers that users use. These issues are frequently encountered by users because most of the designers of the workflow editor programs do not consider these compatibility problems. They write the program working on specific Internet browser and give the program to the users. However, when user tries to enter to the program using an Internet browser which is different that the browser that the program written for, then because of a little compatibility problem, whole program runs incomplete or directly fails. The third and the last problem is that most of the workflow diagram editors work slowly, which causes to make users annoyed. This issue is cause by the usage of the additional programs in the main program. Especially, using plugins cause system to be slower. To sum up, the program that will be designed should not depend on any outer program or plugin, should run on all of the Internet browsers and should run fast enough so that user uses the program with pleasure.

It is important to see sample works on something, because it broadens the horizons of the designers which provide to give people the best product. For this purpose, being open source is the best idea to perform these. However, there is not enough open source programs designed on workflow editors. So, the system that will be designed should be open source.

1.2. Purpose

This document provides a complete description of all the functions and the specifications of the INNOVA IT‘s HTML5 Canvas Workflow Diagram Editor - iFlowEdit. This documentation is written for the purpose of guiding development process of the iFlowEdit.

1.3. Scope

This document contains a complete description of the design of iFlowEdit. All the components and their functions of the product are explained in the document. The intended audiences are code developers of the product. Hence, this report will serve as a guideline throughout the development of the Project.

1.4. Overview

This document contains nine more chapters except this chapter. Second chapter is System Overview in which we will give a detailed design of the system. Then, the Design Considerations follows which contains design issues related to assumptions, dependencies and constraints. The next chapter is the Data Design which includes information and data domain of the system and how it is transformed into structures. In the fifth chapter, architectural design of the system will be given with the subsystems and their components. After that User Interface Designs are explained with screen examples that are created with tools by inspired from similar applications. Then detailed design of the system briefly explained. In the eighth part, libraries and tools that are thought to be used in the project are listed. Then, the planning of the implementation of the project is explained in the eighth part. Finally, document will be finalized with conclusion part.

1.5. Definitions, Acronyms and Abbreviations

HTML:	Hypertext Markup Language
XAML:	Extensible Application Markup Language
WF:	Workflow Foundation
IEEE Std:	Institute of Electrical and Electronics Engineers Standards
SVN:	Apache Subversion
JSON:	JavaScript Object Notation
XML:	Extensible Markup Language
GUI:	Graphical User Interface
UI:	User Interface
CRUD:	Create Read Update Delete
SRS:	Software Requirements Specifications
Config:	Configuration

1.6. References

[1] Projects @ Apache Documentation, http://projects.apache.org/docs/index.html

[2] Software Requirements Specification HTML5 Canvas Workflow Diagram Editor, TriUlti, Dec 25, 2011

[3] Oryx Configuration Specifications, http://oryx-editor.googlecode.com/files/OryxSSS.pdf

[4] JSON and XML, http://www.json.org/xml.html

[5] IEEE Std80, http://standards.ieee.org/findstds/interps/80.html

[6] Ext-JS usage, http://docs.sencha.com/ext-js/4-0/

[7] Processing.js references, http://processingjs.org/reference

[8] XAML explanations, http://msdn.microsoft.com/en-us/library/ms752059.aspx

[9] CSS, http://www.w3.org/Style/CSS

2. System Overview

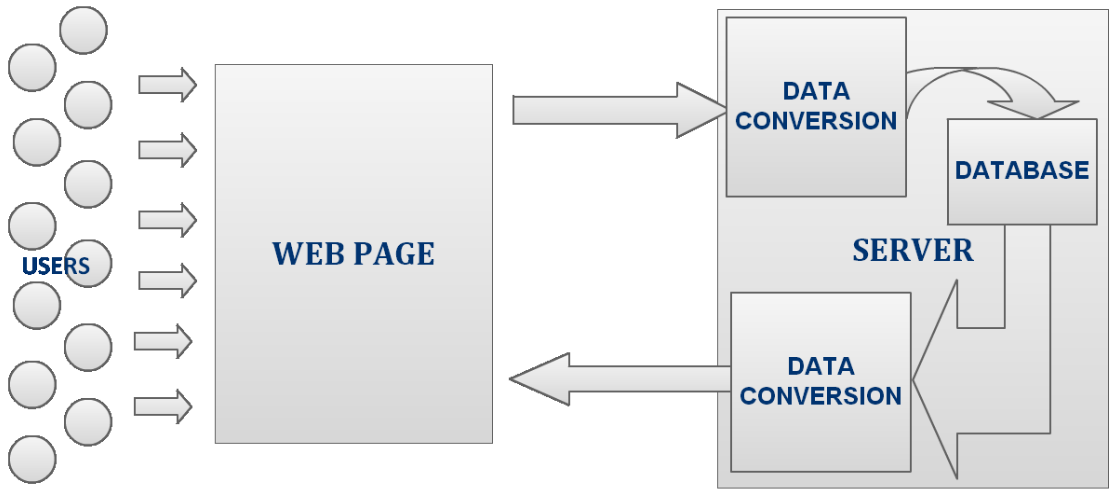


Figure 1 - System Overview

The general overview design of the project is given at Figure 1. System consists of two main parts. The first one is the client side that contains the user interface to create diagrams and their specialties by users. Second one is the server side of the system that provides functionality of dynamically loading and saving of client side of system. Server handles the background processes of the system and holds all of the user data at database. With this way, user can save any diagram data into server side in XAML type to later use for Microsoft WF. There also exist minor parts of server side, which will be used as data conversion of sent and received data. They manipulate the communication data between the server and client. Client side contains two subsystems, which will be user and administrator components with manipulation right on the system with their sub components. Server side contains one subsystem, which will be server component that handles the processes of administrators and users on the system.

3. Design Considerations

3.1. Design Assumptions, Dependencies and Constraints

- There is a connection, e.g., Internet connection and Intranet connection, between client side and server side.
- Data transfer rate between client side and server side is enough to provide a good performance experiences to users.
- Server has enough bandwidth.
- Client side has a web browser which supports HTML5.
- Reports during development phase of the project will be prepared according to IEEEStd830.
- The application shall have a relational database for users and documents.
- The server application shall be implemented using C#.
- The user interface shall be implemented using HTML5, JavaScript, Processing-JS, Ext-JS.
- JSON shall be used to data communication with server application.
- XAML shall be used to store user documents on server.
- All definable options by users shall have default values supplied by the application.
- The server application shall log error messages when an error is detected.
- The application shall be reachable from any browser.
- On any error user data should not be lost.

3.2. Design Goals and Guidelines

During the design of iFlowEdit, there will be some goals and guidelines that shape the structure directly. Those goals and guidelines are listed in an order from the highest priority to the lowest priority.

- System will be designed as simple as it can be done. That is, when a user, who does not have any idea about online editor concept, enters into the system, user can easily start to use the application.

- System will be designed to consume low amount of resources of user computer.

- WF.Net connection will be provided as a main goal of system to be compatible with other programs which are based on WF.Net.

- iFlowEdit will be user friendly. It always guides users in order to prevent them to design invalid diagrams.

- System will be designed by using abstraction principle which is a basic dictum that aims to reduce duplication of information in a program whenever practical by making use of abstractions provided by the programming language.

- Reusability is also an important goal of the project. Because reusable modules and classes reduce implementation time, and facilitates code modifications when a change in implementation is required.

- Maintainability will be considered to ease the additions of new requirements, maintenance in future and environment changes that the program meets.

- SVN will be used as team software process which provides a defined operational process framework that is designed to help teams to organize and produce large-scale software projects of sizes beyond several thousand lines of code.

4. Data Design

4.1. Data Description

4.1.1. User Information Data

User information will be hold in database to store information of any user related data that the system will need to keep. Database diagram of the system is in Figure 2. It contains two tables to hold the information which are user and document tables. User table will be used to access to the information of a user; which are id, username, password, email and role of the user. Every user will have unique id, so id will be the primary key of the user table. Document table will be used to access to the saved data of a user. It holds userId and name of the file stored in a different location. Name of the file will be unique, so name attribute is primary key. A user can have one or more saved data on the folder of saved data.

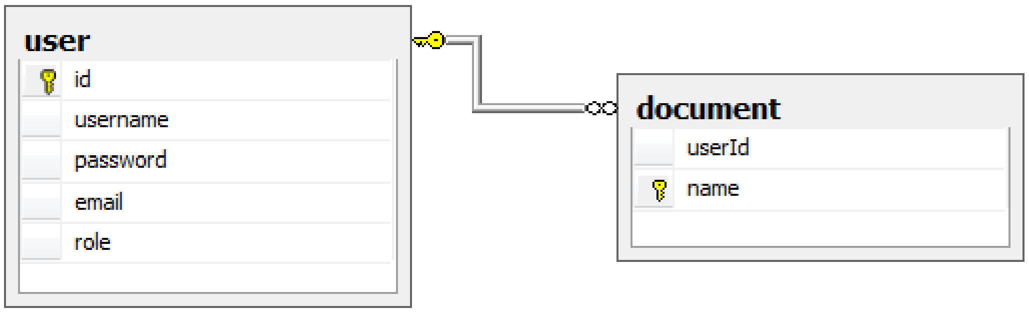


Figure 2 - Database Tables

4.1.2. Server-Client Communication Data

Communication between server and client will be with JSON. Just before sending data from server to client, server side program translates XAML document file to JSON format. Same with this way, right after getting all the data, which sent from client, server translates this data to XAML in order to store in this format.

4.1.3. Document Information Data

Documents will be stored in XAML format of .NET Framework programming model. Any change on user canvas will be stored after the conversion of JSON data. On the other hand, user can load saved data by throwing the data file back from saved data folder and converting it into JSON script. Main aim of the document information folder is to keep the personal data of a user in a specified location.

4.1.4. Configuration Data

The data of each item in the stencil set, e.g., weak entity of entity relation diagram, is called configuration data. Configuration data will be stored in the files and will be in the JSON format. The reasons of choosing JSON are:

1. It is self-descriptive so that human can read it easily.
2. It is lightweight. It needs minimal writing to add information.
3. It is easily processed with JavaScript.

Another way is storing data in XML files. This approach, however, has two pitfalls. One of them is that communication between client side and server side will be done by using JSON, so converting XML to JSON will require an extra job. The other one is that size of the XML file is larger because it needs much more writing to add information.

The comparison below between two types illustrates the situation. Figure 3 shows the XML version and Figure 4 shows the JSON version.

```xml
...
<name>weak entity</name>
<width>100</width>
<height>100</height>
<color>#FFFFFF</color>
<connectionPoints>
    <connectionPoint>top</connectionPoint>
    <connectionPoint>right</connectionPoint>
    <connectionPoint>bottom</connectionPoint>
    <connectionPoint>left</connectionPoint>
</connectionPoints>
...
```

Figure 3 – XML

```json
{
...
    "name": "weak entity",
    "width": "100",
    "height": "100",
    "color": "#FFFFFF",
    "connectionPoints": [
        "connectionPoint": "top",
        "connectionPoint": "right",
        "connectionPoint": "bottom",
        "connectionPoint": "left"
    ]
...
}
```

Figure 4 - JSON

4.2. Data Dictionary

a. ConnectionInstance

connText: Holds the text value that may be added onto the connection. It takes string as a value.

endActivity: Holds which activity is the ending point of the connection. It takes ActivityInstance as a value.

endConnPointIndex: Holds the index of the connection point of the activity. It takes an integer value.

priority: Holds the priority value of connection to make connection on or above other objects. It takes integer as a value.

startActivity: Holds which activity is the starting point of the connection. It takes ActivityInstance as a value.

startConnPointIndex: Holds the index of the connection point of the activity. It takes an integer value.

getConnText: Gets the text of the connection that was written onto it. It takes no parameter and returns string.

getEndActivity: Gets the ending point of the activity. It takes no parameter and returns ActivityInstance.

getEndConnPointIndex: Gets the index of the connection of the ending activity. It takes no parameter and returns integer.

getPriority: Gets the priority value of a connection. It takes no parameter and returns integer.

getStartActivity: Gets the starting point of the activity. It takes no parameter and returns ActivityInstance.

getStartConnPointIndex: Gets the index of the connection of the starting activity. It takes no parameter and returns integer.

setConnText: Sets the text of the connection that may be added into it. It takes string as a parameter and returns void.

setEndConnPointIndex: Sets the index of the connection of the ending activity. It takes integer as a parameter and returns void.

setEndActivity: Sets the ending point of the activity. It takes ActivityInstance as a parameter and returns void.

setPriority: Sets the priority of the connection. It takes integer as a parameter and returns void.

setStartActivity : Sets the starting point of the activity. It takes ActivityInstance as a value and returns void.

setStartConnPointIndex: Sets the index of the connection of the starting activity. It takes integer as a parameter and returns void.

b. ConnectionDefinition

colour: Holds the colour of the connection object. It takes triple value which contains three integers.

name: Holds the name of the connection object. It takes string as a value.

thickness: Holds the thickness of the connection object. It takes integer as a value.

type: Holds the type of the connection object. It takes value which contains string as a value.

draw: The main function of the connection object and it draws the object according to the attributes of it.

c. ActivityInstance

activityText: Holds the text that was added onto the activity object. It takes string as a value.

angle: Holds the angle of the activity object. It takes integer as a value.

locked: Holds the locking condition of the activity whether the activity object is locked or not. It takes boolean as a value.

opacity: Holds the opacity value of the activity object. It takes float as a value.

posx: Holds the coordinate of the activity object on the x-axis. It takes integer as a value.

posy: Holds the coordinate of the activity object on the y-axis. It takes integer as a value.

priority: Holds the priority value of the activity object. It takes integer as a value.

getActivityText: Gets the text value of the activity object that was added on to it. It takes no parameter and returns string.

getAngle: Gets the angle value of the activity object. It takes no parameter and returns integer.

getLocked: Gets the locking condition of the activity object whether it is locked or not. It takes no parameter and returns boolean.

getOpacity: Gets the opacity value of the activity object. It takes no parameter and returns integer.

getPosx: Gets the coordinate of the activity object on x-axis. It takes no parameter and returns integer.

getPosy: Gets the coordinate of the activity object on y-axis. It takes no parameter and returns integer.

setActivityText: Sets the text value of the activity object that was added on to it. It takes string as a parameter and returns void.

setAngle: Sets the angle value of the activity object. It takes integer as a parameter and returns void.

setLocked: Sets the locking condition of the activity object whether it is locked or not. It takes boolean as a parameter and returns void.

setOpacity: Sets the opacity value of the activity object. It takes integer as a parameter and returns void.

setPosx: Sets the coordinate of the activity object on x-axis. It takes integer as a parameter and returns void.

setPosy: Sets the coordinate of the activity object on y-axis. It takes integer as a parameter and returns void.

d. ActivityDefinition

colour: Holds the colour of the activity object. It takes triple value which contains three integers.

connectionPoints: Holds the positions of the all of the connection points of the activity object. It takes a vector object that holds pairs of integer values.

diagramType: Holds the diagram type of the activity object which determines the general properties of the activity. It takes string as a value.

height: Holds the height of the activity object. It comes with default size at the beginning. It takes integer as a value.

name: Holds the name of the activity object. It takes string as a value.

order: Holds the value of the activity objects that stays at stencil set. It determines the order on the stencil set.

shapePath: Holds the path value of the activity object which directs the system to the specific director. It takes string as a value.

width: Holds the width of the activity object. It comes with default size at the beginning. It takes integer as a value.

draw: The main function of the connection object and it draws the object according to the attributes of it.

e. Workflow

activityArray: Holds all of the ActivityInstance objects on the data structure. It takes a vector object which is a series of ActivityInstance objects as a value.

connectionArray: Holds all of the ConnectionInstance objects on the data structure. It takes a vector object which is a series of ConnectionInstance objects as a value.

id: Holds the id number of the workflow object which distinguishes the workflow objects.

title: Holds the title of the workflow document. It takes string as a value.

state: Holds the condition of the workflow whether it is saved or not. It takes boolean as a value.

changeTitle: Changes the title of the workflow document. It takes string as a parameter and returns void.

f. FileController

createDocument: Creates a file that will hold the data of the workflow when a workflow is loaded. It will take user id as a string and fileName as a string in parameter part and returns void.

deleteDocument: Removes a file from the Docs file in the server. FileName as a string is a parameter of the function and it returns void.

readDocument: Reads file from the Docs file, which holds all of the saved data. It will take string fileName as a parameter and returns void.

updateDocument: Changes the document properties. Desired information will be taken as a parameter. UserName, email and password as a string are parameters of the function and it returns void.

g. DatabaseController

createDocument: Creates a file in the Docs file. It takes userId, fileName as a string in parameters part and returns void.

createUser: Creates a user that will use the system. It takes userId, userName,email, password as a string in parameters part and returns void.

deleteDocument: Removes a document from Docs file in the server. It takes fileName as a string in parameter part and return void.

deleteUser: Removes a file from database in the server. userId as a string is a parameter of the function and it returns void.

readDocument: Reads the document that is specified in parameter. It takes fileName as a string in parameter part and return FileStream.

readUser: Reads all of the documents of the user. It takes userId as a string in parameter part and return object array.

updateDocument: Changes document properties. It takes userId, fileName as a string in parameters part and returns void.

updateUser: Changes user properties. It takes userId, userName,email, password as a string in parameters part and returns void.

5. System Architecture

In this part, architecture of the system of the diagram editor will be mentioned in details with whole system diagram.

5.1. Architectural Design

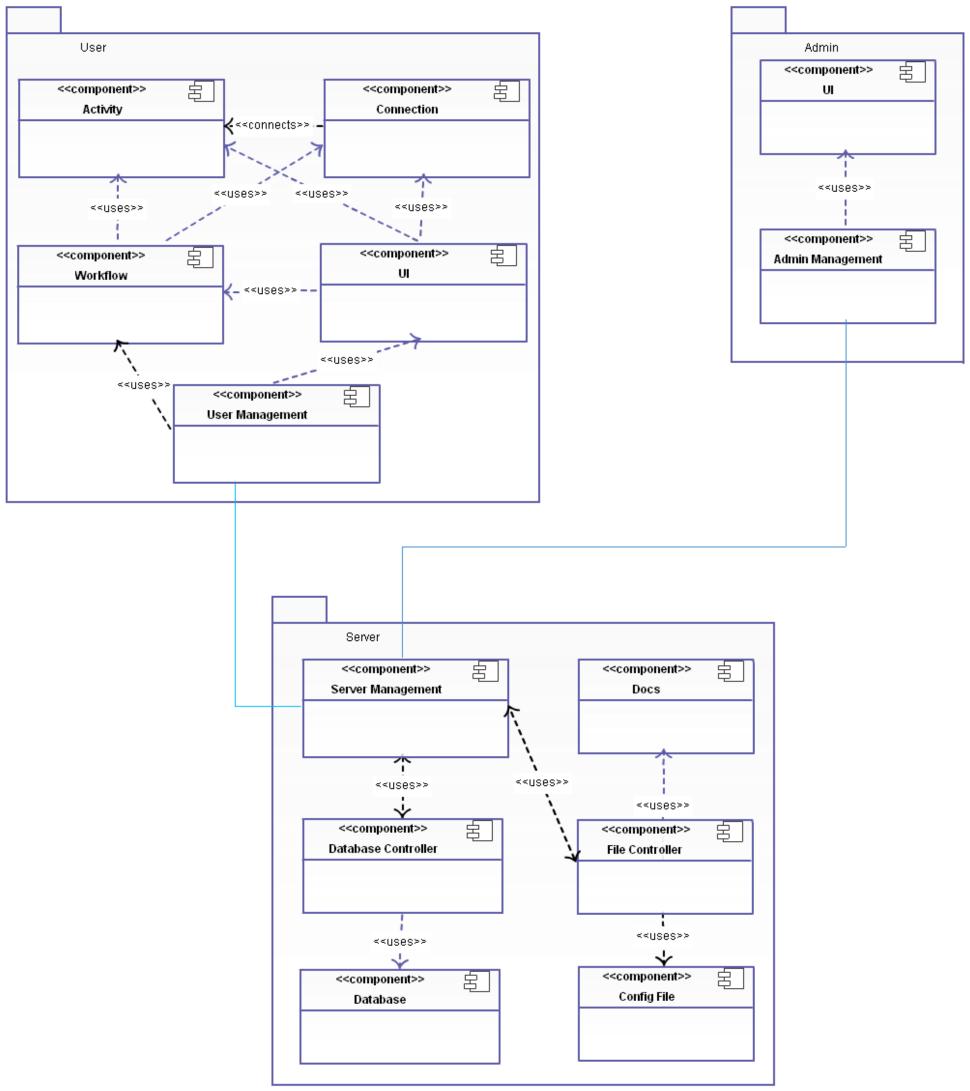


Figure 5 - Component Diagram

5.2. Description of Components

5.2.1. User

5.2.1.1. Processing Narrative for User

User component is responsible for performing every possible action that users of the iFlowEdit may do. These actions are creating workflows, editing workflows, saving workflows, loading workflows and deleting workflows. The actions except editing can be performed no more than one way. Editing action, however, includes creating, editing and deleting activity instances and creating, editing and deleting connection instances. Editing activity can be done by changing any attributes like color, shape or position of that activity. Editing connection can be done by changing any attributes like color, thickness or position of the connection.

5.2.1.2. Interface Description of User

User component can take inputs only from users of the iFlowEdit via GUI. As it is mentioned on SRS Document of iFlowEdit, the system interacts with user by a GUI. UI sub-component is responsible for processing inputs which are taken from GUI. For example, if a user clicks on the save button, UI sub-component calls the necessary functions to save the workflow. Every mouse event and keyboard event that are accepted by the system are the possible inputs of the User component.

User components give output only to users of the iFlowEdit via GUI. UI sub-component is responsible for creating outputs and passing them to the GUI. For example, after processing save action, UI sub-component informs GUI that whether the result is success or failure. Then GUI shows output to the user.

5.2.1.3. Processing Detail of User

As it is mentioned above, users give inputs to system via the GUI. There are four possible processing cycle.

First one is that if a user gives an input about activities, UI sub-component processes input, then uses Activity sub-component. Activity sub-component applies the changes to the activities, then uses Workflow sub-component on the account of the fact that every actions which are performed on the activities effects the workflow which contains that activities. Workflow sub-component applies the changes to workflow, then uses UI sub-component. Finally, UI sub-component gives the output to the user. The graphical representation of the use case is in Figure 6.

Second one is that if a user gives an input about connections, UI sub-component processes input, then uses Connection sub-component. Connection sub-component applies the changes to the activities, then uses Workflow sub-component on the account of the fact that every actions which are performed on the connections effects the workflow which contains that connections. Workflow sub-component applies the changes to workflow, then uses UI sub-component. Finally, UI sub-component gives the output to the user. The graphical representation of the use case is in Figure 7.

Third one is that if a user gives an input directly about workflow, UI sub-component processed input, then uses Workflow sub-component. Workflow sub-component applies the changes to workflow, then uses UI sub-component. Finally, UI sub-component gives the output to the user. The graphical representation of the use case is in Figure 8.

Fourth and the last one is that if a user give an input that concerns server side, UI sub-component processes input, then uses User Management sub-component. User Management sub-component interacts with Server component. After the interaction, it uses UI sub-component. Finally, UI sub-component gives the output to the user. The graphical representation of the use case is in Figure 9.

5.2.1.4. Dynamic Behavior of User

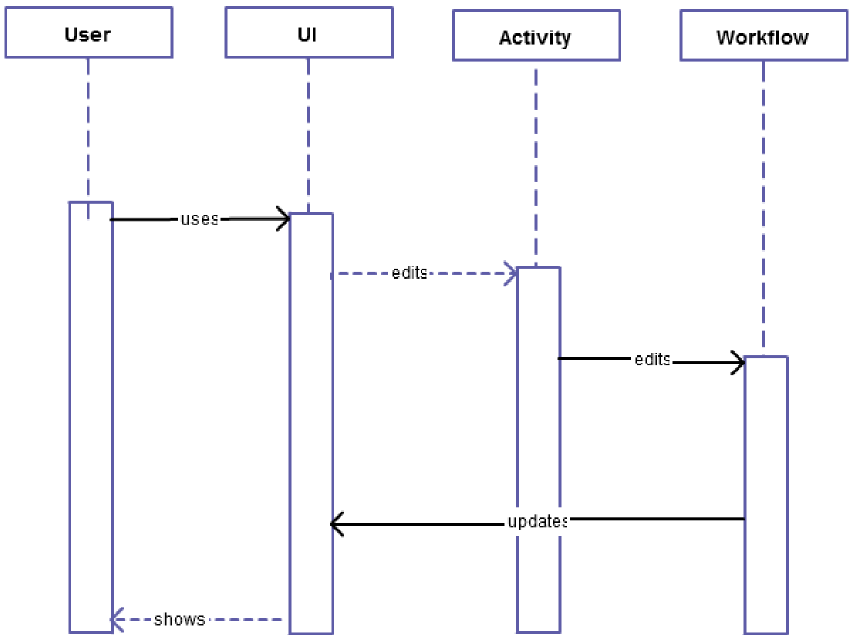


Figure 6 - Sequential Diagram of Use Case 1

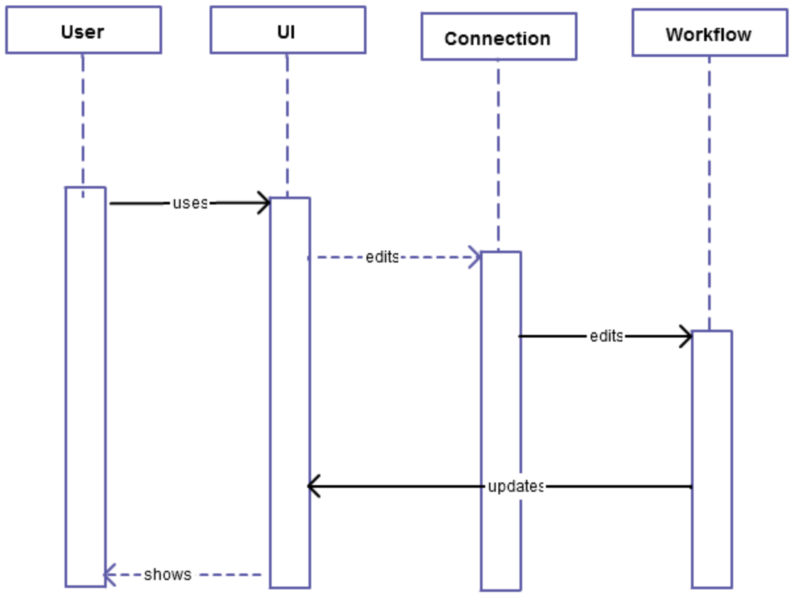


Figure 7 - Sequential Diagram of Use Case 2

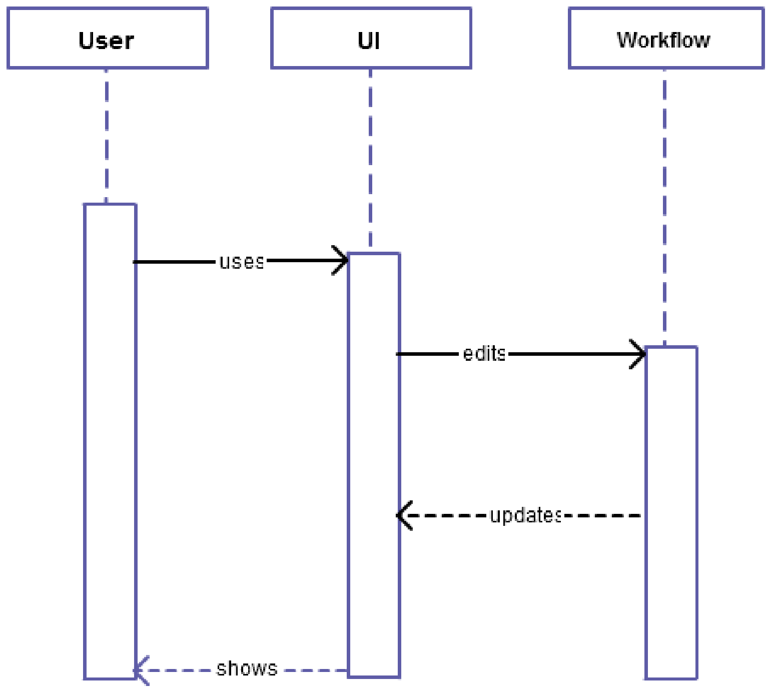


Figure 8 - Sequential Diagram of Use Case 3

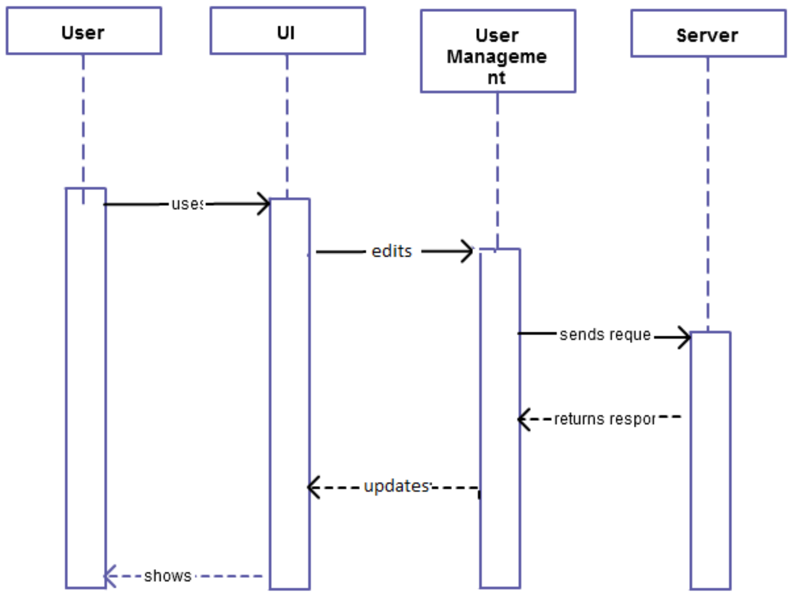


Figure 9 - Sequential Diagram of Use Case 4

5.2.2. Component Admin

5.2.2.1. Processing Narrative for Admin

Admin component is responsible for performing every possible action that admins of the iFlowEdit may do. These actions are the CRUD operations on users of the iFlowEdit. In other words, the actions are creating a user, reading information of a user, updating information of a user and deleting user. A user has three piece of information which are user name, email and password on the database and an admin can reach just two of them which are user name and email. However, on a create action, admin should provide a password for the user.

5.2.2.2. Interface Description of User

Admin component takes inputs from admins of the system via Admin GUI. The component which controls inputs that are taken from GUI is UI sub-component. For example, to create a user, an admin should click create user button, then should provide necessary information about the user, then should click okay button. All of these click events and the information are inputs which are given from the admin.

Admin component gives output to admins of the system via GUI. The component which controls outputs that GUI show to users is UI sub-component. For example, after an admin clicks get user button to read the information of a user, GUI shows the user to the admin. All of the results returned from UI sub-component are outputs of the system.

5.2.2.3. Processing Detail of Admin

Admin component has one possible processing cycle. When admin gives input to the system via Admin GUI, UI sub-component processes the input, then uses Admin Management sub-component. Admin Management sub-component interacts with Server component and gets a result from it. Admin

Management sub-component forwards that result to UI sub-component, then UI sub-component shows the output via Admin GUI. The graphical representation of the use case is in Figure 10.

5.2.2.4. Dynamic Behavior of Admin

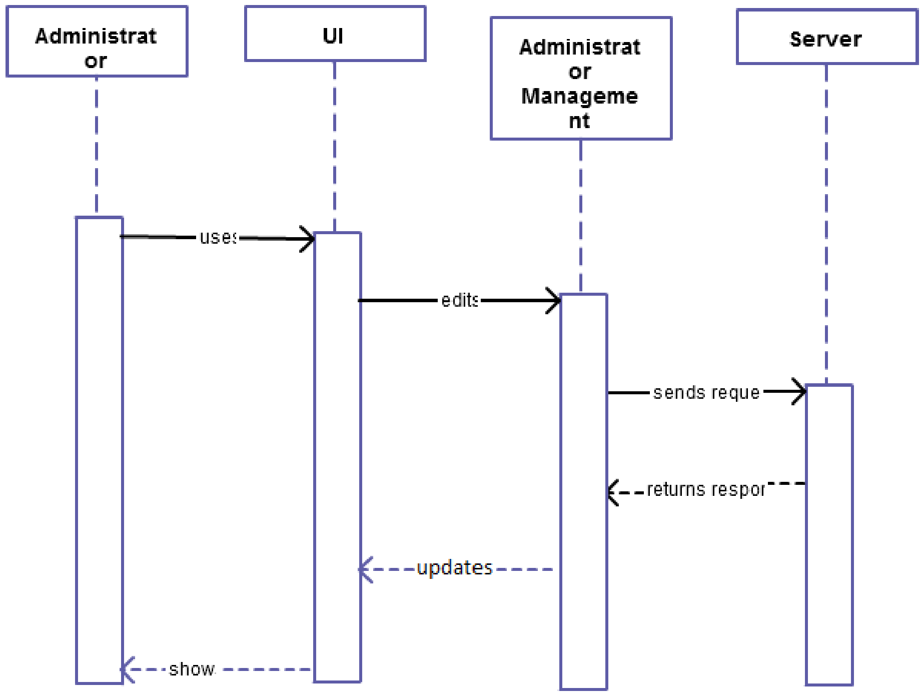


Figure 10 - Sequential Diagram of Administrator Use Case

5.2.3. Component Server

5.2.3.1. Processing Narrative for Server

Server component is responsible for storing all data which includes configuration files and documents which are created by users, holding the database, communicating with client side and sending responds as answers to requests.

5.2.3.2. Interface Description of Server

Server component takes inputs from other two components which are User component and Admin component. The sub-component that is responsible for taking inputs is Server Management. For example, if a read user request is sent from Admin component, Server Management sub-component processes that request first. All requests which are sent from User component and Admin component are inputs of the Server component.

Server component gives outputs to User component and Admin component. The sub-component that is responsible for giving output is Server Management. For example, after processing the input that is taken from Admin component, Server Management sub-component sends responds to Admin component. All responds which are sent from Server Management sub-component are outputs of the Server component.

5.2.3.3. Processing Detail of Server

Server component has four possible processing cycles.

First one is that if a user sends get document request, User component interacts with Server Management sub-component. Server Management sub-component uses File Controller sub-component and informs it about the request. File Controller sub-component uses Docs sub-component to get the document, then passed that file to Server Management sub-component. After that Server Management sub-component sends respond to User component. The graphical representation of the use case is in Figure 11.

Second one is that if a user sends edit or create document request, User component interacts with Server Management sub-component. Server Management passes the information to File Controller sub-component. File Controller sub-component, then, uses Docs sub-component to edit or create the document. After that it informs Server Management sub-component about the process. Finally, Server Management sub-component sends respond to User component. The graphical representation of the use case is in Figure 12.

Third one is that if a user sends get configurations request, User component interacts with Server Management sub-component. Server Management passes the information to File Controller sub-component. File Controller sub-component, then, uses Config File sub-component to get the configuration file, and then passed that file to Server Management sub-component. After that Server Management sub-component sends respond to User component. The graphical representation of the use case is in Figure 13.

Forth and the last one is that if an admin sends any request, Admin component interacts with Server Management sub-component. Server Management uses Database Controller to contact with Database. After that Server Management sends respond to Admin component. The graphical representation of the use case is in Figure 14.

5.2.3.4. Dynamic Behavior of Server

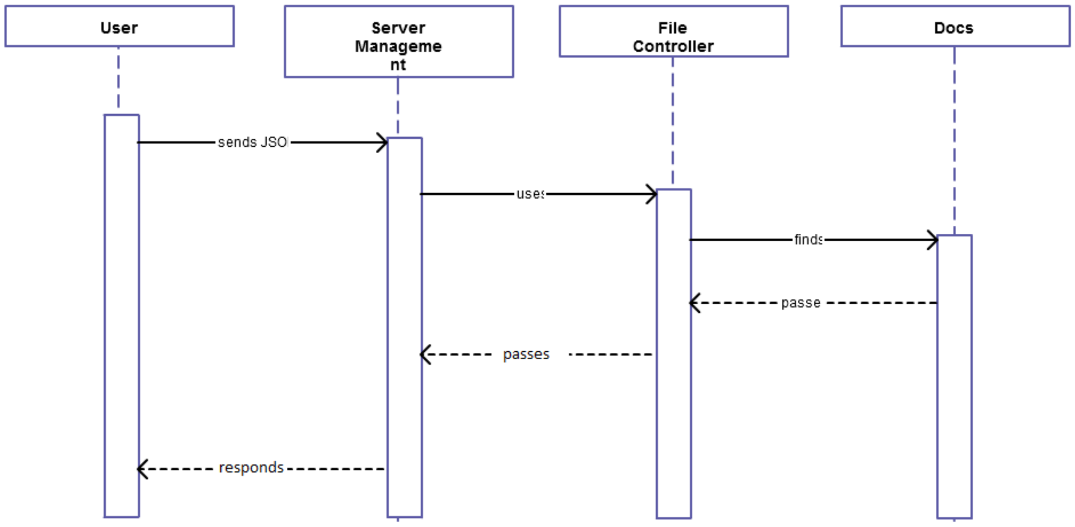


Figure 11 - Sequential Diagram of Server Use Case 1

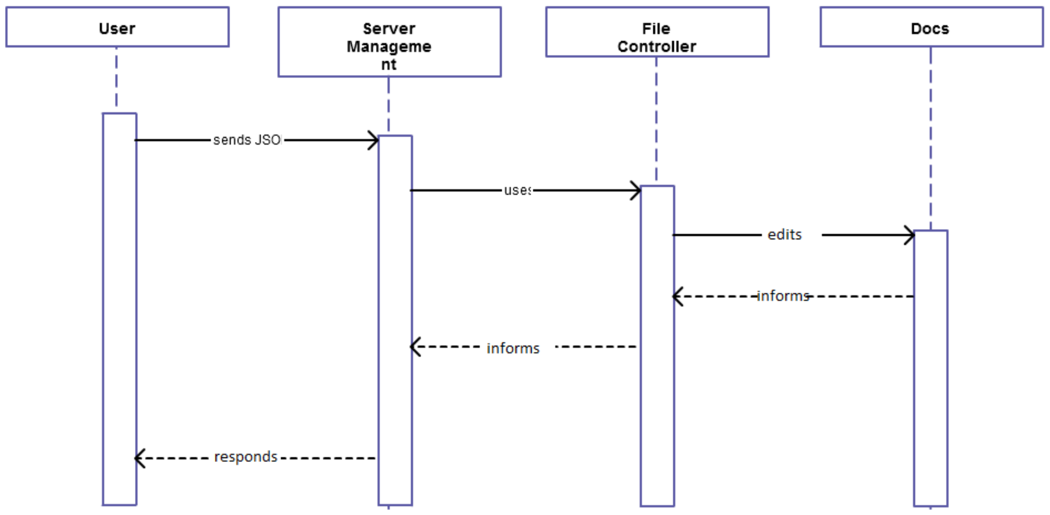


Figure 12 - Sequential Diagram of Server Use Case 2

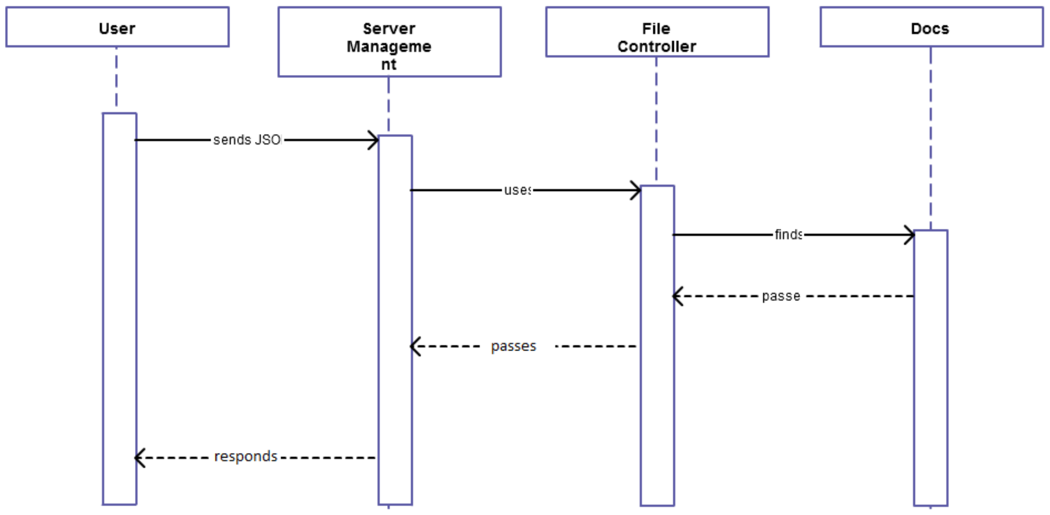


Figure 13 - Sequential Diagram of Server Use Case 3

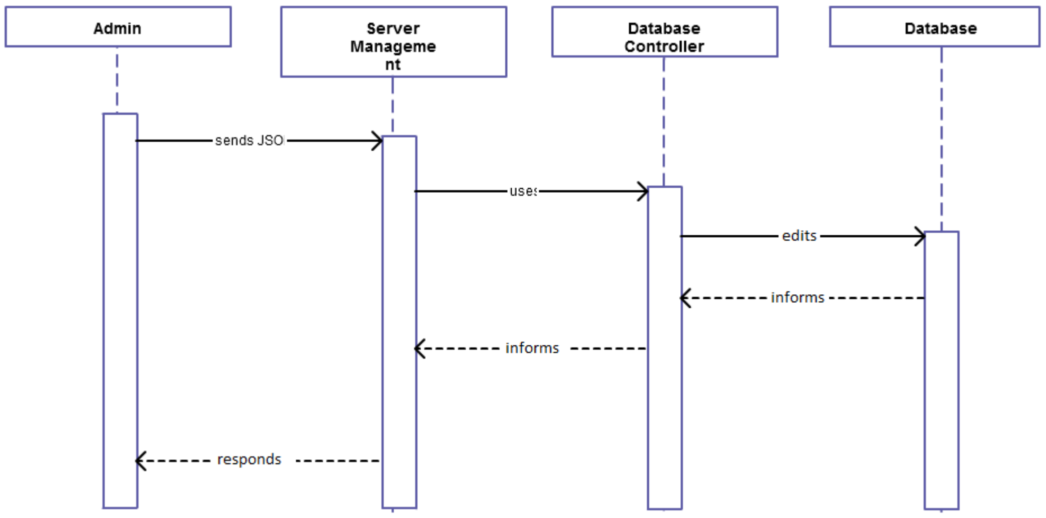


Figure 14 - Sequential Diagram of Server Use Case 4

5.3. Design Rationale

iFlowEdit is a web-based, platform dependent and HTML5 powered online diagram editor. On the account of these qualities of iFlowEdit, the system is separated to two different sub-systems which are client side and server side. Moreover, the system has two types of clients, which are users and admins. So, client side is also separated to two different sub-systems. As a result, system is separated to three components which are User Component, Admin Component and Server Component.

One approach is to separate database from server, but on the account of the fact that database of the system is not large, it is contained by Server component.

Another approach is to separate system to two sub-systems which are Client Component and Server Component. However, this approach makes client side hard to be handled.

6. User Interface Design

6.1. Overview of User Interface

The interaction between the user and the game is done via user interface. Project consists of five main screens. At first step, admin adds users to the system. It is a closed circuit system, which means that none of the users can register or introduces themselves to the system on their-own. If users can log in to the system successfully after be introduced, the interaction between system is started. They can design diagrams with the mouse controls and keyboard key combinations. Users are able to store their diagrams in the server, and open them later on. If they sign-out, the whole interaction will be over.

6.2. Screen Images

6.2.1 User Login Screen

The first step when user opens the iFlowEdit is entering login information in screen which is shown in Figure 15. If given info is correct, user redirects to the design area screen, else a warning is written to the screen.

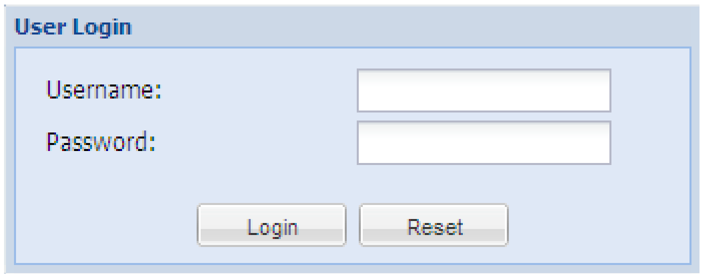


Figure 15 - User Login Screen

6.2.2 Document Selection Screen

In this screen, which is shown in Figure 16, user can select document, which is saved in advance, to act on. A document can be deleted, opened or updated here. If user decides to open the document, user will be redirected to design area screen.

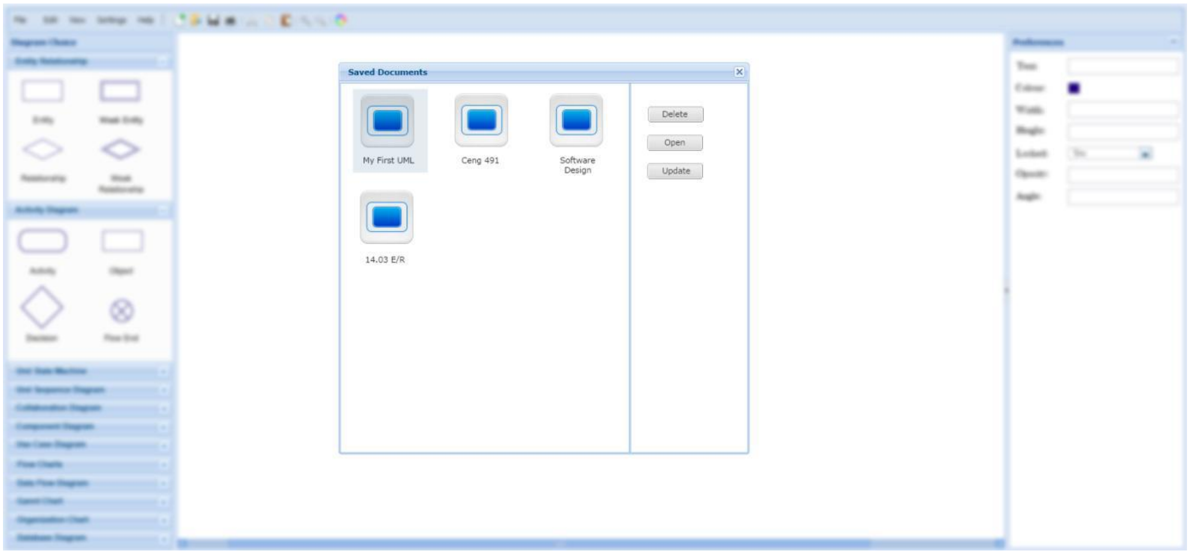


Figure 16 - Document Selection Screen

6.2.3 Design Area Screen

Most of the front-end process is taken in this screen which is shown in Figure 17. User can design the diagram; change any property of it and its children. In addition, user can save and print out the document; zoom-in and zoom-out on the canvas element; change the settings of the document; get a help here.

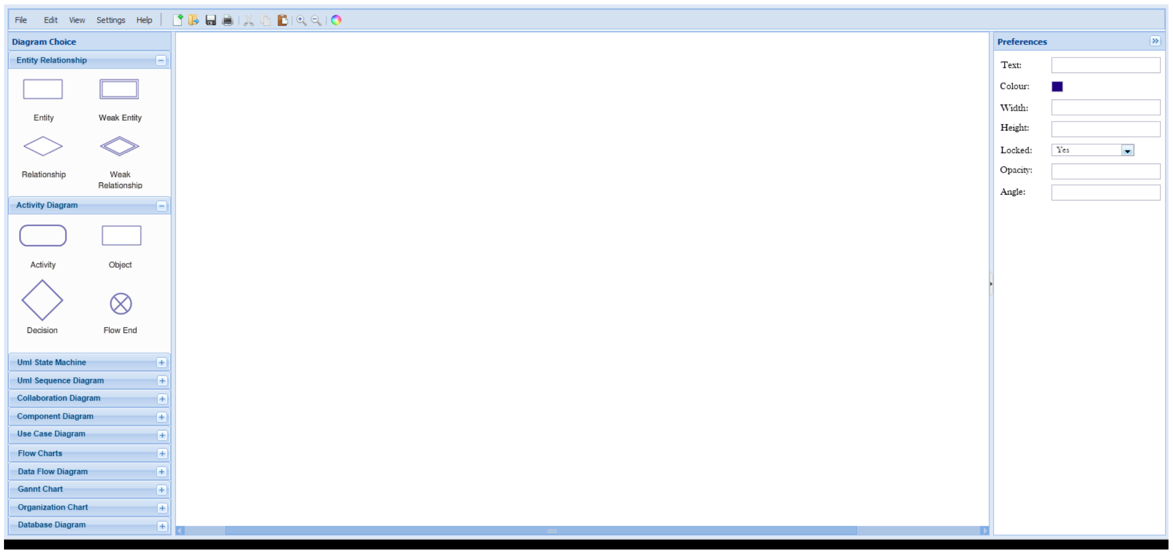


Figure 17 - Design Area Screen

6.2.4 Admin Login Screen

The first step when admin opens the iFlowEdit is entering login information in the screen shown in Figure 18. If given info is correct, admin redirects to the admin area screen, else a warning is written to the screen.

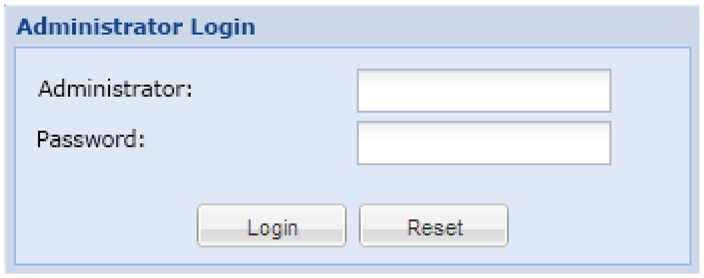
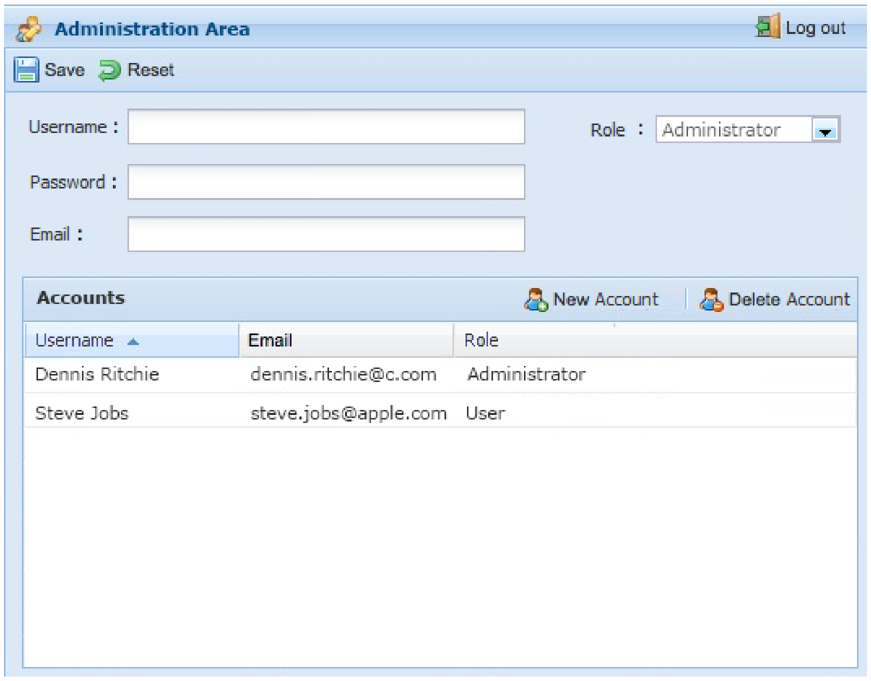


Figure 18 - Administrator Login Screen

6.2.5 Admin Area Screen

In this screen, shown in Figure 19, admin can add new user or administrator to the system; sees, deletes and updates accounts' details. Admin can log out via Log out button.



Username	Email	Role
Dennis Ritchie	dennis.ritchie@c.com	Administrator
Steve Jobs	steve.jobs@apple.com	User

Figure 19 - Administrator Area Screen

6.3. Screen Objects and Actions

The interaction is shown in Figure 20 from the beginning of the admin log in and user log in.

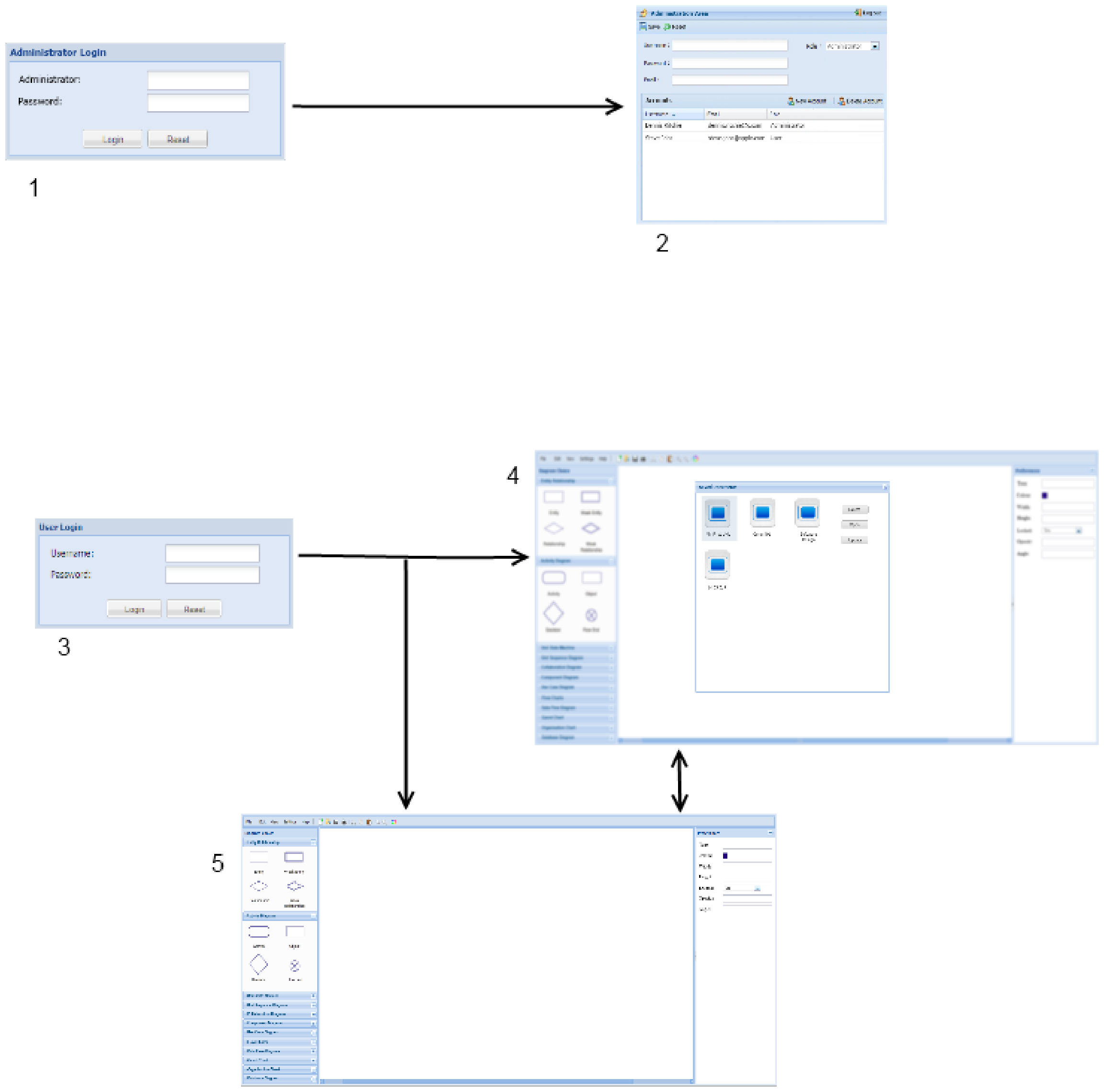


Figure 20 - Screen Objects

The interface denoted with 1 is administrator login; administrator clicks the login button and then goes to admin area, which is denoted by 2. In screen 2, administrator makes what is required then logs out via log out button.

The interface denoted with 3 is user login; user clicks the login button and then goes to whether document selection screen, which is denoted by 4, or design area screen, which is denoted by 5. These last two screens are interchangeable which means that user can pass through from one to another.

7. Detailed Design

7.1. User Component

Classification**:** This component is contained by the user package of the system.

Definition**:** This component of the system is created to provide functionalities for users of the system. All of the user actions will be handled in this component of the system.

Responsibilities**:** The major responsibility of user component is to provide user to do all of the actions on the screen. This component provides user interface to communicate with users and all of the user savings in this component will be hold in the server component of the system. The services that user component provides to the users were explained in the system architecture section.

Constraints**:** The latency caused by the server subsystem is the one of the most important constraints of the system. Data operations take several seconds when there is a huge data load operations caused by the user work flow data. The other important constraint is to wait to make data conversion between two formats, XAML to JSON in server side. This will take several seconds when there exists a huge amount of data to convert.

There would not any storage constraint on user component because this component will run on the user browsers. If user device is capable to run a compatible browser then the user component will run successfully.

In case of network failures, because of the fact that connection is not provided, user management component could not take response from server subsystem and throws connection error message.

***Composition*:** User component has five main subcomponents, which are activity, connection, workflow, UI and user management.

Activity subcomponent is to provide users capability to create activities on the main screen of the system.

Workflow subcomponent is the main component of the user component which holds all of the created activity and connection instances in its memory storage until the save button is pressed.

Connection subcomponent is to provide users capability to create connections between activities on the main screen of the system.

UI subcomponent is the user interface of the user component which provides user and computer interaction. The main screen of the system is created in this component.

User management subcomponent is an interface component of the user component which is designed to create secured communication between user component and server component. Only user management component can interact with outside of the user component.

***Uses & Interactions*:** There is an interaction between user component and server component. All of the user actions which are login, logout, document selection, saving document will be handled with this interaction. For example, user component creates communication with server management subcomponent of server component to reach configuration subcomponent to save the data that was created in the workflow subcomponent of the user component. On the contrary, when user opens a saved document, workflow subcomponent of the user component will interact with server component and request to send the saved data to load into the workflow subcomponent of user component.

There is no direct interaction between user component and administrator component. However, administrator component can make changes on user personal information by using server component.

***Resources*:** The resources used by this component are memory and CPU of the user personal device. In addition to these, Ethernet card is also used to communicate with server. User component firstly access to the server and get the resultant data into the memory of the user device.

***Processing*:** Processing details of this component has been explained in user component part of system architecture section.

***Interface/Exports*:** The user component of the system provides user clients to use the system functions. To achieve this purpose, there exist five classes connected properly. Explanations and details of these classes are mentioned in Data Dictionary section - 4.2.

7.1.1. Activity Subcomponent

Activity subcomponent is to provide users capability to create activities on the main screen of the system. A user cannot access this component directly. When an activity on the main screen is created, activity object will be also created and send to the workflow subcomponent of user component. So, all changes on activity subcomponent will be handled by using user interface that is provided to users. There are two classes in this subcomponent which are ActivityInstance and ActivityDefinition. Data properties and explanations of classes of activity subcomponent are explained on Data Dictionary section - 4.2. In Figure 21, two classes are shown with their relation. ActivityInstance class inherits the properties of ActivityDefinition which is to provide default activity object before a new activity is created. After creating a new activity, ActivityInstance is created which inherits the properties of ActivityDefinition.

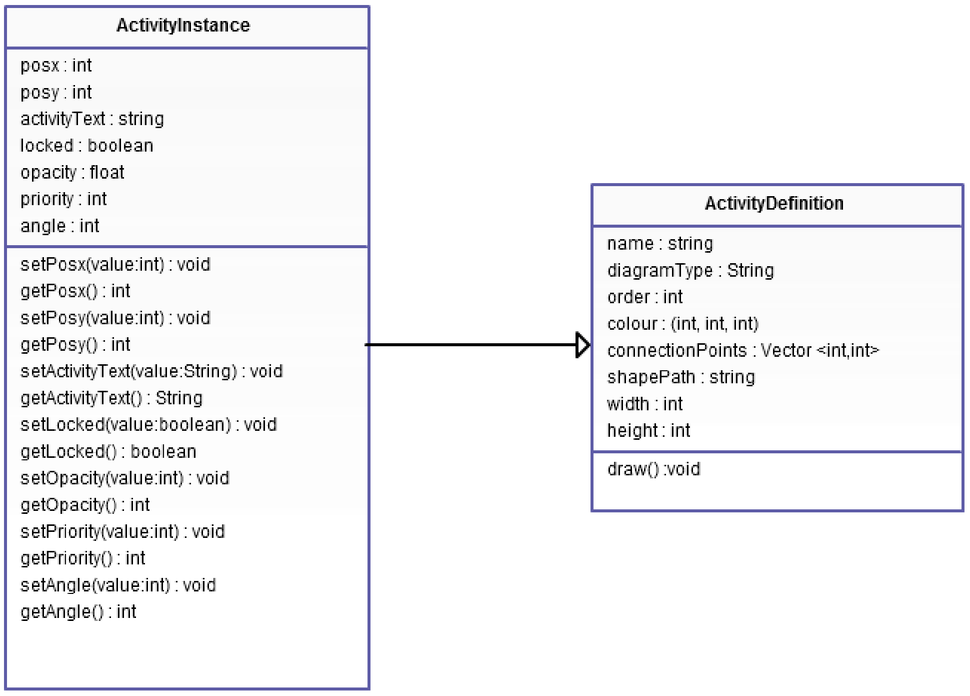


Figure 21 - Activity Diagram

7.1.2. Connection Subcomponent

Connection subcomponent is to provide users capability to create connections between the activities on the main screen of the system. A user cannot access this component directly. When a connection on the main screen is created, connection object will be also created and sent to the workflow subcomponent of user component. So, all changes on connection subcomponent will be handled by using user interface that is provided to users. There are two classes in this subcomponent which are ConnectionInstance and ConnectionDefinition. Data properties and explanations of classes of connection subcomponent are explained on Data Dictionary section - 4.2. In Figure 22, two classes are shown with their relation. ConnectionInstance class inherits the properties of ConnectionDefinition which is to provide default connection object before a new connection is created. After creating a new connection, ConnectionInstance is created which inherits the properties of ConnectionDefinition.

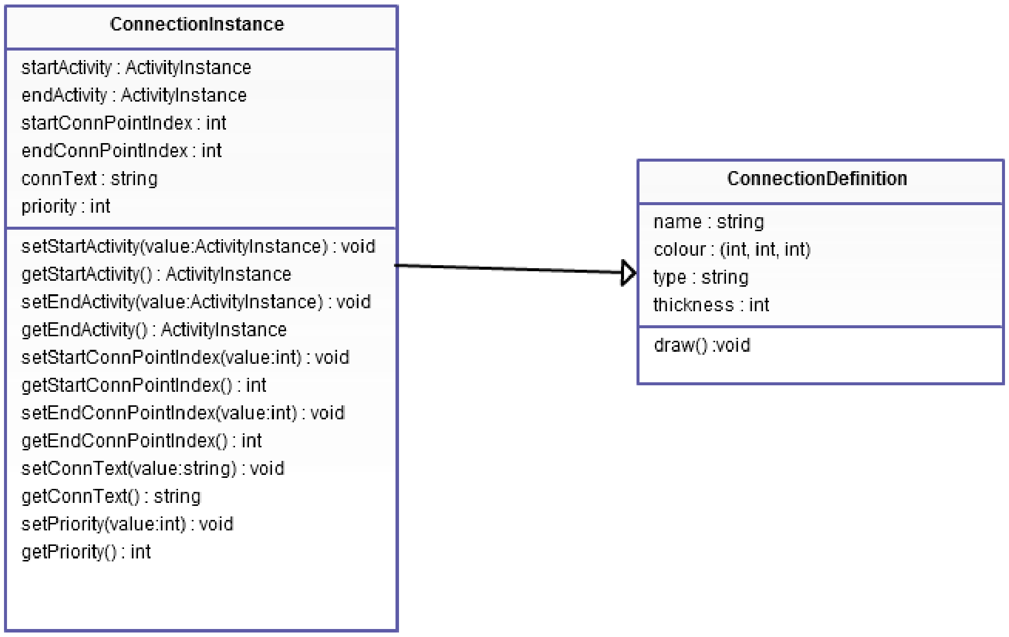

ConnectionInstance
startActivity : ActivityInstance
endActivity : ActivityInstance
startConnPointIndex : int
endConnPointIndex : int
connText : string
priority : int
setStartActivity(value:ActivityInstance) : void
getStartActivity() : ActivityInstance
setEndActivity(value:ActivityInstance) : void
getEndActivity() : ActivityInstance
setStartConnPointIndex(value:int) : void
getStartConnPointIndex() : int
setEndConnPointIndex(value:int) : void
getEndConnPointIndex() : int
setConnText(value:string) : void
getConnText() : string
setPriority(value:int) : void
getPriority() : int
ConnectionDefinition
name : string
colour : (int, int, int)
type : string
thickness : int
draw() :void

Figure 22 - Connection Diagram

7.1.3. Workflow Subcomponent

Workflow subcomponent holds all of the created activity and connection instances in its data structure until the save button is pressed. After saving, all content of the component will be loaded into the server component. The main purpose of this subcomponent is not to save each change made by the user into the server. It increases the performance of the system and avoids unnecessary network traffic. There is a class in this subcomponent, named Workflow. Data properties and explanations of workflow class of workflow subcomponent are explained on Data Dictionary section - 4.2. In Figure 23, this class is shown.

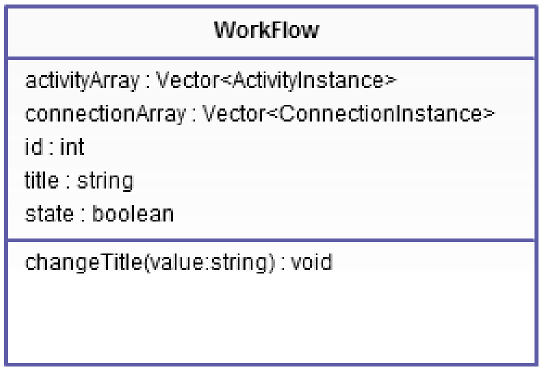

WorkFlow
activityArray : Vector<ActivityInstance>
connectionArray : Vector<ConnectionInstance>
id : int
title : string
state : boolean
changeTitle(value:string) : void

Figure 23 - Workflow Diagram

7.2. Administrator Component

Classification**:** This component is contained by the administrator package of the system.

Definition**:** This component of the system is created to provide functionalities for administrators of the system. All of the administrators' actions will be handled in this component of the system.

Responsibilities**:** The major responsibility of administrator component is to provide administrator to do all of the actions on the screen. This component provides user interface to communicate with administrators and all of the actions done in this component will be hold in the server component of the system. The services that administrator component provides were explained in the system architecture section.

Constraints**:** The latency caused by the server subsystem is the one of the most important constraints of the system. Because of using same network with user subsystem, data operations can take several seconds when there is a huge data load operations on the user subsystem.

In case of network failures, because of the fact that connection is not provided, administrator management component could not take response from server subsystem and throws connection error message.

Composition: Administrator component has two main subcomponents, which are UI and administrator management.

UI subcomponent is the user interface of the administrator component which provides administrator and computer interaction. The screen of the administrator system is created in this component.

Administrator management subcomponent is an interface of the administrator component which is designed to create secured communication between administrator component and server component. Only administrator management component can interact with outside of the administrator component.

Uses & Interactions: There is an interaction between administrator component and server component. All of the administrator actions which are login, logout, user selection, saving user information, creating and deleting users will be handled with this interaction. There is no direct interaction between administrator components with user component. Besides there are no side effects or any known anticipated sub components.

Resources: The resources used by this component are memory and CPU of the user personal device. In addition to these, either Ethernet port or any connection port should be used to communicate with server. Administrator component firstly access to the server and get the resultant data into the memory of the administrator device.

Processing: Processing details of this component has been explained in administrator component part of system architecture section.

Interface/Exports**:** The administrator component of the system provides administrators to use the system functions. To achieve this purpose, there exist two classes connected properly. Explanations and details of these classes are mentioned in Data Dictionary section - 4.3.

7.3. Server Component

Classification**:** This component is contained by the server package of the system.

Definition**:** This component of the system is created to provide functionalities for server of the system. All of the server actions will be handled in this component of the system.

Responsibilities**:** There are two main responsibilities of the server. First one is to store all the data except caches of the users and admins. This data includes the configuration files, saved work flow information files and relational data which hold the general information about users and administrators. The other responsibility is to send responses to the request that users and administrators send.

Constraints**:** There are two constraints that affect the server component most.

First one is the storage capacity of the server. In case the number of users and administrators is very large, the capacity of the storage has a huge importance on the account of the fact that there must be enough space to store all the data on the server.

Second one it the network connection that provides data transformation between server component and user component and data transformation between server component and administrator component. Without network connection, server management sub-component cannot manage to receive requests and send responses to those requests.

Composition**:** User component has six main sub-components, which are docs, config file, file controller, database controller, database and server management.

Docs sub-component stores the documents which are saved to the system by users. These documents hold the necessary information about the workflow of the user. Each document holds the information of just one workflow.

Config file sub-component stores the configuration files which are holds the information of the activities.

File controller sub-component is responsible for the management of the configuration files and documents. In other words, this sub-component is like a pointer to docs sub-component and config file sub-component. For example, if there is a request for a file, this component finds and returns that file. This component avoids the direct connection to aforementioned sub-components.

Database sub-component stores the relational data about users and administrators. This relational data includes the user names and passwords of the users.

Database controller sub-component is responsible for the management of the database. In other words, this sub-component is like a pointer to database. When there is a case that should use the information in the database, database controller sub-component does the necessary work. This component avoids the direct connection to database sub-component.

Server management sub-component is responsible for taking requests from user component and administrator component, processing those requests, communicating with controller sub-components according to the requests and sending responses to user component and administrator component.

Uses & Interactions**:** Server component interacts with other two components which are user component and administrator component. These interactions, however, is just between management sub-components, the other sub-components do not have any direct role in interactions.

User component uses server component to validate the information which is entered at the user login screen, to get configuration files at the beginning and to save and load workflows. For all these operations, it sends requests to server component and acts according to the responses which come from server components.

Administrator component uses server component to validate the information which is entered at the administrator login screen and to add, remove and ban users. For all these operations, it sends requests to server component and acts according to the responses which come from server components.

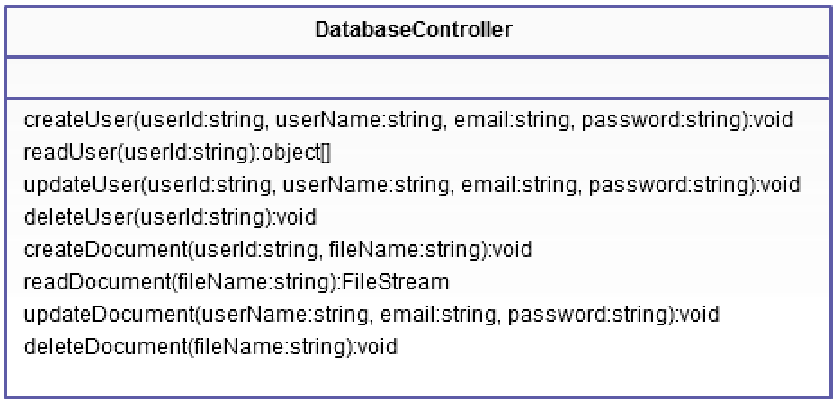
Resources**:** The resources used by server component are server computers with high performance RAMs and hard discs and Ethernet.

Processing**:** Processing details of this component has been explained in server component part of system architecture section.

Interface/Exports**:** Interfaces and exports of server component have classes are mentioned in Data Dictionary section 4.2.

7.3.1. Database Controller Subcomponent

As it is mentioned before, database controller sub-component is like a pointer to database and it is responsible for the management of the database. When there is a case that should use the information in the database, database controller sub-component does the necessary work. This component avoids the direct connection to database sub-component. This sub-component is represented by a class which is called database controller. Database controller class does not have any fields. It has just methods as it can be seen clearly in Figure 24. Each method is responsible for one of the four CRUD operations.



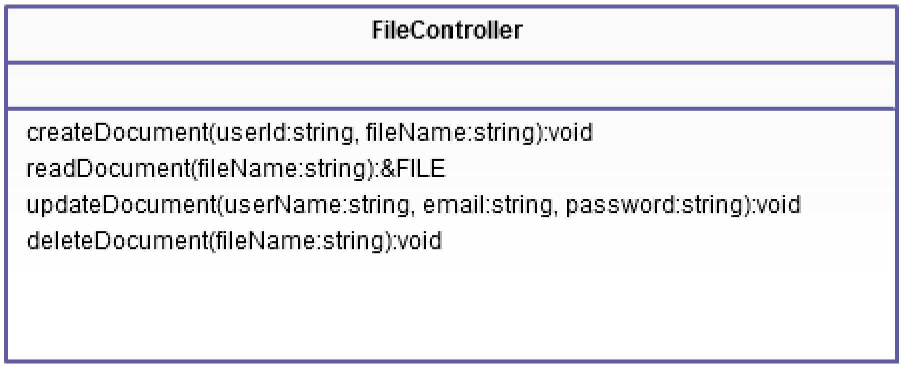
```
DatabaseController

createUser(userId:string, userName:string, email:string, password:string):void
readUser(userId:string):object[]
updateUser(userId:string, userName:string, email:string, password:string):void
deleteUser(userId:string):void
createDocument(userId:string, fileName:string):void
readDocument(fileName:string):FileStream
updateDocument(userName:string, email:string, password:string):void
deleteDocument(fileName:string):void
```

Figure 24 - Database Controller Diagram

7.3.2. File Controller Subcomponent

As it is mentioned before, file controller sub-component is like a pointer to docs sub-component and config file sub-component and it is responsible for the management of the configuration files and documents. This component avoids the direct connection to docs sub-component and config file sub-component. This sub-component is represented by a class which is called file controller. File controller class does not have any fields. It has just methods as it can be seen clearly in Figure 25. Each method is responsible for one of the four CRUD operations on the documents and just read operation on the configuration files.



```
FileController

createDocument(userId:string, fileName:string):void
readDocument(fileName:string):&FILE
updateDocument(userName:string, email:string, password:string):void
deleteDocument(fileName:string):void
```

Figure 25 - File Controller Diagram

8. Libraries and Tools

The technologies that will be used are HTML5, JavaScript, CSS, Ext-JS, Processing-JS, XAML and JSON. HTML5, CSS, JSON and XAML affect functionality indirectly, while JavaScript, Ext-JS and Processing-JS affect functionality directly.

HTML5 will be used to create everything on the user GUI and administrator GUI. Stencil set, workflow, properties box and buttons are some of them.

CSS will be used to give a better look and feel to the user GUI and administrator GUI. Colors, shapes and borders are some of them. [9]

JSON will be used as a message which is shuttled between client side, e.g., user and administrator, and server side. [4]

XAML will be used to store the information of the workflow. It will be used by INNOVA IT with WF.NET. [8]

Processing -JS will be used to draw any element on canvas and update them. [7]

JavaScript will be used to perform user and administrator actions. Specifically, JavaScript will be used to perform button clicks and key presses, to create JSON objects and send them to server, to get JSON objects from server, to change colors, shapes and other design features, to update properties box in the user GUI. [4]

Ext-JS will be used to build skeleton of the user GUI and administrator GUI. It will be also used to add drag and drop functionality to activities in the stencil set. Moreover, it will be used to give stencil set an accordion interaction. [6]

9. Time Planning (Detailed & Finalized Gannt Chart)

9.1. Term 1 Gannt Chart

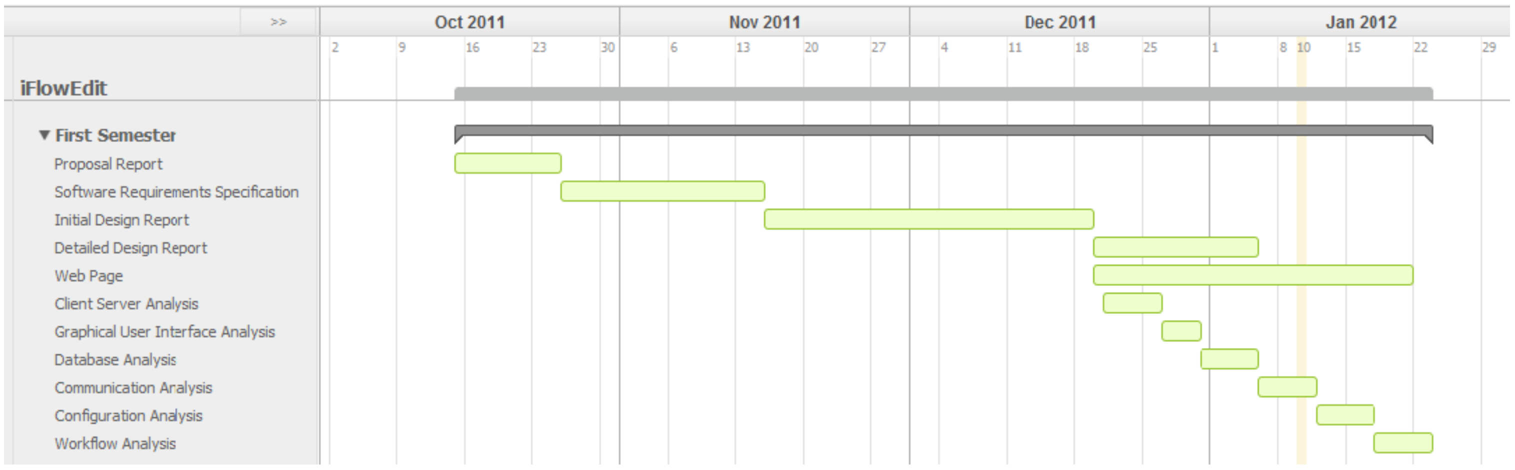


Figure 26 - Term 1 Gantt Chart

9.2. Term 2 Gannt Chart

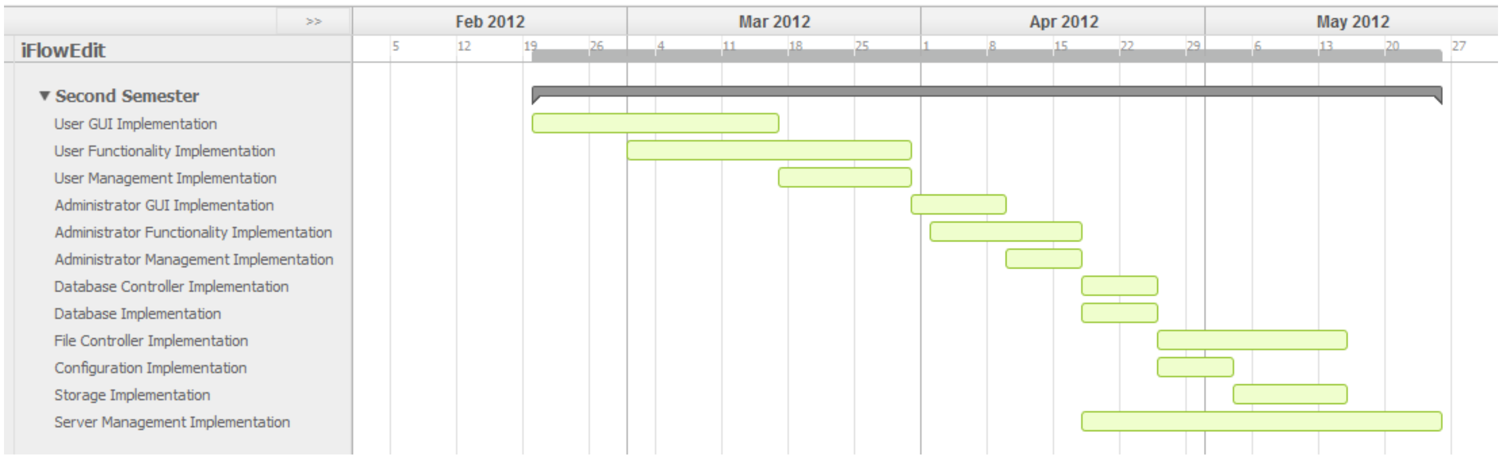


Figure 27 - Term 2 Gantt Chart

10. Conclusion

This Detailed Design Report has been intended to explain how the system will be structured to satisfy the requirements. System components, interfaces and data required for the implementation phase have been briefly described. Descriptions of the components are explained in detail. This final design document has the capability to guide programmers to write the system.